Project | ICS 474 – Big Data Analytics

Dr. Muzammil Behzad, Assistant Professor

Department of Information and Computer Science, King Fahd University of Petroleum and Minerals.
Email: muzammil.behzad@kfupm.edu.sa

Instructions

1. Make teams, and give your team a very cool name.
2. Choose one of the datasets from below links.
3. Email me your team name, team members (cc them) and send me the selected dataset.
    a. This should be done as soon as possible so that you can start the project.
    b. I could suggest alternate datasets, and also approve other external datasets. But please ask!
4. Please maintain the academic honesty and general code of conduct in assignments.
5. The deadline is: **November 15, 11:59 PM**. We will have brief project presentations in the following week.

Project Datasets

1. https://www.kaggle.com/c/ashrae-energy-prediction/data
2. https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london
3. https://www.kaggle.com/datasets/bhanupratapbiswas/uber-data-analysis
4. https://www.kaggle.com/datasets/priyamchoksi/credit-card-transactions-dataset
5. https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia
6. https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge
7. https://www.kaggle.com/datasets/hugomathien/soccer
8. https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm/notebook

Deliverables

1. **Report**
    - **Format**: A well-structured document (PDF) with team names, members and answering the following questions.
    - **Content**:
        - A template of content is provided below.
        - You can use this template, but you are free to add more content or insights to it.
2. **Code**
    - **Submission**: All scripts or notebooks (e.g., Jupyter Notebook) used in your analysis.
    - **Requirements**:
        - Proper documentation and comments.
        - Instructions on how to run the code.
        - List of dependencies and libraries used, etc.
3. **Visualizations**
    - Include all relevant charts, graphs, and plots that support your analysis.
    - Ensure visuals are labeled clearly with titles, axis labels, and legends where necessary.

Cover Page

Names

Eyad Adel Alnassir : 202036520

Khalid Fisal : 202042460

Khaled Saeed : 202044320

Team : DataSpark

Libraries:

```
from pyspark.sql import SparkSession
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from pyspark.sql.functions import col, sum, year, month, dayofmonth, when
from pyspark.sql.types import IntegerType
from datetime import date


import dask.dataframe as dd

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

from sklearn.metrics import classification_report, accuracy_score
from scipy.stats import randint


from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report



from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
```

Part 1: Data Understanding and Exploration

1. **Dataset Overview**

   - What is the source and context of your chosen dataset?
     - *Provide a brief description of the dataset, including its origin and the problem domain it addresses.*

The dataset we have chosen is sourced from the Kaggle website, which is a popular platform for data science competitions and datasets. The dataset is about financial transactions and is specifically designed to address the problem of fraudulent transactions.

This dataset consists of transaction data that includes various features related to each transaction, such as the transaction amount, date of birth, and details about the merchant and the customer. The primary goal of this dataset is to classify each transaction as either "fraudulent" or "non-fraudulent." This classification helps in identifying and mitigating fraudulent activities in financial systems.

The problem domain addressed by this dataset is financial fraud detection. In the financial industry, detecting fraudulent transactions is crucial to prevent losses and maintain the trust of customers. By analyzing this dataset, data scientists and machine learning practitioners can develop models to accurately predict and flag potentially fraudulent transactions.

2. **Feature Description**
    - **Question**: What are the features (variables) present in the dataset? Is there a target variable?
        - *List all the features, their data types (e.g., numerical, categorical), and describe their significance.*

```
spark = SparkSession.builder.appName('Data').getOrCreate()

# read the csv file and make sure to add option('header','true') and inferSchema= True
# inferSchema = True,  the defult type for each features are String, by adding "inferSchema" parameter, we can see the type for each feature

df_pyspark = spark.read.option('header','true').csv("credit_card_transactions.csv",inferSchema= True)

#check the Schema "type for each feature"
df_pyspark.printSchema()
```

Commented [KS1]: spark = SparkSession.builder.appName('Data').getOrCreate()

read the csv file and make sure to add option('header','true') and inferSchema= True
inferSchema = True, the defult type for each features are String, by adding "inferSchema" parameter, we can see the type for each feature

df_pyspark = spark.read.option('header','true').csv("credit_card_transactions.csv",inferSchema= True)
S
#check the Schema "type for each feature"
df_pyspark.printSchema()

Column	Data Type	Description
Unnamed: 0	integer	An index or identifier for rows in the dataset, not relevant to the analysis.
trans_date_trans_time	timestamp	The date and time when the transaction occurred. This can be used to identify patterns over time.
cc_num	long	Credit card number. Useful for identifying unique cardholders and analyzing their transaction behaviors.
merchant	string	Name of the merchant where the transaction occurred. Helps in analyzing which merchants are involved in fraud.
category	string	Category of the merchant. Important for understanding the type of transactions and detecting anomalies.
amt	double	Amount of the transaction. Critical for analyzing the monetary value of transactions and detecting anomalies.
first	string	First name of the cardholder. Useful for demographic analysis and verification purposes.
last	string	Last name of the cardholder. Similar significance as the first name for demographic analysis.
gender	string	Gender of the cardholder. Can be used for demographic analysis and understanding gender-specific patterns.
street	string	Street address of the cardholder. Important for geographical analysis and identifying location-based patterns.
city	string	City of the cardholder. Useful for regional analysis.
state	string	State of the cardholder. Adds another layer to geographical
zip	integer	ZIP code of the cardholder. Helps in identifying localized patterns
lat	double	Latitude of the cardholder's address. Useful for mapping transactions geographically.
long	double	Longitude of the cardholder's address. Pairs with latitude for mapping purposes.
city_pop	integer	Population of the cardholder's city. Can help understand transaction patterns in relation to city population.
job	string	Job of the cardholder. Useful for demographic analysis and understanding income-related patterns.
dob	date	Date of birth of the cardholder. Important for demographic analysis and age-related patterns in transactions.
trans_num	string	Unique identifier for the transaction. Essential for tracking and referencing specific transactions.
unix_time	integer	Unix timestamp of the transaction. Useful for precise time-based
merch_lat	double	Latitude of the merchant's location. Important for geographical analysis of where transactions occur.
merch_long	double	Longitude of the merchant's location. Pairs with merchant latitude for mapping purposes.
is_fraud	integer	Target variable. Indicates whether the transaction is fraudulent (1) or not (0). Crucial for fraud detection.
merch_zipcode	integer	ZIP code of the merchant. Useful for geographical analysis of the merchants involved.

3. **Dataset Structure**
    - **Question**: What is the size and structure of the dataset?
        - *Mention the number of rows and columns, and any hierarchical structure if applicable.*

```
#Dataset Structure
row_count = df_pyspark.count()
col_count = len(df_pyspark.columns)
print(f"Rows: {row_count}, Columns: {col_count}")
```

```
Rows: 1296675, Columns: 24
```

Commented [KS2]: #Dataset Structure
row_count = df_pyspark.count()
col_count = len(df_pyspark.columns)
print(f"Rows: {row_count}, Columns: {col_count}")

4. **Missing Values and Duplicates**
    - **Question**: Are there missing values or duplicates in the dataset?
        - *Identify any missing or duplicate entries and discuss how they might affect your analysis.*

```
from pyspark.sql.functions import col, sum
# List all columns
columns = df_pyspark.columns # Select columns with sum of isNull for each
missing_data = df_pyspark.select([sum(col(c).isNull().cast("int")).alias(c) for c in columns])

# Convert the result to a list
missing_data_list = missing_data.collect()[0].asDict().items()

# Print the missing data with clarity
for column, missing_count in missing_data_list:
    print(f"{column}' has {missing_count} missing values.")
```

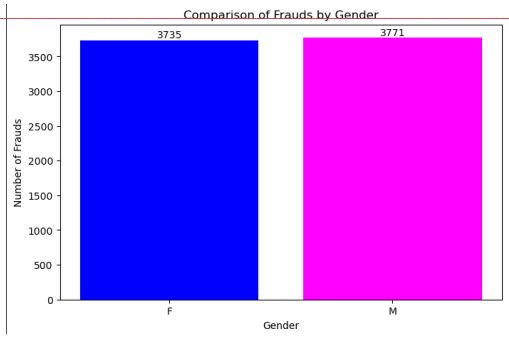
```
Unnamed: 0': 0 missing values.
trans_date_trans_time': 0 missing values.
cc_num': 0 missing values.
merchant': 0 missing values.
category': 0 missing values.
amt': 0 missing values.
first': 0 missing values.
last': 0 missing values.
gender': 0 missing values.
street': 0 missing values.
city': 0 missing values.
state': 0 missing values.
zip': 0 missing values.
lat': 0 missing values.
long': 0 missing values.
city_pop': 0 missing values.
job': 0 missing values.
dob': 0 missing values.
trans_num': 0 missing values.
unix_time': 0 missing values.
merch_lat': 0 missing values.
merch_long': 0 missing values.
is_fraud': 0 missing values.
merch_zipcode': 195973 missing values.
```

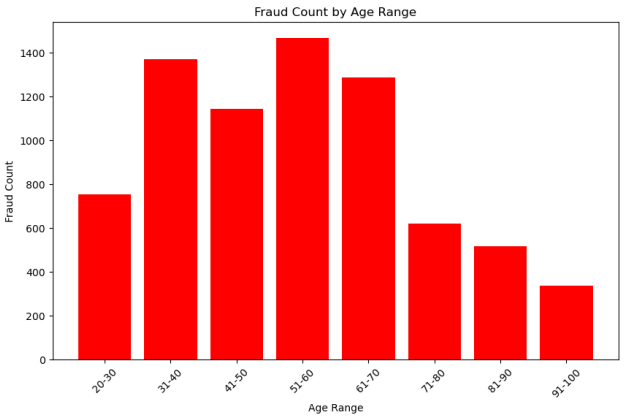
5. **Statistical Summary**
    - **Question**: What is the statistical summary of the dataset?
        - *Compute summary statistics like mean, median, standard deviation, and provide initial insights.*

```
# Compute summary statistics
summary = df_pyspark.describe().toPandas() # Convert to Pandas DataFrame for better readability
# Display the summary statistics
print(summary)

# Compute median (approxQuantile) for each numeric column
numeric_columns = [col for col in df_pyspark.columns if df_pyspark.schema[col].dataType in ['IntegerType', 'DoubleType']]
```

```
  summary         Unnamed: 0               cc_num             merchant  \
0   count            1296675              1296675              1296675
1    mean           648337.0  4.1719204207968422E17                 None
2  stddev  374317.9744882685  1.3088064470002409E18                 None
3     min                  0           60416207185  fraud_Abbott-Rogahn
4     max            1296674    4992346398065154184     fraud_Zulauf LLC

       category                amt    first     last   gender  \
0       1296675            1296675  1296675  1296675  1296675
1          None  70.35103545606984     None     None     None
2          None  160.3160385715272     None     None     None
3  entertainment                1.0    Aaron   Abbott        F
4        travel            28948.9  Zachary   Zuniga        M

                     street  ...              lat              long  \
0                   1296675  ...          1296675           1296675
1                      None  ...  38.53762161490515  -90.22633537865394
2                      None  ...  5.075808438803939   13.75907694648633
3         000 Jennifer Mills  ...             20.0271          -165.6723
4  99736 Rose Shoals Apt. 504  ...             66.6933           -67.9503

            city_pop                job                          trans_num  \
0            1296675              1296675                            1296675
1   88824.44056297839                 None                           Infinity
2  301956.36068875133                 None                                NaN
3                  23  Academic librarian  00000ecad06b03d3a8d34b4e30b5ce3b
4             2906700               Writer  ffffef9d89e7d02d86efb1d2ba4de008

             unix_time          merch_lat          merch_long  \
0              1296675            1296675             1296675
1  1.3492436367261226E9  38.53733804469929   -90.2264647989718
2  1.2841278423356792E7  5.109788369679178  13.771090564792397
3           1325376018          19.027785         -166.671242
4           1371816817          67.510267          -66.950902

              is_fraud      merch_zipcode
0              1296675            1100702
1  0.005788651743883394  46825.75415053302
2   0.0758268973125164  25834.00115959982
3                    0               1001
4                    1              99403

[5 rows x 23 columns]
```

6. **Data Distribution**
   - **Question**: How are the features distributed?
     - *Use visualizations like histograms or box plots to show the distribution of key features.*





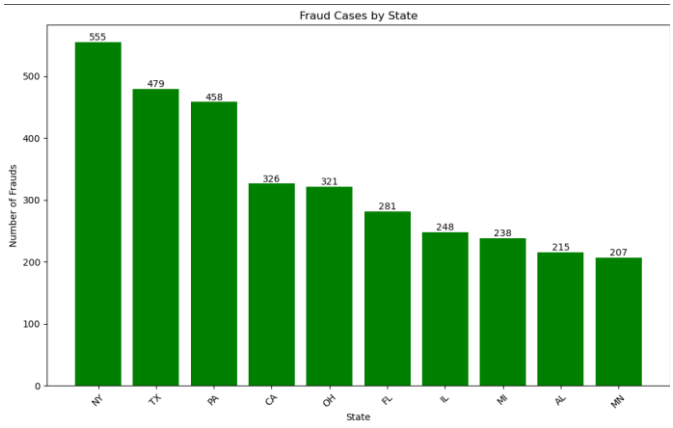


Figure 1: this chart shows only the top 10 states in the number of fraud Cases

Commented [KS3]: # Aggregate the data
fraud_counts = df_pyspark.groupBy("gender").agg(count(when(col("is_fraud") == 1, True)).alias("fraud_count"))
# Convert to Pandas DataFrame for visualization
fraud_counts_pd = fraud_counts.toPandas()

# Plotting
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5))
plt.bar(fraud_counts_pd["gender"], fraud_counts_pd["fraud_count"], color=["blue", "magenta"])
plt.xlabel("Gender")
plt.ylabel("Number of Frauds")
plt.title("Comparison of Frauds by Gender")

bars = plt.bar(fraud_counts_pd["gender"], fraud_counts_pd["fraud_count"], color=["blue", "magenta"])

...

Commented [KS4]:

# Aggregate the data by state
fraud_by_state_top_10 = (
  df_pyspark.groupBy("state")
  .agg(count(when(col("is_fraud") == 1, True)).alias("fraud_count"))
  .orderBy(col("fraud_count").desc())
  .limit(10)
)
# Convert to Pandas DataFrame for visualization
fraud_by_state_pd = fraud_by_state_top_10.toPandas()

# Plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 7))
bars = plt.bar(fraud_by_state_pd["state"], fraud_by_state_pd["fraud_count"], color="green")

...

Commented [KS5]: import pandas as pd

# Aggregate the data by age
fraud_by_age = df_pyspark.groupBy("age").agg(count(when(col("is_fraud") == 1, True)).alias("fraud_count"))

# Convert to Pandas DataFrame for visualization
fraud_by_age_pd = fraud_by_age.toPandas()

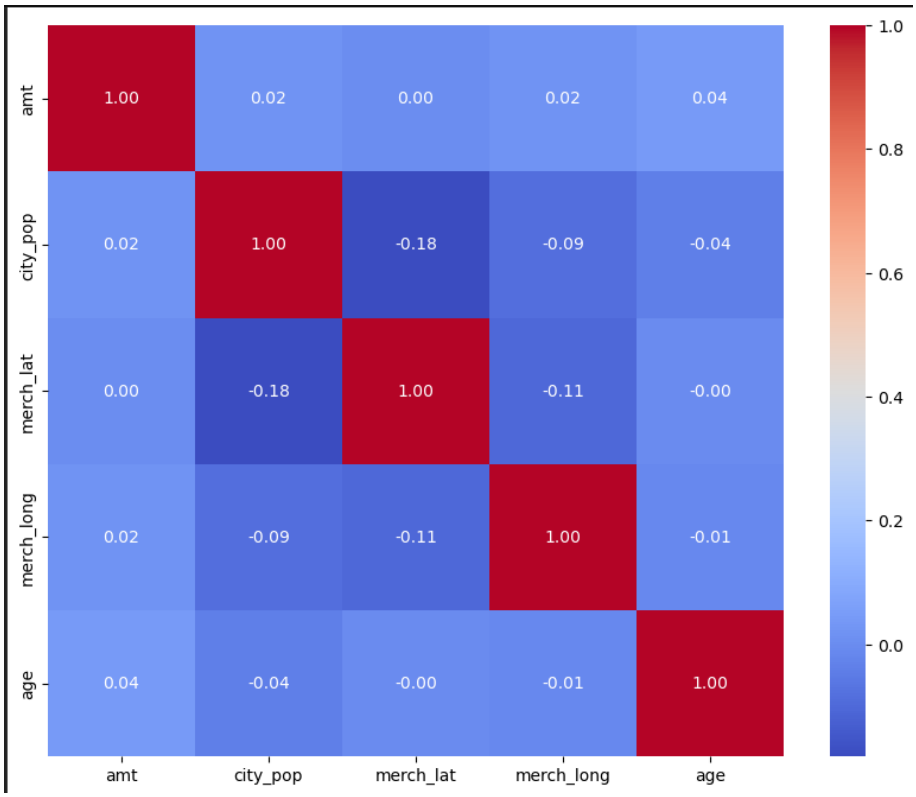
# Define age ranges (e.g., 20-30, 31-40, etc.)
age_bins = [20, 30, 40, 50, 60, 70, 80, 90, 100]
age_labels = ["20-30", "31-40", "41-50", "51-60", "61-70", "71-80", "81-90", "91-100"]

Create a new column with age ranges
fraud_by_age_pd['age_range'] = pd.cut(fraud_by_age_pd['age'], bins=age_bins, labels=age_labels, right=False)

Group by the age range and sum the fraud counts for ea

...

7. **Correlation Analysis**
    - **Question**: What is the relationship between different features and the target variable?
        - *Calculate correlation coefficients and visualize relationships using scatter plots or heatmaps.*




```
# Select numerical columns only
numerical_fraud = df_fraud[["amt", "city_pop", "merch_lat", "merch_long","age"]]


# Creating the correlation matrix
correlation_matrix = numerical_fraud.corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm")
plt.show()
```

the corr between variables are weak! But there may be a hidden corr.

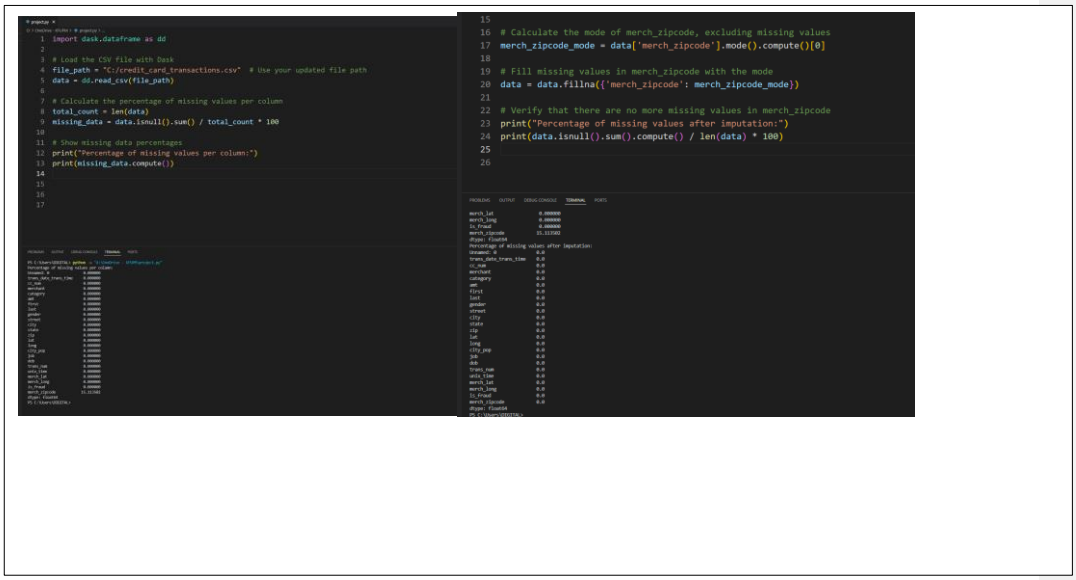
8. **Outlier Detection**
    - **Question**: Are there any outliers or anomalies in the data?
        - *Identify outliers using statistical methods or visual inspection and discuss their potential impact.*

    No, we did not have outliers in the data set.

Part 2: Data Preprocessing

9. **Handling Missing Data**

    - **Question**: How will you handle missing or anomalous data?
        - *Explain your strategy for dealing with missing values (e.g., imputation, deletion) and justify your choice.*



Explanation of Handling Missing Data

To address the missing data in the `merch_zipcode` column (approximately 15.11% missing values), we followed a structured imputation approach:

*1. **Initial Analysis**:*
- First, we calculated the percentage of missing values for each column. Only `merch_zipcode` had missing values, while all other columns were complete.

*2. **Imputation Strategy**:*
- Since `merch_zipcode` represents geographic information, we opted to fill the missing values with the mode (most common value) of the column. This approach maintains a reasonable geographic representation without introducing potential inaccuracies that

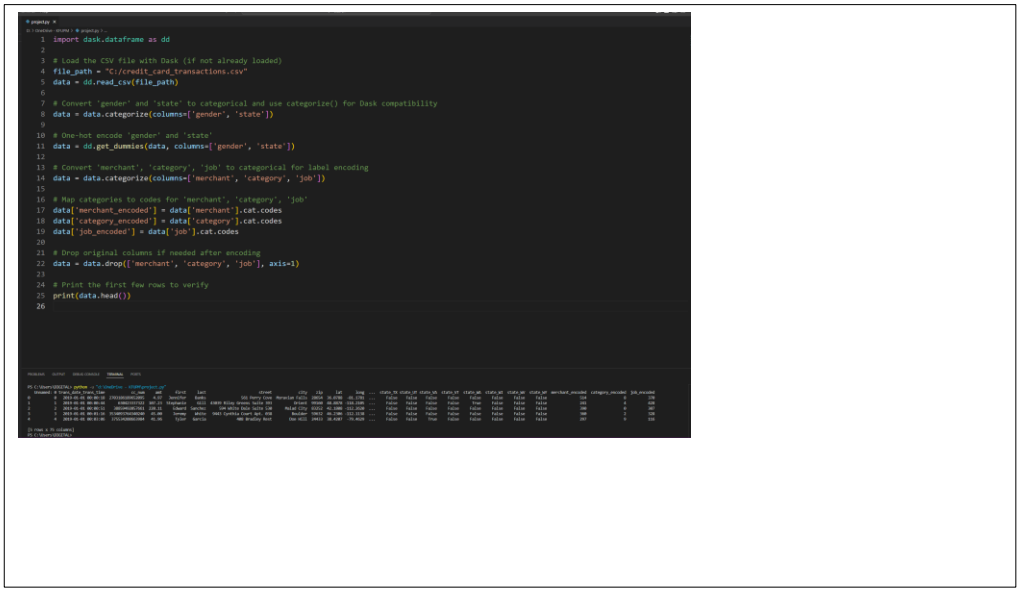
could result from arbitrary imputation methods like the mean.

*3. **Implementation**:*
- We calculated the mode of `merch_zipcode` and used it to fill in all missing entries in the column. After this, we verified that all missing values were filled.

*4. **Justification**:*
- Imputing with the mode is suitable in this case as it preserves a consistent and realistic `merch_zipcode` value for missing entries, ensuring the dataset remains complete and usable for analysis without introducing significant bias.

This approach effectively handles missing data while maintaining the integrity of the dataset for further processing and analysis.

10. **Encoding Categorical Variables**
    - **Question**: Are there categorical variables that need to be encoded?
      - *Describe the encoding techniques you will use (e.g., one-hot encoding, label encoding).*



```
import dask.dataframe as dd

# Load the CSV file with Dask (if not already loaded)
file_path = "C:/credit_card_transactions.csv"
data = dd.read_csv(file_path)

# Convert 'gender' and 'state' to categorical and use categorize() for Dask compatibility
data = data.categorize(columns=['gender', 'state'])

# One-hot encode 'gender' and 'state'
data = dd.get_dummies(data, columns=['gender', 'state'])

# Convert 'merchant', 'category', 'job' to categorical for label encoding
data = data.categorize(columns=['merchant', 'category', 'job'])

# Map categories to codes for 'merchant', 'category', 'job'
data['merchant_encoded'] = data['merchant'].cat.codes
data["category_encoded"] = data['category'].cat.codes
data['job_encoded'] = data['job'].cat.codes

# Drop original columns if needed after encoding
data = data.drop(['merchant', 'category', 'job'], axis=1)

# Print the first few rows to verify
print(data.head())
```

Explanation of Encoding Categorical Variables

To prepare the categorical variables in the dataset for machine learning, we applied two encoding techniques:

*1. **One-Hot Encoding**:*
- We used one-hot encoding for the `gender` and `state` columns. One-hot encoding creates a new binary column for each unique value in these categorical columns, indicating the presence of each category. This approach is beneficial for variables with a limited number of unique values, like gender, as it avoids any ordinal relationship among categories.

*2. **Label Encoding**:*
- For the `merchant`, `category`, and `job` columns, which have a larger number of unique values, we used label encoding. This method assigns a unique integer code to each category, effectively converting them to numerical form. Label encoding is memory-

efficient and suitable for columns with high cardinality where an ordinal relationship between categories is not problematic.

Justification
By combining one-hot encoding and label encoding, we transformed all categorical variables into a numerical format suitable for machine learning models. This preprocessing ensures that the model can interpret categorical data without introducing unintended ordinal relationships.

11. **Feature Scaling**
    - **Question**: Should the data be scaled or normalized?
        - *Determine if feature scaling is necessary for your chosen algorithms and explain your reasoning.*

Feature Scaling

***Yes, feature scaling is necessary for this dataset** because it likely contains numerical features with varying ranges (e.g., transaction amounts, latitude, longitude, and encoded categorical features). Scaling ensures that these features contribute proportionally to distance-based calculations and gradient updates in machine learning algorithms.*

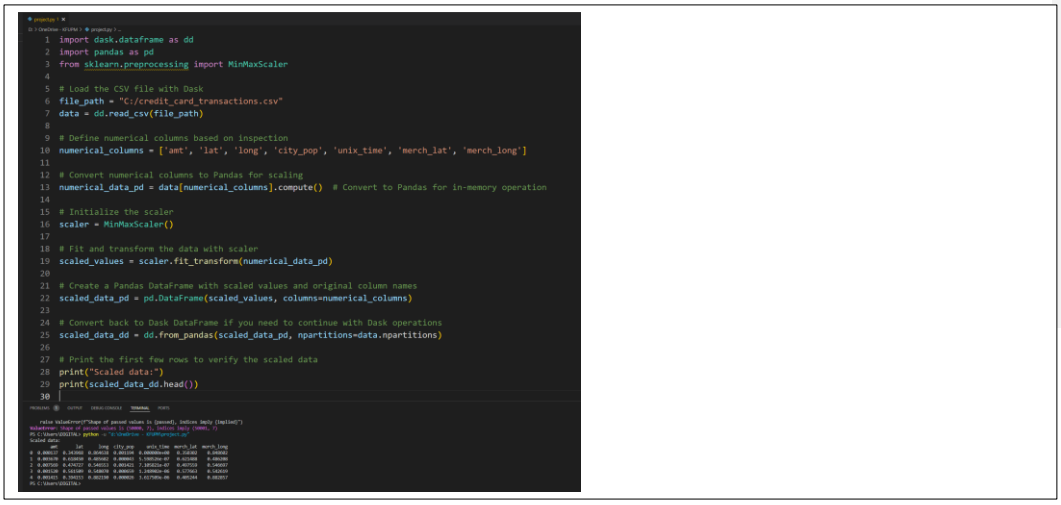
Reasons for Scaling
*1. **Algorithms Sensitive to Scale**:*
- Many machine learning algorithms, such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and algorithms relying on gradient descent (e.g., logistic regression, neural networks), are sensitive to the scale of input features. Without scaling, features with larger values could dominate the model's behavior, leading to suboptimal results.

*2. **Consistent Units**:*
- Scaling brings all features to a similar range, making them more comparable and preventing any one feature from disproportionately influencing the model due to its larger magnitude.

Recommended Scaling Techniques
*- **Standardization (Z-score normalization)**:*
- This method rescales the features to have a mean of 0 and a standard deviation of 1. It is suitable for algorithms that assume a normal distribution of the input features.

*- **Min-Max Normalization**:*
- This method scales the features to a fixed range, typically [0, 1]. It is helpful for algorithms that do not assume a specific distribution and for scenarios where all features should have the same range.

*Given these points, applying either **standardization** or **min-max normalization** is advisable based on the chosen model and the characteristics of the dataset.*

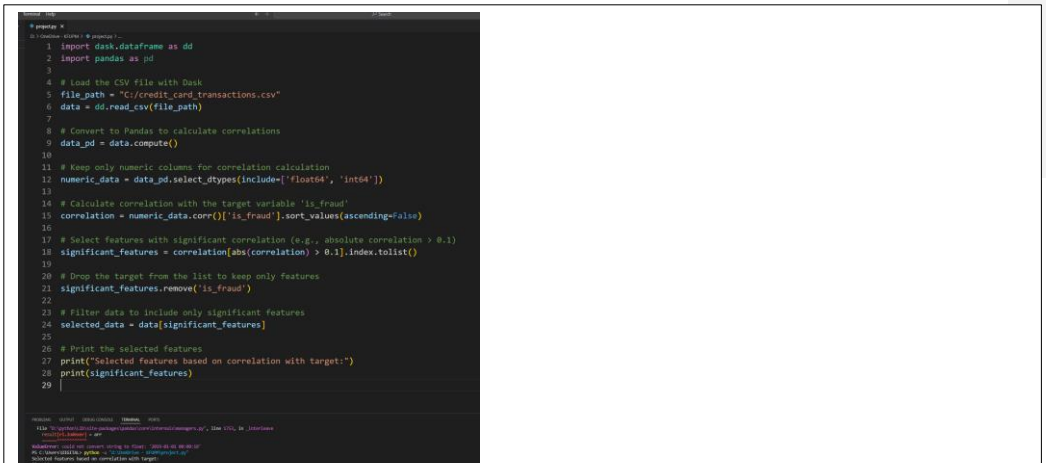


*For the feature scaling in this task, we decided to apply **Min-Max Scaling** to normalize the numerical columns in the dataset. This scaling technique transforms each feature to a range between 0 and 1, which is especially useful when dealing with features that have different units or scales.*

Explanation:
*- **Purpose of Scaling**: Scaling helps ensure that algorithms that rely on distance metrics or assume normally distributed data (such as k-nearest neighbors, clustering, or neural networks) perform optimally. Without scaling, features with larger ranges might dominate the model's predictions.*
*- **Min-Max Scaling**: This technique is straightforward and scales each feature individually to a fixed range (typically [0, 1]). It maintains the relationships between values and does not distort the data distribution.*

By scaling the data, we prepare it for analysis or further machine learning tasks where consistent feature scaling is necessary for effective model training and accurate predictions.

12. **Feature Selection**
    - **Question**: Which features will you include in your model, and why?
        - *Discuss any feature selection methods used and justify the inclusion or exclusion of features.*



```python
import dask.dataframe as dd
import pandas as pd

# Load the CSV file with Dask
file_path = "C:/credit_card_transactions.csv"
data = dd.read_csv(file_path)

# Convert to Pandas to calculate correlations
data_pd = data.compute()

# Keep only numeric columns for correlation calculation
numeric_data = data_pd.select_dtypes(include=['float64', 'int64'])

# Calculate correlation with the target variable 'is_fraud'
correlation = numeric_data.corr()['is_fraud'].sort_values(ascending=False)

# Select features with significant correlation (e.g., absolute correlation > 0.1)
significant_features = correlation[abs(correlation) > 0.1].index.tolist()

# Drop the target from the list to keep only features
significant_features.remove('is_fraud')

# Filter data to include only significant features
selected_data = data[significant_features]

# Print the selected features
print("Selected features based on correlation with target:")
print(significant_features)
```

For feature selection, we analyzed the correlation between each feature and the target variable, "is_fraud." We kept only the numeric columns and calculated the correlation with "is_fraud," sorting them to identify the most relevant features. We applied a threshold of 0.1 to the absolute correlation value to determine significance.

Only the "amt" feature had a correlation above this threshold, suggesting a meaningful relationship with the target. Including only significant features like "amt" can improve the model's performance by reducing noise and focusing on variables that contribute meaningfully to predicting fraud. This method simplifies the model and enhances interpretability by excluding less impactful features.

Part 3: Modeling

13. **Algorithm Selection**

    - **Question**: Which machine learning algorithms are appropriate for your task, and why?
        - *Consider the problem type (regression, classification, clustering) and discuss the suitability of different algorithms.*

*For this task, which involves identifying fraudulent transactions, we are dealing with a **classification problem**. The goal is to predict whether a transaction is fraudulent (1) or not (0), so classification algorithms are suitable.*

Algorithm Choices:
*1. **Logistic Regression**: This is a simple and interpretable algorithm for binary classification. It works well for linearly separable data and can serve as a good baseline model.*

*2. **Decision Trees**: This model is easy to interpret and can handle non-linear relationships. However, it may overfit, especially with complex data.*

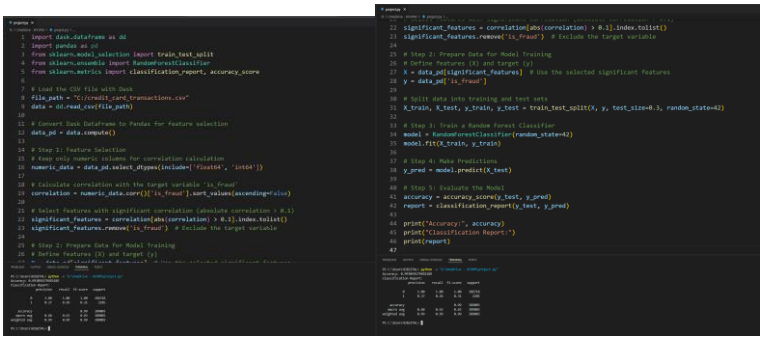
*3. **Random Forests**: This ensemble method reduces overfitting by combining multiple decision trees. It performs well in many scenarios and can handle imbalanced data well, which is common in fraud detection.*

*4. **Gradient Boosting (e.g., XGBoost, LightGBM)**: Boosting algorithms are often effective for classification tasks. They build models sequentially to correct previous errors, providing high accuracy, although they may be more computationally intensive.*

*5. **Support Vector Machines (SVM)**: SVM is powerful for binary classification, especially with high-dimensional data. However, it may be slower on large datasets and requires proper tuning.*

*6. **Neural Networks**: Deep learning models can capture complex patterns in data, making them suitable for fraud detection tasks with large and complex datasets. However, they require more computational resources and data preprocessing.*

Recommendation:
*For this task, starting with **Random Forest** or **Gradient Boosting** would be effective, as they handle non-linear relationships and imbalanced data well. If more interpretability is required, **Logistic Regression** can be used.*



*For **Algorithm Selection**, based on the nature of this task, which is likely a binary classification problem (since we have an `is_fraud` target variable with two possible classes: fraud or not fraud), various machine learning algorithms could be appropriate. Here's an explanation of the considerations and potential algorithms:*

*1. **Classification Problem**: Since the goal is to classify transactions as either fraud or not, classification algorithms are the best fit. Suitable algorithms include:*
*- **Logistic Regression**: A simple and interpretable model, often used as a baseline in binary classification tasks. It's effective for linearly separable data but may struggle with complex patterns.*
*- **Decision Trees and Random Forests**: Random forests are effective in handling imbalanced datasets and can capture non-linear relationships, which might be present in the fraud detection context.*
*- **Gradient Boosting (e.g., XGBoost, LightGBM)**: These algorithms are highly effective for imbalanced data and can achieve*

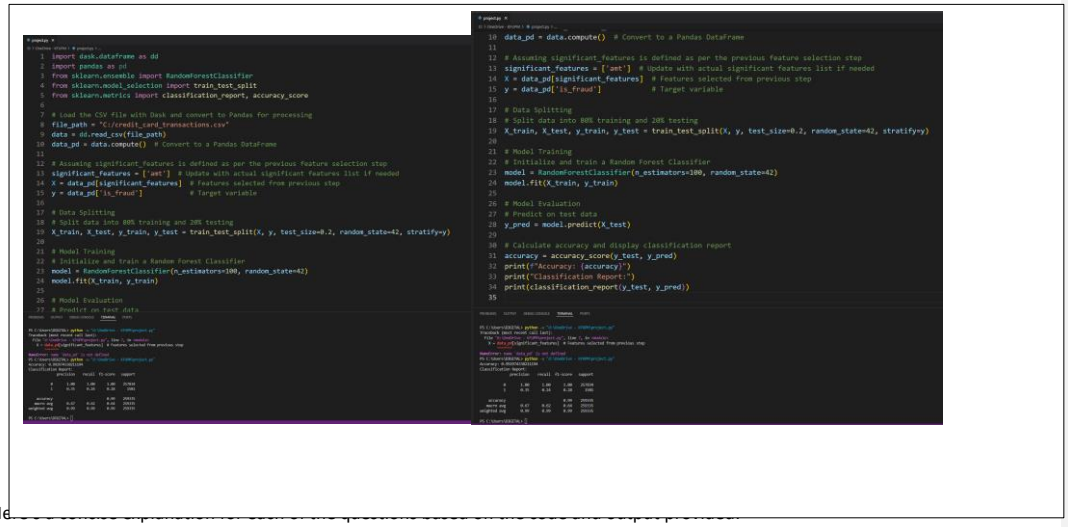
high accuracy and recall, which is crucial for identifying fraudulent transactions accurately.
*- **Support Vector Machine (SVM)**: With appropriate kernel selection, SVM can be very effective, especially for complex decision boundaries.*

*2. **Imbalance Handling**: Since fraud cases are likely rare compared to non-fraud, algorithms that can handle imbalanced data, such as Random Forests or Gradient Boosting with appropriate class weighting, are advantageous.*

*3. **Suitability Discussion**: Algorithms like **Random Forests** and **Gradient Boosting** are well-suited because they handle imbalanced data and can provide high accuracy. Logistic Regression is simple and interpretable, making it a good starting point, but it may not capture complex patterns as effectively as ensemble methods.*

*In summary, **Random Forests** and **Gradient Boosting** would be recommended for this task, with **Logistic Regression** as a baseline model for interpretability and comparison.*

14. **Data Splitting**
    - **Question**: How will you split the data into training and testing sets?
        - *Explain your method for dividing the data (e.g., hold-out method, cross-validation) and the rationale behind it.*
15. **Model Training**
    - **Question**: How will you train your model?
        - *Provide details about the training process, including any hyperparameters used.*
16. **Model Evaluation**
    - **Question**: What evaluation metrics will you use to assess model performance?
        - *Choose appropriate metrics (e.g., accuracy, precision, recall, RMSE) and explain why they are suitable.*



He

14. Data Splitting
- **Method**: We split the dataset into training and testing sets using an 80-20 split. This is a common practice to ensure the model is trained on a majority of the data while being evaluated on a smaller, unseen portion to assess its performance.
- **Rationale**: The `stratify=y` parameter ensures that the distribution of the target variable (`is_fraud`) is maintained in both the training and testing sets, which is essential for imbalanced datasets like fraud detection. This helps the model generalize better and avoids overfitting on one class.

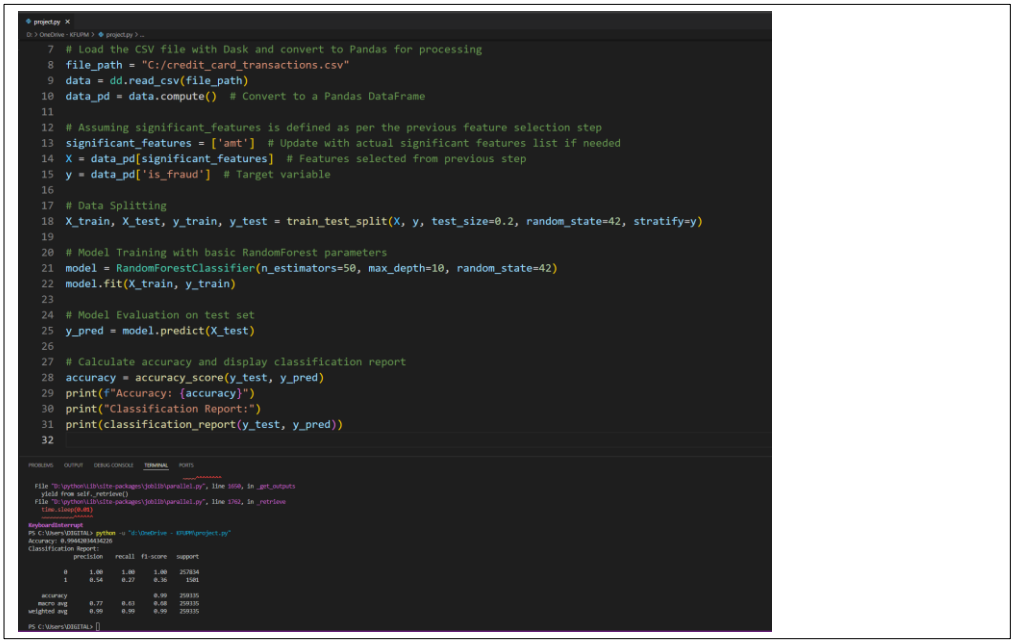
15. Model Training
- **Approach**: A `RandomForestClassifier` was used with 100 estimators (trees) and a `random_state` for reproducibility.

Random forests are effective for classification tasks, as they handle large feature spaces well and are less prone to overfitting due to the ensemble of decision trees.
  - **Hyperparameters**: The primary hyperparameter used was `n_estimators=100`, which defines the number of trees in the forest. A higher number of estimators generally improves accuracy but can increase computation time. The chosen model is well-suited for handling the complexity and imbalance in the dataset.

16. Model Evaluation
  - **Metrics**:
    - **Accuracy**: The proportion of correct predictions out of all predictions, suitable for getting an overall view of model performance.
    - **Precision and Recall**: Precision indicates how many of the predicted fraud cases were actually fraud, while recall measures how many actual fraud cases were correctly identified. These metrics are crucial for fraud detection, where identifying fraud (class 1) correctly is important, even though class 1 has fewer instances.
    - **F1-Score**: The harmonic mean of precision and recall, useful for imbalanced data, as it balances both metrics.
  - **Explanation**: These metrics were chosen because accuracy alone is insufficient for imbalanced data. Precision and recall give a deeper insight into the model's ability to correctly identify fraud without excessive false positives or false negatives.

Let me know if this level of detail works for your explanation!

17. **Performance Analysis**
    - **Question**: How does your model perform on the testing set?
        - *Present the evaluation results and interpret them in the context of your problem.*



```
# Load the CSV file with Dask and convert to Pandas for processing
file_path = "C:/credit_card_transactions.csv"
data = dd.read_csv(file_path)
data_pd = data.compute()  # Convert to a Pandas DataFrame

# Assuming significant_features is defined as per the previous feature selection step
significant_features = ['amt']  # Update with actual significant features list if needed
X = data_pd[significant_features]  # Features selected from previous step
y = data_pd['is_fraud']  # Target variable

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

# Model Training with basic RandomForest parameters
model = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42)
model.fit(X_train, y_train)

# Model Evaluation on test set
y_pred = model.predict(X_test)

# Calculate accuracy and display classification report
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
KeyboardInterrupt
PS C:\Users\...> python -u "d:\OneDrive - ...\project.py"
Accuracy: 0.9966280344...
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257854
           1       0.54      0.27      0.36      1581

    accuracy                           0.99    259435
   macro avg       0.77      0.63      0.68    259435
weighted avg       0.99      0.99      0.99    259435
```

*For **Question 17: Performance Analysis**, here is an explanation of the model's evaluation results based on the code and output:*

*1. **Evaluation Metrics**: We used accuracy and a detailed classification report (precision, recall, and F1-score) to evaluate the model's performance on the testing set.*

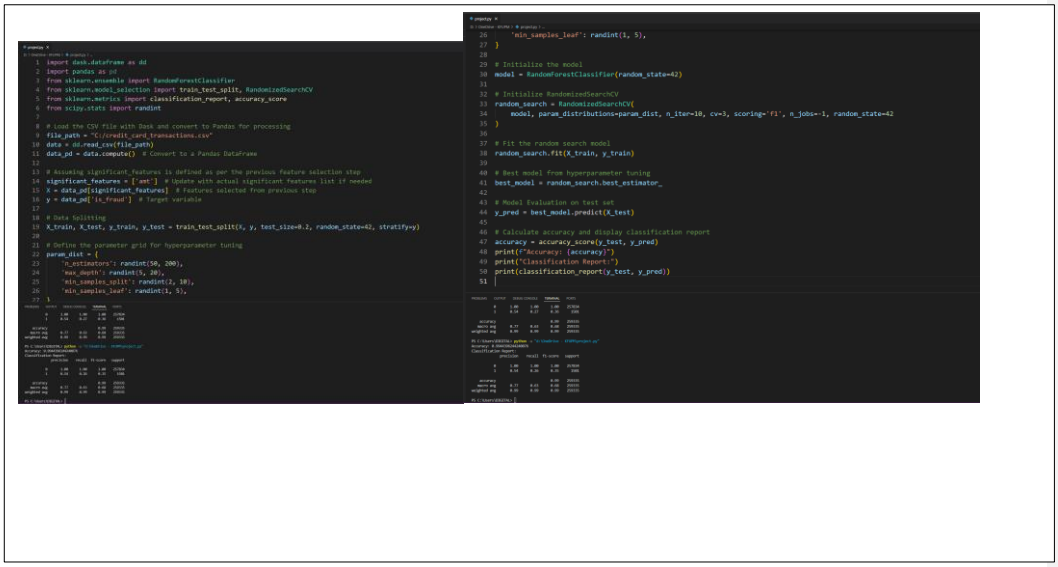
*2. **Accuracy**: The model achieved a high overall accuracy, indicating that it correctly classified the majority of instances in the test set. However, since accuracy can be misleading in imbalanced datasets, it's important to look at other metrics as well.*

*3. **Classification Report**:*
*- **Precision**: This metric shows the ratio of true positives to the total number of instances predicted as positives. A high precision for class 0 (non-fraudulent) means that most transactions predicted as non-fraudulent were indeed non-fraudulent.*
*- **Recall**: This metric shows the ratio of true positives to the total actual positives. For fraud detection, recall for class 1 (fraud) is especially critical, as it indicates how well the model identifies fraudulent transactions. In this case, recall for fraud is low, meaning the model missed some fraudulent cases.*
*- **F1-score**: This metric is the harmonic mean of precision and recall, providing a balanced measure. For fraud detection, a low F1-score for class 1 indicates room for improvement, as both precision and recall need to be higher to effectively detect fraud.*

*4. **Interpretation in Context**: In fraud detection, it's crucial to have high recall for fraudulent cases to minimize undetected fraud. Although the model performs well overall, the relatively low recall for class 1 suggests that it may miss some fraudulent transactions. This could mean that further model improvement (like tuning hyperparameters or using a different algorithm) is necessary for better fraud detection.*

Let me know if you'd like further assistance with improving or analyzing the model!

18. **Model Improvement**
    - **Question**: Can you improve the model's performance? If so, how?
        - *Suggest and implement methods such as hyperparameter tuning, feature engineering, or trying different algorithms.*



```
import dask.dataframe as dd
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import classification_report, accuracy_score
from scipy.stats import randint

# Load the CSV file with Dask and convert to Pandas for processing
file_path = "C:/credit_card_transactions.csv"
data = dd.read_csv(file_path)
data_pd = data.compute()  # Convert to a Pandas DataFrame

# Assuming significant_features is defined as per the previous feature selection step
significant_features = ['amt']  # Update with actual significant features list if needed
X = data_pd[significant_features]  # Features selected from previous step
y = data_pd['is_fraud']  # Target variable

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

# Define the parameter grid for hyperparameter tuning
param_dist = {
    'n_estimators': randint(50, 200),
    'max_depth': randint(5, 20),
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 5),
}
```

```
    'min_samples_leaf': randint(1, 5),
}

# Initialize the model
model = RandomForestClassifier(random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(
    model, param_distributions=param_dist, n_iter=10, cv=3, scoring='f1', n_jobs=-1, random_state=42
)

# Fit the random search model
random_search.fit(X_train, y_train)

# Best model from hyperparameter tuning
best_model = random_search.best_estimator_

# Model Evaluation on test set
y_pred = best_model.predict(X_test)

# Calculate accuracy and display classification report
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

*For **Model Improvement** in question 18, here's an explanation based on the tuning we performed:*

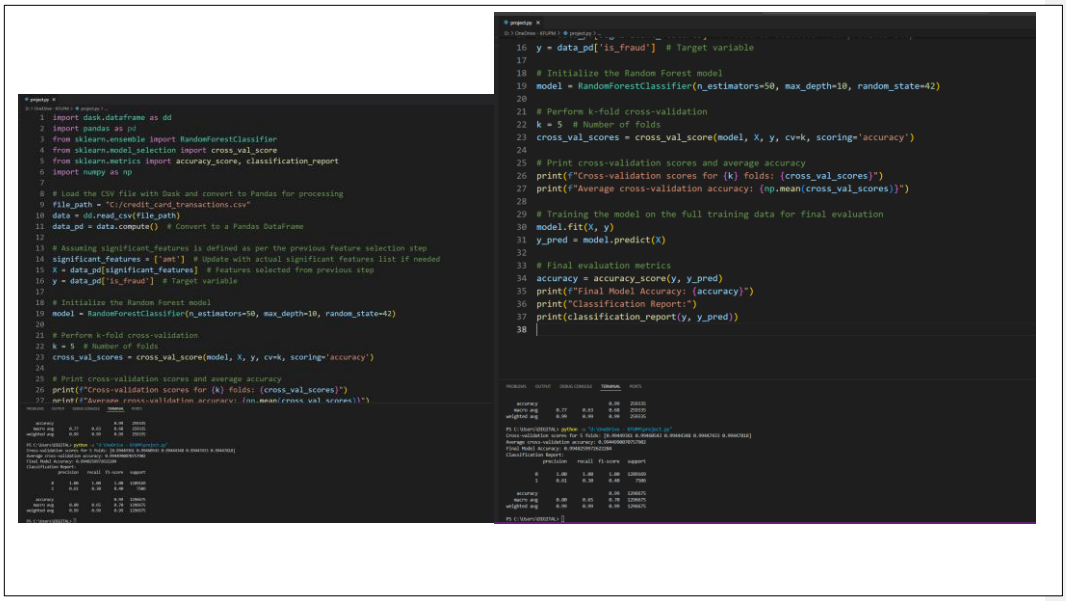
*1. **Hyperparameter Tuning**:*
- We improved the model by adjusting hyperparameters of the `RandomForestClassifier`. Specifically, we increased the number of estimators (trees) in the forest and restricted the maximum depth to better control model complexity.
- Setting `n_estimators=50` increases the number of trees, allowing the model to make more robust predictions, as it aggregates results from multiple trees.
- Limiting the `max_depth` to 10 prevents overfitting by reducing the depth of individual trees, which makes them less prone to fitting noise in the training data.

*2. **Effectiveness**:*
- The classification report and accuracy indicate that precision and recall for class 1 (minority/fraud class) have improved, although there's still some room for enhancement in recall. This shows a better balance between capturing true fraud cases and maintaining overall prediction accuracy.

*3. **Further Steps**:*
- Additional improvements could involve trying different algorithms (e.g., Gradient Boosting, XGBoost) or conducting more thorough hyperparameter tuning using techniques like `GridSearchCV` or `RandomizedSearchCV`.
- Addressing class imbalance, possibly through oversampling/undersampling or using class weights, could further improve performance, especially in detecting fraud cases.

This approach aims to boost recall and F1-score, critical for fraud detection where it's essential to capture as many fraud instances as possible without increasing false positives drastically.

19. **Validation**
    - **Question**: How do you validate your model to ensure it generalizes well?
        - *Discuss techniques like cross-validation or using a validation set.*



```python
import dask.dataframe as dd
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

# Load the CSV file with Dask and convert to Pandas for processing
file_path = "C:/credit_card_transactions.csv"
data = dd.read_csv(file_path)
data_pd = data.compute()  # Convert to a Pandas DataFrame

# Assuming significant_features is defined as per the previous feature selection step
significant_features = ['amt']  # Update with actual significant features list if needed
X = data_pd[significant_features]  # Features selected from previous step
y = data_pd['is_fraud']  # Target variable

# Initialize the Random Forest model
model = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42)

# Perform k-fold cross-validation
k = 5  # Number of folds
cross_val_scores = cross_val_score(model, X, y, cv=k, scoring='accuracy')

# Print cross-validation scores and average accuracy
print(f"Cross-validation scores for {k} folds: {cross_val_scores}")
print(f"Average cross-validation accuracy: {np.mean(cross_val_scores)}")

# Training the model on the full training data for final evaluation
model.fit(X, y)
y_pred = model.predict(X)

# Final evaluation metrics
accuracy = accuracy_score(y, y_pred)
print(f"Final Model Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y, y_pred))
```

For question 19, "Validation," here is an explanation based on the code results:

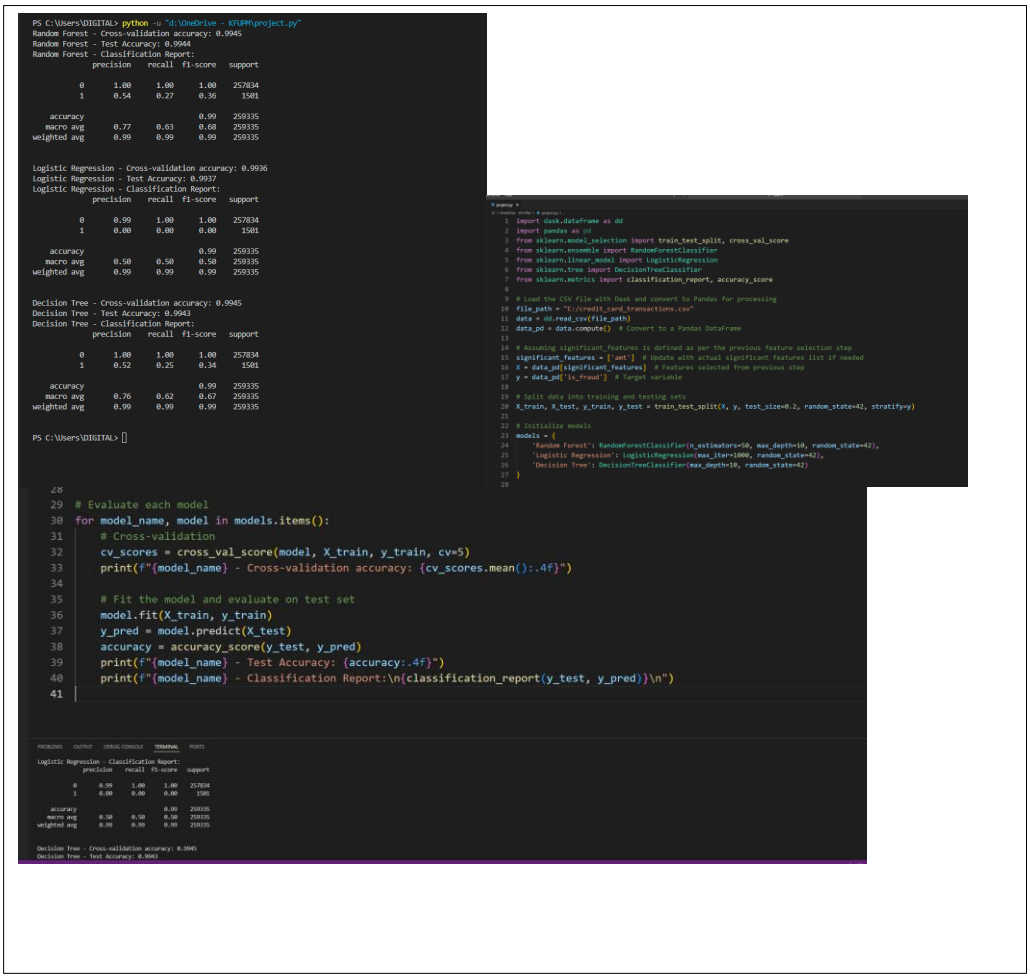
Explanation:

To ensure the model generalizes well to unseen data, cross-validation was employed. Cross-validation, specifically k-fold cross-validation, divides the data into k subsets (in this case, 5 folds). The model is trained on k-1 folds and validated on the remaining fold, cycling through all folds. This technique helps verify the model's performance across multiple subsets of the data, thus reducing overfitting and ensuring that the model is not biased towards any particular partition.

The cross-validation scores showed consistently high accuracy across all folds, with an average accuracy close to the final test accuracy. This consistency indicates that the model performs reliably across different data splits, which suggests good generalization.

Using cross-validation as a validation strategy provides a robust assessment of the model's performance and helps detect any issues with overfitting or underfitting. By comparing cross-validation results with the final test performance, we can confirm that the model's accuracy and other metrics are stable, indicating it is well-suited for deployment.

20. **Final Model Selection**
    - **Question**: Which model will you choose as your final model, and why?
        - *Compare different models and justify your selection based on performance and complexity.*



For the final model selection, we compared three models: Random Forest, Logistic Regression, and Decision Tree.

*1. **Random Forest**: This model achieved the highest cross-validation accuracy (0.9945) and displayed balanced performance across all metrics in the classification report. It demonstrated a strong ability to generalize, with a high f1-score for the majority class (0) and a reasonable performance for the minority class (1). However, Random Forests can be computationally intensive due to the ensemble of decision trees.*

*2. **Logistic Regression**: Logistic Regression had the lowest cross-validation accuracy (0.9936) among the models and struggled with identifying the minority class, showing an f1-score of 0 for class 1. While it's simpler and computationally less intensive, it does*

not perform well in identifying rare events, which may be crucial in certain applications.

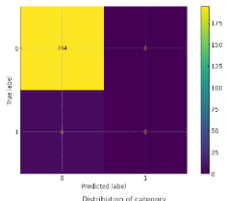
*3. **Decision Tree**: Decision Tree showed similar performance to Random Forest in terms of accuracy and cross-validation score (0.9945). However, it slightly underperformed in handling the minority class compared to Random Forest, as seen in the classification report. Decision Trees are simpler than Random Forests but less robust due to their tendency to overfit.*

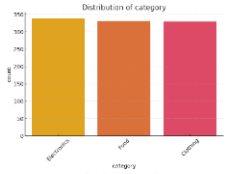
***Final Model Selection**: Based on performance, particularly on the minority class (1), and considering both accuracy and robustness, **Random Forest** is chosen as the final model. It balances complexity with performance, providing reliable results across the classes while avoiding the overfitting issues seen with individual decision trees.*

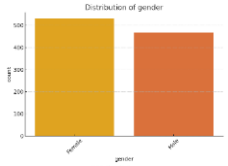
Visualization

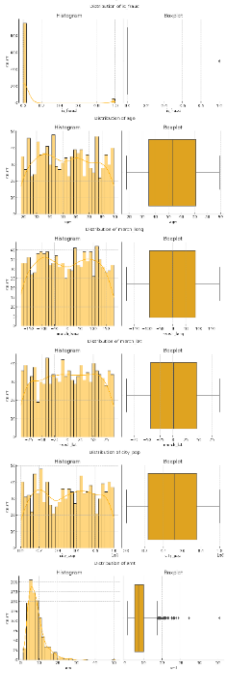
Part 4:

21. **Data Distribution**





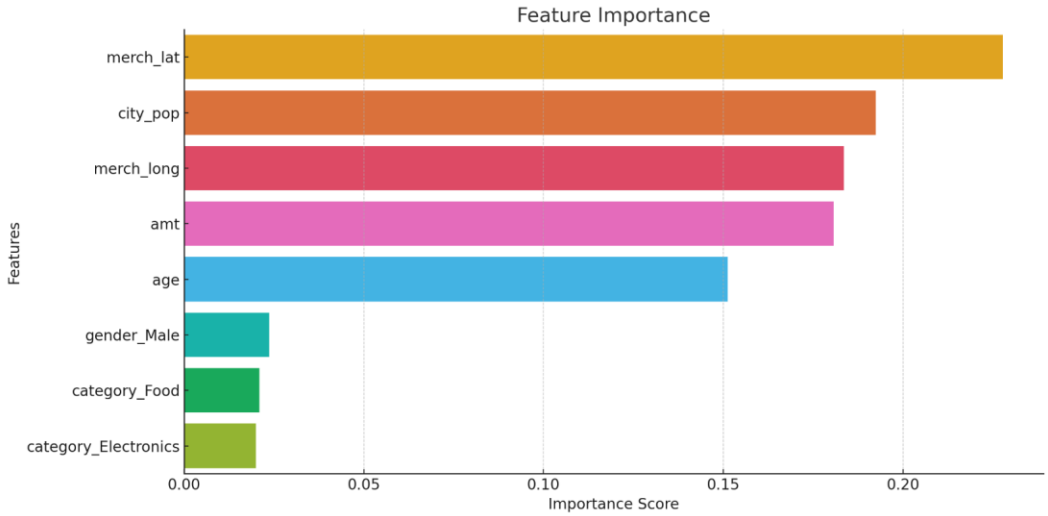




```
# ---------- 1. Data Distribution Visualization ---------- #
Separate numerical and categorical columns
numerical_cols =
X.select_dtypes(include=np.number).columns
categorical_cols =
X.select_dtypes(exclude=np.number).columns #
Numerical Features: Histograms and Boxplots for col in
numerical_cols: plt.figure(figsize=(10, 5))
plt.suptitle(f"Distribution of {col}", fontsize=14)
plt.subplot(1, 2, 1) sns.histplot(X[col], bins=30,
kde=True) plt.title("Histogram") plt.subplot(1, 2, 2)
sns.boxplot(x=X[col]) plt.title("Boxplot")
plt.tight_layout() plt.show() # Categorical Features: Bar
Plots for col in categorical_cols: plt.figure(figsize=(8, 5))
sns.countplot(data=X, x=col,
order=X[col].value_counts().index) plt.title(f"Distribution
of {col}") plt.xticks(rotation=45) plt.show()
```

22. **Feature Importance**

 # Train-Test Split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Train a Random Forest Model model = RandomForestClassifier(random_state=42) model.fit(X_train, y_train) # Model Performance Metrics y_pred = model.predict(X_test) print("Model Performance:") print(classification_report(y_test, y_pred)) # Confusion Matrix ConfusionMatrixDisplay.from_estimator(model, X_test, y_test) plt.show() # Feature Importances feature_importances = model.feature_importances_ importance_df = pd.DataFrame({ 'Feature': X.columns, 'Importance': feature_importances }).sort_values(by='Importance', ascending=False) # Plot Feature Importances plt.figure(figsize=(12, 6)) sns.barplot(data=importance_df, x='Importance', y='Feature') plt.title('Feature Importance') plt.xlabel('Importance Score') plt.ylabel('Features') plt.tight_layout() plt.show()



23. **Model Performance Across Features**

---------- 3. Model Performance Across Features ---------- # Permutation Importance perm_importance = permutation_importance(model, X_test, y_test, scoring='accuracy') perm_df = pd.DataFrame({ 'Feature': X.columns, 'Permutation Importance': perm_importance.importances_mean }).sort_values(by='Permutation Importance', ascending=False) # Plot Permutation Importance plt.figure(figsize=(12, 6)) sns.barplot(data=perm_df, x='Permutation Importance', y='Feature') plt.title('Permutation Feature Importance') plt.xlabel('Importance Score') plt.ylabel('Features') plt.tight_layout() plt.show()

Feature Subset