

Document Title: Credit Card Fraud Detection Project Plan

Introduction

Problem Statement

- Credit card fraud poses a significant financial risk to both cardholders and financial institutions. The aim of this project is to design and implement a credit card fraud detection system to identify and prevent fraudulent transactions.

Objectives: Develop an efficient and accurate fraud detection model. Minimize false positives to avoid inconveniencing legitimate cardholders. Enhance the security of credit card transactions.

PROBLEM STATEMENT: Developing a real-time credit card fraud detection system is a critical task that involves several steps, from data preprocessing to model deployment. Below, I'll outline a comprehensive project plan for building such a system:

1. Data Collection:- Obtain access to historical credit card transaction data, including both legitimate and fraudulent transactions. Ensure data privacy and security compliance.

2. Data Preprocessing:

Data Cleaning: Handle missing values, duplicate records, and outliers.

Data Transformation: Convert categorical variables into numerical representations (e.g., one-hot encoding).

Scaling: Normalize or standardize numerical features.

Feature Selection: Identify and select relevant features that contribute to fraud detection.

Time-Based Features: Create features based on transaction timestamps, such as hour of the day, day of the week, or time since the last transaction.

3. Exploratory Data Analysis (EDA): - Visualize and analyze the data to gain insights into the distribution of legitimate and fraudulent transactions. Explore correlations between features and fraud. Determine if there's class imbalance (likely), and decide on strategies to address it.

4. Feature Engineering: - Create new features that may improve fraud detection, such as transaction amounts relative to the cardholder's historical spending patterns. Experiment with different feature engineering techniques to enhance model performance.

5. Model Selection: - Choose appropriate machine learning algorithms for fraud detection. Common choices include:

Logistic Regression , Random Forest , Gradient Boosting , Neural Networks , Anomaly Detection Models (e.g., Isolation Forest, One-Class SVM). Consider ensemble methods to combine multiple models for better performance.

6. Data Splitting: - Split the dataset into training, validation, and testing sets. Ensure that the test set is representative of real-world data.

7. Model Training: - Train selected models using the training data. Tune hyperparameters using the validation set through techniques like cross-validation.

8. Model Evaluation:- Evaluate models using appropriate metrics, such as:

Accuracy, Precision and Recall, F1-Score, ROC-AUC . Use a confusion matrix to analyze model performance.

9. Addressing Class Imbalance: - Implement techniques to handle class imbalance, such as oversampling, undersampling, or using algorithms designed for imbalanced data.

10. Threshold Selection: - Choose a threshold for the model's predicted probabilities to classify transactions as either fraudulent or legitimate. Adjust the threshold based on the desired trade-off between false positives and false negatives.

11. Model Deployment: - Deploy the selected model(s) into a real-time production environment. Implement an API or interface for real-time credit card fraud detection.

12. Monitoring and Maintenance:- Continuously monitor the model's performance in production. Update the model periodically with new data to adapt to evolving fraud patterns. Implement alerting mechanisms for suspicious activities.

13. Documentation and Reporting: - Document the entire project, including data sources, preprocessing steps, model architectures, and deployment procedures. Create reports and dashboards for stakeholders to track fraud detection performance.

14. Compliance and Security: - Ensure that the system complies with regulatory requirements and data security standards (e.g., GDPR, PCI DSS).

15. User Training: - Train relevant personnel on using and interpreting the fraud detection system.

DESIGN THINKING:

Design Thinking is an iterative approach to problem-solving and product development that emphasizes understanding user needs and delivering innovative solutions. Here's how you can apply Design Thinking principles to the development of a credit card fraud detection system using the provided dataset:

1. Empathize:- Understand the stakeholders, including customers, financial institutions, and regulatory bodies. Conduct interviews and surveys to gather insights into their pain points, concerns, and expectations regarding credit card fraud detection. Explore the experiences of both victims of fraud and legitimate users.

2. Define:- Clearly define the problem: create a problem statement that captures the essence of the credit card fraud detection challenge, such as "How might we improve the accuracy of credit card fraud detection while minimizing false positives?" . Identify specific user needs and requirements based on the insights gathered during the empathize phase.

3. Ideate:

Generate ideas for addressing the problem:- Brainstorm potential features or functionalities that could enhance fraud detection.Explore different algorithms, techniques, and approaches for modeling and detecting fraud.Consider user-friendly interfaces and real-time alerting mechanisms.

4. Prototype:

Create a low-fidelity prototype or mockup of the fraud detection system:-Design the user interface for both data input (if applicable) and output.Develop a simplified version of the machine learning model for testing purposes.Implement a basic alerting system for fraudulent transactions.

5. Test:- Gather feedback by presenting the prototype to potential users and stakeholders.Collect data on how well the prototype addresses user needs and whether it effectively detects fraud. Use user feedback to refine the prototype and iterate on the design.

6. Iterate:

Based on the feedback and test results, make necessary improvements to the system:- Adjust the machine learning model based on the performance metrics.Refine the user interface to make it more intuitive.Enhance the alerting and reporting mechanisms.

7. Implement: - Develop the full-scale credit card fraud detection system based on the refined prototype.Integrate it with the chosen dataset and machine learning models.Ensure data privacy and security measures are in place.

8. Test (Again):- Conduct rigorous testing of the complete system in a controlled environment.Use simulated and historical data to assess the system's performance.

9. Deploy:-Deploy the system to a real-world environment, such as a financial institution's infrastructure.Continuously monitor its performance and fine-tune as necessary.

10. Evaluate and Iterate (Continuous Improvement):- Regularly evaluate the system's effectiveness in detecting fraud while minimizing false positives.Gather feedback from users and stakeholders to identify areas for improvement.Iterate on the system's design and functionality to adapt to evolving fraud patterns and user needs.

DATA PREPROSESING:

Data preprocessing is a crucial step in building a credit card fraud detection system. In this phase, you will clean the data, handle missing values, and normalize features to ensure that the dataset is suitable for training machine learning models.

1. Data Cleaning:Identify and handle missing values: Determine the extent of missing data in each column.Decide on a strategy to handle missing values, such as imputation (mean, median, mode), removal of rows/columns, or using advanced imputation techniques.Remove duplicates: Check for and remove duplicate transactions if they exist.Outlier detection: Identify and potentially handle outliers, as extreme values can affect model performance.Check for data consistency and correctness: Verify that data types are appropriate, and values fall within expected ranges (e.g., transaction amounts, timestamps).

2. **Handling Class Imbalance:**In credit card fraud detection, there is often a significant class imbalance, with many more legitimate transactions than fraudulent ones. Address class imbalance by using techniques such as oversampling the minority class (fraudulent transactions) or undersampling the majority class (legitimate transactions). You can also explore synthetic data generation methods like Synthetic Minority Over-sampling Technique (SMOTE).
3. **Normalization and Scaling:**Normalize or standardize numerical features to bring them to a common scale. Common methods include z-score normalization (subtract mean and divide by standard deviation) or min-max scaling (scale to a specific range, e.g., [0, 1]). Feature scaling ensures that features contribute equally to model training and prevents certain features from dominating others.
4. **Encoding Categorical Variables:**If your dataset includes categorical variables (e.g., merchant information, card details), encode them into numerical format. Common techniques include one-hot encoding or label encoding, depending on the nature of the categorical data.
5. **Data Splitting:**Split the dataset into training, validation, and testing sets. A common split is 70-15-15 or 80-10-10, depending on the dataset size.
6. **Data Imbalance Handling (Again):**When splitting the data into training and validation sets, ensure that both sets maintain the class balance (i.e., have a similar proportion of fraudulent and legitimate transactions).
7. **Save Preprocessed Data:**Save the cleaned and preprocessed data in a format that is easily accessible for model training. Common formats include CSV, HDF5.

FEATURE ENGINEERING:

Feature engineering plays a crucial role in enhancing the performance of a credit card fraud detection system. By creating additional features that capture meaningful information from the data, you can improve the model's ability to identify fraudulent transactions.

1. **Transaction Frequency: Number of Transactions in a Time Window:**Calculate the number of transactions made by a cardholder within specific time windows (e.g., the last hour, day, week). Unusual spikes in transaction frequency could indicate fraud.

Transaction Rate:Calculate the rate of transactions (transactions per hour or day) for each cardholder. A sudden increase in transaction rate may be suspicious.

2. **Amount Deviations: Transaction Amount Deviation:**Compute the deviation of each transaction amount from the cardholder's historical spending patterns. You can calculate this deviation using metrics like z-scores or percentage differences.

Average Transaction Amount:Calculate the average transaction amount for each cardholder. Sudden deviations from this average may indicate fraud.

Maximum Transaction Amount:Identify the maximum transaction amount for each cardholder. Unusually large transactions could be flagged.

Minimum Transaction Amount: Determine the minimum transaction amount for each cardholder. Extremely small transactions may also be suspicious.

Amount Clustering: Group transaction amounts into clusters based on their similarity and assign a cluster label to each transaction. Detect anomalies based on transactions that do not belong to any cluster or belong to a very small cluster.

3. Time-Based Features:
Time Since Last Transaction: Calculate the time elapsed since the cardholder's last transaction. Large time gaps between transactions may raise suspicions.

Time Between Transactions: Compute the time intervals between consecutive transactions. Unusually short or long time intervals may indicate fraud.

Hour of the Day: Create a categorical feature representing the hour of the day when each transaction occurred. Fraudulent activity may exhibit patterns at specific hours.

Day of the Week: Similarly, create a categorical feature for the day of the week. Fraudulent activity may vary by weekdays or weekends.

4. Aggregated Features:

Average Transaction Amount per Merchant: Calculate the average transaction amount for each merchant for each cardholder. Identify deviations from typical spending at specific merchants.

Transaction Frequency per Merchant: Compute the number of transactions made at each merchant. Sudden changes in transaction frequency with a particular merchant can be indicative of fraud.
Transaction Amount Deviation by Merchant: Calculate the deviation of transaction amounts at each merchant from the cardholder's overall spending pattern.

MODEL SELECTION:

Ensemble Methods, specifically Random Forest or Gradient Boosting (e.g., XGBoost or LightGBM). Here's why:

- **Ensemble Learning:** Ensemble methods combine the predictions of multiple machine learning models to improve overall performance. Credit card fraud detection often deals with imbalanced datasets (a small percentage of transactions are fraudulent), and ensemble methods can effectively handle this class imbalance.
- **High Accuracy:** Random Forest and Gradient Boosting are known for their high predictive accuracy. They can capture complex patterns and relationships in the data, which is crucial for detecting fraudulent transactions that may not follow typical patterns.
- **Robustness:** These algorithms are robust to outliers and noisy data, which is essential for real-world financial data where anomalies can occur due to various reasons.
- **Feature Importance:** Random Forest and Gradient Boosting provide feature importance scores, which can help in understanding which features (transaction

attributes) are most relevant for fraud detection. This insight can be valuable for fraud analysts.

- **Ease of Use:** While they are more complex than some other algorithms, Random Forest and Gradient Boosting are relatively easy to implement and require less hyperparameter tuning compared to deep learning models.
- **Scalability:** These algorithms can handle large datasets efficiently, making them suitable for real-time or batch processing in credit card transaction systems.

MODEL TRAINING:

1. Data Preparation: Ensure that you have split your preprocessed data into training and validation sets. Verify that both sets maintain class balance (fraudulent and legitimate transactions) if you previously addressed class imbalance.

2. Model Initialization: Instantiate the chosen machine learning model with the selected hyperparameters.

3. Model Training: Train the model using the training data. Use the validation data to monitor the model's performance during training. This helps you detect overfitting and fine-tune hyperparameters if needed.

4. Hyperparameter Tuning (Optional): If your model has hyperparameters that need tuning (e.g., learning rate, max depth for trees), perform hyperparameter optimization. Techniques like grid search or random search can help you find the best hyperparameters.

5. Cross-Validation (Optional): To ensure robustness of the model, consider using k-fold cross-validation. This involves splitting the data into k subsets (folds), training and validating the model k times, and averaging the results.

6. Model Evaluation: Evaluate the model's performance using various metrics, such as accuracy, precision, recall, F1-score, and ROC-AUC, on both the training and validation datasets. Use a confusion matrix to gain insights into the model's performance in terms of false positives and false negatives.

7. Interpretability (Optional): If using a complex model (e.g., neural network), consider methods for model interpretability. This helps in understanding why the model makes specific predictions, which can be crucial for fraud detection systems.

8. Model Saving: Once satisfied with the model's performance, save the trained model to disk for future use.

9. Model Serialization and Deployment: Serialize the model so that it can be easily deployed in a production environment. Deploy the model in a real-time or batch processing system, depending on the operational requirements of your fraud detection system.

10. Monitoring and Maintenance: Continuously monitor the deployed model's performance in a production environment. Implement automated processes for model retraining using new data to adapt to evolving fraud patterns.

11. Documentation: Document the model's architecture, hyperparameters, and training process for reference and future updates.

EVALUATION:

Accuracy:Accuracy measures the overall correctness of the model's predictions, including both true positives and true negatives.However, accuracy can be misleading in highly imbalanced datasets where most transactions are legitimate. A high accuracy may be achieved by simply classifying all transactions as legitimate.

Precision:High precision indicates a low rate of false positives, which is crucial in fraud detection to avoid inconveniencing legitimate cardholders.

Recall (Sensitivity or True Positive Rate):Recall measures the proportion of true positive predictions among all actual positive cases.High recall is important for capturing as many fraudulent transactions as possible, minimizing false negatives.

F1-Score:The F1-score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance.

Using Formula: $2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$

The F1-score is useful when there is an imbalance between the classes, as it considers both false positives and false negatives.

Receiver Operating Characteristic Area Under the Curve (ROC-AUC):ROC-AUC measures the area under the Receiver Operating Characteristic (ROC) curve, which plots the True Positive Rate (Recall) against the False Positive Rate at various threshold values.A higher ROC-AUC indicates a better ability to distinguish between fraudulent and legitimate transactions.

```
#import libraries
```

```
Import pandas as pd
```

```
Import numpy as np
```

```
Import matplotlib.pyplot as plt
```

```
From sklearn.ensemble import
```

```
#Read the Dataset
```

```
# Loading the data
```

```
Df = pd.read_csv('credit_card_detection.csv')
```

Df

dataset size

Data.shape

#fully described data

Data.describe

Fraud =data[data['class']==1]

Valid=data[data['class']==0]

Fvalue=len(fraud)/len(valid)

Print(fvalue)

Print('Fraud cases:{}'.format(len(data['class']==1))))

Print('valid transaction:{}'.format(len(data[data['class']==0])))

fraud.Amount.describe()

valid.Amount.describe()

Df.columns

Df.isnull().sum()

Df.head()

Df['IssFraud?'].value_counts()

**Independent=df[['User', 'Card', 'Year', 'Month', 'Day','UseChip',
 'MerchantName', 'MerchantCity', 'MerchantState', 'Zip', 'MCC']]**

Dependent=df[['IssFraud?']]

Print(independent)


```

Def quanQual(df):

    Quan=[]

    Qual=[]

    For columnName in df.columns:

        #print(columnName)

        If(df[columnName].dtypes=='O'):

            #print("qual")

            Qual.append(columnName)

        Else:

            #print("quan")

            Quan.append(columnName)

    Return quan,qual

Quan,qual=quanQual(df)

Quan

Import numpy as np

From sklearn.impute import SimpleImputer

Imp=SimpleImputer(missing_values=np.nan,strategy="mean",copy=True)

Imp.fit(df[quan])

Datan=imp.transform(df[quan])

Datan

Datan=pd.DataFrame(datan,columns=quan)

Import numpy as np

```

```
From sklearn.impute import SimpleImputer
```

```
Imp=SimpleImputer(missing_values=np.nan,strategy="most_frequent")
```

```
Imp.fit(df[qual])
```

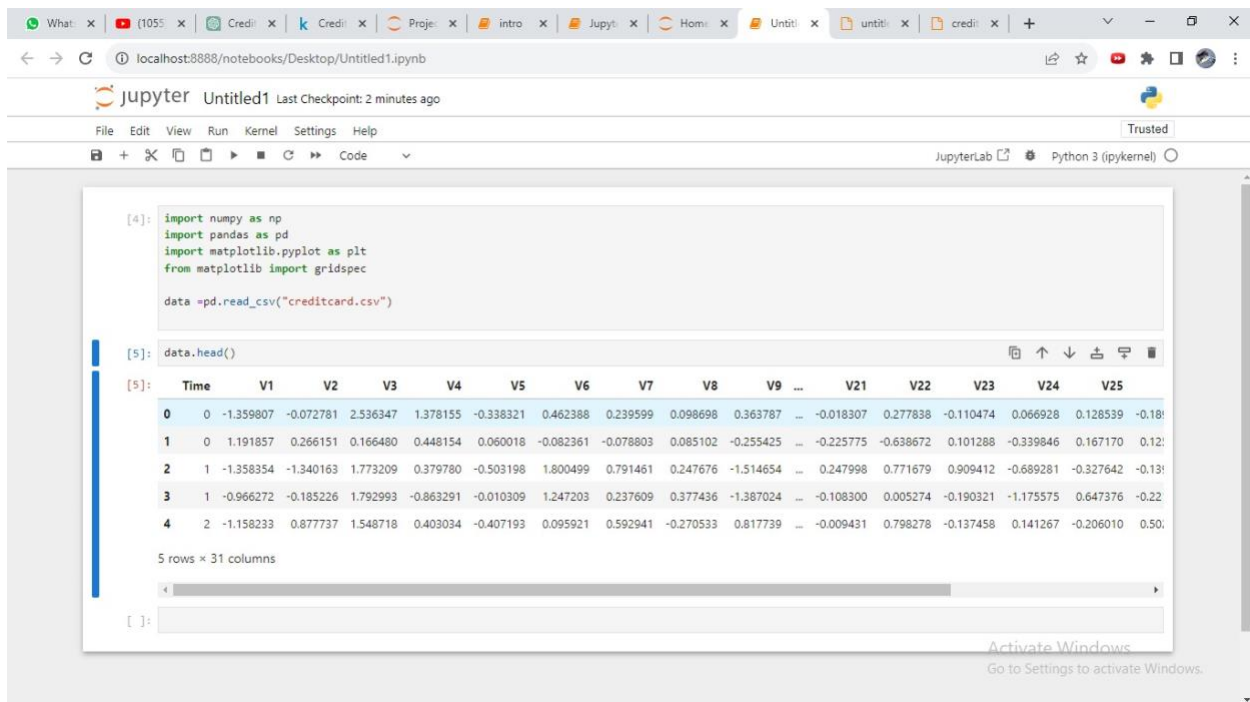
```
Datal=imp.transform(df[qual])
```

```
Df=pd.concat([datan,datal],axis=1)
```

```
Csv=df.to_csv("Preprocessed_credit_card_detection.csv",index=False)
```

OUTPUT:

IMPORTING LIBRARIES



The screenshot shows a Jupyter Notebook window titled 'Untitled1' with the following code in the first cell:

```
[4]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import gridspec

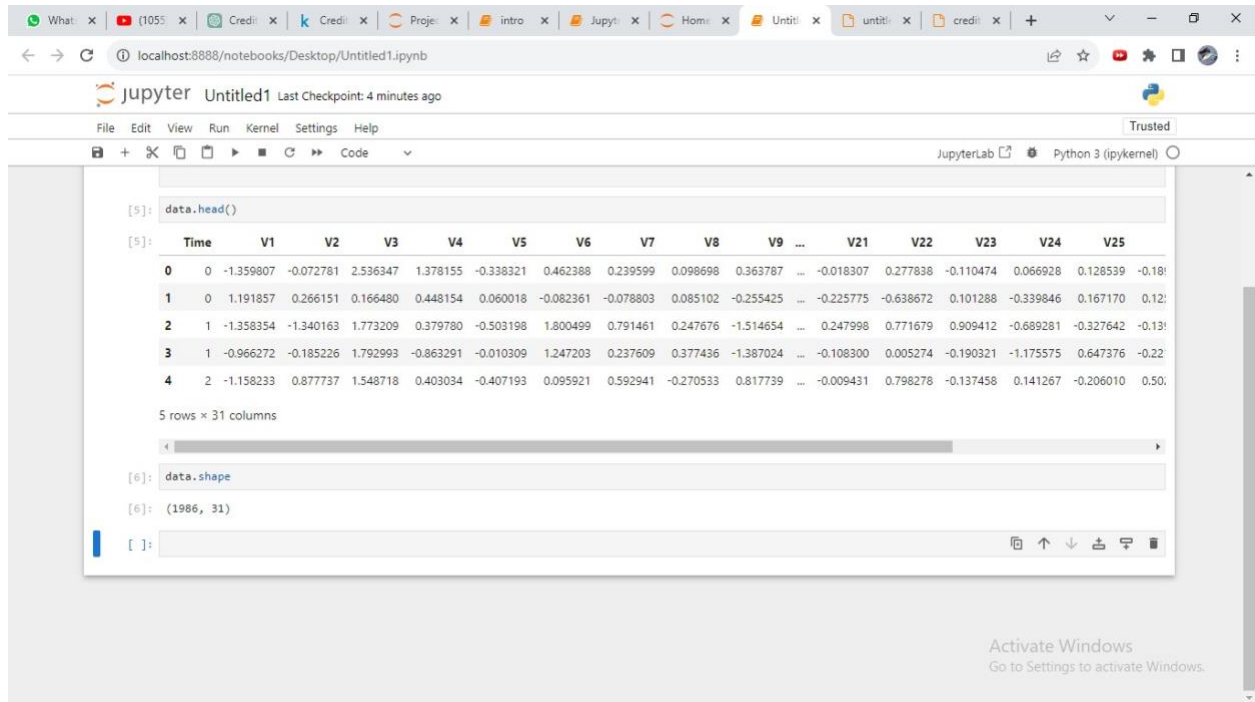
data = pd.read_csv("creditcard.csv")
```

The second cell contains the command `data.head()`, which has been executed, resulting in a preview of the first 5 rows of the dataset. The preview shows columns: Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, ..., V21, V22, V23, V24, V25. The data is as follows:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.181
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.121
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.131
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.501

Below the table, it indicates '5 rows x 31 columns'. At the bottom right of the notebook interface, there is a watermark that says 'Activate Windows. Go to Settings to activate Windows.'

DATA SIZE AND DATASET



A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/Desktop/Untitled1.ipynb'. The notebook title is 'Untitled1' and it was last checkpointed 4 minutes ago. The menu bar includes File, Edit, View, Run, Kernel, Settings, and Help. The toolbar shows various icons for file operations and execution. The code cell contains two lines of Python code: `data.head()` and `data.shape`. The output of `data.head()` shows a table with 5 rows and 31 columns. The columns are labeled Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, ..., V21, V22, V23, V24, V25. The output of `data.shape` is `(1986, 31)`.

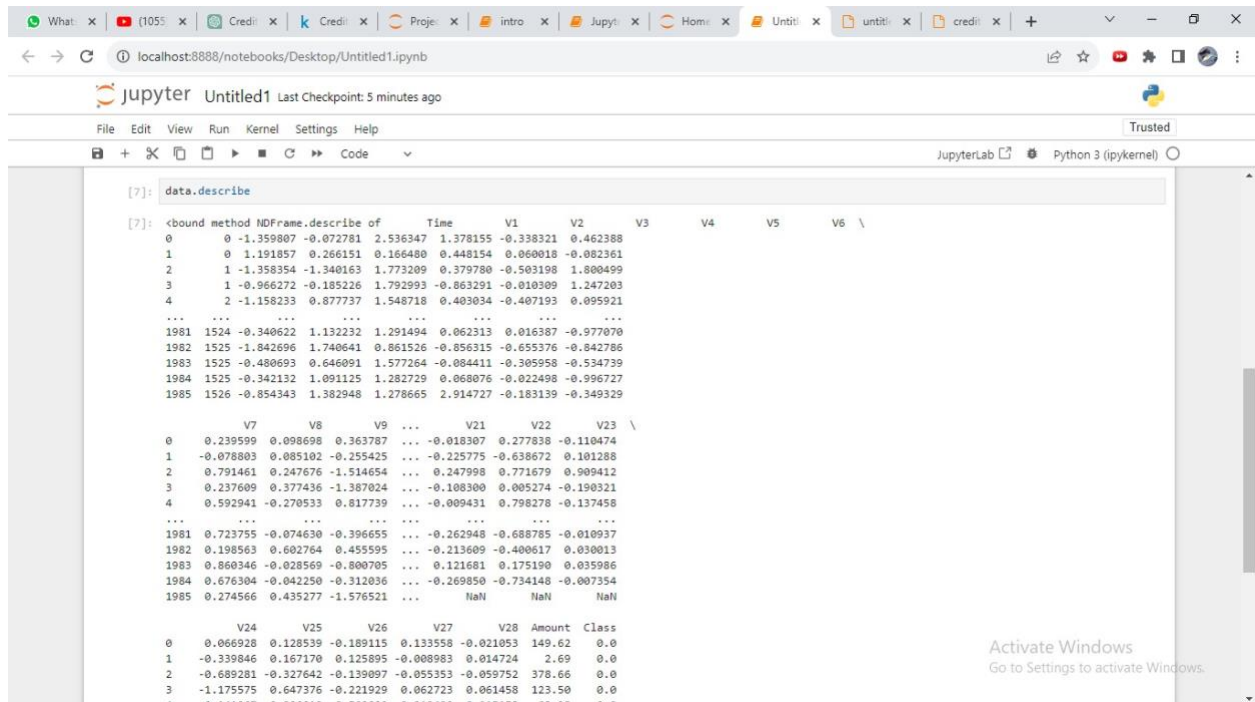
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.181
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.12
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.13
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.22
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.50

5 rows x 31 columns

`data.shape`

`(1986, 31)`

FULLY DESCRIBED DATA



A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/Desktop/Untitled1.ipynb'. The notebook title is 'Untitled1' and it was last checkpointed 5 minutes ago. The menu bar includes File, Edit, View, Run, Kernel, Settings, and Help. The toolbar shows various icons for file operations and execution. The code cell contains the line `data.describe()`. The output shows a summary of the data, including the number of non-null observations, the mean, standard deviation, and the minimum and maximum values for each column. The columns are labeled Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, ..., V21, V22, V23, V24, V25, Amount, and Class.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	Amount	Class
count	1986	1986	1986	1986	1986	1986	1986	1986	1986	1986	...	1986	1986	1986	1986	1986	1986	1986
mean	0.5	-0.0001	-0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	...	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
std	0.5	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001	...	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001
min	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.181	0
max	2	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.12	2

What: x (1055: x Credit: x Credit: x Proj: x intro: x Jupy: x Home: x Untit: x unti: x credi: x +

localhost:8888/notebooks/Desktop/Untitled1.ipynb

Jupyter Untitled1 Last Checkpoint: 12 minutes ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (ipykernel)

```
1 -0.339846 0.167170 0.125895 -0.008983 0.014724 2.69 0.0
2 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752 378.66 0.0
3 -1.175575 0.647376 -0.221929 0.062723 0.061458 123.50 0.0
4 0.141267 -0.206010 0.502292 0.219422 0.215153 69.99 0.0
...
1981 0.334061 -0.160025 0.071779 0.245128 0.098336 5.35 0.0
1982 0.512611 -0.077087 0.286218 0.586012 0.352610 1.00 0.0
1983 0.557665 -0.112301 0.337154 -0.015602 0.051504 80.70 0.0
1984 0.319161 -0.179146 0.073683 0.241932 0.097139 3.59 0.0
1985 NaN NaN NaN NaN NaN NaN NaN
[1986 rows x 31 columns]>
[8]: fraud = data[data['Class']==1]
valid = data[data['Class']==0]
fvalue=len(fraud)/len(valid)
print(fvalue)
print('Fraud cases :{}'.format(len(data[data['Class']==1])))
print('valid transactions :{}'.format(len(data[data['Class']==0])))
0.0010085728693898135
Fraud cases :2
valid transactions :1983
```

Activate Windows
Go to Settings to activate Windows.

What: x (1055: x Credit: x Credit: x Proj: x intro: x Jupy: x Home: x Untit: x unti: x credi: x +

localhost:8888/notebooks/Desktop/Untitled1.ipynb

Jupyter Untitled1 Last Checkpoint: 15 minutes ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (ipykernel)

```
[1986 rows x 31 columns]>
[8]: fraud = data[data['Class']==1]
valid = data[data['Class']==0]
fvalue=len(fraud)/len(valid)
print(fvalue)
print('Fraud cases :{}'.format(len(data[data['Class']==1])))
print('valid transactions :{}'.format(len(data[data['Class']==0])))
0.0010085728693898135
Fraud cases :2
valid transactions :1983
[9]: fraud.Amount.describe()
valid.Amount.describe()
[9]: count 1983.000000
mean 68.404892
std 241.572682
min 0.000000
25% 4.950000
50% 15.090000
75% 63.285000
max 7712.430000
Name: Amount, dtype: float64
```

Activate Windows
Go to Settings to activate Windows.

