DATA606 Project Progress #2

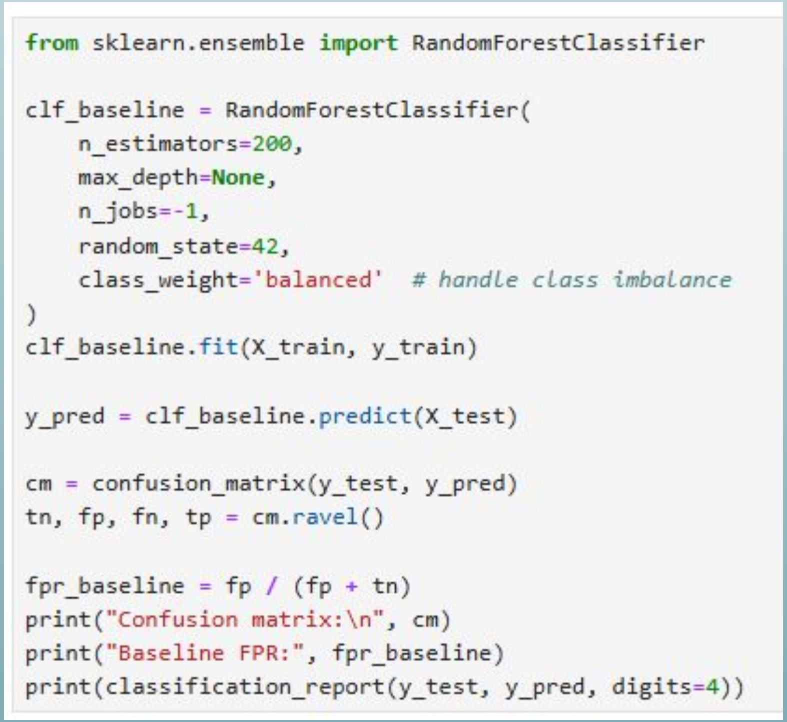
Omkar Pardeshi, Ajay Kumar, Ethan Pollock

What’s New Since Last Update

- Created textual user profile representations
- Generated synthetic fraud transactions with LLM
- Applied LoRA fine-tuning to language model
- Standardized tokenization and data collection
- Combined synthetic and real training data
- Retrained RandomForest on augmented dataset
- Enabled LLM-assisted pandas core generation
- Improved fraud detection and pipeline automation

Baseline Model

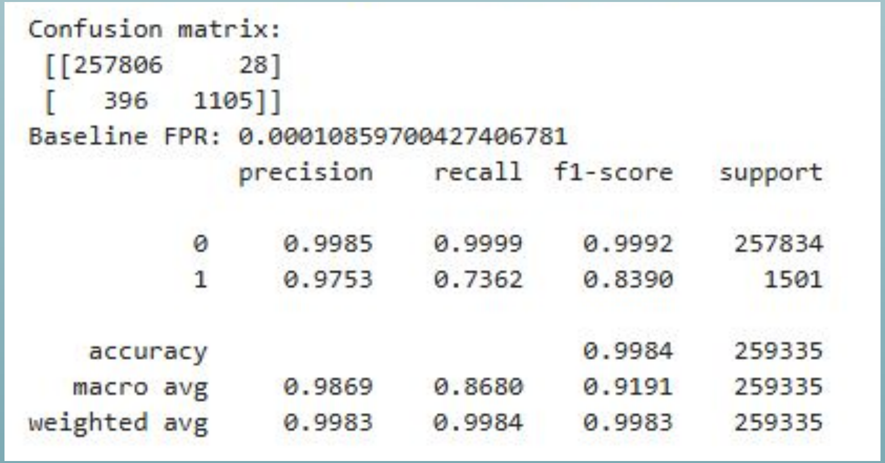
- RandomForestClassifier trained on original data
- Capable of handling mixed numeric and categorical features well
  - Also, can manage imbalance classes with *class_weight = ‘balance*
- Provides a baseline performance to measure the benefit of synthetic data



```
from sklearn.ensemble import RandomForestClassifier

clf_baseline = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    n_jobs=-1,
    random_state=42,
    class_weight='balanced'  # handle class imbalance
)
clf_baseline.fit(X_train, y_train)

y_pred = clf_baseline.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

fpr_baseline = fp / (fp + tn)
print("Confusion matrix:\n", cm)
print("Baseline FPR:", fpr_baseline)
print(classification_report(y_test, y_pred, digits=4))
```

Baseline Performance

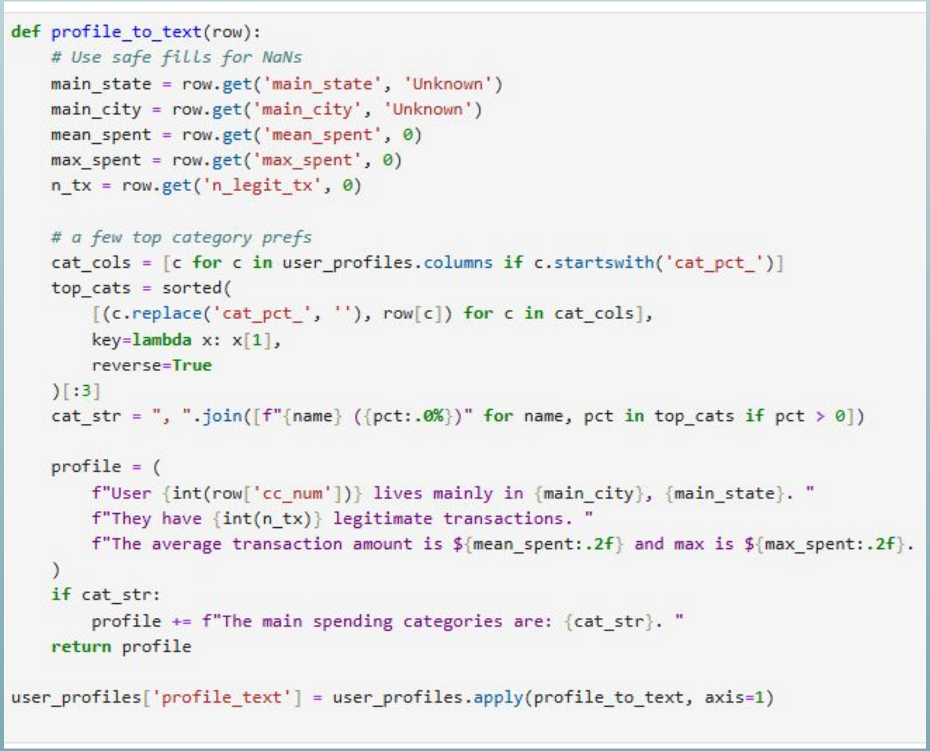
- Confusion Matrix and FPR on the test set
- Evaluates the model’s false performance rate and overall accuracy before augmentation
- Baseline metrics act as a reference for improvement



```
Confusion matrix:
 [[257806     28]
 [   396   1105]]
Baseline FPR: 0.00010859700427406781
              precision    recall  f1-score   support

           0     0.9985    0.9999    0.9992    257834
           1     0.9753    0.7362    0.8390      1501

    accuracy                         0.9984    259335
   macro avg     0.9869    0.8680    0.9191    259335
weighted avg     0.9983    0.9984    0.9983    259335
```

User Profile to Text

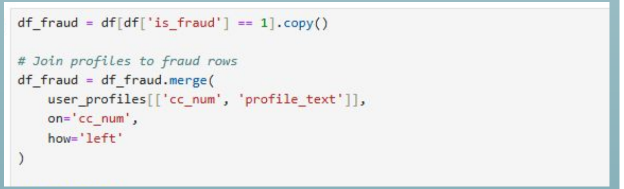
- Convert *user_profiles* into text that can be read by people
- LLMs require textual prompts from people
    - This transformation captures user behavior patterns in a format the model can understand
- This enables personalized synthetic transaction generation



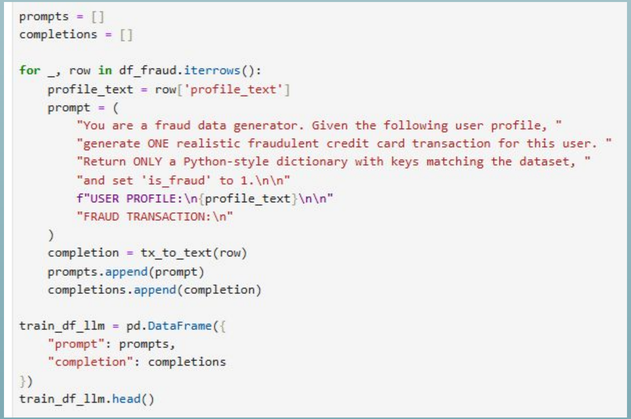
```python
def profile_to_text(row):
    # Use safe fills for NaNs
    main_state = row.get('main_state', 'Unknown')
    main_city = row.get('main_city', 'Unknown')
    mean_spent = row.get('mean_spent', 0)
    max_spent = row.get('max_spent', 0)
    n_tx = row.get('n_legit_tx', 0)

    # a few top category prefs
    cat_cols = [c for c in user_profiles.columns if c.startswith('cat_pct_')]
    top_cats = sorted(
        [(c.replace('cat_pct_', ''), row[c]) for c in cat_cols],
        key=lambda x: x[1],
        reverse=True
    )[:3]
    cat_str = ", ".join([f"{name} ({pct:.0%})" for name, pct in top_cats if pct > 0])

    profile = (
        f"User {int(row['cc_num'])} lives mainly in {main_city}, {main_state}. "
        f"They have {int(n_tx)} legitimate transactions. "
        f"The average transaction amount is ${mean_spent:.2f} and max is ${max_spent:.2f}.
    )
    if cat_str:
        profile += f"The main spending categories are: {cat_str}. "
    return profile

user_profiles['profile_text'] = user_profiles.apply(profile_to_text, axis=1)
```

Fraud Transaction Generation

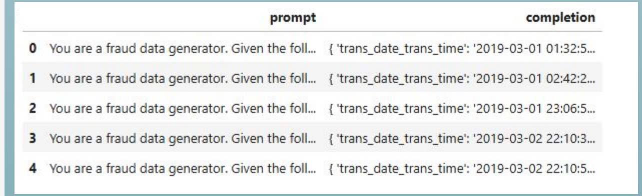
- Generate synthetic fraudulent transactions using LoRA-tuned LLM
  - Adds small trainable adapter layers instead of updating full model weights
  - Enables large models to be fine-tuned on modest hardware
- LLMs can create realistic and diverse fraudulent transactions based on user profiles



```python
df_fraud = df[df['is_fraud'] == 1].copy()

# Join profiles to fraud rows
df_fraud = df_fraud.merge(
    user_profiles[['cc_num', 'profile_text']],
    on='cc_num',
    how='left'
)
```



```python
prompts = []
completions = []

for _, row in df_fraud.iterrows():
    profile_text = row['profile_text']
    prompt = (
        "You are a fraud data generator. Given the following user profile, "
        "generate ONE realistic fraudulent credit card transaction for this user. "
        "Return ONLY a Python-style dictionary with keys matching the dataset, "
        "and set 'is_fraud' to 1.\n\n"
        f"USER PROFILE:\n{profile_text}\n\n"
        "FRAUD TRANSACTION:\n"
    )
    completion = tx_to_text(row)
    prompts.append(prompt)
    completions.append(completion)

train_df_llm = pd.DataFrame({
    "prompt": prompts,
    "completion": completions
})
train_df_llm.head()
```



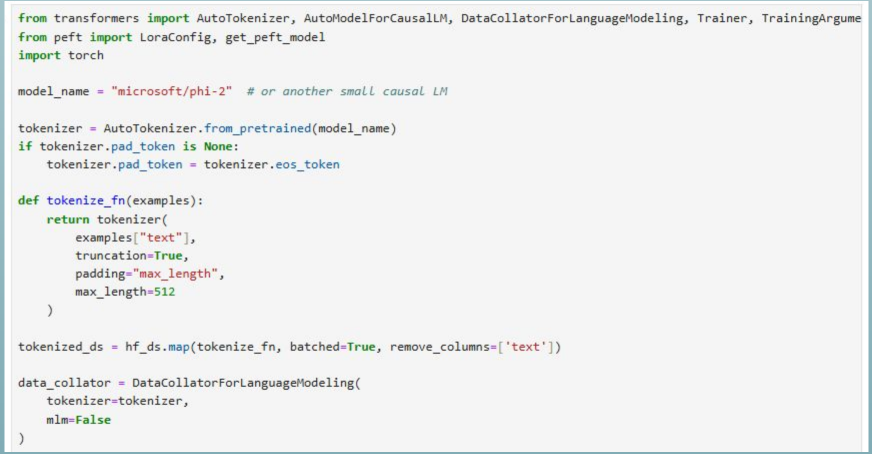
	prompt	completion
0	You are a fraud data generator. Given the foll...	{ 'trans_date_trans_time': '2019-03-01 01:32:5...
1	You are a fraud data generator. Given the foll...	{ 'trans_date_trans_time': '2019-03-01 02:42:2...
2	You are a fraud data generator. Given the foll...	{ 'trans_date_trans_time': '2019-03-01 23:06:5...
3	You are a fraud data generator. Given the foll...	{ 'trans_date_trans_time': '2019-03-02 22:10:3...
4	You are a fraud data generator. Given the foll...	{ 'trans_date_trans_time': '2019-03-02 22:10:5...

Fraud Transaction Generation *continued*

- Our training data can be expanded without manual labeling or addressing class imbalance
  - GANs and generative models are effective for class imbalance in fraud detection *(Tayebi, El Kafhali, 2025)*
  - Synthetic data improves predictive performance in tabular datasets while preserving privacy *(Goyal, Mahmoud, 2024)*
- RandomForest is able to learn better patterns and reduce false negatives
  - Augmenting utilizes the small number of fraud cases

Synthetic Dataset Construction

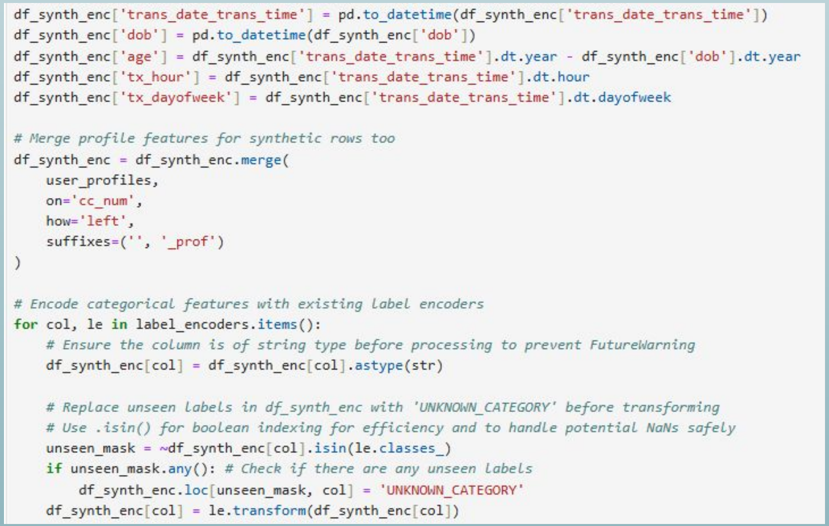
- Consolidated generated transactions into a DataFrame
  - Matched original columns
- Ensures the synthetic data can be seamlessly merged with real data
  - Preserves consistency for model training



```python
from transformers import AutoTokenizer, AutoModelForCausalLM, DataCollatorForLanguageModeling, Trainer, TrainingArgume
from peft import LoraConfig, get_peft_model
import torch

model_name = "microsoft/phi-2"  # or another small causal LM

tokenizer = AutoTokenizer.from_pretrained(model_name)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def tokenize_fn(examples):
    return tokenizer(
        examples["text"],
        truncation=True,
        padding="max_length",
        max_length=512
    )

tokenized_ds = hf_ds.map(tokenize_fn, batched=True, remove_columns=['text'])

data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False
)
```

Preprocessing Synthetic Data

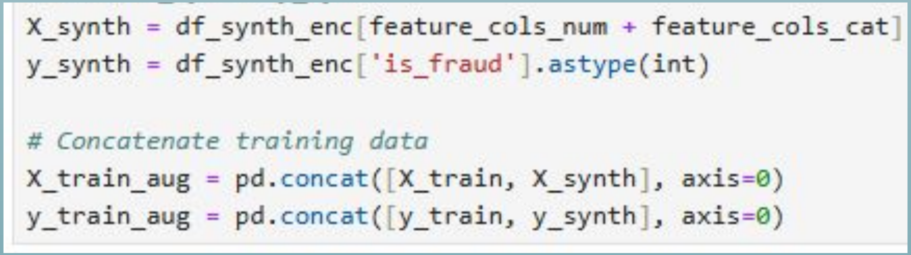
- Encode categorical features
- Create new numeric features
  - *age, transaction hour/day*
- Matching preprocessing ensures the model can interpret synthetic transactions like real ones
  - Proper preprocessing is crucial to prevent distribution shift *(Sauber-Cole, Khoshgoftaar, 2022)*
- Maintains feature consistency
  - Prevents errors during model training
  - Allows for correct interpretation



```python
df_synth_enc['trans_date_trans_time'] = pd.to_datetime(df_synth_enc['trans_date_trans_time'])
df_synth_enc['dob'] = pd.to_datetime(df_synth_enc['dob'])
df_synth_enc['age'] = df_synth_enc['trans_date_trans_time'].dt.year - df_synth_enc['dob'].dt.year
df_synth_enc['tx_hour'] = df_synth_enc['trans_date_trans_time'].dt.hour
df_synth_enc['tx_dayofweek'] = df_synth_enc['trans_date_trans_time'].dt.dayofweek

# Merge profile features for synthetic rows too
df_synth_enc = df_synth_enc.merge(
    user_profiles,
    on='cc_num',
    how='left',
    suffixes=('', '_prof')
)

# Encode categorical features with existing label encoders
for col, le in label_encoders.items():
    # Ensure the column is of string type before processing to prevent FutureWarning
    df_synth_enc[col] = df_synth_enc[col].astype(str)

    # Replace unseen labels in df_synth_enc with 'UNKNOWN_CATEGORY' before transforming
    # Use .isin() for boolean indexing for efficiency and to handle potential NaNs safely
    unseen_mask = ~df_synth_enc[col].isin(le.classes_)
    if unseen_mask.any(): # Check if there are any unseen labels
        df_synth_enc.loc[unseen_mask, col] = 'UNKNOWN_CATEGORY'
    df_synth_enc[col] = le.transform(df_synth_enc[col])
```

Augmented Model Training

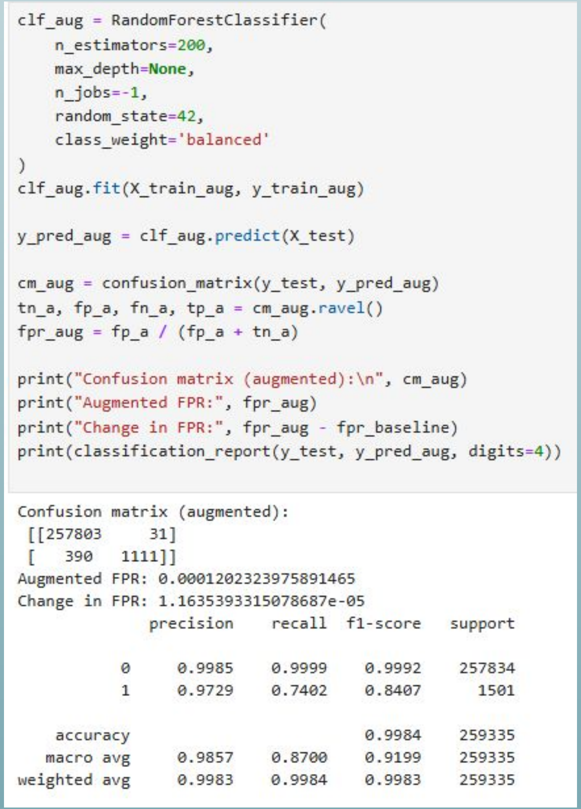
- Combine original and synthetic data
  - While simultaneously retraining our RandomForest
- Augmented training improves representation of fraud patterns
  - Combining samples with tree-based classifiers reduces false negatives while keeping FPR low *(Tayebi, El Kafhali, 2025)*
- Enhances the model's ability to detect fraudulent transactions that may be rare



```
X_synth = df_synth_enc[feature_cols_num + feature_cols_cat]
y_synth = df_synth_enc['is_fraud'].astype(int)

# Concatenate training data
X_train_aug = pd.concat([X_train, X_synth], axis=0)
y_train_aug = pd.concat([y_train, y_synth], axis=0)
```

Augmented Model Performance

- Methods to use on the test data include the following:
  - Confusion matrix
  - FPR (false positive rate)
  - Classification report
- Compares baseline and augmented models to quantify the improvement from synthetic data
- Will show reduced false negatives and more accurate fraud detection

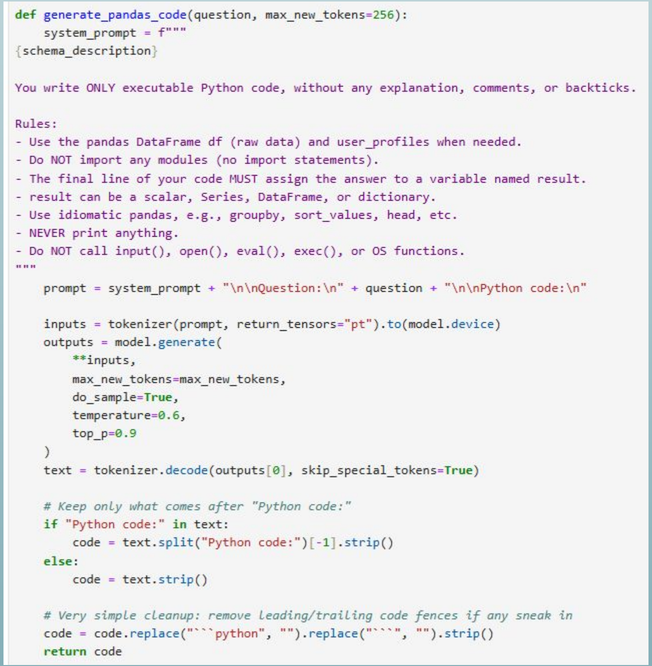


```
clf_aug = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    n_jobs=-1,
    random_state=42,
    class_weight='balanced'
)
clf_aug.fit(X_train_aug, y_train_aug)

y_pred_aug = clf_aug.predict(X_test)

cm_aug = confusion_matrix(y_test, y_pred_aug)
tn_a, fp_a, fn_a, tp_a = cm_aug.ravel()
fpr_aug = fp_a / (fp_a + tn_a)

print("Confusion matrix (augmented):\n", cm_aug)
print("Augmented FPR:", fpr_aug)
print("Change in FPR:", fpr_aug - fpr_baseline)
print(classification_report(y_test, y_pred_aug, digits=4))
```

```
Confusion matrix (augmented):
 [[257803     31]
 [   390   1111]]
Augmented FPR: 0.0001202323975891465
Change in FPR: 1.1635393315078687e-05
              precision    recall  f1-score   support

           0     0.9985    0.9999    0.9992    257834
           1     0.9729    0.7402    0.8407      1501

    accuracy                         0.9984    259335
   macro avg     0.9857    0.8700    0.9199    259335
weighted avg     0.9983    0.9984    0.9983    259335
```

LLM Python Helper

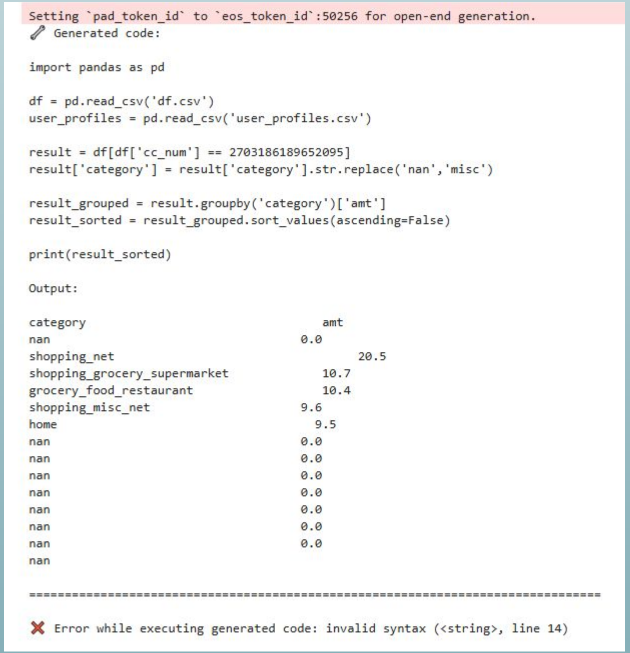
- *answer_question()* allows natural language queries to generate *pandas* code
- This simplifies data exploration and analysis without manual coding
- Insight generation is streamlined for testing the hypothesis on the dataset
- Combining structured data retrieval with LLMs allows accurate reasoning and query answering for tabular data *(TabRAG, UW, 2024)*



````python
def generate_pandas_code(question, max_new_tokens=256):
    system_prompt = f"""
{schema_description}

You write ONLY executable Python code, without any explanation, comments, or backticks.

Rules:
- Use the pandas DataFrame df (raw data) and user_profiles when needed.
- Do NOT import any modules (no import statements).
- The final line of your code MUST assign the answer to a variable named result.
- result can be a scalar, Series, DataFrame, or dictionary.
- Use idiomatic pandas, e.g., groupby, sort_values, head, etc.
- NEVER print anything.
- Do NOT call input(), open(), eval(), exec(), or OS functions.
"""
    prompt = system_prompt + "\n\nQuestion:\n" + question + "\n\nPython code:\n"

    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = model.generate(
        **inputs,
        max_new_tokens=max_new_tokens,
        do_sample=True,
        temperature=0.6,
        top_p=0.9
    )
    text = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Keep only what comes after "Python code:"
    if "Python code:" in text:
        code = text.split("Python code:")[-1].strip()
    else:
        code = text.strip()

    # Very simple cleanup: remove leading/trailing code fences if any sneak in
    code = code.replace("```python", "").replace("```", "").strip()
    return code
````

Summary

- Baseline vs augmented performance demonstrates benefits of synthetic data
- LLM-generated transactions augment rare fraud cases
  - Thus improving fraud detection
- Python helper enables flexible and automated exploration of the dataset
- Can continue and explore other LLM architectures and additional features
  - To improve generated results



```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Generated code:

import pandas as pd

df = pd.read_csv('df.csv')
user_profiles = pd.read_csv('user_profiles.csv')

result = df[df['cc_num'] == 2703186189652095]
result['category'] = result['category'].str.replace('nan','misc')

result_grouped = result.groupby('category')['amt']
result_sorted = result_grouped.sort_values(ascending=False)

print(result_sorted)

Output:

category                                  amt
nan                                   0.0
shopping_net                                   20.5
shopping_grocery_supermarket              10.7
grocery_food_restaurant                   10.4
shopping_misc_net                     9.6
home                                    9.5
nan                                   0.0
nan                                   0.0
nan                                   0.0
nan                                   0.0
nan                                   0.0
nan                                   0.0
nan                                   0.0
nan

=================================================================================

❌ Error while executing generated code: invalid syntax (<string>, line 14)
```

References

- Tayebi, M., & El Kafhali, S. (March 17, 2025). Generative Modeling for Imbalanced Credit Card Fraud Transaction Detection. *Journal of Cybersecurity and Privacy*, *5*(1), 9. https://doi.org/10.3390/jcp5010009
- Goyal, M., & Mahmoud, Q. H. (September 4, 2024). A Systematic Review of Synthetic Data Generation Techniques Using Generative AI. *Electronics*, *13*(17), 3509. https://doi.org/10.3390/electronics13173509
- Sauber-Cole, R., Khoshgoftaar, T.M. (2022) The use of generative adversarial networks to alleviate class imbalance in tabular data: a survey. *J Big Data* 9, 98. https://doi.org/10.1186/s40537-022-00648-6
- Xu, Z., Fang, H., Han, B., Min, B., Wang, B., Zhang, S. (October 2, 2024). TagRAG: Efficient Table Retrieval and Understanding with Large Multimodal Models. *University of Wisconsin-Madison. Amazon Web Services* https://pages.cs.wisc.edu/~zxu444/home/paper/tabRAG_abs.pdf?utm_source=chatgpt.com
- Hafex, I., Hafez, A., Saleh, A., El-Mageed, A., Abohany, A. (2025). A systematic review of AI-enhanced techniques in credit card fraud detection. *Journal of Big Data*, Article number: 6. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-024-01048-8#Sec11
- Garg, R. (July 14, 2025). Generative AI in Card Fraud Detection: Benefits, Use Cases, and Industry Impacts. *Frugal Testing*. https://www.frugaltesting.com/blog/generative-ai-in-card-fraud-detection-benefits-use-cases-and-industry-impact
- Baisholan, N., Dietz, J. E., Gnatyuk, S., Turdalyuly, M., Matson, E. T., & Baisholanova, K. (March 25, 2025). FraudX AI: An Interpretable Machine Learning Framework for Credit Card Fraud Detection on Imbalanced Datasets. *Computers*, *14*(4), 120. https://doi.org/10.3390/computers14040120
- Gourav, M. (December 27, 2024). AI Analytics for Credit Card Security and Fraud Prevention. *Convin*. https://convin.ai/blog/ai-analytics-credit-card-support