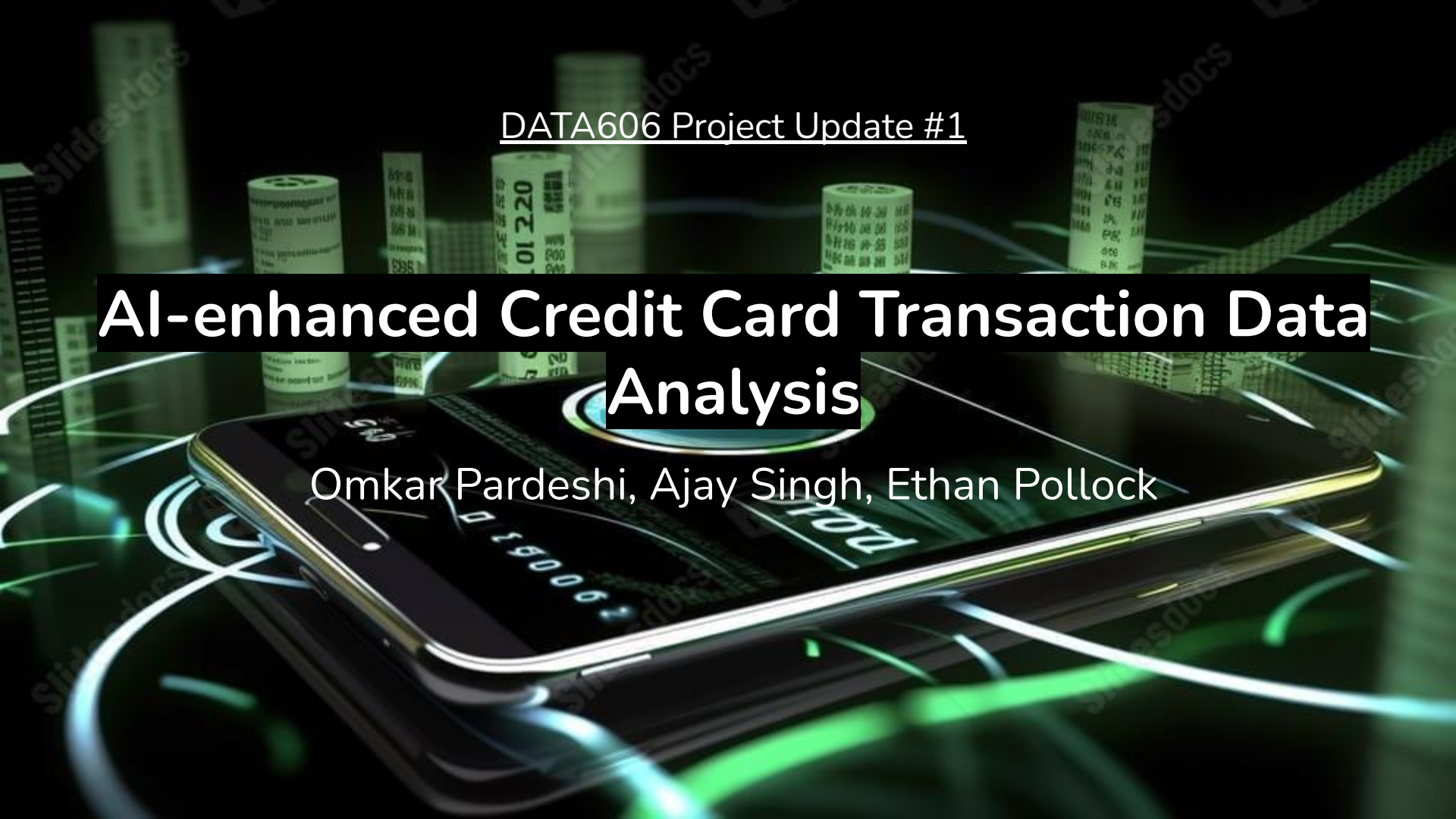
DATA606 Project Update #1

AI-enhanced Credit Card Transaction Data Analysis

Omkar Pardeshi, Ajay Singh, Ethan Pollock



Where We Left From

- Dataset including 1.3 million rows of credit card transaction data
- Dates range from December 31, 2018 to June 21, 2020
- Variables include:
  - Credit card number
  - Merchant/store
  - Transaction category
  - Transaction amount
  - First name, Last name
  - Gender of card holder
  - Address details of cardholder

- Models selected to test
  - Random Forest
  - Support Vector Machines
  - Deep Learning Neural Networks
  - Gradient Boosted Trees
    - Specifically XGBoost
  - Clustering and Unsupervised Learning
    - Specifically K-Mean and Hierarchical Clustering

Feature	Traditional Fraud Detection	AI & ML Fraud Detection
Core Approach	Rule-based, relies on human expertise	Automated, driven by machine learning models and pattern recognition
Advantages	✅ Human oversight ✅ Simple to implement	✅ Real-time analysis ✅ Adaptive learning ✅ Detects non-linear patterns
Disadvantages	❌ High false positives ❌ Limited scalability ❌ Only detects linear patterns	❌ Data dependent ❌ Complex to implement
Response Time	Slow, after manual review	Instant, real-time scoring with generative AI tools
Scalability	Limited, needs constant manual rule updates	Scalable and self-improving over time
Accuracy & Adaptability	Low – static rules can't evolve quickly	Highly adaptable to new fraud detection solutions
Use in Business	Traditional banking or small-scale platforms	Used in fintech, e-wallets, and modern digital payment services

Progress To Date

- ❏ Package Installation and Imports
- ❏ Data Cleaning
- ❏ Feature Engineering
- ❏ Clustering
- ❏ Predictive Modeling
- ❏ Explainability
- ❏ Interactive EDA Dashboard
- ❏ PDF Report Generation

Package Installation and Imports

- *pip* automatically installs any missing packages
  - scikit-learn
  - pandas
  - matplotlib
  - seaborn
  - shap
  - joblib
  - xgboost

```
# ## 0) Install required packages (if missing)
import sys, subprocess, pkg_resources
required = ['scikit-learn','pandas','matplotlib','seaborn','shap','joblib','xgboost']
installed = {pkg.key for pkg in pkg_resources.working_set}
to_install = [p for p in required if p not in installed]
if to_install:
    print("Installing:", to_install)
    subprocess.check_call([sys.executable, '-m', 'pip', 'install'] + to_install)
else:
    print("All required packages already installed.")
```

```
All required packages already installed.
/tmp/ipython-input-2972911722.py:2: DeprecationWarning: pkg_resources is deprecated as
  import sys, subprocess, pkg_resources
```

Package Installation and Imports

- Data Handling & Visualization
  - pandas
  - mumpy
  - matplotlib
- Modeling & Evaluation
  - RandomForestClassifier
  - K-Means
  - classification_report
- Preprocessing & Pipelines
  - StandardScalar
  - OneHotEncoder
  - Pipeline
  - ColumnTransformer
- Utility & Explainability
  - joblib
  - shap

```
# ## 1) Standard imports
import pandas as pd, numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix
import joblib
import shap
print('OK')
```

```
OK
```

Data Cleaning

- Displays:
  - Data types
  - Non-null counts
  - Memory usage
- Check for missing values across all columns
- Provides summary statistics for the *amt* column
- Show class distribution for the *is_fraud* target variable

```
# EDA
print(df.info())
print('\nMissing values per column:\n', df.isnull().sum())
if 'amt' in df.columns:
    display(df['amt'].describe())
if 'is_fraud' in df.columns:
    display(df['is_fraud'].value_counts(dropna=False))
```

```
 10  city                   1296675 non-null  object
 11  state                  1296675 non-null  object
 12  zip                    1296675 non-null  int64
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64
 16  job                    1296675 non-null  object
 17  dob                    1296675 non-null  object
 18  trans_num              1296675 non-null  object
 19  unix_time              1296675 non-null  int64
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64
 23  merch_zipcode          1100702 non-null  float64
dtypes: float64(6), int64(6), object(12)
memory usage: 237.4+ MB
None
```

```
Missing values per column:
 Unnamed: 0                 0
trans_date_trans_time       0
cc_num                      0
merchant                    0
category                    0
amt                         0
first                       0
last                        0
gender                      0
street                      0
city                        0
state                       0
zip                         0
lat                         0
long                        0
city_pop                    0
job                         0
dob                         0
trans_num                   0
unix_time                   0
merch_lat                   0
merch_long                  0
is_fraud                    0
merch_zipcode          195973
dtype: int64
count    1.296675e+06
mean     7.035104e+01
std      1.603160e+02
min      1.000000e+00
25%      9.650000e+00
50%      4.752000e+01
75%      8.314000e+01
max      2.894890e+04
Name: amt, dtype: float64
is_fraud
0    1289169
1       7506
Name: count, dtype: int64
```

Feature Engineering

- Execute key transformations
  - Datetime Parsing
    - Extract *hour*, *day*, and *month* from *trans_date_trans_time*
  - Monetary Feature
    - Applies log transformation from *amt* to *amt_log1p* to reduce skew
  - Age Calculation
    - Derives *age* in years from *dob*
- Adds temporal, scaled, and demographic features that can capture behavioral pattern
  - Important in fraud detection

```
# Feature engineering
df2 = df.copy()

# Parse datetime if present
if 'trans_date_trans_time' in df2.columns:
    df2['trans_date_trans_time'] = pd.to_datetime(df2['trans_date_trans_time'], errors='coerce')
    df2['trans_hour'] = df2['trans_date_trans_time'].dt.hour
    df2['trans_day'] = df2['trans_date_trans_time'].dt.day
    df2['trans_month'] = df2['trans_date_trans_time'].dt.month

# Amount features
if 'amt' in df2.columns:
    df2['amt_log1p'] = np.log1p(df2['amt'].astype(float))

# Age from dob if exists
if 'dob' in df2.columns:
    df2['dob'] = pd.to_datetime(df2['dob'], errors='coerce')
    df2['age'] = ((pd.Timestamp.now() - df2['dob']).dt.days / 365.25).fillna(0).astype(int)

display(df2.head())
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	unix_time	merch_lat	merch_long	is_fraud	merch_zipcode	trans_hour	trans_day	trans_month	amt_log1
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	...	1325376018	36.011293	-82.048315	0	28705.0	0	1	1	1.78674
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greens Suite 393	...	1325376044	49.159047	-118.186462	0	NaN	0	1	1	4.68425
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Suite	...	1325376051	43.150704	-112.154481	0	83236.0	0	1	1	5.39866

Clustering

- Group similar transaction and/or users using K-Means clustering
- Scales selected numeric features with StandardScalar
- Applies K-Means clustering
  - 2-4 clusters based on data size
- Assigns each record a cluster label
- Unsupervised segmentation can help uncover hidden patterns
  - E.g., high-risk transaction clusters

```
# Select numeric features for clustering (fallback if columns missing)
clust_features = [c for c in ['amt','amt_log1p','trans_hour','city_pop','age'] if c in df2.columns]
print('Clustering on:', clust_features)

if clust_features:
    Xc = df2[clust_features].fillna(0).astype(float)
    scaler = StandardScaler()
    Xs = scaler.fit_transform(Xc)
    kmeans = KMeans(n_clusters=min(4, max(2, len(df2)//5)), random_state=42)
    df2['cluster'] = kmeans.fit_predict(Xs)
    print('Cluster counts:')
    display(df2['cluster'].value_counts())
else:
    print('No clustering features found.')
```

```
Clustering on: ['amt', 'amt_log1p', 'trans_hour', 'city_pop', 'age']
Cluster counts:
cluster
0    431275
3    418567
1    404949
2     41884
Name: count, dtype: int64
```

Anomaly Detection

- Detect potentially fraudulent transactions using unsupervised learning
- Use IsolationForest that allows 1% contamination threshold
- Output
  - Adds *anomaly_score* as -1 for anomaly and 1 for normal
  - Adds binary anomaly flag for easy filtering
- Detects rare and suspicious behavior without requiring labeled fraud data

```
# Anomaly detection with IsolationForest
ad_features = [c for c in ['amt','amt_log1p','trans_hour','city_pop'] if c in df2.columns]
if ad_features:
    Xad = df2[ad_features].fillna(0).astype(float)
    iso = IsolationForest(n_estimators=100, contamination=0.01, random_state=42)
    df2['anomaly_score'] = iso.fit_predict(Xad)  # -1 anomaly, 1 normal
    df2['anomaly'] = df2['anomaly_score'] == -1
    display(df2[['amt'] + [c for c in ad_features if c!='amt'] + ['anomaly']].head(8))
else:
    print('No features available for anomaly detection.')
```

	amt	amt_log1p	trans_hour	city_pop	anomaly
0	4.97	1.786747	0	3495	False
1	107.23	4.684259	0	149	False
2	220.11	5.398660	0	4154	False
3	45.00	3.828641	0	1939	False
4	41.96	3.760269	0	99	False
5	94.63	4.560487	0	2158	False
6	44.54	3.818591	0	2691	False
7	71.65	4.285653	0	6018	False

Predictive Modeling

- Train a supervised learning classifier using labeled data (*is_fraud*)
- Pipeline components
  - Preprocessing: scales numerics and applies one-hot encoding to categoricals
    - Numerical: *amt*, *trans_hour*, *city_pop*, *age*
    - Categorical: *category*, *merchant*, *job*, *state*
  - Classifier: RandomForestClassifier with 100 trees

```
# Prepare data for supervised modeling
if 'is_fraud' in df2.columns:
    target = 'is_fraud'
    # Features selection: numeric + some categorical (limited)
    numeric_feats = [c for c in ['amt','amt_log1p','trans_hour','trans_day','trans_month','city_pop','age'] if c in df2.columns]
    cat_feats = [c for c in ['category','merchant','job','state'] if c in df2.columns]
    X = df2[numeric_feats + cat_feats].copy()
    y = df2[target].astype(int).fillna(0)

    # Simple preprocessing pipeline
    numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])
    from sklearn import __version__ as sklver
    ohe_kwargs = {'handle_unknown': 'ignore'}
    # use correct arg name depending on sklearn version
    if int(sklver.split('.')[1]) >= 4:
        ohe_kwargs['sparse_output'] = False
    else:
        ohe_kwargs['sparse'] = False

    cat_transformer = Pipeline(steps=[('ohe', OneHotEncoder(**ohe_kwargs))])

    preprocessor = ColumnTransformer(transformers=[
        ('num', numeric_transformer, numeric_feats),
        ('cat', cat_transformer, cat_feats)
    ], remainder='drop')

    clf = Pipeline(steps=[('pre', preprocessor),
                          ('clf', RandomForestClassifier(n_estimators=100, random_state=42))])
```

Predictive Modeling

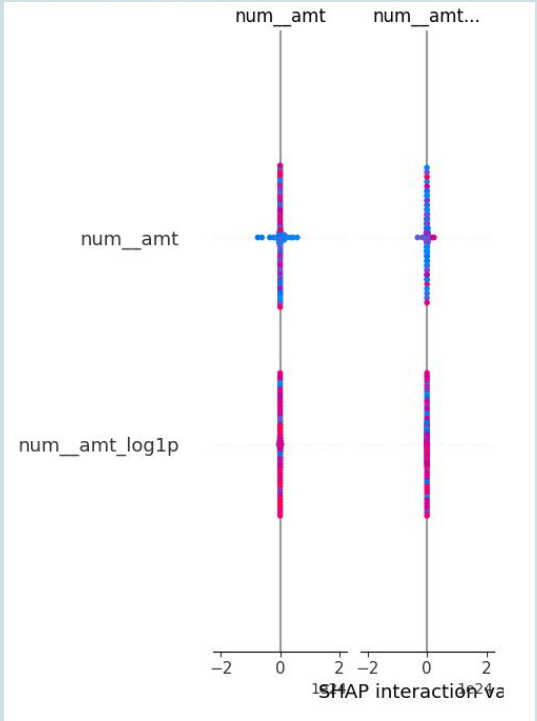
- Data is split into train/test sets (stratified)
- Trains and evaluates pipelines:
  - classification_report
  - ROC AUC
- Saves final model to *fraud_model.joblib*
- Modular and version-safe pipeline ensures reliable, repeatable training with performance reporting

```
    # train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.unique())>1 else None)
    clf.fit(X_train, y_train)
    ypred = clf.predict(X_test)
    print('Classification report:')
    print(classification_report(y_test, ypred, zero_division=0))
    if hasattr(clf[-1], 'predict_proba') and len(y_test)>0:
        yproba = clf.predict_proba(X_test)[:,1]
        try:
            print('ROC AUC:', roc_auc_score(y_test, yproba))
        except Exception as e:
            print('ROC AUC could not be computed:', e)
    # Save model
    joblib.dump(clf, 'fraud_model.joblib')
    print('Model saved to fraud_model.joblib')
else:
    print('Target column `is_fraud` not found — skipping supervised modeling.')
```

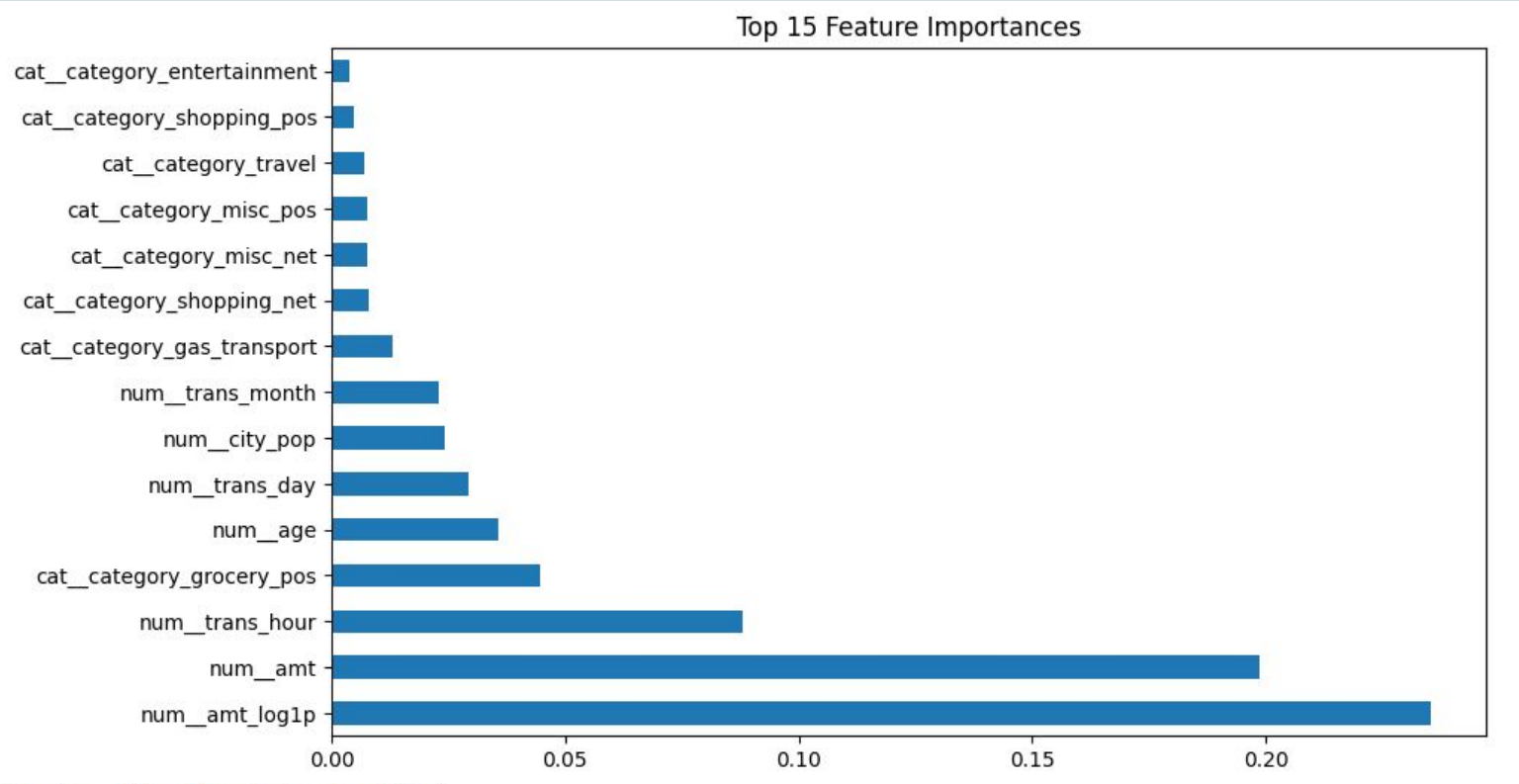
```
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257834
           1       0.97      0.71      0.82      1501

    accuracy                           1.00    259335
   macro avg       0.99      0.86      0.91    259335
weighted avg       1.00      1.00      1.00    259335

ROC AUC: 0.9911300629380465
Model saved to fraud_model.joblib
```

Explainability

- Automatically adjusts for SHAP API changes (legacy to modern)
- SHAP Summary Plot:
  - Uses top 100 transformed training samples for clarity
  - Visualizes feature impact and direction on model predictions
- Adds transparency to model decisions, build trust, and aids debugging or stakeholder communication



Explainability



Explainability

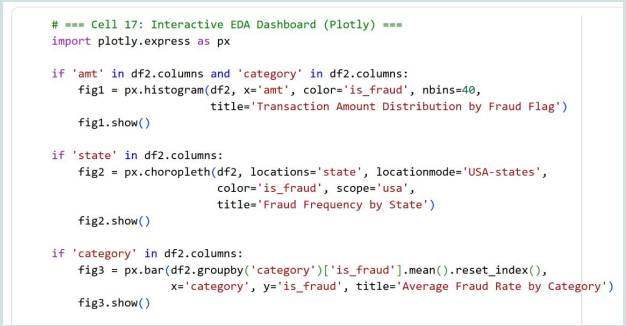
- Models tested:
  - RandomForestClassifier
  - GradientBoostingClassifier
  - LogisticRegression
- Reuses preprocessing pipeline for fair comparison
- Measures performance using ROC AUC on test data
- Outputs sorted table of model names against ROC AUC scores
- Helps identify the best-performing baseline model quickly without manual training

```python
# === Cell 16: AutoML-style quick model comparison ===
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score

models = {
    "RandomForest": RandomForestClassifier(n_estimators=100, random_state=42),
    "GradientBoosting": GradientBoostingClassifier(random_state=42),
    "LogisticRegression": LogisticRegression(max_iter=1000)
}

results = []
for name, m in models.items():
    pipe = Pipeline(steps=[('pre', preproc), ('clf', m)])
    pipe.fit(X_train, y_train)
    ypred = pipe.predict(X_test)
    yprob = pipe.predict_proba(X_test)[:,1] if hasattr(pipe, 'predict_proba') else ypred
    auc = roc_auc_score(y_test, yprob) if len(np.unique(y_test)) > 1 else np.nan
    results.append({'Model': name, 'ROC_AUC': auc})

pd.DataFrame(results).sort_values('ROC_AUC', ascending=False)
```

	Model	ROC_AUC
0	RandomForest	0.991130
1	GradientBoosting	0.971749
2	LogisticRegression	0.848463

Interactive EDA Dashboard

- Intuitive and interactive visualizations to explore fraud patterns in dataset
  - Histogram: distribution of transaction amounts, colored by fraud flag
  - Choropleth Map: fraud frequency by US State (geographic heatmap)
  - Bar Chart: average fraud rate by transaction category
- Plotly.express allows for dynamic and interactive plotting
- Enables stakeholders to explore trends, hotspots, and anomalies visually
  - Ideal for presentations and dashboards to stakeholders



```python
# === Cell 17: Interactive EDA Dashboard (Plotly) ===
import plotly.express as px

if 'amt' in df2.columns and 'category' in df2.columns:
    fig1 = px.histogram(df2, x='amt', color='is_fraud', nbins=40,
                        title='Transaction Amount Distribution by Fraud Flag')
    fig1.show()

if 'state' in df2.columns:
    fig2 = px.choropleth(df2, locations='state', locationmode='USA-states',
                         color='is_fraud', scope='usa',
                         title='Fraud Frequency by State')
    fig2.show()

if 'category' in df2.columns:
    fig3 = px.bar(df2.groupby('category')['is_fraud'].mean().reset_index(),
                  x='category', y='is_fraud', title='Average Fraud Rate by Category')
    fig3.show()
```

PDF Report Generation

- Creates a professionally formatted PDF summarizing project results
  - *reportlab* allowed for structured document creation
- Includes contents generated from these models
  - Title and timestamp
  - Dataset overview
  - Model performance
    - ROC AUC scores for all tested models
  - Key insights:
    - RandomForest was the top performer
    - SHAP explained key fraud indicators
    - Visual dashboards highlight fraud trends by amount, category, and location

```python
# === Cell 18: Generate Summary PDF Report ===
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
from reportlab.lib.styles import getSampleStyleSheet
import datetime

summary_path = "CreditCard_Fraud_Report.pdf"
doc = SimpleDocTemplate(summary_path, pagesize=letter)
styles = getSampleStyleSheet()
story = []

story.append(Paragraph("<b>Credit Card Fraud Detection Report</b>", styles['Title']))
story.append(Spacer(1, 12))
story.append(Paragraph(f"Generated on: {datetime.datetime.now()}", styles['Normal']))
story.append(Spacer(1, 12))

# Add dataset info
story.append(Paragraph(f"Rows: {df2.shape[0]}, Columns: {df2.shape[1]}", styles['Normal']))
story.append(Spacer(1, 12))

# Add model results
if 'results' in locals():
    story.append(Paragraph("<b>Model Comparison (ROC AUC)</b>", styles['Heading2']))
    for _, r in pd.DataFrame(results).iterrows():
        story.append(Paragraph(f"{r['Model']}: {round(r['ROC_AUC'], 3)}", styles['Normal']))
        story.append(Spacer(1, 6))

# Add key insights
story.append(Paragraph("<b>Key Insights:</b>", styles['Heading2']))
story.append(Paragraph("• RandomForest performed best among tested models.", styles['Normal']))
story.append(Paragraph("• SHAP analysis shows top drivers of fraud risk.", styles['Normal']))
story.append(Paragraph("• Interactive charts visualize fraud by amount, category, and state.", styles['Normal']))

doc.build(story)
print(f"✅ Report generated: {summary_path}")
```

AI Aspects to be Included

- LLM integration to explain the features causing fraud detections using prompts.
- Explain the insights produced by models using LLM
- Agents that simulate fraud data to decrease the False Positive Rates.

Next Steps

- Creation and utilization of API keys for FRED and OpenAI
  - Integrate economic elements into model analysis
- Additional generative AI aspects?
  - Expand on insights that can be produced from fraudulent detection model
- Being able to segment data into smaller sets
  - Potentially limited generation of certain arrays due to the demand on local computer RAM

References

- A systematic review of AI-enhanced techniques in credit card fraud detection | Journal of Big Data
- Generative AI in Card Fraud Detection: Benefits, Use Cases, and Industry Impact
- FraudX AI: An Interpretable Machine Learning Framework for Credit Card Fraud Detection on Imbalanced Datasets
- AI Analytics for Credit Card Security and Fraud Prevention