Enhancing AI Coding Agents for Data Science via Few-Shot Example Retrieval

Aaron Tamte
tamte.aa@northeastern.edu
Northeastern University

April 23, 2025

Abstract

This paper presents the design, implementation, and evaluation of a specialized Information Retrieval (IR) system aimed at enhancing AI coding agents within the data science domain. The system retrieves relevant code snippets and explanatory text from a corpus of high-performing Kaggle notebooks to provide targeted examples for few-shot prompting. Unlike general code search, the presented approach focuses on semantic relevance for specific tasks, aiming to improve the quality, correctness, and contextual appropriateness of code generated by AI agents, particularly for complex data science problems. Two retrieval strategies are compared: a baseline using raw code embeddings and an enhanced method leveraging LLM-generated annotations [8]. Evaluation using Precision@3 demonstrates the superiority of the annotation-based approach, highlighting the importance of semantic enrichment for effective few-shot example retrieval.

1 Introduction

AI coding agents show promise in automating data science tasks, but often struggle to generate high-quality, contextually appropriate code for complex problems without relevant examples. General code search frequently falls short, failing to provide the specific, nuanced guidance needed for effective few-shot prompting (e.g., retrieving specific feature scaling techniques when requested, not just any data loading code). To address this gap, this paper describes the design and implementation of a specialized Information Retrieval (IR) system aimed exclusively at enhancing the capabilities of AI coding agents operating within the data science domain. Specifically, the system retrieves relevant code snippets and explanatory text segments from a corpus of high-performing Kaggle notebooks. The primary goal is to provide targeted, high-quality examples suitable for **few-shot prompting** of the AI agent. By supplying contextually appropriate examples directly related to the agent's current task, the system aims to significantly improve the quality, correctness, and relevance of the code generated by the agent, particularly for complex tasks prevalent in Kaggle competitions.

The proposed IR system functions as an external knowledge source within a Retrieval-Augmented Generation (RAG) framework [3] for the coding agent. The agent identifies a need for external examples based on its current task and formulates a query describing the data science problem or technique it is addressing (e.g., "handling imbalanced data with SMOTE," [1] "time series forecasting using ARIMA"). It then invokes the IR system as a **tool** via a synchronous API call, passing the formulated query and awaiting the results before proceeding with code generation. The IR system processes the query, retrieves relevant notebook chunks (defined here as individual notebook cells), and returns a small set of top-ranked results.

The significance of the approach lies in its potential to make AI coding agents more effective and reliable. Large language models often struggle to generate accurate, complex code without relevant examples [7]. The system addresses the gap by grounding the agent's generation process in proven code patterns and explanations sourced from successful human-written notebooks, tailored specifically for few-shot learning.

2 Information Need, Relevance, and Results

2.1 Information Need and Queries

The information need arises from the AI agent's requirement for illustrative examples to complete its current coding task. Queries, formulated by the agent itself, are natural language descriptions of data science concepts, techniques, or challenges. Examples include:

- "Implement K-Fold cross-validation"

- "Feature scaling techniques for numerical data"
- "Using TensorFlow for image classification"
- "Optimize XGBoost hyperparameters"

The system assumes these agent-generated queries provide reasonably clear descriptions of the target task. It processes these queries using vector embeddings (via `all-MiniLM-L6-v2` [5]) to capture semantic meaning.

2.2 Relevance Criteria

Relevance is defined strictly by a snippet's suitability as a **few-shot example** for the agent [7]. A relevant result is a code or documentation chunk (notebook cell) that effectively demonstrates the concept or technique described in the agent's query. While assessment involves some inherent subjectivity, the key aspects considered are:

- **Task Alignment:** The chunk must directly address the specific data science concept or technique described in the agent's query.
- **Clarity and Conciseness:** The example should be presented in a clear, understandable, and reasonably concise manner suitable for prompting.
- **Implicit Correctness:** Sourced from high-performing notebooks, the code is *assumed* (though not explicitly verified) to be largely correct for the demonstrated task.
- **Few-Shot Suitability:** The chunk must appear useful for inclusion in a prompt to guide the agent, often implying it is relatively self-contained and clearly illustrates the target pattern or technique.

Given the goal of few-shot prompting, the system prioritizes precision over recall. High precision in the top results is prioritized, accepting that some niche but relevant snippets might be missed, to ensure the agent receives highly reliable examples for prompting. The objective is to find a small number of highly relevant examples.

2.3 Results Organization

The system returns results as a **ranked list** based on semantic similarity (Euclidean distance between query and chunk embeddings). Consistent with the few-shot prompting objective, the agent is expected to primarily utilize the top few results. Therefore, the design focuses on delivering **three high-quality examples** per query.

3 Methodology

3.1 Data Corpus

Kaggle notebooks served as the source corpus, chosen for their relevance to data science and available performance indicators (votes). To ensure quality, we implemented a strict filtering criterion: only notebooks with 10 or more votes were included in our corpus. The threshold served as a community-validated quality signal, helping eliminate low-effort implementations while retaining notebooks that demonstrated proven utility to data scientists. We acknowledge that vote counts might not represent uniform quality across competitions of vastly different scales, but consider it a practical heuristic for the project's scope. A total of 1,646 notebooks were initially acquired via the Kaggle API that met this vote criterion.

3.2 Preprocessing and Annotation

The preprocessing pipeline operated on the corpus of acquired notebook content, which was persisted in an SQLite database. Our chunking strategy is cell-oriented: we associate individual code cells with their immediately preceding markdown context to create more semantically complete units. For instance, a markdown cell explaining a technique like SMOTE is combined with the immediately following code cell implementing it to form a single, context-rich chunk. The approach aims to capture both the explanatory narrative and the practical implementation. During chunking, code cells falling below a configurable token threshold (50 tokens) were excluded to filter out potentially uninformative snippets. The resulting chunks, containing the code content, associated context, and provenance information (notebook ID, cell index), were stored in a structured database, yielding a final corpus of 16,762 processed chunks for retrieval.

Subsequently, an automated annotation process enriched the chunk database. The process utilized the WizardCoder-Python-13B-V1.0 Large Language Model (LLM) [6] to generate structured metadata for each chunk, including a concise summary, relevant keywords, example natural language queries for which the chunk would be relevant, and a textual assessment indicating the chunk's suitability as a high-quality few-shot example (stored within the 'annotationContext' field). Annotation generation took approximately 16 hours to complete for the entire corpus on an NVIDIA RTX 3090 GPU.

Below is an example of a code chunk (ID: 16145) and its corresponding LLM-generated annotation:

Code Content:

```
features = [x for x in X_train.columns if x not in col_del]
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
folds = 3
kf = KFold(n_splits = folds, shuffle = True, random_state=seed)
y_preds11 = np.zeros(X_test.shape[0])
i = 0
for tr_idx, val_idx in kf.split(X_train, y_train):
    i+=1
    clf = xgb.XGBClassifier(
        n_estimators=800,
        max_depth=9,
        learning_rate=0.03,
        subsample=0.9,
        colsample_bytree=0.9,
        tree_method='gpu_hist'
    )

    X_tr = X_train[features].iloc[tr_idx, :]
    y_tr = y_train.iloc[tr_idx]
    clf.fit(X_tr, y_tr)
    del X_tr
    y_preds11+= clf.predict_proba(X_test[features])[:,1] / folds
    if debug:
        print("debug:",roc_auc_score(y_test, clf.predict_proba(X_test[features])[:,1] / folds)
    )
    del clf

gc.collect()
if debug:
    print("debug:",roc_auc_score(y_test, y_preds11))
    print("debug:",roc_auc_score(y_test, y_preds11*0.5 + y_preds*0.5))
```

Annotation Context:

```
{
  "annotation": "This code demonstrates how to perform cross-validation using the XGBoost
    classifier in Python.",
  "keywords": [
    "cross validation",
    "xgboost",
    "classification",
    "machine learning",
    "roc auc score"
  ],
  "example_queries": [
    "how to perform cross validation with xgboost",
    "example of kfold cross validation in python",
    "using roc auc score for model evaluation"
  ],
  "useful": "yes"
}
```

3.3 Indexing and Retrieval Model

Vector space modeling formed the core of the retrieval mechanism. For both the naive and custom implementations, dense vector representations (embeddings) of the selected textual units (either raw code or synthesized annotations) were generated using the `all-MiniLM-L6-v2` sentence transformer model [5]. The model was chosen for its balance of performance and computational efficiency in capturing semantic meaning. To facilitate efficient retrieval, these embeddings, along with associated metadata, were stored and indexed using the LanceDB vector database [4]. LanceDB manages the creation of vector indices internally, enabling rapid nearest neighbor searches based on Euclidean (L2) distance by default. Such a setup allows the system to efficiently

identify the stored chunks whose embeddings exhibit the highest similarity to the embedding of a given user query.

3.4 Naive Implementation

The baseline or "naive" implementation focused solely on the raw code content for retrieval. For each chunk identified during preprocessing (Section 3.2), an embedding was generated using the `all-MiniLM-L6-v2` sentence transformer model applied directly to the code content. The preceding markdown context and the LLM-generated annotations were disregarded for the baseline approach. The resulting code embeddings, along with associated chunk metadata, were then ingested into a LanceDB table. Retrieval involved embedding the user query using the same `all-MiniLM-L6-v2` model and performing a vector similarity search within LanceDB to find the chunks with the closest code embeddings.

3.5 LLM Annotated Implementation

The second implementation explored a strategy centered on semantic enrichment through auxiliary metadata. The LLM annotated approach leveraged the structured annotations previously generated by a Large Language Model for each code chunk (detailed in Section 3.2). Rather than relying solely on the source code, the LLM annotated method synthesized a textual representation from the annotation data, encompassing elements such as the descriptive summary, curated keywords, and illustrative example queries. The underlying hypothesis was that embeddings derived from the semantically rich, purpose-oriented text would yield superior retrieval performance compared to embeddings generated from the raw code. Consequently, the `all-MiniLM-L6-v2` sentence transformer model was applied to these synthesized annotation texts to produce embeddings. These annotation-derived embeddings were subsequently stored in a LanceDB table, along with associated chunk metadata, forming the basis for similarity search in this LLM annotated configuration.

3.6 Limitations

A key limitation is the reliance on Kaggle votes as a proxy for notebook quality. While practical for the project's scope, votes do not guarantee code correctness, reusability, or adherence to best practices. Furthermore, the current cell-based chunking strategy may not adequately capture dependencies in code that spans multiple cells. Additionally, the system lacks mechanisms for code verification or ensuring that retrieved snippets are compatible with the agent's specific runtime environment and dependencies; a more robust system might incorporate static analysis or environment checks.

4 Evaluation Metrics

The primary evaluation metric is **Precision@3 (P@3)** [9]. The metric directly aligns with the goal of providing three high-quality few-shot examples to the AI agent. P@3 measures the proportion of the top 3 retrieved results that are judged relevant according to the criteria outlined in Section 2.2. The focus on the top-k results is appropriate because the agent will likely only use a small number of examples for its few-shot prompt. Metrics focusing on recall (e.g., R-Precision, MAP) are less critical here, as retrieving *all* potentially relevant snippets is not the primary objective.

5 Evaluation

5.1 Evaluation Setup

A quantitative evaluation was conducted to assess the effectiveness of the retrieval system, particularly comparing the baseline (code-based embedding) and LLM annotated (annotation-based embedding [8]) approaches. The evaluation involved constructing a representative set of natural language queries mirroring the expected input from an AI coding agent operating in the data science domain. Example queries focused on common tasks like data cleaning, feature engineering, model training, and evaluation (e.g., "apply SMOTE for imbalanced classification" [1], "scale features using `MinMaxScaler`", "train a `RandomForestRegressor`").

For each query in the test set, manual relevance judgments were performed by assessing the suitability of retrieved chunks as few-shot examples [7], adhering to the criteria outlined in Section 2.2 (Task Alignment, Clarity, Conciseness, Few-Shot Suitability). Given the focus on providing a small number of high-quality examples for prompting, the primary metric chosen was Precision@3 (P@3), as established in Section **??**. The P@3 score was calculated for both the baseline and LLM annotated retrieval implementations across the entire

query set. The approach allowed for a direct comparison of their ability to rank highly relevant few-shot examples within the top 3 results.

5.2 Results

The evaluation yielded the following aggregate Precision@3 (P@3) scores across the 10 test queries:

- **Baseline Implementation (Code Embedding) P@3: 0.4667**
- **LLM Annotated Implementation (Annotation Embedding) P@3: 0.6667**

The LLM annotated implementation significantly outperformed the baseline approach, achieving a P@3 score approximately 43% higher (0.6667 vs 0.4667). Figure 1 provides a broader comparison including Precision@5 and MRR@5 metrics.

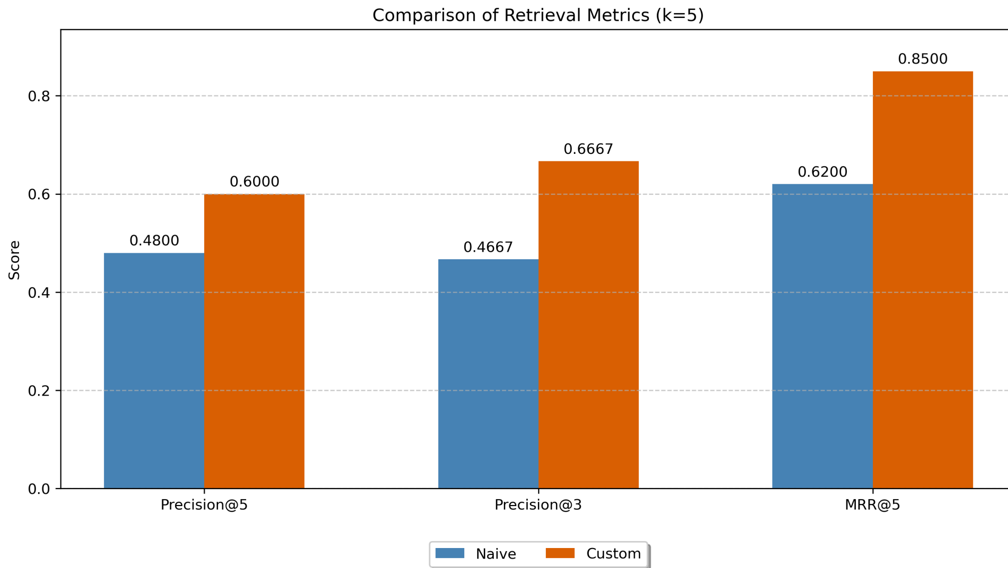

Figure 1: Comparison of aggregate retrieval metrics (Precision@5, Precision@3, MRR@5) for the Baseline (Naive) and LLM Annotated (Custom) systems across all test queries. The LLM Annotated approach achieved higher Precision@3 (0.6667 vs 0.4667).

Table 1 presents a detailed breakdown of Precision@3 scores for each individual query, highlighting specific areas where each retrieval approach performed well or struggled. Figure 2 visually contrasts these per-query results.

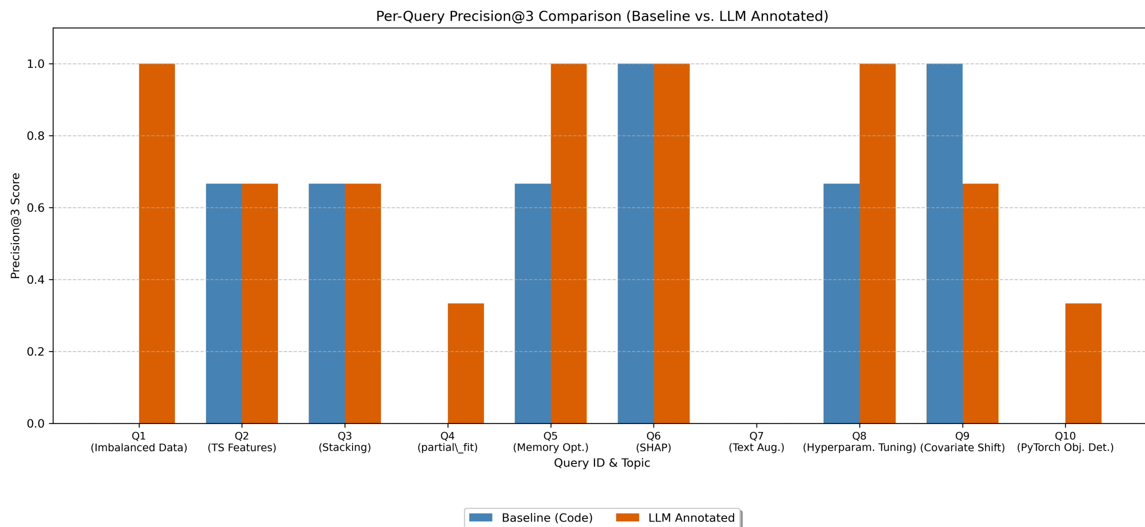

Figure 2: Comparison of Precision@3 scores for each individual query for the Baseline and LLM Annotated systems.

Table 1: Precision@3 Scores per Query for Baseline and LLM Annotated Systems

Query	Baseline P@3	LLM Annotated P@3
How to handle highly imbalanced data in financial fraud detection?	0.000	**1.000**
Show feature engineering techniques for time series demand forecasting.	0.667	0.667
Implement model stacking for a classification task.	0.667	0.667
Use `scikit-learn` `partial_fit` for incremental learning on large datasets.	0.000	**0.333**
Best practices for optimizing memory usage when reading large CSV files with `pandas`.	0.667	**1.000**
How to interpret XGBoost or LightGBM models using SHAP values?	1.000	1.000
Apply text augmentation techniques for an NLP classification task.	0.000	0.000
Perform hyperparameter tuning using `Optuna` for a machine learning model.	0.667	**1.000**
How to handle covariate shift between training and test data?	**1.000**	0.667
Create an image processing pipeline with augmentations in PyTorch for object detection.	0.000	**0.333**

5.3 Analysis and Discussion

An analysis of queries where the baseline implementation performed poorly (achieving low P@3 scores, see Appendix) reveals several patterns:

1. **Retrieval of Thematically Related but Technically Incorrect Code:** For queries targeting specific techniques within a broader domain (e.g., handling data imbalance), the system often retrieved code snippets related to the general domain (like exploratory data analysis for fraud detection) but failed to retrieve examples demonstrating the *specific technique* requested. The pattern was particularly evident for Query 1 (`"apply SMOTE for imbalanced classification"`), where the top baseline results included general data loading and preprocessing steps from fraud detection notebooks, but actual SMOTE implementation snippets were ranked much lower or absent from the top 3. Relevant results, when present in the corpus, were often ranked outside the top 3, indicating the raw code embedding struggled to differentiate between the general topic and the specific method sought.

2. **Failure to Retrieve Specific Functions or Abstract Concepts:** For queries requesting the use of a particular function (e.g., `scikit-learn`'s `partial_fit`) or describing a more abstract process (e.g., "text augmentation techniques"), the top-ranked baseline results often contained code from the correct library or domain but did not actually implement the specified function or process. The behavior was observed in Query 4 (targeting `partial_fit`) and Query 7 (requesting "text augmentation techniques"), both of which yielded zero relevant results in the top 3 for the baseline system. For example, the query for text augmentation might retrieve general NLP setup code (like tokenizer loading) rather than code demonstrating specific augmentation methods like back-translation or synonym replacement. The observation suggests that embedding raw code alone may not effectively capture the semantics of specific, less commonly explicitly named functions or abstract processes without stronger contextual cues.

3. **Potential Over-reliance on Surface Keywords:** The errors suggest the baseline approach might be matching keywords present in the code (e.g., "fraud", "scikit-learn", "NLP", "PyTorch", "detection") without fully understanding the specific task or technique described in the natural language query. Such behavior contributes to the issues seen in Q1 (matching "fraud" but not `SMOTE` [1]) and Q4 (matching concepts related to large datasets but not `partial_fit`). Similarly, for Query 10 (PyTorch object detection), the baseline results included general object detection setup or unrelated framework code (TensorFlow/Keras) instead of the requested PyTorch pipeline. For a query like `"Optimize XGBoost hyperparameters"` (Q8, where baseline P@3 was 0.667), while performance wasn't detrimental, there's a risk the system might return code that simply *imports* XGBoost or uses it with default parameters, rather than demonstrating

hyperparameter tuning techniques like `GridSearchCV` or `RandomizedSearchCV` (or potentially `Optuna` [2]), leading to superficially relevant but ultimately incorrect results.

A comparative analysis reveals distinct patterns where the LLM annotated approach [8] outperformed the baseline. The LLM annotated method excelled particularly on queries requiring specific techniques within broader domains, such as finding `SMOTE` implementations for imbalanced data (Q1) or locating `partial_fit` usage (Q4). In contrast, the baseline method often returned code related only to the general topic (fraud detection, large datasets). This result suggests the LLM-generated annotations (summaries, keywords, example queries) provided crucial semantic context. It allowed the model to differentiate between, for example, general hyperparameter tuning code and specific `Optuna` implementations (Q8), where the baseline method sometimes faltered.

Conversely, both methods struggled with the "text augmentation" query (Q7), indicating potential gaps in the corpus or the annotations for the specific NLP technique. For queries with very distinctive code patterns or keywords, like memory optimization using `reduce_mem_usage` (Q5) or SHAP value calculation (Q6), both methods performed well. However, the LLM annotated results were often slightly more comprehensive or precisely targeted. The baseline method occasionally performed marginally better when highly specific code identifiers were present, as seen potentially with "covariate shift" and the `relax_data` function (Q9).

Overall, the comparison strongly supports the hypothesis that leveraging semantic annotations significantly improves retrieval relevance for task-oriented few-shot prompting compared to relying solely on raw code embeddings. This is especially true for queries demanding nuanced understanding beyond surface-level code structure or keywords. These findings underscore the potential limitations of using raw code embeddings alone for nuanced, task-oriented retrieval for few-shot prompting. The system might retrieve code from the correct domain but fail to provide a relevant example of the specific technique needed by the AI agent.

6 Conclusion

The paper presented and evaluated an Information Retrieval system tailored for enhancing AI coding agents via few-shot prompting within a RAG framework [3], utilizing high-vote Kaggle notebooks as a source. A baseline approach using raw code embeddings was compared against an LLM annotated method leveraging LLM-generated semantic annotations [8]. The evaluation focused on Precision@3 (P@3), aligning with the goal of providing a small number of high-quality examples.

The LLM annotated method achieved significantly superior performance compared to the baseline implementation. A key finding from the error analysis was the baseline system's tendency to over-index on topical keywords present in code (e.g., matching "fraud" broadly) while failing to capture the specific algorithmic technique requested (e.g., `SMOTE`). This finding highlights a limitation of relying solely on raw code embeddings for tasks requiring fine-grained semantic understanding.

The results demonstrate that semantically enriched representations, like those used in the LLM annotated approach, become increasingly crucial for retrieving truly useful few-shot examples. This importance grows especially as agent queries become more specific and complex. The present work establishes the feasibility of targeted retrieval for few-shot code generation and underscores the importance of moving beyond surface-level code similarity towards deeper semantic understanding for the application.

7 Future Directions

Future work could build upon these findings in several innovative directions to significantly advance the capabilities of IR systems for few-shot code generation. Research could explore semantic code segmentation, moving beyond simple cell-based chunking towards techniques like Abstract Syntax Trees (ASTs) or control flow graphs to create more logically coherent code examples, even across multiple original cells. Another avenue involves developing task-specific and structure-aware embeddings tailored to the semantics of code, potentially using graph neural networks or fine-tuned language models. Enhancements to the LLM annotated approach could include dynamic, multi-modal annotation and retrieval, incorporating visualizations or execution summaries alongside code and text. Furthermore, adaptive retrieval ensembles could dynamically blend different methods based on query characteristics. Integrating conversational query refinement or decomposition would allow the AI agent to clarify its needs more effectively. Shifting evaluation towards agent-in-the-loop methods, measuring the utility of examples in downstream tasks and using feedback loops for learning, represents a significant step forward. Finally, incorporating proactive quality assurance through automated code review, static analysis, or execution checks could ensure retrieved examples are not only relevant but also high-quality and robust, moving beyond simple proxies like Kaggle votes.

8 References

References

[1] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. *SMOTE: Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research, 16:321–357, 2002.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework*. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[3] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. In Advances in Neural Information Processing Systems 33 (NeurIPS 2020), 2020.

[4] *LanceDB: Open-Source Vector Database for AI Applications*. Available at: https://lancedb.com, 2023. Note: Whitepaper and documentation.

[5] Nils Reimers and Sentence-Transformers Team. *all-MiniLM-L6-v2: 384-dimensional Sentence Embedding Model*. Available at: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, 2020.

[6] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. *WizardCoder: Empowering Code Large Language Models with Evol-Instruct*. arXiv preprint arXiv:2306.08568, 2023.

[7] Xiaonan Li, Kai Lv, Hang Yan, Tianyang Lin, Zhuwei, Yuan Ni, Guotong Xie, Xiaoling Wang, and Xipeng Qiu. *Unified Demonstration Retriever for In-Context Learning*. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL), 2023.

[8] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. *Document Expansion by Query Prediction*. arXiv preprint arXiv:1904.08375, 2019.

[9] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. Chapter 8 (Evaluation of IR Systems).

Appendix: Detailed Query Results

This appendix provides the detailed retrieval results (Top 3) for each evaluated query, comparing the naive (code embedding) and LLM annotated (annotation embedding) systems. Relevance judgments (✓ / *imes*) are indicated based on the criteria outlined in the main paper.

Note on Relevance: For the relevance judgments indicated below, a retrieved code chunk was considered relevant if it directly addressed the query **or** if it contained a clear call to a function or class instantiation directly related to the query (e.g., calling `reduce_mem_usage` for a memory optimization query, or instantiating `StackingClassifier` for a stacking query). Retrieving the definition of such called functions or classes is assumed to be a trivial subsequent step for the requesting agent.

Query 1: `How to handle highly imbalanced data in financial fraud detection?`

Narrative: The AI agent is working on a classification problem with imbalanced classes. Standard training might bias the model towards the majority class. The agent needs examples of applying the SMOTE algorithm (using ‘imblearn‘ or similar) to oversample the minority class, creating a balanced dataset to improve model performance on identifying minority instances.

Naive Implementation Results

Rank 1
Chunk ID: 16437 **Distance:** 1.0680 **Status:** Not Relevant

```
# Let's get the frequency of the cases with higher proportion of Fraud
props_30 = train.groupby("id_30")['isFraud'].value_counts(normalize=True).unstack()
props_30 = props_30.sort_values(by=1, ascending = False).head(20) # sort by fraud and get top 20
id_30_top = props_30.index.tolist()
props_30_c = train.groupby("id_30")['isFraud'].value_counts()
props_30_c.loc[id_30_top]
# --- Next Cell ---
props_31 = train.groupby("id_31")['isFraud'].value_counts(normalize=True).unstack()
props_31 = props_31.sort_values(by=1, ascending = False).head(20) # sort by fraud and get top 20
id_31_top = props_31.index.tolist()
props_31_c = train.groupby("id_31")['isFraud'].value_counts()
props_31_c.loc[id_31_top]
```

Rank 2
Chunk ID: 16391 **Distance:** 1.0985 **Status:** Not Relevant

```
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn.tree as tree
import sklearn.ensemble as ensem
from sklearn.model_selection import train_test_split
# --- Next Cell ---
identity = pd.read_csv("../input/train_identity.csv",header=0)
transaction = pd.read_csv("../input/train_transaction.csv",header=0)
# --- Next Cell ---
tempData = transaction[["TransactionAmt","ProductCD","card4","isFraud"]]
# --- Next Cell ---
tempData.head()
# --- Next Cell ---
#how many frauddata and  non fraud records
tempData.isFraud.value_counts()
```

Rank 3
Chunk ID: 16672 **Distance:** 1.1766 **Status:** Not Relevant

```
high_correlated_cols(df_bc, plot=True)
# --- Next Cell ---
drop_list = high_correlated_cols(df_bc)
# --- Next Cell ---
df_bc.drop(drop_list, axis=1)
# --- Next Cell ---
high_correlated_cols(df_bc.drop(drop_list, axis=1), plot=True)
# --- Next Cell ---
df_fraud = pd.read_csv("../input/ieee-fraud-detection/train_transaction.csv")
# --- Next Cell ---
df_fraud.head()
```

```
# --- Next Cell ---
check_df(df_fraud)
# --- Next Cell ---
drop_list = high_correlated_cols(df_fraud, plot=True)
```

LLM Annotated Implementation Results

Rank 1

Chunk ID: 14593 **Distance:** 1.1786 **Status:** Relevant

```
#Using SMOTE for class imbalance in target
from imblearn.over_sampling import SMOTE
from collections import Counter
sm = SMOTE(random_state=42)
X_resampled, y_resampled = sm.fit_resample(X, y)
print('Resampled dataset shape %s' % Counter(y_resampled))
# --- Next Cell ---
# Now we split the resmpled train set into train and validation sets
x_train_res, x_val_res, y_train_res, y_val_res = train_test_split(X_resampled,
                                                    y_resampled,
                                                    test_size = 0.3,
                                                    random_state=2019)
```

Rank 2

Chunk ID: 14544 **Distance:** 1.2081 **Status:** Relevant

```
y = sample.target
x = sample.drop(columns=['target'])
smotenn = combine.SMOTEENN(random_state=0, ratio=0.5)
resamp_x, resamp_y= smotenn.fit_resample(x, y)
# Transform the resampled data into principal components
pca = PCA(n_components=2)
resamp = pd.DataFrame(np.hstack((np.vstack(resamp_y), resamp_x)))

resamp_0 = resamp[resamp[0] == 0.0]
resamp_1 = resamp[resamp[0] == 1.0]
orig_0 = sample[sample.target == 0]
orig_1 = sample[sample.target == 1]

orig_no = pca.fit_transform(orig_0)
orig_yes = pca.fit_transform(orig_1)
resamp_no = pca.fit_transform(resamp_0)
resamp_yes = pca.fit_transform(resamp_1)

ono_x = orig_no[:, 0]
ono_y = orig_no[:, 1]
oyes_x = orig_yes[:, 0]
oyes_y = orig_yes[:, 1]
rno_x = resamp_no[:, 0]
rno_y = resamp_no[:, 1]
ryes_x = resamp_yes[:, 0]
ryes_y = resamp_yes[:, 1]

fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs= axs.flatten()
axs[0].set_title('Original Data')
axs[0].scatter(ono_x, ono_y, label='Original Class0')
axs[0].scatter(oyes_x, oyes_y, label='Original Class1')
axs[1].set_title('Oversampled Minority Class')
axs[1].scatter(oyes_x, oyes_y, label='Original Class1')
axs[1].scatter(ryes_x, ryes_y, label='Oversampled Class1')
axs[3].set_title('Undersampled Majority Class')
axs[3].scatter(ono_x, ono_y, label='Original Class0')
axs[3].scatter(rno_x, rno_y, label='Undersampled Class0')
axs[2].set_title('More Balanced Data')
axs[2].scatter(rno_x, rno_y, label='Undersampled Class0')
axs[2].scatter(ryes_x, ryes_y, label='Oversampled Class1')
axs[0].legend()
axs[1].legend()
axs[2].legend()
axs[3].legend()

plt.show()
```

Rank 3

Chunk ID: 14542 **Distance:** 1.2104 **Status:** Relevant

```
y = sample.target
x = sample.drop(columns=['target'])
smo = over_sampling.SMOTE(random_state=0, ratio=0.5)
resamp_x, resamp_y= smo.fit_resample(x, y)
# Transform the resampled data into principal components
pca = PCA(n_components=2)
resamp = pd.DataFrame(np.hstack((np.vstack(resamp_y), resamp_x)))

resamp_0 = resamp[resamp[0] == 0.0]
resamp_1 = resamp[resamp[0] == 1.0]
orig_0 = sample[sample.target == 0]
orig_1 = sample[sample.target == 1]

orig_no = pca.fit_transform(orig_0)
orig_yes = pca.fit_transform(orig_1)
resamp_no = pca.fit_transform(resamp_0)
resamp_yes = pca.fit_transform(resamp_1)

ono_x = orig_no[:, 0]
ono_y = orig_no[:, 1]
oyes_x = orig_yes[:, 0]
oyes_y = orig_yes[:, 1]
rno_x = resamp_no[:, 0]
rno_y = resamp_no[:, 1]
ryes_x = resamp_yes[:, 0]
ryes_y = resamp_yes[:, 1]

fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs= axs.flatten()
axs[0].set_title('Original Data')
axs[0].scatter(ono_x, ono_y, label='Original Class0')
axs[0].scatter(oyes_x, oyes_y, label='Original Class1')
axs[1].set_title('Oversampled Minority Class')
axs[1].scatter(oyes_x, oyes_y, label='Original Class1')
axs[1].scatter(ryes_x, ryes_y, label='Oversampled Class1')
axs[2].set_title('More Balanced Data')
axs[2].scatter(ono_x, ono_y, label='Original Class0')
axs[2].scatter(ryes_x, ryes_y, label='Oversampled Class1')
axs[0].legend()
axs[1].legend()
axs[2].legend()
fig.delaxes(axs[3])
plt.show()
```

Query 2: `Show feature engineering techniques for time series demand forecasting.`

Narrative: The agent needs to implement a regression model using Random Forest. It requires a straightforward code example showing how to instantiate, train (‘fit‘), and potentially predict with ‘scikit-learn‘’s ‘RandomForestRegressor‘ to ensure correct usage within its code generation flow.

Naive Implementation Results

Rank 1

Chunk ID: 6554 **Distance:** 1.0110 **Status:** Relevant

```
def date_features(df):

    df["date"] = pd.to_datetime(df["date"])
    df["day"] = df.date.dt.day
    df["month"] = df.date.dt.month
    df["week_day"] = df.date.dt.weekday

    df.drop(columns="date", inplace=True)

    return df

def sales_features(df):

    df.sell_price.fillna(0, inplace=True)

    return df

def demand_features(df):

    df["lag_t28"] = df["demand"].transform(lambda x: x.shift(28))
    df["rolling_mean_t7"] = df["demand"].transform(lambda x:x.shift(28).rolling(7).mean())
    df['rolling_mean_t30'] = df['demand'].transform(lambda x: x.shift(28).rolling(30).mean())
    df['rolling_mean_t60'] = df['demand'].transform(lambda x: x.shift(28).rolling(60).mean())
    df['rolling_mean_t90'] = df['demand'].transform(lambda x: x.shift(28).rolling(90).mean())
```

```
    df['rolling_mean_t180'] = df['demand'].transform(lambda x: x.shift(28).rolling(180).mean())
    df['rolling_std_t7'] = df['demand'].transform(lambda x: x.shift(28).rolling(7).std())
    df['rolling_std_t30'] = df['demand'].transform(lambda x: x.shift(28).rolling(30).std())

    df.fillna(0, inplace=True)

    return df
```

Rank 2
Chunk ID: 7042 Distance: 1.0796 Status: Relevant

```
# seed everything
seed_everything(42)
# reading data
data, calendar, sell_prices, submission, train_fold_df, valid_fold_df = read_data()
# get feature columns, also ignoring some features because we dont have enough ram and they overffit
features = [col for col in data.columns if col not in ['id', 'demand', 'part', 'date', 'wm_yr_wk', 'mean_demand_month', '
     std_demand_month', 'max_demand_month', 'mean_demand_week', 'std_demand_week',
                                                  'max_demand_week']]

print(f'We are training with {len(features)} fetures')
data.tail()
```

Rank 3
Chunk ID: 5592 Distance: 1.0991 Status: Not Relevant

```
# The pre-processing idea has been benefited greatly from
# https://www.analyticsvidhya.com/blog/2020/10/multivariate-multi-step-time-series-forecasting-using-stacked-lstm-sequence-to-
     sequence-autoencoder-in-tensorflow-2-0-keras/
# https://aeturrell.github.io/coding-for-economists/time-series.html
# https://coderedirect.com/questions/673308/interpolate-pandas-df
# https://www.kaggle.com/limweixuan1994/store-sales-predictions
# and many more~
## a huge shout-out to them

import os
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OrdinalEncoder

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import SimpleRNN, LSTM, Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Reshape
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import BatchNormalization

import warnings
warnings.filterwarnings('ignore')
```

LLM Annotated Implementation Results

Rank 1
Chunk ID: 7727 Distance: 0.7673 Status: Relevant

```
def making_train_test_data(df_train ,df_test):
    df_train = making_train_data(df_train)
    df_test = making_test_data(df_test)
    print("processing train test data")
    max_train_days = df_train['days'].max()
    min_test_days = df_test['days'].min()
    shift_data = 6
    df_test = pd.concat([df_train.loc[max_train_days - 28 * shift_data <= df_train['days'], :], df_test.loc[df_test['days'] >
     max_train_days, :]]).reset_index(drop=True)

#      shift_days_set = [28, 29, 30]
#      for i in shift_days_set:
#          df_train['demand_{}day_ago'.format(i)] = df_train.groupby(['item_id', 'store_id'])['demand'].transform(lambda x: x.shift
     (i))
#          df_test['demand_{}day_ago'.format(i)] = df_test.groupby(['item_id', 'store_id'])['demand'].transform(lambda x: x.shift(i
     ))
#          gc.collect()
```

```
    rolling_days_set = [2, 3, 5, 7, 14, 28, 56, 140]
    for i in rolling_days_set:
        df_train['demand_{}day_mean'.format(i)] = df_train.groupby(['item_id', 'store_id'])['demand'].transform(lambda x: x.shift
     (28).rolling(i).mean())
#           df_train['demand_{}day_max'.format(i)] = df_train.groupby(['item_id', 'store_id'])['demand'].transform(lambda x: x.shift
     (28).rolling(i).max())

        df_test['demand_{}day_mean'.format(i)] = df_test.groupby(['item_id', 'store_id'])['demand'].transform(lambda x: x.shift
     (28).rolling(i).mean())
#           df_test['demand_{}day_max'.format(i)] = df_test.groupby(['item_id', 'store_id'])['demand'].transform(lambda x: x.shift
     (28).rolling(i).max())
        df_train = reduce_mem_usage(df_train)
        df_test = reduce_mem_usage(df_test)
        gc.collect()

    df_test = df_test.loc[df_test['days'] >= min_test_days, :]
    df_test = reduce_mem_usage(df_test)
    gc.collect()

    return df_train, df_test
```

Rank 2
Chunk ID: 7855 **Distance:** 0.8915 **Status:** Not Relevant

```
def plot_results(fcst, y_eval, rmse, algo, item):
    fig = plt.figure(figsize=(11, 5))
    ax = fig.add_subplot()
    ax.plot(fcst, color='red', label='Forecast')
    ax.plot(y_eval, color='blue', label='Ground Truth')
    ax.set_title(f' {algo} for {item}, RMSE: {rmse}')
    ax.grid()
    ax.legend()
    plt.show()
# --- Next Cell ---
choice_data = train_data.copy()
choice_data['d_val'] = choice_data[d_cols].mean(axis=1)
choice_data.drop(columns=d_cols,inplace=True)
# --- Next Cell ---
print(f'Time Series Averages: Min {choice_data.d_val.min()} Max:{choice_data.d_val.max()} Median: {choice_data.d_val.median()}')
```

Rank 3
Chunk ID: 11104 **Distance:** 0.9056 **Status:** Relevant

```
# Adding min and max price of the item
item_price = (
    train_monthly.sort_values("date_block_num")
    .groupby(["item_id"], as_index=False)
    .agg({"item_price": [np.min, np.max]})
)
item_price.columns = ["item_id", "hist_min_item_price", "hist_max_item_price"]
train_monthly = pd.merge(train_monthly, item_price, on="item_id", how="left")

# Adding average price difference w.r.t the current month with min and max historic price
train_monthly["price_increase"] = (
    train_monthly["item_price"] - train_monthly["hist_min_item_price"]
)
train_monthly["price_decrease"] = (
    train_monthly["hist_max_item_price"] - train_monthly["item_price"]
)

# Adding first month of selling of a product
train_monthly["first_selling_date_block"] = train_monthly.groupby("item_id")[
    "date_block_num"
].min()

# Adding a boolean if a product is a newly launched product or not
train_monthly["is_new_product"] = (
    train_monthly["first_selling_date_block"] == train_monthly["date_block_num"]
)
```

Query 3: Implement model stacking for a classification task.

Narrative: The agent is preparing features for a scale-sensitive algorithm (e.g., SVM, KNN). It needs to normalize numerical features to a [0, 1] range. The agent requires a code snippet demonstrating the use of ‘scikit-learn‘’s ‘MinMaxScaler‘ to apply this transformation correctly, preventing features with large values from dominating the model.

Naive Implementation Results

Rank 1
Chunk ID: 3761 **Distance:** 0.8279 **Status:** Relevant

```
# Define best models:
best_classifiers = {
    "LGBM" : LGBMClassifier(**best_params["LGBM"], verbose=-1, random_state=0),
    "CatBoost" : CatBoostClassifier(**best_params["CatBoost"], verbose=False, random_state=0),
    "xgb_tunned" : XGBClassifier(**best_xgb_params)
}
# --- Next Cell ---
for name, model in best_classifiers.items():
    score = score_model(X_train, y_train, model)
    print(f"Model: {name}, Score: {score:.5f}")
# --- Next Cell ---
# Create a list of the base models for the StackingClassifier
estimators = [
    ('LGBM', best_classifiers['LGBM']),
    ('CatBoost', best_classifiers['CatBoost']),
    ('XGBoost', best_classifiers['xgb_tunned'])
]
```

Rank 2
Chunk ID: 16401 **Distance:** 0.9685 **Status:** Relevant

```
# Specify model tree for StackNet
models = [[l1_clf1, l1_clf2, l1_clf3], # Level 1
          [l2_clf1]] # Level 2
# --- Next Cell ---
from pystacknet.pystacknet import StackNetClassifier

# Specify parameters for stacked model and begin training
model = StackNetClassifier(models,
                           metric="auc",
                           folds=3,
                           restacking=False,
                           use_retraining=True,
                           use_proba=True, # To use predict_proba after training
                           random_state=seed,
                           n_jobs=-1,
                           verbose=1)

# Fit the entire model tree
model.fit(X_train, y_train)
```

Rank 3
Chunk ID: 10958 **Distance:** 1.0133 **Status:** Not Relevant

```
test_preds = stacking_preds(meta_model, y_train_pred, y_val_pred, y_test_pred, y_train, y_valid)
```

LLM Annotated Implementation Results

Rank 1
Chunk ID: 8360 **Distance:** 0.7984 **Status:** Not Relevant

```
oof_lgb_stack, prediction_lgb_stack, feature_importance = train_model(X=train_stack, X_test=test_stack, params=params, model_type=
      'lgb', plot_feature_importance=True)
# --- Next Cell ---
plt.figure(figsize=(18, 8))
plt.subplot(2, 3, 1)
plt.plot(y_tr, color='g', label='y_train')
plt.plot(oof_lgb, color='b', label='lgb')
plt.legend(loc=(1, 0.5));
plt.title('lgb');
plt.subplot(2, 3, 2)
plt.plot(y_tr, color='g', label='y_train')
plt.plot(oof_xgb, color='teal', label='xgb')
plt.legend(loc=(1, 0.5));
plt.title('xgb');
plt.subplot(2, 3, 3)
plt.plot(y_tr, color='g', label='y_train')
plt.plot(oof_svr, color='red', label='svr')
plt.legend(loc=(1, 0.5));
plt.title('svr');
plt.subplot(2, 3, 4)
```

```
plt.plot(y_tr, color='g', label='y_train')
plt.plot(oof_cat, color='b', label='cat')
plt.legend(loc=(1, 0.5));
plt.title('cat');
plt.subplot(2, 3, 5)
plt.plot(y_tr, color='g', label='y_train')
plt.plot(oof_lgb_stack, color='gold', label='stack')
plt.legend(loc=(1, 0.5));
plt.title('blend');
plt.legend(loc=(1, 0.5));
plt.suptitle('Predictions vs actual');
plt.subplot(2, 3, 6)
plt.plot(y_tr, color='g', label='y_train')
plt.plot((oof_lgb + oof_xgb + oof_svr + oof_svr1 + oof_r + oof_cat) / 6, color='gold', label='blend')
plt.legend(loc=(1, 0.5));
plt.title('blend');
plt.legend(loc=(1, 0.5));
plt.suptitle('Predictions vs actual');
```

Rank 2
Chunk ID: 15289 **Distance:** 0.8074 **Status:** Relevant

```
print("Train dataset (rows, cols):",trainset.values.shape, "\nTest dataset (rows, cols):",testset.values.shape)
# --- Next Cell ---
class Ensemble(object):
    def __init__(self, n_splits, stacker, base_models):
        self.n_splits = n_splits
        self.stacker = stacker
        self.base_models = base_models

    def fit_predict(self, X, y, T):
        X = np.array(X)
        y = np.array(y)
        T = np.array(T)

        folds = list(StratifiedKFold(n_splits=self.n_splits, shuffle=True, random_state=314).split(X, y))

        S_train = np.zeros((X.shape[0], len(self.base_models)))
        S_test = np.zeros((T.shape[0], len(self.base_models)))
        for i, clf in enumerate(self.base_models):

            S_test_i = np.zeros((T.shape[0], self.n_splits))

            for j, (train_idx, test_idx) in enumerate(folds):
                X_train = X[train_idx]
                y_train = y[train_idx]
                X_holdout = X[test_idx]


                print ("Base model %d: fit %s model | fold %d" % (i+1, str(clf).split('(')[0], j+1))
                clf.fit(X_train, y_train)
                cross_score = cross_val_score(clf, X_train, y_train, cv=3, scoring='roc_auc')
                print("cross_score [roc-auc]: %.5f [gini]: %.5f" % (cross_score.mean(), 2*cross_score.mean()-1))
                y_pred = clf.predict_proba(X_holdout)[:,1]

                S_train[test_idx, i] = y_pred
                S_test_i[:, j] = clf.predict_proba(T)[:,1]
            S_test[:, i] = S_test_i.mean(axis=1)

        results = cross_val_score(self.stacker, S_train, y, cv=3, scoring='roc_auc')
        # Calculate gini factor as 2 * AUC - 1
        print("Stacker score [gini]: %.5f" % (2 * results.mean() - 1))

        self.stacker.fit(S_train, y)
        res = self.stacker.predict_proba(S_test)[:,1]
        return res
```

Rank 3
Chunk ID: 15338 **Distance:** 0.8333 **Status:** Relevant

```
# Base models
lgb_model_1 = LGBMClassifier(**lgb_params_1)

lgb_model_2 = LGBMClassifier(**lgb_params_2)

lgb_model_3 = LGBMClassifier(**lgb_params_3)
# --- Next Cell ---
# Stacker models
log_model = LogisticRegression()

et_model = ExtraTreesClassifier(n_estimators=100, max_depth=6, min_samples_split=10, random_state=10)

mlp_model = MLPClassifier(max_iter=7, random_state=42)
```

```
# --- Next Cell ---
# Mode 2 run
stack = Ensemble(mode=2,
        n_splits=3,
        stacker_2 = (log_model, et_model),
        stacker_1 = (log_model, et_model, mlp_model),
        base_models = (lgb_model_1, lgb_model_2, lgb_model_3))

y_pred = stack.fit_predict(train, target_train, test)
```

Query 4: `Use scikit-learn partial_fit for incremental learning on large datasets.`

Narrative: The agent is tasked with training a model on a dataset too large for available memory. It needs examples of incremental learning using ‘scikit-learn‘’s ‘partial$_fit‘method(availableinestimatorslike‘SGDClassifier‘, ‘Passive$

Naive Implementation Results

Rank 1
Chunk ID: 2030 **Distance:** 1.0430 **Status:** Not Relevant

```
def feature_engineering(val, clfs, target=None):
    # Cluster and Dimensional mapping analysis for each data
    if clfs[0] is None:
        clfs[0] = MiniBatchKMeans(n_clusters=8, random_state=0, init="random").fit(val[:,1:4])
    km = clfs[0].predict(val[:,1:4])
    km_oh = np.zeros((val.shape[0],8), dtype=np.uint8) # discrete value change to One-hot
    for i in range(8):
        idx = np.where(km==0)[0]
        km_oh[idx,i] = 1
    if clfs[1] is None:
        clfs[1] = TruncatedSVD(n_components=2, n_iter=10, random_state=0).fit(val[:,1:4])
    svd = clfs[1].transform(val[:,1:4])
    # Per-user statistics
    print("Per-user statistics")
    cp = 0
    sp = 0
    usrm = np.zeros((val.shape[0], 5*val.shape[1]-10), dtype=np.float16)
    for i in range(val.shape[0]):
        if cp > val[i,0]:
            for t in range(val.shape[1]-2):
                usrm[sp:i,5*t] = np.mean(val[sp:i,t+1])
                usrm[sp:i,5*t+1] = np.std(val[sp:i,t+1])
                usrm[sp:i,5*t+2] = np.max(val[sp:i,t+1])
                usrm[sp:i,5*t+3] = np.min(val[sp:i,t+1])
                usrm[sp:i,5*t+4] = (i-sp)/val.shape[0]
            sp = i
        cp = val[i,0]
    for t in range(val.shape[1]-2):
        usrm[sp:,5*t] = np.mean(val[sp:,t+1])
        usrm[sp:,5*t+1] = np.std(val[sp:,t+1])
        usrm[sp:,5*t+2] = np.max(val[sp:,t+1])
        usrm[sp:,5*t+3] = np.min(val[sp:,t+1])
        usrm[sp:,5*t+4] = (val.shape[0]-sp)/val.shape[0]
    iskinetic = np.stack([(val[:,4]>=32).astype(np.uint8), (val[:,4]>=64).astype(np.uint8)]).transpose((1,0))
    # Cluster and Dimensional mapping analysis for each user/task
    print("Cluster and Dimensional mapping analysis for each user/task")
    if clfs[2] is None:
        clfs[2] = MiniBatchKMeans(n_clusters=8, random_state=0, init="random").fit(usrm)
    kmu = clfs[2].predict(usrm)
    kmu_oh = np.zeros((val.shape[0],8), dtype=np.uint8) # discrete value change to One-hot
    for i in range(8):
        idx = np.where(kmu==0)[0]
        kmu_oh[idx,i] = 1
    del kmu
    gc.collect()
    if clfs[3] is None:
        clfs[3] = TruncatedSVD(n_components=2, n_iter=10, random_state=0).fit(usrm)
    svdu = clfs[3].transform(usrm)
    gc.collect()
    # Merge waypoints
    marged = np.hstack([val[:,1:4],km_oh,svd])
    # Moving average and variance within the same user
    print("Moving average and variance within the same user/task")
    wnd = np.hstack([moving_window_by_group(val[:,0], marged[:,:15], 5) for kernel_size in FEAT_WINDOW_SIZE])
    # Analyze the entire merge data
    print("Analyze the entire merge data")
    usrv = np.hstack([svd,svdu])
    if clfs[4] is None:
        clfs[4] = [LinearRegression().fit(usrv, target[:,i]) for i in range(3)]
    reg = np.stack([clfs[4][i].predict(usrv) for i in range(3)]).transpose((1,0))
```

```
    del usrv
    gc.collect()
    if clfs[5] is None:
        clfs[5] = TruncatedSVD(n_components=2, n_iter=10, random_state=0).fit(marged)
    svdm = clfs[5].transform(marged)
    # Marge all
    return np.hstack([marged,wnd,reg,svdm,usrm,kmu_oh,svdu,iskinetic])
```

Rank 2
Chunk ID: 4104 **Distance:** 1.0432 **Status:** Not Relevant

```
!pip install ydata_profiling
# --- Next Cell ---
import ydata_profiling as pp

profile = pp.ProfileReport(df_train[:4000], explorative=True)

profile.to_file("/kaggle/working/EDA_Train.html")
# --- Next Cell ---
!pip install lazypredict
# --- Next Cell ---
from lazypredict.Supervised import LazyClassifier


clf = LazyClassifier(verbose=1,ignore_warnings=True, custom_metric=None)
models,predictions = clf.fit(X.iloc[:6000], X.iloc[6000:], Y.iloc[:6000], Y.iloc[6000:])

models
# --- Next Cell ---
from xgboost import XGBClassifier
from scipy.stats import uniform, randint
from sklearn.model_selection import RandomizedSearchCV
```

Rank 3
Chunk ID: 8461 **Distance:** 1.0613 **Status:** Not Relevant

```
# Create a training file with simple derived features
rows = 150_000
segments = int(np.floor(train.shape[0] / rows))

X_tr = pd.DataFrame(index=range(segments), dtype=np.float64,
                       columns=['mean', 'std', 'max', 'min',
                               'mean_change_abs', 'mean_change_rate', 'abs_max', 'abs_min',
                               'std_first_50000', 'std_last_50000', 'std_first_10000', 'std_last_10000',
                               'avg_first_50000', 'avg_last_50000', 'avg_first_10000', 'avg_last_10000',
                               'min_first_50000', 'min_last_50000', 'min_first_10000', 'min_last_10000',
                               'max_first_50000', 'max_last_50000', 'max_first_10000', 'max_last_10000',
                               'max_to_min', 'max_to_min_diff', 'count_big', 'sum',
                               'mean_change_rate_first_50000', 'mean_change_rate_last_50000', 'mean_change_rate_first_10000', '
     mean_change_rate_last_10000','q70','q75','q60','q65','q85',"q90",'q80','q95','q99','Hilbert_mean','Hann_window_mean','
     classic_sta_lta1_mean','classic_sta_lta2_mean','classic_sta_lta3_mean','classic_sta_lta4_mean','Moving_average_700_mean','
     Moving_average_1500_mean','Moving_average_3000_mean','Moving_average_6000_mean','exp_Moving_average_300_mean','
     exp_Moving_average_3000_mean','exp_Moving_average_30000_mean','MA_700MA_std_mean','MA_700MA_BB_high_mean','
     MA_700MA_BB_low_mean','MA_400MA_std_mean','MA_400MA_BB_high_mean','MA_400MA_BB_low_mean','MA_1000MA_std_mean'])
y_tr = pd.DataFrame(index=range(segments), dtype=np.float64,
                       columns=['time_to_failure'])

total_mean = train['acoustic_data'].mean()
total_std = train['acoustic_data'].std()
total_max = train['acoustic_data'].max()
total_min = train['acoustic_data'].min()
total_sum = train['acoustic_data'].sum()
total_abs_max = np.abs(train['acoustic_data']).sum()

for segment in tqdm_notebook(range(segments)):
    seg = train.iloc[segment*rows:segment*rows+rows]
    x = pd.Series(seg['acoustic_data'].values)
    y = seg['time_to_failure'].values[-1]

    y_tr.loc[segment, 'time_to_failure'] = y
    X_tr.loc[segment, 'mean'] = x.mean()
    X_tr.loc[segment, 'std'] = x.std()
    X_tr.loc[segment, 'max'] = x.max()
    X_tr.loc[segment, 'min'] = x.min()


    X_tr.loc[segment, 'mean_change_abs'] = np.mean(np.diff(x))
    X_tr.loc[segment, 'mean_change_rate'] = np.mean(np.nonzero((np.diff(x) / x[:-1]))[0])
    X_tr.loc[segment, 'abs_max'] = np.abs(x).max()
    X_tr.loc[segment, 'abs_min'] = np.abs(x).min()

    X_tr.loc[segment, 'std_first_50000'] = x[:50000].std()
    X_tr.loc[segment, 'std_last_50000'] = x[-50000:].std()
```

```
    X_tr.loc[segment, 'std_first_10000'] = x[:10000].std()
    X_tr.loc[segment, 'std_last_10000'] = x[-10000:].std()

    X_tr.loc[segment, 'avg_first_50000'] = x[:50000].mean()
    X_tr.loc[segment, 'avg_last_50000'] = x[-50000:].mean()
    X_tr.loc[segment, 'avg_first_10000'] = x[:10000].mean()
    X_tr.loc[segment, 'avg_last_10000'] = x[-10000:].mean()

    X_tr.loc[segment, 'min_first_50000'] = x[:50000].min()
    X_tr.loc[segment, 'min_last_50000'] = x[-50000:].min()
    X_tr.loc[segment, 'min_first_10000'] = x[:10000].min()
    X_tr.loc[segment, 'min_last_10000'] = x[-10000:].min()

    X_tr.loc[segment, 'max_first_50000'] = x[:50000].max()
    X_tr.loc[segment, 'max_last_50000'] = x[-50000:].max()
    X_tr.loc[segment, 'max_first_10000'] = x[:10000].max()
    X_tr.loc[segment, 'max_last_10000'] = x[-10000:].max()

    X_tr.loc[segment, 'max_to_min'] = x.max() / np.abs(x.min())
    X_tr.loc[segment, 'max_to_min_diff'] = x.max() - np.abs(x.min())
    X_tr.loc[segment, 'count_big'] = len(x[np.abs(x) > 500])
    X_tr.loc[segment, 'sum'] = x.sum()

    X_tr.loc[segment, 'mean_change_rate_first_50000'] = np.mean(np.nonzero((np.diff(x[:50000]) / x[:50000][:-1]))[0])
    X_tr.loc[segment, 'mean_change_rate_last_50000'] = np.mean(np.nonzero((np.diff(x[-50000:]) / x[-50000:][:-1]))[0])
    X_tr.loc[segment, 'mean_change_rate_first_10000'] = np.mean(np.nonzero((np.diff(x[:10000]) / x[:10000][:-1]))[0])
    X_tr.loc[segment, 'mean_change_rate_last_10000'] = np.mean(np.nonzero((np.diff(x[-10000:]) / x[-10000:][:-1]))[0])

     #new:'q70','q75','q60','q65'
    X_tr.loc[segment, 'q70'] = np.quantile(x, 0.70)
    X_tr.loc[segment, 'q75'] = np.quantile(x, 0.75)
    X_tr.loc[segment, 'q60'] = np.quantile(x, 0.60)
    X_tr.loc[segment, 'q65'] = np.quantile(x, 0.65)
    X_tr.loc[segment, 'q85'] = np.quantile(x, 0.85)
    X_tr.loc[segment, 'q90'] = np.quantile(x, 0.90)
    X_tr.loc[segment, 'q80'] = np.quantile(x, 0.80)
    X_tr.loc[segment, 'q95'] = np.quantile(x, 0.95)
    X_tr.loc[segment, 'q99'] = np.quantile(x, 0.99)


    X_tr.loc[segment, 'Hilbert_mean'] = np.abs(hilbert(x)).mean()
    X_tr.loc[segment, 'Hann_window_mean'] = (convolve(x, hann(150), mode='same') / sum(hann(150))).mean()
    X_tr.loc[segment, 'classic_sta_lta1_mean'] = classic_sta_lta(x, 500, 10000).mean()
    X_tr.loc[segment, 'classic_sta_lta2_mean'] = classic_sta_lta(x, 5000, 100000).mean()
    X_tr.loc[segment, 'classic_sta_lta3_mean'] = classic_sta_lta(x, 3333, 6666).mean()
    X_tr.loc[segment, 'classic_sta_lta4_mean'] = classic_sta_lta(x, 10000, 25000).mean()
    X_tr.loc[segment, 'Moving_average_700_mean'] = x.rolling(window=700).mean().mean(skipna=True)
    X_tr.loc[segment, 'Moving_average_1500_mean'] = x.rolling(window=1500).mean().mean(skipna=True)
    X_tr.loc[segment, 'Moving_average_3000_mean'] = x.rolling(window=3000).mean().mean(skipna=True)
    X_tr.loc[segment, 'Moving_average_6000_mean'] = x.rolling(window=6000).mean().mean(skipna=True)
    ewma = pd.Series.ewm
    X_tr.loc[segment, 'exp_Moving_average_300_mean'] = (ewma(x, span=300).mean()).mean(skipna=True)
    X_tr.loc[segment, 'exp_Moving_average_3000_mean'] = ewma(x, span=3000).mean().mean(skipna=True)
    X_tr.loc[segment, 'exp_Moving_average_30000_mean'] = ewma(x, span=6000).mean().mean(skipna=True)
    no_of_std = 2
    X_tr.loc[segment, 'MA_700MA_std_mean'] = x.rolling(window=700).std().mean()
    X_tr.loc[segment,'MA_700MA_BB_high_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] + no_of_std * X_tr.loc[segment, '
     MA_700MA_std_mean']).mean()
    X_tr.loc[segment,'MA_700MA_BB_low_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] - no_of_std * X_tr.loc[segment, '
     MA_700MA_std_mean']).mean()
    X_tr.loc[segment, 'MA_400MA_std_mean'] = x.rolling(window=400).std().mean()
    X_tr.loc[segment,'MA_400MA_BB_high_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] + no_of_std * X_tr.loc[segment, '
     MA_400MA_std_mean']).mean()
    X_tr.loc[segment,'MA_400MA_BB_low_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] - no_of_std * X_tr.loc[segment, '
     MA_400MA_std_mean']).mean()
    X_tr.loc[segment, 'MA_1000MA_std_mean'] = x.rolling(window=1000).std().mean()
```

LLM Annotated Implementation Results

Rank 1

Chunk ID: 3024 **Distance:** 0.9755 **Status:** Relevant

```
print(clf.predict([[-0.8, -1]]))
# --- Next Cell ---
clf_pf = GaussianNB()
clf_pf.partial_fit(X, Y, np.unique(Y))
# --- Next Cell ---
print(clf_pf.predict([[-0.8, -1]]))
# --- Next Cell ---
import numpy as np
rng = np.random.RandomState(1)
X = rng.randint(5, size=(6, 100))
Y = np.array([1, 2, 3, 4, 4, 5])
from sklearn.naive_bayes import BernoulliNB
clf = BernoulliNB()
clf.fit(X, Y)
```

```
print(clf.predict(X[2:3]))
```

Rank 2
Chunk ID: 3952 **Distance:** 0.9786 **Status:** Not Relevant

```
inputs.head()
# --- Next Cell ---

x_train, x_val, y_train, y_val = train_test_split(inputs, target, test_size=0.2, random_state=42)
print(x_train.shape)
print(y_train.shape)

print(x_val.shape)
print(y_val.shape)
# --- Next Cell ---
pipe = Pipeline([("scaler", MinMaxScaler()), ("model", KNeighborsClassifier(n_neighbors=12))])
pipe.fit(x_train, y_train)
print(pipe.score(x_val, y_val))
# --- Next Cell ---
pipe2 = Pipeline([("scaler", MinMaxScaler()),("model", DecisionTreeClassifier(random_state=42, max_depth=9,max_leaf_nodes=28,
     max_features=12))])
pipe2.fit(x_train, y_train)
print(pipe2.score(x_val, y_val))
```

Rank 3
Chunk ID: 10001 **Distance:** 0.9957 **Status:** Not Relevant

```
# Train model with full data
clf = LGBMClassifier(**best_params)
clf.fit(df, bin_target)
# --- Next Cell ---
def write_submission(model, env):
    days = env.get_prediction_days()
    for (market_obs_df, news_obs_df, predictions_template_df) in days:
        news_obs_df = preprocess_news(news_obs_df)
        # Unstack news
        index_df = unstack_asset_codes(news_obs_df)
        news_unstack = merge_news_on_index(news_obs_df, index_df)
        # Group and and get aggregations (mean)
        news_obs_agg = group_news(news_unstack)

        # Join market and news frames
        market_obs_df['date'] = market_obs_df.time.dt.date
        obs_df = market_obs_df.merge(news_obs_agg, how='left', on=['assetCode', 'date'])
        del market_obs_df, news_obs_agg, news_obs_df, news_unstack, index_df
        gc.collect()
        obs_df = obs_df[obs_df.assetCode.isin(predictions_template_df.assetCode)]

        # Drop cols that are not features
        feats = [c for c in obs_df.columns if c not in ['date', 'assetCode', 'assetName', 'time']]

        preds = model.predict_proba(obs_df[feats])[:, 1] * 2 - 1
        sub = pd.DataFrame({'assetCode': obs_df['assetCode'], 'confidence': preds})
        predictions_template_df = predictions_template_df.merge(sub, how='left').drop(
            'confidenceValue', axis=1).fillna(0).rename(columns={'confidence':'confidenceValue'})

        env.predict(predictions_template_df)
        del obs_df, predictions_template_df, preds, sub
        gc.collect()
    env.write_submission_file()

write_submission(clf, env)
```

Query 5: Best practices for optimizing memory usage when reading large CSV files with pandas.

Narrative: The agent is processing a large pandas DataFrame and needs to reduce its memory footprint for efficiency. It seeks examples of common optimization functions (often named ‘reduce$_m em_u sage$‘)$thatdowncastnumericaldtypesto$

Naive Implementation Results

Rank 1
Chunk ID: 13915 **Distance:** 0.8238 **Status:** Relevant

```
# import Dataset to play with it
train= pd.read_csv("../input/train.csv")
test = pd.read_csv('../input/test.csv')
# --- Next Cell ---
sample_submission = pd.read_csv('../input/sample_submission.csv')
sample_submission.head()
# --- Next Cell ---
train.shape, test.shape, sample_submission.shape
# --- Next Cell ---
train.head(5)
# --- Next Cell ---
#Based on this great kernel https://www.kaggle.com/arjanso/reducing-dataframe-memory-size-by-65
def reduce_mem_usage(df):
    start_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",start_mem_usg," MB")
    NAlist = [] # Keeps track of columns that have missing values filled in.
    for col in df.columns:
        if df[col].dtype != object:  # Exclude strings

            # Print current column type
            print("******************************")
            print("Column: ",col)
            print("dtype before: ",df[col].dtype)

            # make variables for Int, max and min
            IsInt = False
            mx = df[col].max()
            mn = df[col].min()

            # Integer does not support NA, therefore, NA needs to be filled
            if not np.isfinite(df[col]).all():
                NAlist.append(col)
                df[col].fillna(mn-1,inplace=True)

            # test if column can be converted to an integer
            asint = df[col].fillna(0).astype(np.int64)
            result = (df[col] - asint)
            result = result.sum()
            if result > -0.01 and result < 0.01:
                IsInt = True


            # Make Integer/unsigned Integer datatypes
            if IsInt:
                if mn >= 0:
                    if mx < 255:
                        df[col] = df[col].astype(np.uint8)
                    elif mx < 65535:
                        df[col] = df[col].astype(np.uint16)
                    elif mx < 4294967295:
                        df[col] = df[col].astype(np.uint32)
                    else:
                        df[col] = df[col].astype(np.uint64)
                else:
                    if mn > np.iinfo(np.int8).min and mx < np.iinfo(np.int8).max:
                        df[col] = df[col].astype(np.int8)
                    elif mn > np.iinfo(np.int16).min and mx < np.iinfo(np.int16).max:
                        df[col] = df[col].astype(np.int16)
                    elif mn > np.iinfo(np.int32).min and mx < np.iinfo(np.int32).max:
                        df[col] = df[col].astype(np.int32)
                    elif mn > np.iinfo(np.int64).min and mx < np.iinfo(np.int64).max:
                        df[col] = df[col].astype(np.int64)

            # Make float datatypes 32 bit
            else:
                df[col] = df[col].astype(np.float32)

            # Print new column type
            print("dtype after: ",df[col].dtype)
            print("******************************")

    # Print final result
    print("___MEMORY USAGE AFTER COMPLETION:___")
    mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage is: ",mem_usg," MB")
    print("This is ",100*mem_usg/start_mem_usg,"% of the initial size")
    return df, NAlist
```

Rank 2
Chunk ID: 2252 **Distance:** 0.8391 **Status:** Not Relevant

```
df_subjects = pd.read_csv(os.path.join(root, 'subjects.csv'))
df_subjects.head()
# --- Next Cell ---
# list of all tdcsfog csv file path
```

```
tdcs_file_path = glob.glob(os.path.join(root, 'train', 'tdcsfog', '*.csv'), recursive=True)

# In this notebook, we limit the number of files to be read in order to reduce the time required for model training.
tdcs_file_path = tdcs_file_path

print(f'the number of files to be read: {len(tdcs_file_path)}')
```

Rank 3
Chunk ID: 1863 **Distance:** 0.8461 **Status:** Relevant

```
def reduce_memory_usage(df):

    start_mem = df.memory_usage().sum() / 1024 ** 2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype.name
        if ((col_type != 'datetime64[ns]') & (col_type != 'category')):
            if (col_type != 'object'):
                c_min = df[col].min()
                c_max = df[col].max()

                if str(col_type)[:3] == 'int':
                    if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                        df[col] = df[col].astype(np.int8)
                    elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                        df[col] = df[col].astype(np.int16)
                    elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                        df[col] = df[col].astype(np.int32)
                    elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                        df[col] = df[col].astype(np.int64)

                else:
                    if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                        df[col] = df[col].astype(np.float16)
                    elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                        df[col] = df[col].astype(np.float32)
                    else:
                        pass
            else:
                df[col] = df[col].astype('category')
    mem_usg = df.memory_usage().sum() / 1024 ** 2
    print("Memory usage became: ",mem_usg," MB")

    return df
```

LLM Annotated Implementation Results

Rank 1
Chunk ID: 16349 **Distance:** 0.5680 **Status:** Relevant

```
train_identity = pd.read_csv("train_identity.csv")
train_transaction = pd.read_csv("train_transaction.csv")

test_identity = pd.read_csv("test_identity.csv")
test_transaction = pd.read_csv("test_transaction.csv")

train_identity = reduce_mem_usage(train_identity)
train_transaction = reduce_mem_usage(train_transaction)
test_identity = reduce_mem_usage(test_identity)
test_transaction = reduce_mem_usage(test_transaction)
# --- Next Cell ---
# Since the number of columns are too large, we can expand it using pd.set_option()
pd.set_option('display.max_columns', None)
# --- Next Cell ---
train_transaction['TransactionID'].value_counts().sort_values(ascending = False)
```

Rank 2
Chunk ID: 7899 **Distance:** 0.6082 **Status:** Relevant

```
def read_data():
    print('Reading files...')
    calendar = pd.read_csv('/kaggle/input/m5-forecasting-accuracy/calendar.csv')
    calendar = reduce_mem_usage(calendar)
    print('Calendar has {} rows and {} columns'.format(calendar.shape[0], calendar.shape[1]))

    sell_prices = pd.read_csv('/kaggle/input/m5-forecasting-accuracy/sell_prices.csv')
    sell_prices = reduce_mem_usage(sell_prices)
    print('Sell prices has {} rows and {} columns'.format(sell_prices.shape[0], sell_prices.shape[1]))
```

```
    sales_train_val = pd.read_csv('/kaggle/input/m5-forecasting-accuracy/sales_train_validation.csv')
    print('Sales train validation has {} rows and {} columns'.format(sales_train_val.shape[0], sales_train_val.shape[1]))

    submission = pd.read_csv('/kaggle/input/m5-forecasting-accuracy/sample_submission.csv')

    return calendar, sell_prices, sales_train_val, submission
```

Rank 3
Chunk ID: 7660 **Distance:** 0.6380 **Status:** Relevant

```
# read data files
calendar = pd.read_csv(input_path + '/calendar.csv')
calendar = reduce_mem_usage(calendar)
sell_prices = pd.read_csv(input_path + '/sell_prices.csv')
sell_prices = reduce_mem_usage(sell_prices)
sales_train_val = pd.read_csv(input_path + '/sales_train_validation.csv')
sales_train_val = reduce_mem_usage(sales_train_val)
sample_submission = pd.read_csv(input_path + '/sample_submission.csv')
# --- Next Cell ---
NUM_ITEMS = sales_train_val.shape[0]  # 30490
DAYS_PRED = sample_submission.shape[1] - 1  # 28
```

Query 6: How to interpret XGBoost or LightGBM models using SHAP values?

Narrative: The agent needs to explain the predictions of a trained model by calculating feature contributions. It requires code examples showing how to use the SHAP library to compute SHAP values, enabling interpretation of feature importance and model behavior.

Naive Implementation Results

Rank 1
Chunk ID: 10644 **Distance:** 0.8326 **Status:** Relevant

```
import optuna  # pip install optuna


def objective(trial):
    x = trial.suggest_float("x", -7, 7)
    y = trial.suggest_float("y", -7, 7)
    return (x - 1) ** 2 + (y + 3) ** 2


study = optuna.create_study()
study.optimize(objective, n_trials=200)  # number of iterations

study.best_params
# --- Next Cell ---
study.best_value
# --- Next Cell ---
import shap  # pip install shap
import xgboost as xgb

# Load and train a model
X, y = shap.datasets.diabetes()
clf = xgb.XGBRegressor().fit(X, y)

# Explain model's predictions with SHAP
explainer = shap.Explainer(clf)
shap_values = explainer(X)

# Visualize the predictions' explanation
shap.plots.beeswarm(shap_values)
```

Rank 2
Chunk ID: 10107 **Distance:** 0.8533 **Status:** Relevant

```
# Fit and predict
evals_result = {}
m = lgb.train(
    lgb_params, dtrain, num_boost_round=1000, valid_sets=(dvalid,), valid_names=('valid',),
```

```
    verbose_eval=25, early_stopping_rounds=20, evals_result=evals_result
)
# --- Next Cell ---
lgb.plot_importance(m);
# --- Next Cell ---
lgb.plot_importance(m, importance_type='gain');
# --- Next Cell ---
shap_explainer = shap.TreeExplainer(m)
# --- Next Cell ---
%%time
sample = X.sample(frac=0.50, random_state=100)
shap_values = shap_explainer.shap_values(sample)
```

Rank 3
Chunk ID: 10077 **Distance:** 0.9561 **Status:** Relevant

```
# DF, based on which importance is checked
X_importance = X_test

# Explain model predictions using shap library:
explainer = shap.TreeExplainer(lgb_model)
shap_values = explainer.shap_values(X_importance)
# --- Next Cell ---
# Plot summary_plot
shap.summary_plot(shap_values, X_importance)
# --- Next Cell ---
# Plot summary_plot as barplot:
shap.summary_plot(shap_values, X_importance, plot_type='bar')
# --- Next Cell ---
X_importance.returnsClosePrevRaw10_lag_3_mean.value_counts()
```

LLM Annotated Implementation Results

Rank 1
Chunk ID: 10078 **Distance:** 0.6641 **Status:** Relevant

```
plt.hist(X_importance.returnsClosePrevRaw10_lag_3_mean, bins=100)
# --- Next Cell ---
shap.dependence_plot("returnsClosePrevRaw10_lag_3_mean", shap_values, X_importance)
# --- Next Cell ---
shap.dependence_plot("volume", shap_values, X_importance)
# --- Next Cell ---
X_interaction = X_importance.iloc[:500,:]

shap_interaction_values = shap.TreeExplainer(lgb_model).shap_interaction_values(X_interaction)
# --- Next Cell ---
shap.summary_plot(shap_interaction_values, X_interaction)
# --- Next Cell ---
# Raw dependence plot:

shap.dependence_plot(
    "returnsClosePrevRaw10_lag_3_mean",
    shap_values, X_importance)
# --- Next Cell ---
# Interaction values dependence plot capturing main effects:

shap.dependence_plot(
    ("returnsClosePrevRaw10_lag_3_mean", "returnsClosePrevRaw10_lag_3_mean"),
    shap_interaction_values, X_interaction)
```

Rank 2
Chunk ID: 12129 **Distance:** 0.7035 **Status:** Relevant

```
temp = X_test.drop(columns='lgb_pred').sample(10000)
explainer = shap.TreeExplainer(lgb_model)
shap_values = explainer.shap_values(temp)
shap.summary_plot(shap_values, temp, max_display=30)
# --- Next Cell ---
fig = go.Figure(data=[
    go.Bar(name='Prediction',x=data.date,y=data.lgb_pred),
    go.Bar(name='Target',x=data.date,y=data.target)
])
fig.update_layout(
    title='Mean Prediction and Target Values by Month',
    xaxis={'title':'Month','type':'category'},
    yaxis={'title':'Mean Value'},
    legend={'yanchor':'top','y':1.05,'xanchor':'left','x':0.01},
    template='plotly_dark'
)
```

```
text = '''
    The model may not be adequately accounting for<br>
    the December sales spike. There is no systemic under or<br>
    over-prediction visible for the test month, November.
'''
fig.add_annotation(
    yref='paper', y=1.1,
    xref='paper', x=0.7,
    text=text,
    font={'size':11},
    showarrow=False)

text = '''
    Prediction mean is very close to the  <br>
    target mean in our validation set.
'''
fig.add_annotation(
    xref='paper', x=0.95,
    yref='paper', y=0.58,
    text=text,
    font={'size':11},
    showarrow=True, arrowhead=1)
fig.show()
```

Rank 3

Chunk ID: 8023 **Distance:** 0.7048 **Status:** Relevant

```
del sub,sub2
# --- Next Cell ---
gc.collect()
# --- Next Cell ---
shap_values = model.get_feature_importance(tst_pool,type="ShapValues")
expected_value = shap_values[0,-1]
shap_values = shap_values[:,:-1]
# --- Next Cell ---
# we used one of the test data for shap value calculation
shap.initjs()
shap.force_plot(expected_value, shap_values[3,:], tst.iloc[3,:])
# --- Next Cell ---
%%time
shap.summary_plot(shap_values, tst, plot_type='bar')
```

Query 7: Apply text augmentation techniques for an NLP classification task.

Narrative: The agent is working on an NLP task with limited data and needs to augment the text dataset to improve model generalization. It requires examples of text augmentation techniques like back-translation, synonym replacement, or using specific libraries to create synthetic training samples.

Naive Implementation Results

Rank 1

Chunk ID: 11918 **Distance:** 1.1900 **Status:** Not Relevant

```
from autoviml.Auto_NLP import Auto_NLP
nlp_column = 'Message'
target = 'Category'
train_nlp, test_nlp, nlp_pipeline, predictions = Auto_NLP(
                nlp_column, train, test, target, score_type='balanced_accuracy',
                modeltype='Classification',top_num_features=200, verbose=0,
                build_model=True)
# --- Next Cell ---
from sklearn.metrics import accuracy_score, balanced_accuracy_score
print(accuracy_score(test[target].values, test_nlp[target].values))
# --- Next Cell ---
nlp = setup(data = train, target = 'Message', session_id = 1)
```

Rank 2

Chunk ID: 14186 **Distance:** 1.2058 **Status:** Not Relevant

```
nlp = spacy.load('en_core_web_sm')

print(nlp.Defaults.stop_words)
print(len(nlp.Defaults.stop_words))
# --- Next Cell ---
words = ['is', 'and', 'Tesla', 'you', 'IS', 'AND']

for word in words:
    print(f"{word}: is stop word: {nlp.vocab[word].is_stop}")
# --- Next Cell ---
# We can add our own stop word
nlp.Defaults.stop_words.add('btw')
nlp.Defaults.stop_words.add('u')

sentence = 'Where was u ? I was looking for you btw session...'
for word in sentence.split():
    print(f"{word:{20}}: is stop word: {nlp.vocab[word].is_stop}")
```

Rank 3
Chunk ID: 11298 Distance: 1.2227 Status: Not Relevant

```
from sklearn.datasets import fetch_20newsgroups
data = fetch_20newsgroups(subset="all")["data"]
# --- Next Cell ---
from sklearn import datasets
# --- Next Cell ---
!pip install sentence_transformers
# --- Next Cell ---
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('distilbert-base-nli-mean-tokens')
embeddings = model.encode(data, show_progress_bar=True)
# --- Next Cell ---
embeddings
# --- Next Cell ---
!pip install umap-learn
```

LLM Annotated Implementation Results

Rank 1
Chunk ID: 11298 Distance: 0.8902 Status: Not Relevant

```
from sklearn.datasets import fetch_20newsgroups
data = fetch_20newsgroups(subset="all")["data"]
# --- Next Cell ---
from sklearn import datasets
# --- Next Cell ---
!pip install sentence_transformers
# --- Next Cell ---
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('distilbert-base-nli-mean-tokens')
embeddings = model.encode(data, show_progress_bar=True)
# --- Next Cell ---
embeddings
# --- Next Cell ---
!pip install umap-learn
```

Rank 2
Chunk ID: 14335 Distance: 0.8954 Status: Not Relevant

```
def augment(x, y, t=2):

    xs, xn = [], []

    for i in range(t // 2):
        mask = y == 0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
        featnum = x1.shape[1] // 200 - 1

        for c in range(200):
            np.random.shuffle(ids)
            x1[:, [c] + [200 + featnum * c + idc for idc in range(featnum)]] = x1[ids][:, [c] + [200 + featnum * c + idc for idc
      in range(featnum)]]
        xn.append(x1)

    for i in range(t):
        mask = y > 0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
```

```
        featnum = x1.shape[1] // 200 - 1

        for c in range(200):
            np.random.shuffle(ids)
            x1[:, [c] + [200 + featnum * c + idc for idc in range(1)]] = x1[ids][:, [c] + [200 + featnum * c + idc for idc in
      range(1)]]
        xs.append(x1)

    xs = np.vstack(xs)
    xn = np.vstack(xn)
    ys = np.ones(xs.shape[0])
    yn = np.zeros(xn.shape[0])
    x = np.vstack([x, xs, xn])
    y = np.concatenate([y, ys, yn])

    return x, y
```

Rank 3

Chunk ID: 10374 **Distance:** 0.9393 **Status:** Not Relevant

```
news_df = pd.read_pickle('../input/data-preparation-memory-optimization/news_df.pkl')
# --- Next Cell ---
news_df['date'] = news_df.time.dt.date
time_news = news_df.groupby('date').headline.apply(' '.join).reset_index()
# --- Next Cell ---
del news_df; gc.collect()
# --- Next Cell ---
# The function "text_to_wordlist" is from
# https://www.kaggle.com/currie32/quora-question-pairs/the-importance-of-cleaning-text

def text_to_wordlist(text, remove_stopwords=False, stem_words=False):
    # Clean the text, with the option to remove stopwords and to stem words.

    # Convert words to lower case and split them
    text = text.lower().split()

    # Optionally, remove stop words
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        text = [w for w in text if not w in stops]

    text = " ".join(text)

    # Clean the text
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "cannot ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\.", " ", text)
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\/", " ", text)
    text = re.sub(r"\^", " ^ ", text)
    text = re.sub(r"\+", " + ", text)
    text = re.sub(r"\-", " - ", text)
    text = re.sub(r"\=", " = ", text)
    text = re.sub(r"'", " ", text)
    text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
    text = re.sub(r":", " : ", text)
    text = re.sub(r" e g ", " eg ", text)
    text = re.sub(r" b g ", " bg ", text)
    text = re.sub(r" u s ", " american ", text)
    text = re.sub(r"\0s", "0", text)
    text = re.sub(r" 9 11 ", "911", text)
    text = re.sub(r"e - mail", "email", text)
    text = re.sub(r"j k", "jk", text)
    text = re.sub(r"\s{2,}", " ", text)

    # Optionally, shorten words to their stems
    if stem_words:
        text = text.split()
        stemmer = SnowballStemmer('english')
        stemmed_words = [stemmer.stem(word) for word in text]
        text = " ".join(stemmed_words)

    # Return a list of words
    return(text)
```

Query 8: `Perform hyperparameter tuning using Optuna for a machine learning model.`

Narrative: The agent is using an XGBoost model and needs to tune its hyperparameters for better performance. It requires code examples demonstrating hyperparameter optimization methods like ‘GridSearchCV‘, ‘RandomizedSearchCV‘, or Bayesian optimization (e.g., using ‘Optuna‘) applied specifically to XGBoost.

Naive Implementation Results

Rank 1
Chunk ID: 4468 **Distance:** 0.8400 **Status:** Relevant

```
# Hyperparameter Tuning with Optuna

def objective(trial):
    callback = [PyTorchLightningPruningCallback(trial, monitor="val_loss")]

    # set input_chunk_length, between 21 and 365 days
    input_chunk_length = trial.suggest_int("input_chunk_length", 63, 270)

    # Other hyperparameters
    num_stacks = trial.suggest_int("num_stacks", 1, 3)
    num_blocks = trial.suggest_int("num_blocks", 1, 3)
    num_layers = trial.suggest_int("num_layers", 1, 3)
    layer_exp = trial.suggest_int("layer_exp", 7, 10)
    #layer_widths = 2 ** layer_exp
    dropout = trial.suggest_float("dropout", 0.01, 0.2, step=0.01)
    lr = trial.suggest_float("lr", 5e-5, 0.1, log=True)

    # build and train the N-HiTS model with these hyper-parameters:
    model = build_fit_nhits_model(
                    input_chunk_length=input_chunk_length,
                    num_stacks=num_stacks,
                      num_blocks=num_blocks,
                    num_layers=num_layers,
                    layer_exp=layer_exp,
                      dropout=dropout,
                          lr=lr,
                      likelihood=None,
                      callbacks=callback,
                      #max_samples=365
    )

    # Evaluate how good it is on the validation set
    preds = model.predict(series=train, past_covariates=NHiTS_covariates, n=val_len)
    rmsles = rmsle(val, preds, n_jobs=-1, verbose=True)
    rmsle_val = np.mean(rmsles)

    return rmsle_val if rmsle_val != np.nan else float("inf")

def print_callback(study, trial):
    print(f"Current value: {trial.value}, Current params: {trial.params}")
    print(f"Best value: {study.best_value}, Best params: {study.best_trial.params}")


torch.cuda.empty_cache()

study_nhits = optuna.create_study(direction="minimize")

study_nhits.optimize(objective, n_trials=5, callbacks=[print_callback])

# Finally, print the best value and best hyperparameters:
print(f"Best value: {study_nhits.best_value}, Best params: {study_nhits.best_trial.params}")
```

Rank 2
Chunk ID: 4289 **Distance:** 0.8688 **Status:** Not Relevant

```
# 1. MLPClassifier with tuning hyperparameters
from sklearn.neural_network import MLPClassifier

mlp_params = {
    'activation': 'relu',
    'solver': 'adam',
    'early_stopping': True,
}

mlp = MLPClassifier(**mlp_params)

mlp_params_tuned = {
    'hidden_layer_sizes': [(10,),(20,),(50,)],
    'alpha': [0.01, 0.1],
    'learning_rate_init': [0.0001, 0.0005, 0.001],
    'max_iter': [200, 500],
```

```
}

mlp_randomized = RandomizedSearchCV(mlp, mlp_params_tuned, n_iter=10, cv=5, verbose=1, random_state=random_state, n_jobs=-1)
mlp_randomized.fit(X_train_scaled, y_train)

print(f"Best parameters: {mlp_randomized.best_params_}")
print(f"Best score: {mlp_randomized.best_score_}")

# Validate the model using the best parameters
best_mlp = mlp_randomized.best_estimator_
y_pred = best_mlp.predict(X_val_scaled)
print(f"Accuracy for best MLP model: {accuracy_score(y_val, y_pred)}")
```

Rank 3
Chunk ID: 3116 Distance: 0.9034 Status: Relevant

```
drop_svc = True
# --- Next Cell ---
import optuna
from sklearn.svm import SVC
from sklearn.metrics import f1_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

if drop_svc:
    print("Ignored.")
else:
    # Function for SVM optimization using Optuna
    def objective_svm(trial, X_train, X_test, y_train, y_test, num_classes):
        # Hyperparameters to be optimized
        kernel = trial.suggest_categorical('kernel', ['linear', 'rbf', 'poly', 'sigmoid'])

        param = {
            'C': trial.suggest_loguniform('C', 1e-3, 1e2),
            'kernel': kernel,
            'gamma': trial.suggest_categorical('gamma', ['scale', 'auto']),
            'degree': trial.suggest_int('degree', 2, 5) if kernel == 'poly' else 3,  # Only optimize 'degree' if kernel is 'poly'
            'random_state': RANDOM_SEED
        }

        # SVM Model with preprocessing pipeline
        model = Pipeline([
            ('scaler', StandardScaler()),  # Scale features
            ('svc', SVC(**param))  # Apply SVM with the chosen parameters
        ])

        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)

        # Compute the F1 score
        f1 = f1_score(y_test, y_pred, average='weighted')

        return f1

    # Check if SVM results already exist
    model_name = 'SVC'
    if model_name in df_results_optimized['Model'].values:
        print(f"{model_name} results already exist in df_results:")
        print(df_results_optimized[df_results_optimized['Model'] == model_name])
    else:
        print(f"Launching Optuna study for {model_name}...")

        # Dictionary to store optimized results for SVM
        optimized_results_svm = {}

        # Iterate through each dataset for SVM optimization
        for dataset_name, (X_train, X_test, y_train, y_test) in datasets_to_optimize.items():
            print(f"Optimizing SVM for {dataset_name} dataset...")

            num_classes = len(set(y_train))

            # Create an Optuna study for SVM
            study_svm = optuna.create_study(direction='maximize')

            # Optimize the study for 100 iterations
            study_svm.optimize(lambda trial: objective_svm(trial, X_train, X_test, y_train, y_test, num_classes),
                           n_trials=OPTUNA_TRIALS, timeout=60)  # 1 minute per trial


            # Best hyperparameters
            best_params_svm = study_svm.best_params

            # Train the model using the best hyperparameters
```

```
        model_svm = Pipeline([
            ('scaler', StandardScaler()),
            ('svc', SVC(**best_params_svm))
        ])
        model_svm.fit(X_train, y_train)

        # Make predictions with the optimized model
        y_pred_svm = model_svm.predict(X_test)

        # Compute optimized F1-score
        optimized_f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

        # Store the optimized F1-score in the dictionary
        optimized_results_svm[dataset_name] = optimized_f1_svm

        # Compare F1 before and after optimization (from df_results)
        before_f1_svm = df_results[(df_results['Model'] == 'SVC') & (df_results['Dataset'] == dataset_name)]['F1-score'].
    values[0]

        print(f"Dataset: {dataset_name}")
        print(f"F1-score before optimization (SVM): {before_f1_svm:.4f}")
        print(f"F1-score after optimization (SVM): {optimized_f1_svm:.4f}\n")

    # Create a DataFrame for optimized SVM results
    df_results_optimized_svm = pd.DataFrame.from_dict(
        optimized_results_svm,
        orient='index',
        columns=['Optimized F1-score']
    )
    df_results_optimized_svm.reset_index(inplace=True)
    df_results_optimized_svm.rename(columns={'index': 'Dataset'}, inplace=True)
    df_results_optimized_svm['Model'] = 'SVC'

    # Merge with existing df_results_optimized
    df_results_optimized = pd.concat([df_results_optimized, df_results_optimized_svm], ignore_index=True)

    # Save the updated results to CSV
    output_file_optimized = OUTPUT_DATA_DIR + "model_results_optimized.csv"
    df_results_optimized.to_csv(output_file_optimized, index=False)

    print(f"Optimized results (including SVC) saved to {output_file_optimized}")
```

LLM Annotated Implementation Results

Rank 1
Chunk ID: 10775 **Distance:** 0.5326 **Status:** Relevant

```
import matplotlib.pyplot as plt
import numpy as np
import optuna
import pandas as pd
import seaborn as sns

optuna.logging.set_verbosity(optuna.logging.WARNING)
# --- Next Cell ---
import optuna  # pip install optuna


def objective(trial):
    x = trial.suggest_float("x", -7, 7)
    y = trial.suggest_float("y", -7, 7)
    return (x - 1) ** 2 + (y + 3) ** 2
```

Rank 2
Chunk ID: 10776 **Distance:** 0.6231 **Status:** Relevant

```
study = optuna.create_study()
study.optimize(objective, n_trials=100)  # number of iterations
# --- Next Cell ---
study.best_params
# --- Next Cell ---
len(study.trials)
# --- Next Cell ---
study.optimize(objective, n_trials=100)
# --- Next Cell ---
study.best_params
# --- Next Cell ---
study = optuna.create_study()
type(study)
# --- Next Cell ---
def objective(trial: optuna.Trial):
    """Conventional optimization function
    signature for optuna.
```

```
    """
    custom_metric = ...
    return custom_metric
```

Rank 3
Chunk ID: 2630 **Distance:** 0.6445 **Status:** Relevant

```
'''
# ExtraTree Algorithm
def optimize_et(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 1000, 2000, step=100),
        'max_depth': trial.suggest_int('max_depth', 5, 15),
        'max_features': trial.suggest_float('max_features', 0.5, 1.0),
        'random_state': 42
    }
    model = ExtraTreesRegressor(**params)
    score = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_root_mean_squared_error').mean()
    return score

study_et = optuna.create_study(direction='maximize')
study_et.optimize(optimize_et, n_trials=20)
best_et_params = study_et.best_params
'''
# --- Next Cell ---
best_xgb_params = {'n_estimators': 1400, 'max_depth': 3, 'learning_rate': 0.018293416542962154, 'subsample': 0.8383691260878259, '
     colsample_bytree': 0.5013210246761113}
best_lgbm_params = {'n_estimators': 1800, 'max_depth': 4, 'num_leaves': 65, 'learning_rate': 0.007228355080148622, '
     feature_fraction': 0.6638871805164953}
best_catboost_params = {'iterations': 1800, 'depth': 6, 'learning_rate': 0.04017750463622224}
best_rf_params = {'n_estimators': 1800, 'max_depth': 13, 'min_samples_split': 4, 'min_samples_leaf': 1}
best_et_params = {'n_estimators': 2000, 'max_depth': 13, 'max_features': 0.5964598821528422}
```

Query 9: `How to handle covariate shift between training and test data?`

Narrative: The agent needs to diagnose potential covariate shift between training and test sets, which could harm model generalization. It seeks examples of techniques, sometimes involving functions like ‘relax$_data$‘, $usedtodetectdistrib$

Naive Implementation Results

Rank 1
Chunk ID: 15855 **Distance:** 0.9185 **Status:** Relevant

```
train, test = relax_data(train, test, 'C11')
plot_numerical('C11')
# --- Next Cell ---
print('Covariate shift after data relaxation:', covariate_shift('C11'))
# --- Next Cell ---
plot_numerical('C12')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('C12'))
# --- Next Cell ---
train, test = relax_data(train, test, 'C12')
plot_numerical('C12')
# --- Next Cell ---
print('Covariate shift after data relaxation:', covariate_shift('C12'))
```

Rank 2
Chunk ID: 15852 **Distance:** 0.9286 **Status:** Relevant

```
train, test = relax_data(train, test, 'C6')
plot_numerical('C6')
# --- Next Cell ---
print('Covariate shift after data relaxation:', covariate_shift('C6'))
# --- Next Cell ---
plot_numerical('C7')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('C7'))
# --- Next Cell ---
train, test = relax_data(train, test, 'C7')
plot_numerical('C7')
# --- Next Cell ---
print('Covariate shift after data relaxation:', covariate_shift('C7'))
```

Rank 3
Chunk ID: 15854 **Distance:** 0.9984 **Status:** Relevant

```
print('Covariate shift after data relaxation:', covariate_shift('C9'))
# --- Next Cell ---
plot_numerical('C10')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('C10'))
# --- Next Cell ---
train, test = relax_data(train, test, 'C10')
plot_numerical('C10')
# --- Next Cell ---
print('Covariate shift after data relaxation:', covariate_shift('C10'))
# --- Next Cell ---
plot_numerical('C11')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('C11'))
```

LLM Annotated Implementation Results

Rank 1
Chunk ID: 15848 **Distance:** 1.0360 **Status:** Relevant

```
plot_numerical('V7')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('V7'))
# --- Next Cell ---
plot_numerical('V258')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('V258'))
# --- Next Cell ---
train, test = relax_data(train, test, 'V258')
plot_numerical('V258')
# --- Next Cell ---
plot_numerical('V294')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('V294'))
# --- Next Cell ---
train, test = relax_data(train, test, 'V294')
plot_numerical('V294')
```

Rank 2
Chunk ID: 15855 **Distance:** 1.1985 **Status:** Relevant

```
train, test = relax_data(train, test, 'C11')
plot_numerical('C11')
# --- Next Cell ---
print('Covariate shift after data relaxation:', covariate_shift('C11'))
# --- Next Cell ---
plot_numerical('C12')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('C12'))
# --- Next Cell ---
train, test = relax_data(train, test, 'C12')
plot_numerical('C12')
# --- Next Cell ---
print('Covariate shift after data relaxation:', covariate_shift('C12'))
```

Rank 3
Chunk ID: 15858 **Distance:** 1.2144 **Status:** Not Relevant

```
print('Covariate shift:', covariate_shift('D4'))
# --- Next Cell ---
plot_numerical('D5')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('D5'))
# --- Next Cell ---
plot_numerical('D6')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('D6'))
# --- Next Cell ---
plot_numerical('D7')
# --- Next Cell ---
print('Covariate shift:', covariate_shift('D7'))
# --- Next Cell ---
plot_numerical('D8')
```

Query 10: `Create an image processing pipeline with augmentations in PyTorch for object detection.`

Narrative: The agent needs to implement an object detection pipeline using the PyTorch framework. It requires an example showing setup, data loading, model instantiation (e.g., Faster R-CNN from 'torchvision'), inference, and potentially fine-tuning steps for a typical object detection task.

Naive Implementation Results

Rank 1
Chunk ID: 1070 **Distance:** 0.8550 **Status:** Not Relevant

```
from imageai.Detection import ObjectDetection
model_weight_path = "../input/imageairepo/imageai/resnet50_v2.0.1.h5"

execution_path = os.getcwd()
detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(model_weight_path)
detector.loadModel()
# --- Next Cell ---
for i in range(n_rounds):
    batch = next(img_generator)
    for j, prediction in enumerate(batch):
        image = filenames[i * batch_size + j]
        detections = detector.detectObjectsFromImage(input_image=image_path+image, output_image_path="image_with_box.png",
     minimum_percentage_probability = 75)
        pred_str = ""
        labels = ""
        for eachObject in detections:
            if eachObject["name"] in rev:
                pred_str += rev[eachObject["name"]] + " " + str(float(eachObject["percentage_probability"])/100) + " 0.1 0.1 0.9
     0.9"
                pred_str += " "
                labels += eachObject['name'] + ", " + str(round(float(eachObject['percentage_probability'])/100, 1))
                labels += " | "
        if labels != "":
            plt.figure(figsize=(12,12))
            plt.imshow(plt.imread("image_with_box.png"))
            plt.show()

            print ("Labels Detected: ")
            print (labels)
            print ()
            print ("Prediction String: ")
            print (pred_str)

    if i == 10:
        break
```

Rank 2
Chunk ID: 1074 **Distance:** 0.8856 **Status:** Not Relevant

```
from tensorflow.python.keras import backend as K
sess = K.get_session()
# --- Next Cell ---
from __future__ import division
import time
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np
import cv2
#from util import *
import argparse
import os
import os.path as osp
#from darknet import Darknet
import pickle as pkl
import pandas as pd
import random
from PIL import Image, ImageDraw, ImageFont #sajin
def arg_parse():
    """
    Parse arguements to the detect module
```

```
    """

    parser = argparse.ArgumentParser(description='YOLO v3 Detection Module')

    parser.add_argument("--images", dest = 'images', help =
                        "Image / Directory containing images to perform detection upon",
                        default = "../input/google-ai-open-images-object-detection-track/test/challenge2018_test/00001a21632de752.
     jpg", type = str)
    parser.add_argument("--det", dest = 'det', help =
                        "Image / Directory to store detections to",
                        default = "det", type = str)
    parser.add_argument("--bs", dest = "bs", help = "Batch size", default = 1)
    parser.add_argument("--confidence", dest = "confidence", help = "Object Confidence to filter predictions", default = 0.5)
    parser.add_argument("--nms_thresh", dest = "nms_thresh", help = "NMS Threshhold", default = 0.4)
    parser.add_argument("--cfg", dest = 'cfgfile', help =
                        "Config file",
                        default = "../input/yolov3cfg/yolov3.cfg", type = str)
    parser.add_argument("--weights", dest = 'weightsfile', help =
                        "weightsfile",
                        default = "../input/yolov3-weights/yolov3.weights", type = str)
    parser.add_argument("--reso", dest = 'reso', help =
                        "Input resolution of the network. Increase to increase accuracy. Decrease to increase speed",
                        default = "416", type = str)

    return parser.parse_args()

def get_test_input(image, input_dim):
    img = cv2.imread(image)
    img = cv2.resize(img, (input_dim, input_dim))
    img_ =  img[:,:,::-1].transpose((2,0,1))
    img_ = img_[np.newaxis,:,:,:]/255.0
    img_ = torch.from_numpy(img_).float()
    img_ = Variable(img_)
def getmodel(cfgfile, weightsfile):
        #Set up the neural network
    print("Loading network.....")
    model = Darknet(cfgfile)
    model.load_weights(weightsfile)
    print("Network successfully loaded")
    return model
def load_classes(namesfile):
    fp = open(namesfile, "r")
    names = fp.read().split("\n")[:-1]
    f = []
    #names = [x if x=="tvmonitor": "television" else: x for x in names]
    for x in names:
        if x == "tvmonitor":
            f.append("television")
        elif x == "aeroplane":
            f.append("airplane")
        elif x == "pottedplant":
            f.append("houseplant")
        elif x == "cell phone":
            f.append("mobile phone")
        elif x == "cup":
            f.append("coffee cup")
        elif x == "diningtable":
            f.append("kitchen & dining room table")
        elif x == "sofa":
            f.append("sofa bed")
        elif x == "motorbike":
            f.append("motorcycle")
        elif x == "cow":
            f.append("cattle")
        elif x == "microwave":
            f.append("microwave oven")
        elif x == "remote":
            f.append("remote control")
        elif x == "sports ball":
            f.append("ball")
        else:
            f.append(x)
    names = f
    return names
def detect(model, images_i, bs_i, confidence_i, nms_thresh_i, class_path, weightsfile, cfgfile, reso):
    #args = arg_parse()

    images = images_i
    batch_size = int(bs_i)
    confidence = float(confidence_i)
    nms_thesh = float(nms_thresh_i)
    #scales = scales_i
    start = 0
    CUDA = torch.cuda.is_available()
    print("CUDA:",str(CUDA))
    num_classes = 80
    classes = load_classes(class_path)
    '''
    #Set up the neural network
    print("Loading network.....")
```

```
model = Darknet(cfgfile)
model.load_weights(weightsfile)
print("Network successfully loaded")
'''
model.net_info["height"] = reso
inp_dim = int(model.net_info["height"])
assert inp_dim % 32 == 0
assert inp_dim > 32

#If there's a GPU availible, put the model on GPU
if CUDA:
    model.cuda()
#Set the model in evaluation mode
model.eval()

read_dir = time.time()
#Detection phase
try:
    imlist = [osp.join(osp.realpath('.'), images, img) for img in os.listdir(images)]
except NotADirectoryError:
    imlist = []
    imlist.append(osp.join(osp.realpath('.'), images))
    #print("Not a directory...")
except FileNotFoundError:
    print ("No file or directory with the name {}".format(images))
    exit()

#if not os.path.exists(args.det):
    #os.makedirs(args.det)

load_batch = time.time()
#loaded_ims = [cv2.imread(x) for x in imlist]

#im_batches = list(map(prep_image, loaded_ims, [inp_dim for x in range(len(imlist))]))
batches = list(map(prep_image, imlist, [inp_dim for x in range(len(imlist))]))
im_batches = [x[0] for x in batches]
orig_ims = [x[1] for x in batches]
im_dim_list = [x[2] for x in batches]
#im_dim_list = [(x.shape[1], x.shape[0]) for x in loaded_ims]
im_dim_list = torch.FloatTensor(im_dim_list).repeat(1,2)
if CUDA:
    im_dim_list = im_dim_list.cuda()

leftover = 0
if (len(im_dim_list) % batch_size):
    leftover = 1

if batch_size != 1:
    num_batches = len(imlist) // batch_size + leftover
    im_batches = [torch.cat((im_batches[i*batch_size : min((i +  1)*batch_size,
                        len(im_batches))]))  for i in range(num_batches)]

i=0
write = False
#model(get_test_input(inp_dim, CUDA), CUDA)
start_det_loop = time.time()
if CUDA:
    im_dim_list = im_dim_list.cuda()

start_det_loop = time.time()
objs = {}

for batch in im_batches:
#load the image
    start = time.time()
    if CUDA:
        batch = batch.cuda()
    with torch.no_grad():
        prediction, pred_old = model(Variable(batch), CUDA)
    #print("PREDICTION:",prediction)
    #prediction_old = prediction[0]
    #prediction = prediction[1]
    prediction = write_results(prediction, confidence, num_classes, nms_conf = nms_thesh)
    #pred_old = write_results(pred_old, confidence, num_classes, nms_conf = nms_thesh)
    #print("PREDICTION: ",prediction)
    #print("PREDICTION_OLD: ",pred_old)
    end = time.time()

    if type(prediction) == int:
        i += 1
        continue
    end = time.time()
    prediction[:,0] += i*batch_size

    if not write:                      #If we have't initialised output
        output = prediction
        write = 1
    else:
        output = torch.cat((output,prediction))

    for im_num, image in enumerate(imlist[i*batch_size: min((i +  1)*batch_size, len(imlist))]):
```

```
        im_id = i*batch_size + im_num
        #print("{0:20s} predicted in {1:6.3f} seconds".format(image.split("/")[-1], (end - start)/batch_size))
        print("{0:20s} {1:s}".format("Objects Detected:", ""))
        print("----------------------------------------------------------")
        image_id = image.split("/")[-1].split(".")[0]

    i += 1
    if CUDA:
        torch.cuda.synchronize()
try:
    output
except NameError:
    print ("No detections were made")
    return ['']#exit()

im_dim_list = torch.index_select(im_dim_list, 0, output[:,0].long())

scaling_factor = torch.min(inp_dim/im_dim_list,1)[0].view(-1,1)


output[:,[1,3]] -= (inp_dim - scaling_factor*im_dim_list[:,0].view(-1,1))/2
output[:,[2,4]] -= (inp_dim - scaling_factor*im_dim_list[:,1].view(-1,1))/2

output[:,1:5] /= scaling_factor

for i in range(output.shape[0]):
    output[i, [1,3]] = torch.clamp(output[i, [1,3]], 0.0, im_dim_list[i,0])
    output[i, [2,4]] = torch.clamp(output[i, [2,4]], 0.0, im_dim_list[i,1])

output_recast = time.time()
class_load = time.time()
colors = pkl.load(open("../input/pallete/pallete", "rb"))

draw = time.time()
sub_str = " " #initialise to a space value
def write(x, batches, results):
    c1 = tuple(x[1:3].int())
    c2 = tuple(x[3:5].int())
    img = results[int(x[0])]
    cls = int(x[-1])
    color = random.choice(colors)
    label = "{0}".format(classes[cls])
    cv2.rectangle(img, c1, c2,color, 1)
    #sajin
    #d_class_label[classes[cls]]
    height = img.shape[0]
    width = img.shape[1]
    x_min = np.around(c1[0].data.numpy()/width,decimals = 2)
    y_min = np.around(c1[1].data.numpy()/height,decimals = 2)
    x_max = np.around(c2[0].data.numpy()/width,decimals = 2)
    y_max = np.around(c2[1].data.numpy()/height,decimals = 2)
    confi = np.around(x[-3].data.numpy(), decimals = 2)
    #print("iiii..",t,tf.divide(t ,img.shape[0] ) )
    #print("Box ",image_id,c1, c2, classes[cls],d_class_label[classes[cls].lower()],img.shape)
    #print("Box1 ",image_id,d_class_label[classes[cls].lower()],confi,x_min,y_min,x_max,y_max)
    sub_str = d_class_label[classes[cls].lower()]+' '+str(confi)+' '+str(x_min)+' '+str(y_min)+' '+str(x_max)+' '+str(y_max)
    #font = ImageFont.truetype(font='../input/firamonomedium/FiraMono-Medium.otf',size=np.floor(3e-2 * img.size[1] + 0.5).
  astype('int32'))
    #t_size = cv2.getTextSize(label, font, 2 , 1)[0]
    #end
    t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_COMPLEX_SMALL, 1 , 1)[0] # 1 to 2 Sajin
    c2 = c1[0] + t_size[0] + 3, c1[1] + t_size[1] + 4
    cv2.rectangle(img, c1, c2,color, -1)
    cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, [225,255,255], 1);
    #cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), font, 1, [225,255,255], 1); #sajin
    return sub_str


sub_str =  list(map(lambda x: write(x, im_batches, orig_ims), output))
#print("Final...",sub_str)
#det_names = pd.Series(imlist).apply(lambda x: "{}/det_{}".format(args.det,x.split("/")[-1]))
#print("Det names: ",det_names)
det_names = "output_image.png"
#list(map(cv2.imwrite, det_names, loaded_ims))
#list(map(cv2.imwrite, "im.png", loaded_ims))
#list(map(cv2.imshow, det_names, loaded_ims))
#cv2.waitKey(0)
#cv2.destroyAllWindows()
#
'''
import matplotlib.pyplot as plt
for img in orig_ims:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    print("Img width:",img.shape[0])
    #plt.xticks(np.arange(img.shape[0], img.shape[0]+1, 100))
    plt.figure(figsize=(12,12))
    plt.imshow(img)
    plt.show()

end = time.time()
```

```
    print("SUMMARY")
    print("----------------------------------------------------------")
    print("{:25s}: {}".format("Task", "Time Taken (in seconds)"))
    print()
    print("{:25s}: {:2.3f}".format("Reading addresses", load_batch - read_dir))
    print("{:25s}: {:2.3f}".format("Loading batch", start_det_loop - load_batch))
    print("{:25s}: {:2.3f}".format("Detection (" + str(len(imlist)) +  " images)", output_recast - start_det_loop))
    print("{:25s}: {:2.3f}".format("Output Processing", class_load - output_recast))
    print("{:25s}: {:2.3f}".format("Drawing Boxes", end - draw))
    print("{:25s}: {:2.3f}".format("Average time_per_img", (end - load_batch)/len(imlist)))
    print("----------------------------------------------------------")
    '''

    torch.cuda.empty_cache()
    return sub_str
```

Rank 3

Chunk ID: 504 **Distance:** 0.9268 **Status:** Not Relevant

```
# test_image_dir = '/kaggle/working/models/research/object_detection/test_images/ducky/test/'
# test_images_np = []
# for i in range(1, 50):
#     image_path = os.path.join(test_image_dir, 'out' + str(i) + '.jpg')
#     test_images_np.append(np.expand_dims(
#     load_image_into_numpy_array(image_path), axis=0))

# # Again, uncomment this decorator if you want to run inference eagerly
# @tf.function
# def detect(input_tensor):
#     preprocessed_image, shapes = detection_model.preprocess(input_tensor);prediction_dict = detection_model.predict(
      preprocessed_image, shapes)
#     return detection_model.postprocess(prediction_dict, shapes)

# # Note that the first frame will trigger tracing of the tf.function, which will
# # take some time, after which inference should be fast.

# label_id_offset = 1
# for i in range(len(test_images_np)):
#     input_tensor = tf.convert_to_tensor(test_images_np[i], dtype=tf.float32)
#     detections = detect(input_tensor)

#     plot_detections(test_images_np[i][0], detections['detection_boxes'][0].numpy(), detections['detection_classes'][0].numpy().
      astype(np.uint32) + label_id_offset, detections['detection_scores'][0].numpy(),category_index, figsize=(15, 20), image_name
      ="gif_frame_" + ('%02d' % i) + ".jpg")
```

LLM Annotated Implementation Results

Rank 1

Chunk ID: 1074 **Distance:** 0.7396 **Status:** Relevant

```
from tensorflow.python.keras import backend as K
sess = K.get_session()
# --- Next Cell ---
from __future__ import division
import time
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np
import cv2
#from util import *
import argparse
import os
import os.path as osp
#from darknet import Darknet
import pickle as pkl
import pandas as pd
import random
from PIL import Image, ImageDraw, ImageFont #sajin
def arg_parse():
    """
    Parse arguements to the detect module

    """

    parser = argparse.ArgumentParser(description='YOLO v3 Detection Module')

    parser.add_argument("--images", dest = 'images', help =
                        "Image / Directory containing images to perform detection upon",
                        default = "../input/google-ai-open-images-object-detection-track/test/challenge2018_test/00001a21632de752.
      jpg", type = str)
    parser.add_argument("--det", dest = 'det', help =
                        "Image / Directory to store detections to",
```

```
                        default = "det", type = str)
    parser.add_argument("--bs", dest = "bs", help = "Batch size", default = 1)
    parser.add_argument("--confidence", dest = "confidence", help = "Object Confidence to filter predictions", default = 0.5)
    parser.add_argument("--nms_thresh", dest = "nms_thresh", help = "NMS Threshhold", default = 0.4)
    parser.add_argument("--cfg", dest = 'cfgfile', help =
                        "Config file",
                        default = "../input/yolov3cfg/yolov3.cfg", type = str)
    parser.add_argument("--weights", dest = 'weightsfile', help =
                        "weightsfile",
                        default = "../input/yolov3-weights/yolov3.weights", type = str)
    parser.add_argument("--reso", dest = 'reso', help =
                        "Input resolution of the network. Increase to increase accuracy. Decrease to increase speed",
                        default = "416", type = str)

    return parser.parse_args()

def get_test_input(image, input_dim):
    img = cv2.imread(image)
    img = cv2.resize(img, (input_dim, input_dim))
    img_ =  img[:,:,::-1].transpose((2,0,1))
    img_ = img_[np.newaxis,:,:,:]/255.0
    img_ = torch.from_numpy(img_).float()
    img_ = Variable(img_)
def getmodel(cfgfile, weightsfile):
        #Set up the neural network
    print("Loading network.....")
    model = Darknet(cfgfile)
    model.load_weights(weightsfile)
    print("Network successfully loaded")
    return model
def load_classes(namesfile):
    fp = open(namesfile, "r")
    names = fp.read().split("\n")[:-1]
    f = []
    #names = [x if x=="tvmonitor": "television" else: x for x in names]
    for x in names:
        if x == "tvmonitor":
            f.append("television")
        elif x == "aeroplane":
            f.append("airplane")
        elif x == "pottedplant":
            f.append("houseplant")
        elif x == "cell phone":
            f.append("mobile phone")
        elif x == "cup":
            f.append("coffee cup")
        elif x == "diningtable":
            f.append("kitchen & dining room table")
        elif x == "sofa":
            f.append("sofa bed")
        elif x == "motorbike":
            f.append("motorcycle")
        elif x == "cow":
            f.append("cattle")
        elif x == "microwave":
            f.append("microwave oven")
        elif x == "remote":
            f.append("remote control")
        elif x == "sports ball":
            f.append("ball")
        else:
            f.append(x)
    names = f
    return names
def detect(model, images_i, bs_i, confidence_i, nms_thresh_i, class_path, weightsfile, cfgfile, reso):
    #args = arg_parse()

    images = images_i
    batch_size = int(bs_i)
    confidence = float(confidence_i)
    nms_thesh = float(nms_thresh_i)
    #scales = scales_i
    start = 0
    CUDA = torch.cuda.is_available()
    print("CUDA:",str(CUDA))
    num_classes = 80
    classes = load_classes(class_path)
    '''
    #Set up the neural network
    print("Loading network.....")
    model = Darknet(cfgfile)
    model.load_weights(weightsfile)
    print("Network successfully loaded")
    '''
    model.net_info["height"] = reso
    inp_dim = int(model.net_info["height"])
    assert inp_dim % 32 == 0
    assert inp_dim > 32

    #If there's a GPU availible, put the model on GPU
    if CUDA:
```

```
    model.cuda()
#Set the model in evaluation mode
model.eval()

read_dir = time.time()
#Detection phase
try:
    imlist = [osp.join(osp.realpath('.'), images, img) for img in os.listdir(images)]
except NotADirectoryError:
    imlist = []
    imlist.append(osp.join(osp.realpath('.'), images))
    #print("Not a directory...")
except FileNotFoundError:
    print ("No file or directory with the name {}".format(images))
    exit()

#if not os.path.exists(args.det):
    #os.makedirs(args.det)

load_batch = time.time()
#loaded_ims = [cv2.imread(x) for x in imlist]

#im_batches = list(map(prep_image, loaded_ims, [inp_dim for x in range(len(imlist))]))
batches = list(map(prep_image, imlist, [inp_dim for x in range(len(imlist))]))
im_batches = [x[0] for x in batches]
orig_ims = [x[1] for x in batches]
im_dim_list = [x[2] for x in batches]
#im_dim_list = [(x.shape[1], x.shape[0]) for x in loaded_ims]
im_dim_list = torch.FloatTensor(im_dim_list).repeat(1,2)
if CUDA:
    im_dim_list = im_dim_list.cuda()

leftover = 0
if (len(im_dim_list) % batch_size):
    leftover = 1

if batch_size != 1:
    num_batches = len(imlist) // batch_size + leftover
    im_batches = [torch.cat((im_batches[i*batch_size : min((i +  1)*batch_size,
                        len(im_batches))]))  for i in range(num_batches)]

i=0
write = False
#model(get_test_input(inp_dim, CUDA), CUDA)
start_det_loop = time.time()
if CUDA:
    im_dim_list = im_dim_list.cuda()

start_det_loop = time.time()
objs = {}

for batch in im_batches:
#load the image
    start = time.time()
    if CUDA:
        batch = batch.cuda()
    with torch.no_grad():
        prediction, pred_old = model(Variable(batch), CUDA)
    #print("PREDICTION:",prediction)
    #prediction_old = prediction[0]
    #prediction = prediction[1]
    prediction = write_results(prediction, confidence, num_classes, nms_conf = nms_thesh)
    #pred_old = write_results(pred_old, confidence, num_classes, nms_conf = nms_thesh)
    #print("PREDICTION: ",prediction)
    #print("PREDICTION_OLD: ",pred_old)
    end = time.time()

    if type(prediction) == int:
        i += 1
        continue
    end = time.time()
    prediction[:,0] += i*batch_size

    if not write:                      #If we have't initialised output
        output = prediction
        write = 1
    else:
        output = torch.cat((output,prediction))

    for im_num, image in enumerate(imlist[i*batch_size: min((i +  1)*batch_size, len(imlist))]):
        im_id = i*batch_size + im_num
        #print("{0:20s} predicted in {1:6.3f} seconds".format(image.split("/")[-1], (end - start)/batch_size))
        print("{0:20s} {1:s}".format("Objects Detected:", ""))
        print("----------------------------------------------------------")
        image_id = image.split("/")[-1].split(".")[0]

    i += 1
    if CUDA:
        torch.cuda.synchronize()
try:
    output
```

```
except NameError:
    print ("No detections were made")
    return ['']#exit()

im_dim_list = torch.index_select(im_dim_list, 0, output[:,0].long())

scaling_factor = torch.min(inp_dim/im_dim_list,1)[0].view(-1,1)


output[:,[1,3]] -= (inp_dim - scaling_factor*im_dim_list[:,0].view(-1,1))/2
output[:,[2,4]] -= (inp_dim - scaling_factor*im_dim_list[:,1].view(-1,1))/2

output[:,1:5] /= scaling_factor

for i in range(output.shape[0]):
    output[i, [1,3]] = torch.clamp(output[i, [1,3]], 0.0, im_dim_list[i,0])
    output[i, [2,4]] = torch.clamp(output[i, [2,4]], 0.0, im_dim_list[i,1])

output_recast = time.time()
class_load = time.time()
colors = pkl.load(open("../input/pallete/pallete", "rb"))

draw = time.time()
sub_str = " " #initialise to a space value
def write(x, batches, results):
    c1 = tuple(x[1:3].int())
    c2 = tuple(x[3:5].int())
    img = results[int(x[0])]
    cls = int(x[-1])
    color = random.choice(colors)
    label = "{0}".format(classes[cls])
    cv2.rectangle(img, c1, c2,color, 1)
    #sajin
    #d_class_label[classes[cls]]
    height = img.shape[0]
    width = img.shape[1]
    x_min = np.around(c1[0].data.numpy()/width,decimals = 2)
    y_min = np.around(c1[1].data.numpy()/height,decimals = 2)
    x_max = np.around(c2[0].data.numpy()/width,decimals = 2)
    y_max = np.around(c2[1].data.numpy()/height,decimals = 2)
    confi = np.around(x[-3].data.numpy(), decimals = 2)
    #print("iiii..",t,tf.divide(t ,img.shape[0] ) )
    #print("Box ",image_id,c1, c2, classes[cls],d_class_label[classes[cls].lower()],img.shape)
    #print("Box1 ",image_id,d_class_label[classes[cls].lower()],confi,x_min,y_min,x_max,y_max)
    sub_str = d_class_label[classes[cls].lower()]+' '+str(confi)+' '+str(x_min)+' '+str(y_min)+' '+str(x_max)+' '+str(y_max)
    #font = ImageFont.truetype(font='../input/firamonomedium/FiraMono-Medium.otf',size=np.floor(3e-2 * img.size[1] + 0.5).
  astype('int32'))
    #t_size = cv2.getTextSize(label, font, 2 , 1)[0]
    #end
    t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_COMPLEX_SMALL, 1 , 1)[0] # 1 to 2 Sajin
    c2 = c1[0] + t_size[0] + 3, c1[1] + t_size[1] + 4
    cv2.rectangle(img, c1, c2,color, -1)
    cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, [225,255,255], 1);
    #cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), font, 1, [225,255,255], 1); #sajin
    return sub_str


sub_str =  list(map(lambda x: write(x, im_batches, orig_ims), output))
#print("Final...",sub_str)
#det_names = pd.Series(imlist).apply(lambda x: "{}/det_{}".format(args.det,x.split("/")[-1]))
#print("Det names: ",det_names)
det_names = "output_image.png"
#list(map(cv2.imwrite, det_names, loaded_ims))
#list(map(cv2.imwrite, "im.png", loaded_ims))
#list(map(cv2.imshow, det_names, loaded_ims))
#cv2.waitKey(0)
#cv2.destroyAllWindows()
#
'''
import matplotlib.pyplot as plt
for img in orig_ims:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    print("Img width:",img.shape[0])
    #plt.xticks(np.arange(img.shape[0], img.shape[0]+1, 100))
    plt.figure(figsize=(12,12))
    plt.imshow(img)
    plt.show()

end = time.time()

print("SUMMARY")
print("----------------------------------------------------------")
print("{:25s}: {}".format("Task", "Time Taken (in seconds)"))
print()
print("{:25s}: {:2.3f}".format("Reading addresses", load_batch - read_dir))
print("{:25s}: {:2.3f}".format("Loading batch", start_det_loop - load_batch))
print("{:25s}: {:2.3f}".format("Detection (" + str(len(imlist)) +  " images)", output_recast - start_det_loop))
print("{:25s}: {:2.3f}".format("Output Processing", class_load - output_recast))
print("{:25s}: {:2.3f}".format("Drawing Boxes", end - draw))
print("{:25s}: {:2.3f}".format("Average time_per_img", (end - load_batch)/len(imlist)))
print("----------------------------------------------------------")
```

```
    '''

    torch.cuda.empty_cache()
    return sub_str
```

Rank 2
Chunk ID: 1073 **Distance:** 0.7635 **Status:** Not Relevant

```
f = open("../input/class-labels-500/class-descriptions-500.csv","r")
d_class_label = {}
for rec in f:
    line = rec.split(",")
    d_class_label[line[1][:-1].lower()] = line[0]
d_class_label["remote control"] = "/m/0qjjc"
d_class_label["frisbee"] = "/m/0df_n8"
print(d_class_label)
# --- Next Cell ---
model = Darknet("../input/yolov3cfg/yolov3.cfg")
inp = get_test_input("../input/google-ai-open-images-object-detection-track/test/challenge2018_test/00001a21632de752.jpg",416)
#print(inp)
pred = model(inp, torch.cuda.is_available())
print ("Pred..",pred)
#f = open("ouput.csv","w")
#print (pred)

model = Darknet("../input/yolov3cfg/yolov3.cfg")
model.load_weights("../input/yolov3-weights/yolov3.weights")
```

Rank 3
Chunk ID: 1075 **Distance:** 0.7694 **Status:** Not Relevant

```
coco_classes = "../input/coco-classesv2/coco_classesv2.txt"
weightsfile = "../input/yolov3-weights/yolov3.weights"
cfgfile = "../input/yolov3cfg/yolov3.cfg"
img_path = "../input/google-ai-open-images-object-detection-track/test/challenge2018_test/00000b4dcff7f799.jpg"
#00001a21632de752.jpg"
'''
batches = 1
model = getmodel(cfgfile, weightsfile)
sub_str1 = detect(model,img_path,batches,0.5,0.4, coco_classes, weightsfile, cfgfile, "416")
print(sub_str1)
'''


# --- Next Cell ---
#model = Darknet("../input/yolov3cfg/yolov3.cfg")
#inp = get_test_input("../input/google-ai-open-images-object-detection-track/test/challenge2018_test/00001a21632de752.jpg",416)
#print(inp)
#pred = model(inp, torch.cuda.is_available())
#print (pred)
#print (pred.shape)
```