Capstone Project

License & Disclaimer

Author: Abigail de Guzman

Course: Post Graduate Diploma in AI and Machine Learning

Institution: AIM School of Executive Education and Lifelong Learning

Capstone Project Step 1: Problem Understanding & Framing

Business Context

Credit card fraud poses significant financial and reputational risks to financial institutions. Given the high volume of daily transactions, manual review is impractical, making automated fraud detection systems essential for identifying potentially fraudulent activity in a timely manner.

Data Science Problem

The objective of this project is to develop a machine learning model that can accurately identify fraudulent credit card transactions based on historical transaction data. This problem is framed as a **binary classification task**, where each transaction is classified as either fraudulent (1) or non-fraudulent (0).

Evaluation Metrics & Success Criteria

Due to the highly imbalanced nature of credit card fraud data, traditional accuracy is insufficient as a standalone performance metric. Model evaluation therefore focuses on **ROC-AUC** to measure overall discriminative ability, complemented by **Precision–Recall analysis** to assess the trade-off between detecting fraudulent transactions (recall) and minimizing false positives (precision).

Business KPIs

From a business perspective, success is measured by the model's ability to maximize fraud detection rates while minimizing false positives that may inconvenience legitimate customers. The model should support risk-based decision-making and allow for configurable classification thresholds based on an organization's tolerance for fraud losses versus customer friction.

Data Preprocessing

```
# Import Libraries for Data Analysis and Modeling

# Data processing
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Train/test split and preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Evaluation metrics
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    precision_recall_curve,
    confusion_matrix
)
```

```
# Data Upload and Exploration
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
file_path = '/content/drive/MyDrive/AIM - PGD AI and ML/Capstone Proj
```

```
# Load Credit Card dataset
df = pd.read_csv('/content/drive/MyDrive/AIM - PGD AI and ML/Capstone
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.23959
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.07880
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.79146
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.23760
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.59294

5 rows × 31 columns

```
df.shape
```

```
(284807, 31)
```

Dataset Overview

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
```

```
  16  V16     284807 non-null  float64
  17  V17     284807 non-null  float64
  18  V18     284807 non-null  float64
  19  V19     284807 non-null  float64
  20  V20     284807 non-null  float64
  21  V21     284807 non-null  float64
  22  V22     284807 non-null  float64
  23  V23     284807 non-null  float64
  24  V24     284807 non-null  float64
  25  V25     284807 non-null  float64
  26  V26     284807 non-null  float64
  27  V27     284807 non-null  float64
  28  V28     284807 non-null  float64
  29  Amount  284807 non-null  float64
  30  Class   284807 non-null  int64
 dtypes: float64(30), int64(1)
 memory usage: 67.4 MB
```

```
df.describe()
```

	Time	V1	V2	V3	V4
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01

8 rows × 31 columns

Data Dictionary

The dataset used in this project is the publicly available **Credit Card Fraud Dataset** released by the Machine Learning Group of Université Libre de Bruxelles. The data contains anonymized credit card transactions made by European cardholders in September 2013.

Feature	Data Type	Description
Time	float	Number of seconds elapsed between a given transaction and the first transaction in the da
V1–V28	float	Anonymized numerical features obtained via Principal Component Analysis (PCA) to protec
Amount	float	Transaction amount in euros
Class	int	Target variable indicating whether a transaction is fraudulent (1) or non-fraudulent (0)

Due to confidentiality constraints, the original transaction attributes are not available, and only the transformed PCA components are provided. As a result, feature interpretation focuses on relative importance and model behavior rather than semantic meaning.

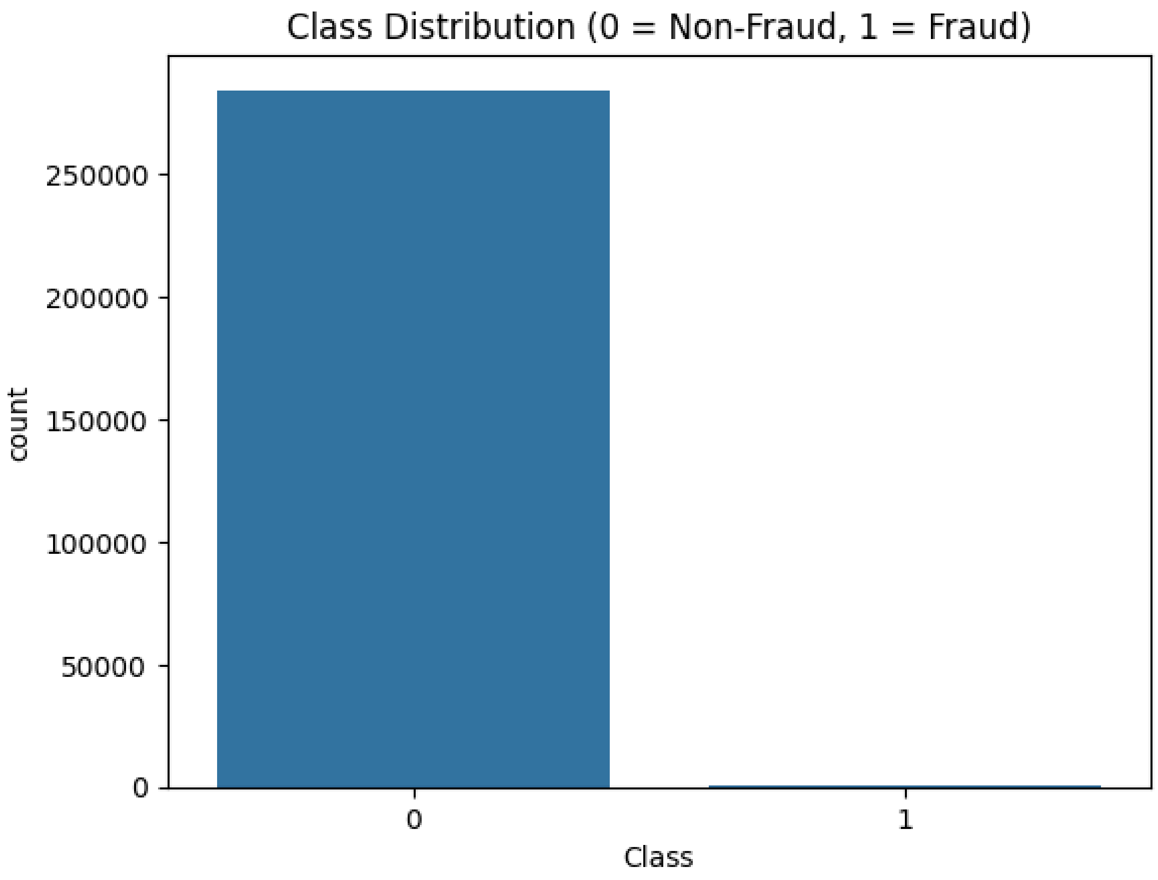
Data Quality Checks

```
df.isnull().sum()
```

The dataset was checked for missing values across all features, and no missing observations were found. As a result, no imputation or removal of records was required prior to model training.

Target Variable Distribution

```
sns.countplot(x="Class", data=df)
plt.title("Class Distribution (0 = Non-Fraud, 1 = Fraud)")
plt.show()

df["Class"].value_counts(normalize=True)
```



```
       proportion
Class
0        0.998273
1        0.001727

dtype: float64
```

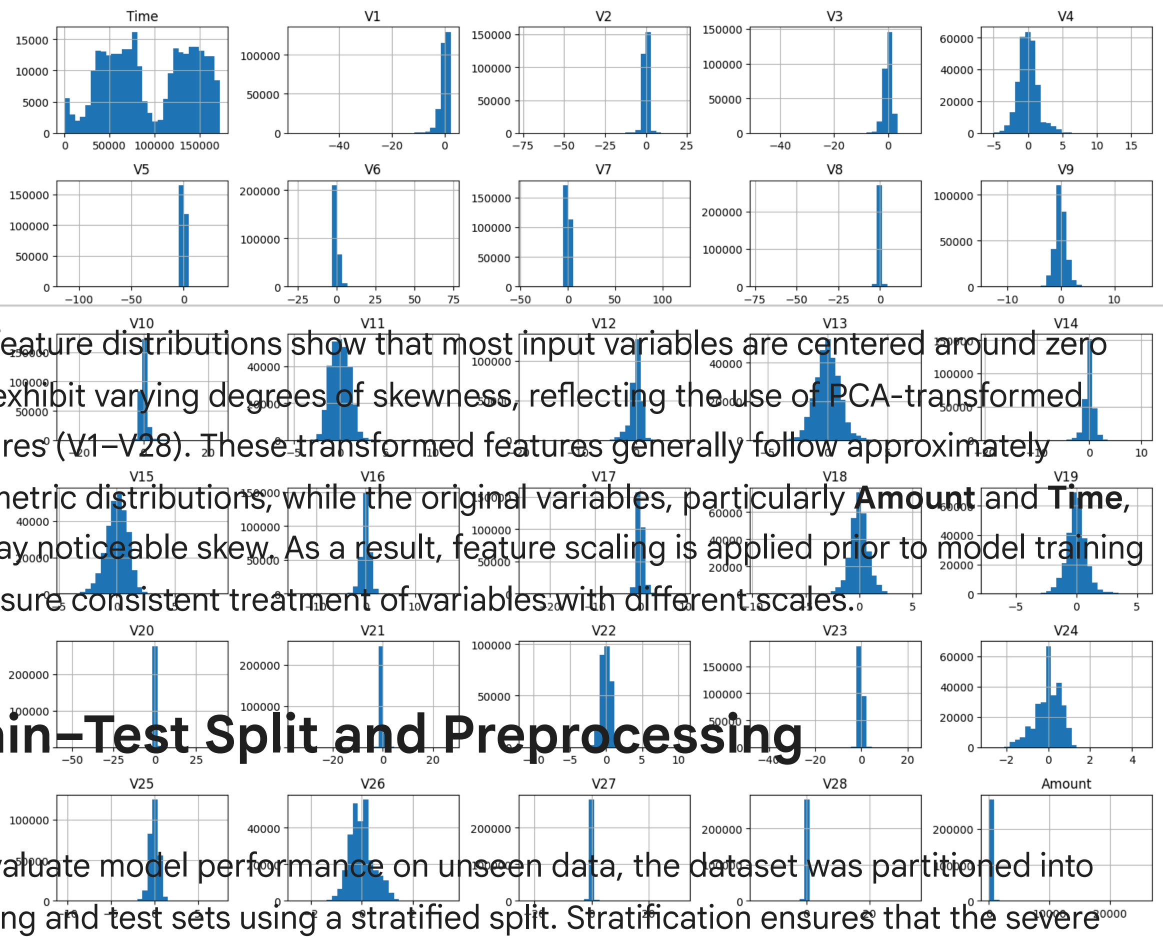
The target variable exhibits a severe class imbalance, with fraudulent transactions accounting for approximately 0.17% of all observations. This imbalance reflects real-

```
        0
Time    0
V1      0
V2      0
V3      0
V4      0
V24     0
V25     0
V27     0
V28     0
Amount  0
Class   0

dtype: int64
```

world fraud detection scenarios, where the vast majority of transactions are legitimate. In such settings, conventional accuracy-based evaluation is misleading, as a model can achieve high accuracy by predicting all transactions as non-fraudulent. To address this, model evaluation and training strategies in later stages emphasize imbalance-aware metrics and techniques, including ROC-AUC, Precision–Recall analysis, and class-weighted learning.

Feature Distributions

```
df.drop(columns=["Class"]).hist(
    figsize=(15, 12),
    bins=30
)
plt.tight_layout()
plt.show()
```



The feature distributions show that most input variables are centered around zero and exhibit varying degrees of skewness, reflecting the use of PCA-transformed features (V1–V28). These transformed features generally follow approximately symmetric distributions, while the original variables, particularly **Amount** and **Time**, display noticeable skew. As a result, feature scaling is applied prior to model training to ensure consistent treatment of variables with different scales.

Train–Test Split and Preprocessing

To evaluate model performance on unseen data, the dataset was partitioned into training and test sets using a stratified split. Stratification ensures that the severe class imbalance observed in the target variable is preserved across both subsets.

```
X = df.drop(columns=["Class"])
y = df["Class"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=42
)
```

An 80/20 train–test split was used, with a fixed random seed to ensure reproducibility of results.

Feature Scaling

Feature scaling was applied to standardize the input variables prior to model training. Since the dataset contains features with different magnitudes (particularly the original `Time` and `Amount` variables), standardization ensures that all features contribute proportionally during optimization, which is especially important for distance-based and gradient-based models.

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

The `Time` feature was retained to preserve temporal transaction patterns. In a production setting, this variable may be further engineered into cyclical or window-based features to better capture periodic transaction behavior.

Model 1: Logistic Regression

Logistic Regression was used as a baseline model due to its simplicity, interpretability, and common use in binary classification tasks. To address the severe class imbalance in the dataset, class weighting was applied to penalize misclassification of fraudulent transactions more heavily during training.

```
lr = LogisticRegression(
    max_iter=1000,
    class_weight="balanced"
)

lr.fit(X_train_scaled, y_train)

y_pred_lr = lr.predict(X_test_scaled)
y_proba_lr = lr.predict_proba(X_test_scaled)[:, 1]

print(classification_report(y_test, y_pred_lr))
print("ROC AUC:", roc_auc_score(y_test, y_proba_lr))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56864
           1       0.06      0.92      0.11        98

    accuracy                           0.98     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.98      0.99     56962

ROC AUC: 0.9720834996210077
```

The baseline model achieves strong overall discrimination as indicated by ROC-AUC, but exhibits low precision for the fraud class, reflecting the expected trade-off between fraud detection and false positives in highly imbalanced data.

Model 2: Random Forest

Random Forest was selected as a second model to capture non-linear relationships and feature interactions that may not be well represented by a linear baseline. To mitigate overfitting given the extreme class imbalance, the model was constrained through depth and leaf-size parameters, and class weighting was applied to emphasize correct identification of fraudulent transactions.

```
rf = RandomForestClassifier(
    n_estimators=30,         # ↓ from 100
    max_depth=10,            # caps tree complexity
    min_samples_leaf=50,     # prevents overfitting
    class_weight="balanced",
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
y_proba_rf = rf.predict_proba(X_test)[:, 1]

print(classification_report(y_test, y_pred_rf))
print("ROC AUC:", roc_auc_score(y_test, y_proba_rf))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.62      0.87      0.72        98

    accuracy                           1.00     56962
   macro avg       0.81      0.93      0.86     56962
weighted avg       1.00      1.00      1.00     56962

ROC AUC: 0.9764099699390165
```

Compared to the Logistic Regression baseline, the Random Forest model achieves a modest improvement in overall discrimination and substantially higher precision for the fraud class. This indicates a better balance between detecting fraudulent transactions and reducing false positives, making the model more suitable for practical fraud detection scenarios.

Model Comparison Table

The performance of the baseline Logistic Regression model and the Random Forest model was compared using ROC-AUC to evaluate their overall discriminative ability under severe class imbalance.

```
results = pd.DataFrame({
    "Model": ["Logistic Regression", "Random Forest"],
    "ROC AUC": [
        roc_auc_score(y_test, y_proba_lr),
        roc_auc_score(y_test, y_proba_rf)
    ]
})

results
```

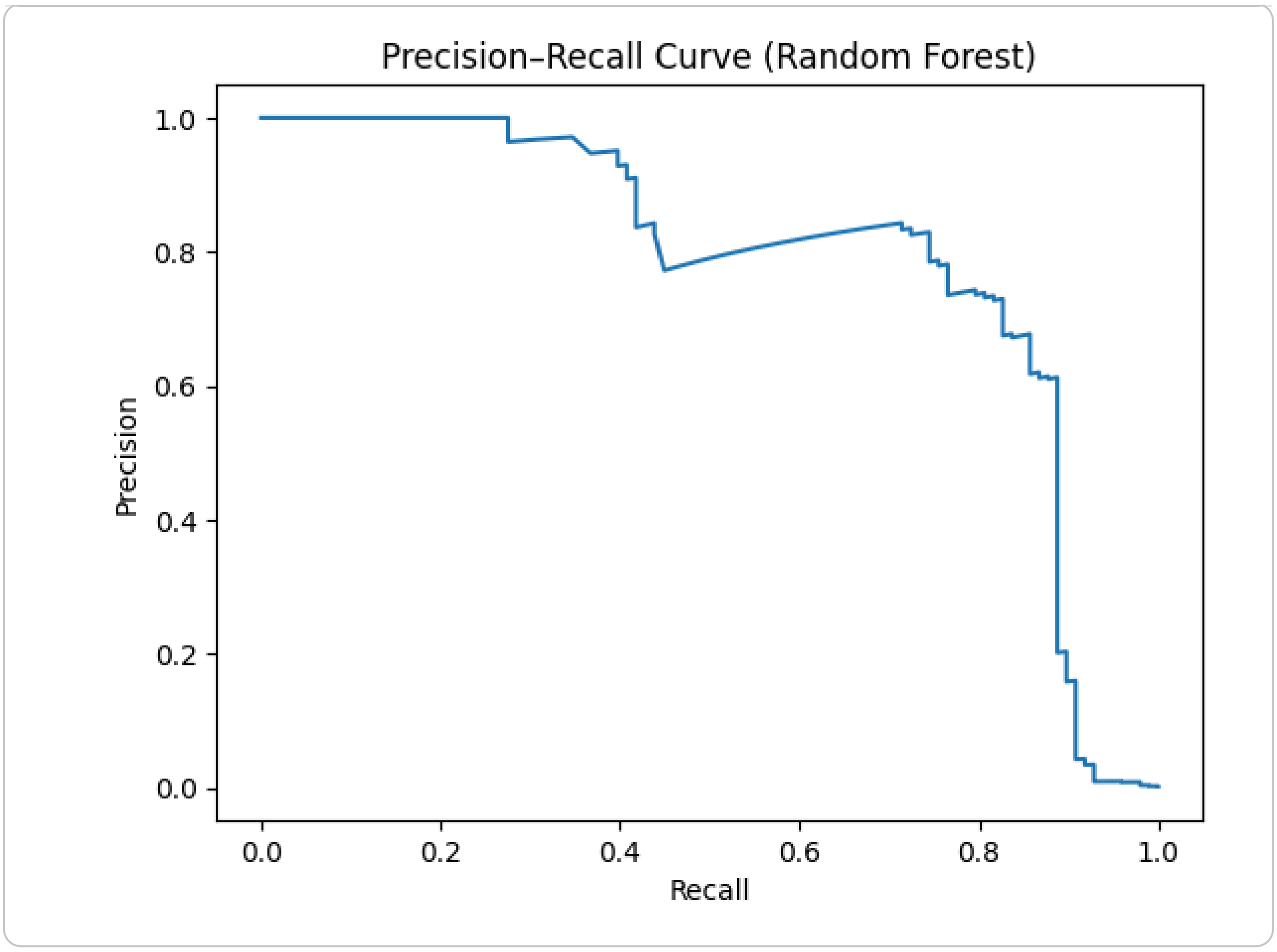
	Model	ROC AUC
0	Logistic Regression	0.972083
1	Random Forest	0.976410

Next steps: Generate code with results New interactive sheet

The Random Forest model outperforms the Logistic Regression baseline, achieving a higher ROC-AUC and substantially improved precision for the fraud class. This suggests that the Random Forest better captures complex, non-linear patterns in the data while reducing false positives, making it more suitable for practical fraud detection scenarios.

Precision–Recall Curve

```
precision, recall, thresholds = precision_recall_curve(y_test, y_prob

plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision–Recall Curve (Random Forest)")
plt.show()
```



The precision–recall curve illustrates the trade-off between identifying fraudulent transactions and minimizing false positives in a highly imbalanced setting. As recall increases, precision declines, reflecting the rising cost of incorrectly flagging legitimate transactions as fraud. This analysis underscores the importance of threshold selection when deploying the model in production, depending on the acceptable balance between fraud detection performance and operational burden.

Model Explainability (SHAP)

To improve interpretability and provide insight into the model's decision-making process, SHAP (SHapley Additive exPlanations) was applied to the trained model. SHAP values quantify the contribution of each feature to individual predictions, enabling analysis of which variables most strongly influence fraud classification.

```
# Install SHAP and import SHAP to support model explainability analys

!pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-
Requirement already satisfied: numpy>=2 in /usr/local/lib/python3.12/d
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dis
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.1
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.1
Requirement already satisfied: typing-extensions in /usr/local/lib/pyt
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/loca
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/li
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/d
```

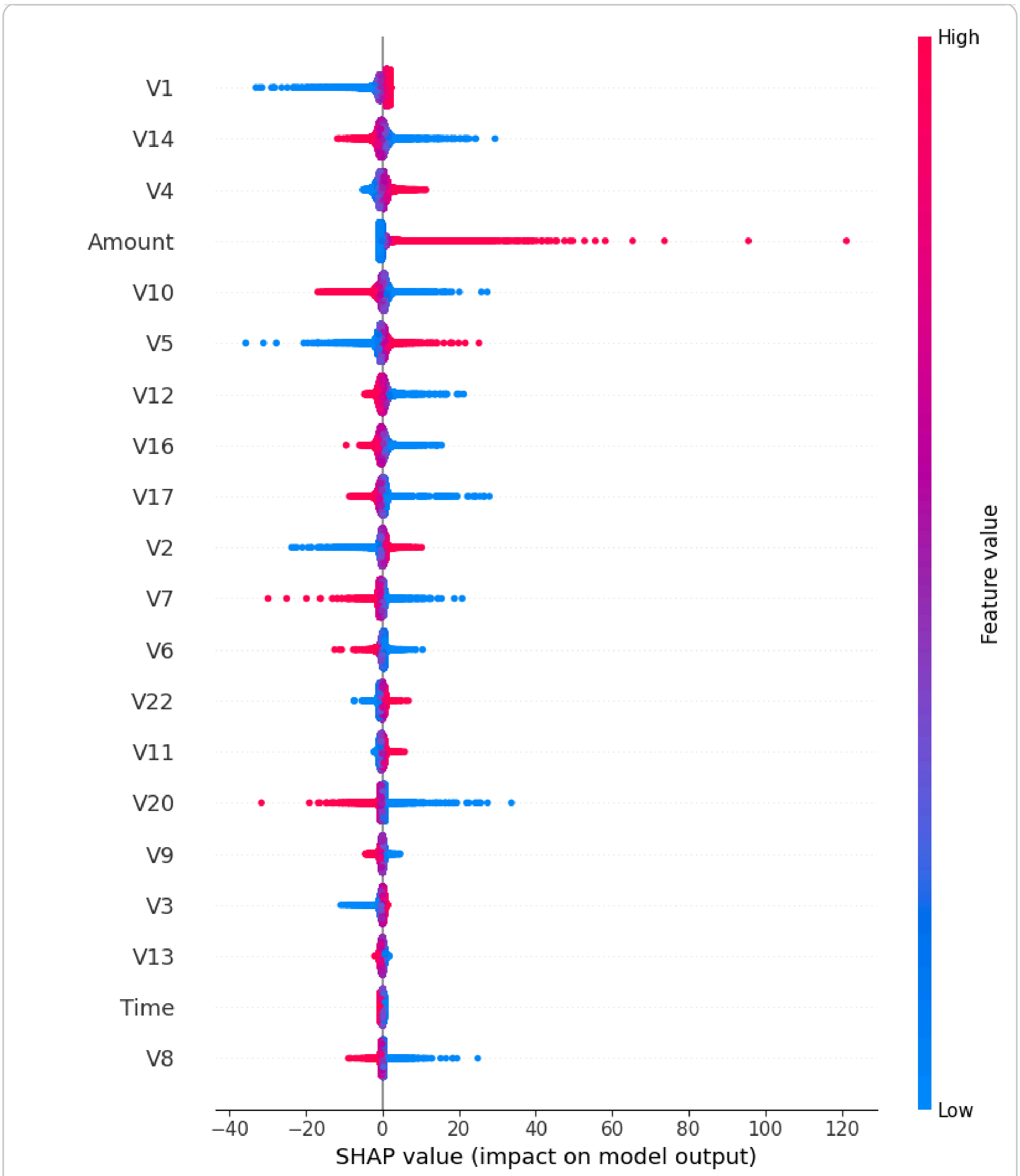
```
import shap

# Used a small background sample for speed
background = X_train_scaled[np.random.choice(X_train_scaled.shape[0],
```

SHAP Explainer Setup

```
explainer = shap.LinearExplainer(lr, background)
shap_values = explainer.shap_values(X_test_scaled)
```

SHAP summary plot

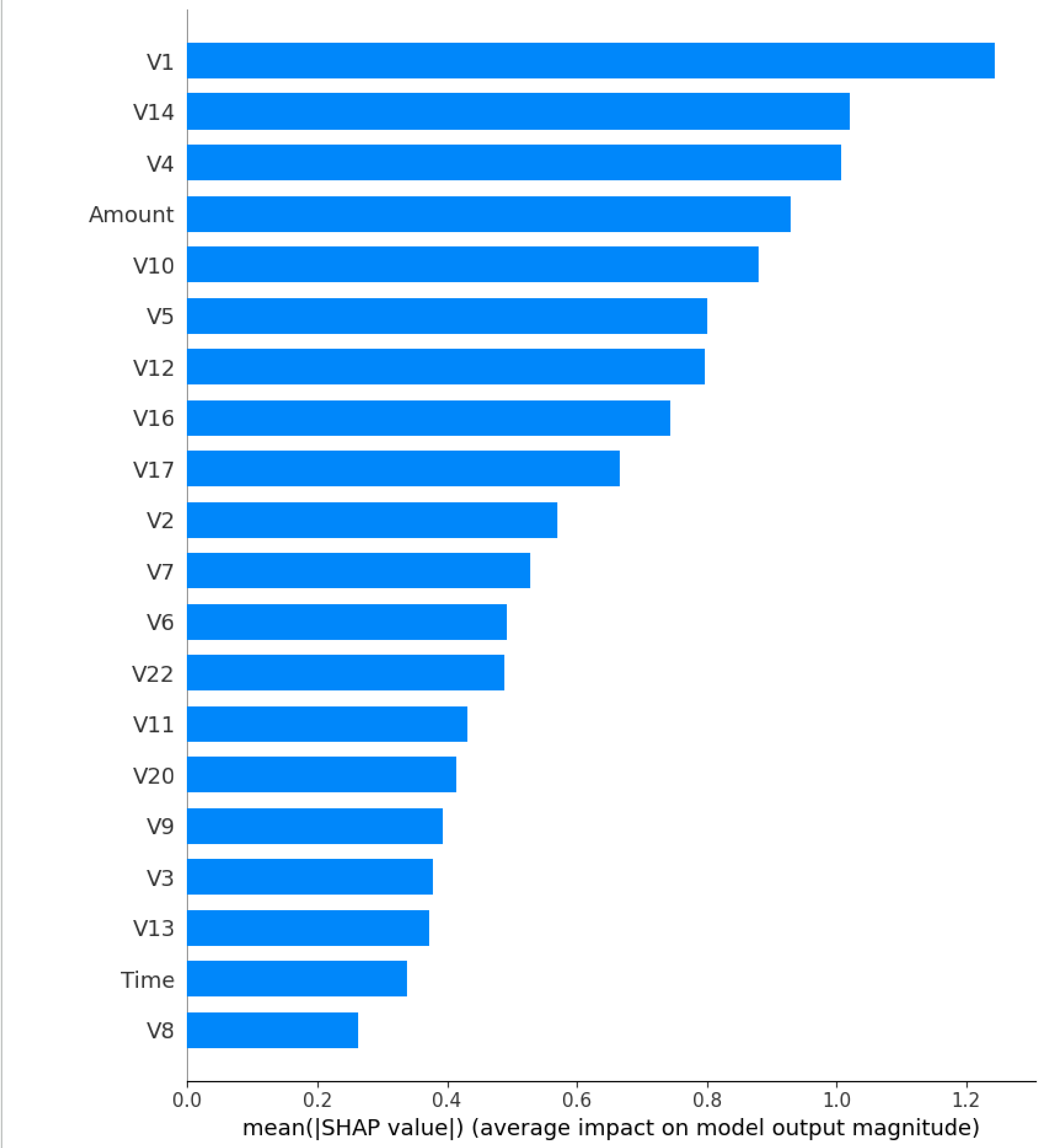
```
shap.summary_plot(
    shap_values,
    X_test_scaled,
    feature_names=X.columns,
    show=True
)
```



The SHAP summary plot visualizes the global impact of each feature on the model's predictions. Features are ranked by their average absolute SHAP value, indicating their overall importance in driving fraud classification.

Higher SHAP values (shown in red) correspond to feature values that increase the likelihood of a transaction being classified as fraudulent, while lower values (blue) decrease the predicted fraud risk. Due to feature anonymization via PCA, the analysis focuses on relative importance and directional influence rather than semantic interpretation of individual variables.

```
shap.summary_plot(
    shap_values,
    X_test_scaled,
    feature_names=X.columns,
    plot_type="bar",
    show=True
)
```



The bar plot aggregates SHAP values across all observations to highlight the most influential features in the model. Variables such as V1, V4, V14, and Amount emerge as the strongest contributors to fraud predictions, indicating that both transformed behavioral signals and transaction magnitude play a role in identifying suspicious activity.

Overall, SHAP analysis enhances model transparency by revealing which features consistently influence fraud predictions. While individual feature semantics are obscured due to PCA-based anonymization, the explainability results provide confidence that the model relies on meaningful transaction patterns rather than spurious correlations. This level of interpretability is critical for risk-sensitive domains such as financial fraud detection.

Bias Analysis: Transaction Amount as a Proxy

To assess whether model errors disproportionately affect transactions of different monetary values, transaction amount was used as a proxy variable for bias analysis. While transaction amount is not a protected attribute, it can reveal potential operational bias, such as whether high-value or low-value transactions are more likely to be incorrectly flagged as fraudulent.

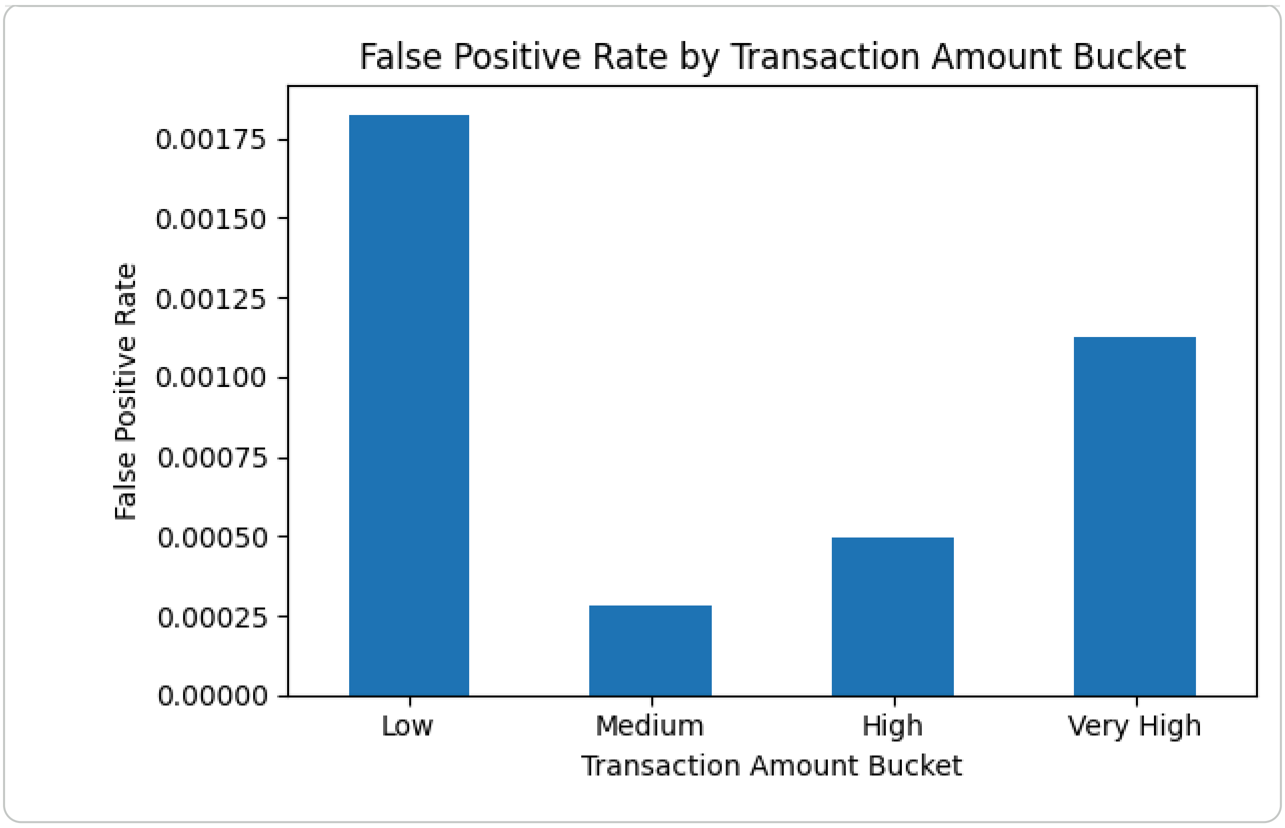
```
# Create amount buckets
df_test = X_test.copy()
df_test["Actual"] = y_test.values
df_test["Predicted"] = y_pred_rf
df_test["Amount"] = df.loc[X_test.index, "Amount"]

df_test["Amount_Bucket"] = pd.qcut(
    df_test["Amount"],
    q=4,
    labels=["Low", "Medium", "High", "Very High"]
)

# False positive rate by bucket
bias_summary = df_test.groupby("Amount_Bucket").apply(
    lambda x: ((x["Predicted"] == 1) & (x["Actual"] == 0)).mean()
)

bias_summary
```

```
/tmp/ipython-input-585588598.py:14: FutureWarning: The default of obse
  bias_summary = df_test.groupby("Amount_Bucket").apply(
/tmp/ipython-input-585588598.py:14: DeprecationWarning: DataFrameGroup
  bias_summary = df_test.groupby("Amount_Bucket").apply(
                          0
 Amount_Bucket
 
     Low            0.001825
   Medium           0.000280
    High            0.000493
  Very High         0.001124

dtype: float64
```

The table reports the false positive rate within each transaction amount bucket. False positives are defined as legitimate transactions incorrectly classified as fraud. Lower false positive rates indicate reduced customer friction for transactions within that amount range.

Chart 1: False Positive Rate by Amount Bucket

```
bias_summary.plot(
    kind="bar",
    figsize=(6, 4)
)
plt.ylabel("False Positive Rate")
plt.xlabel("Transaction Amount Bucket")
plt.title("False Positive Rate by Transaction Amount Bucket")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

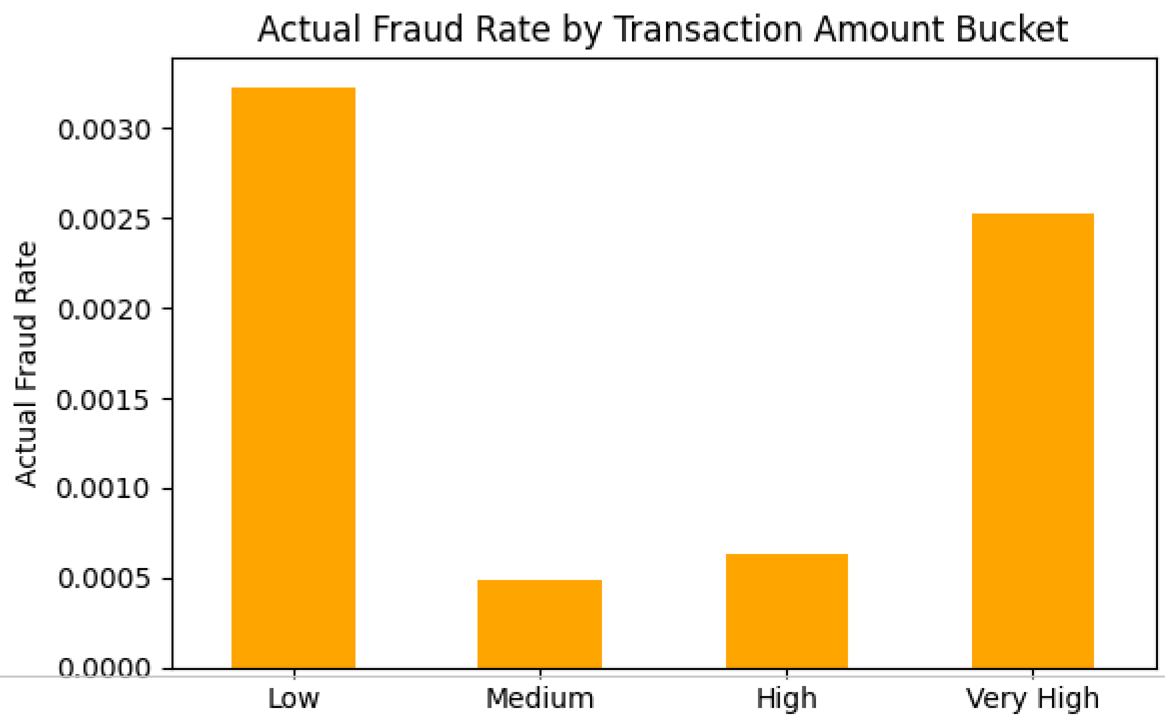


The false positive rate varies across transaction amount buckets, with the highest rate observed in the lowest and very high transaction ranges. This suggests that the model is more conservative when evaluating unusually small or unusually large transactions, potentially reflecting their higher perceived risk profiles. While this behavior may be operationally justified, it also highlights areas where customer friction could increase.

Chart 2: Fraud Rate by Amount Bucket

```
fraud_rate = df_test.groupby("Amount_Bucket")["Actual"].mean()

fraud_rate.plot(
    kind="bar",
    figsize=(6, 4),
    color="orange"
)
plt.ylabel("Actual Fraud Rate")
plt.xlabel("Transaction Amount Bucket")
plt.title("Actual Fraud Rate by Transaction Amount Bucket")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-2226202152.py:1: FutureWarning: The default of obse
  fraud_rate = df_test.groupby("Amount_Bucket")["Actual"].mean()
```



The actual fraud rate by transaction amount bucket shows that fraudulent activity is more prevalent in low and very high transaction ranges. This partially aligns with the observed false positive patterns, indicating that the model's behavior is influenced by underlying risk distribution rather than arbitrary thresholds.

Overall, this proxy-based bias analysis suggests that the model's error patterns broadly reflect the underlying fraud distribution across transaction amounts. However, the elevated false positive rates at the extremes indicate opportunities for refinement, such as amount-specific thresholds or differentiated review workflows. In a production setting, such adjustments could help balance fraud prevention effectiveness with customer experience.

End of Notebook