Name: Priansh Madan

Batch: E4

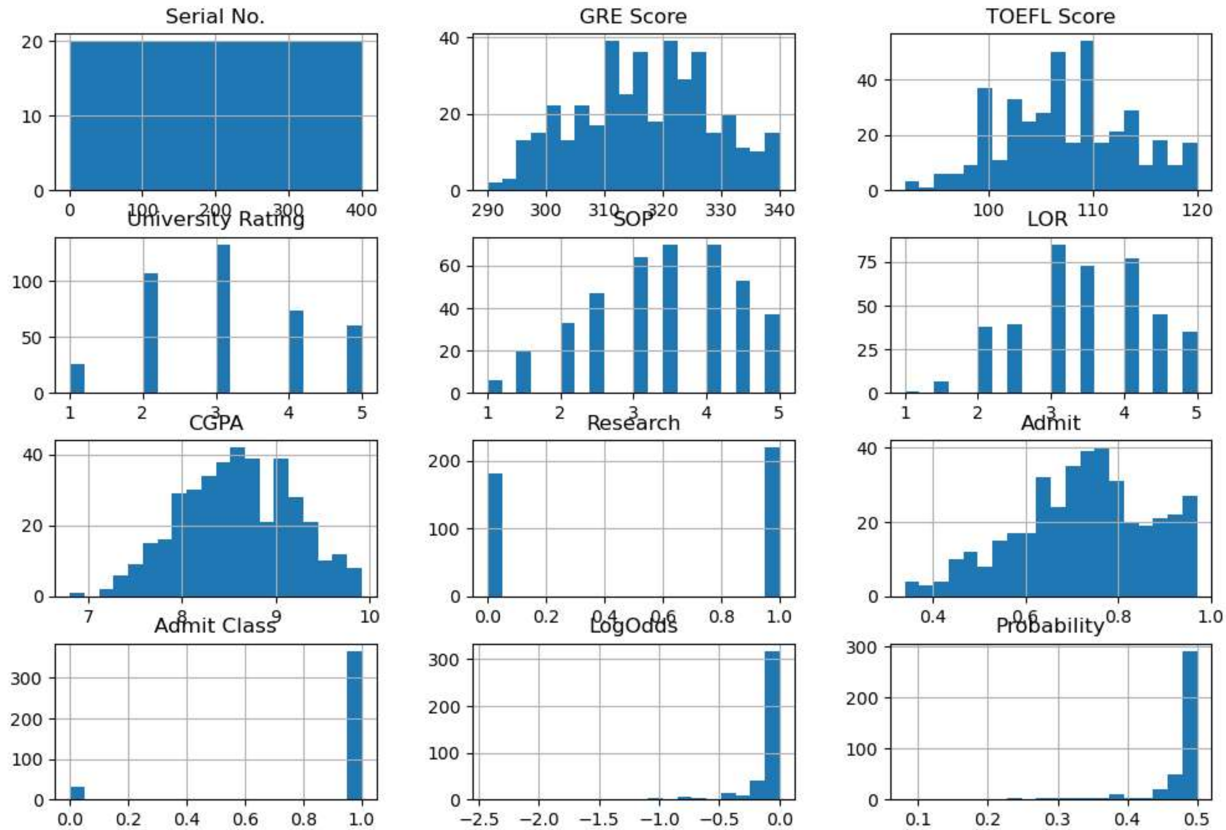
Roll no.: 58

1. Perform logistic regression on the admission dataset

a) Import the Admission_Predict dataset, display it, and visualize various columns:

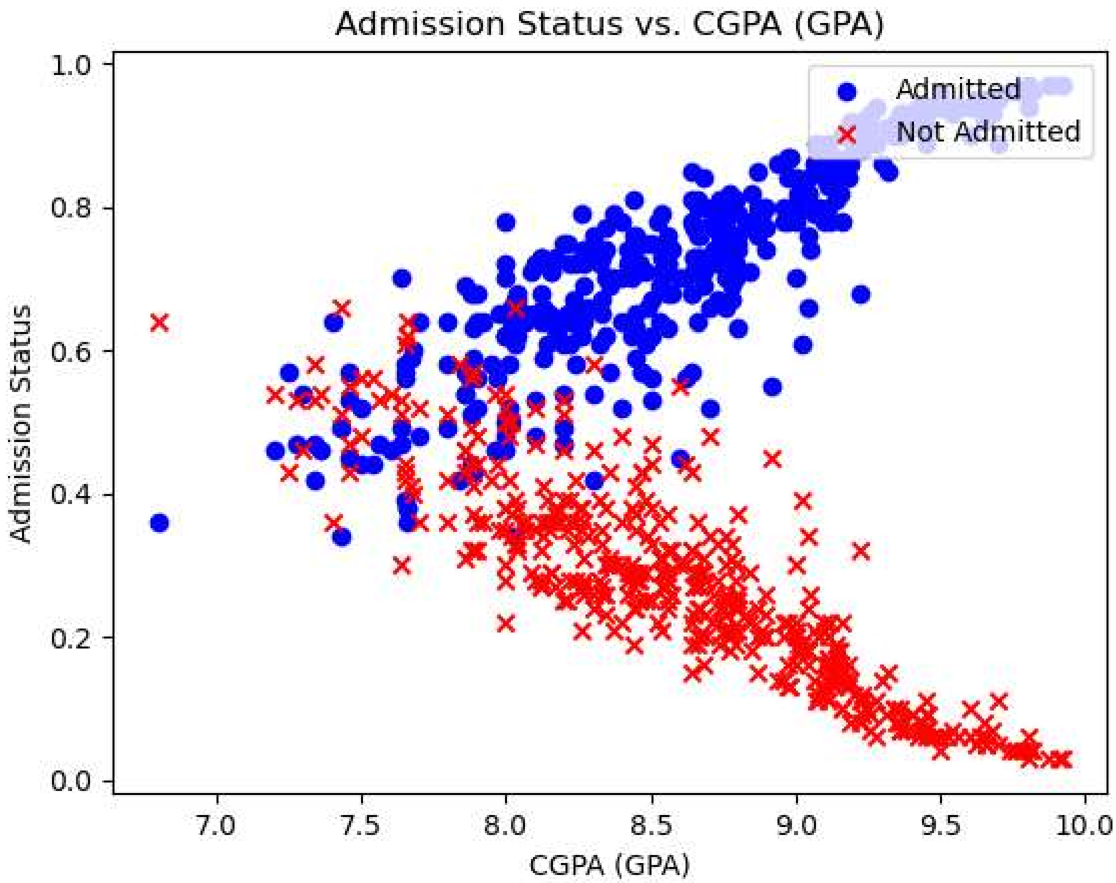
In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = pd.read_csv("Admission_Predict (1).csv")

print(data.head())
data.hist(bins=20, figsize=(12, 8))
plt.show()
```

```
   Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0           1        337          118                  4  4.5   4.5  9.65
1           2        324          107                  4  4.0   4.5  8.87
2           3        316          104                  3  3.0   3.5  8.00
3           4        322          110                  3  3.5   2.5  8.67
4           5        314          103                  2  2.0   3.0  8.21

   Research  Admit  Admit Class   LogOdds  Probability
0         1   0.92            1 -0.000671     0.499832
1         1   0.76            1 -0.009353     0.497662
2         1   0.72            1 -0.164149     0.459055
3         1   0.80            1 -0.018318     0.495421
4         0   0.65            1 -0.084027     0.479006
```



b) Plot the dataset on GPA vs. admit score:

In [ ]:
```
# b) Plot the dataset on GPA vs. admit score.
plt.scatter(data['CGPA'], data['Admit'], marker='o', c='b', label='Admitted')
plt.scatter(data['CGPA'], 1 - data['Admit'], marker='x', c='r', label='Not Admitted')
plt.xlabel('CGPA (GPA)')
plt.ylabel('Admission Status')
plt.legend(loc='upper right')
plt.title('Admission Status vs. CGPA (GPA)')
plt.show()
```



Find the slope and intercept of the line to fit:

In [ ]:
```
data['Admit Class'] = (data['Admit'] >= 0.5).astype(int)
data.to_csv('Admission_Predict.csv', index=False)

X = data[['CGPA']].values  # Reshape feature to (n_samples, n_features)
y = data['Admit Class'].values  # Reshape target to (n_samples, )
X=X.reshape(-1,1)
model = LogisticRegression()
model.fit(X, y)

slope = model.coef_[0][0]
intercept = model.intercept_[0]
print(f"Slope: {slope}, Intercept: {intercept}")
```

```
Slope: 3.3834358054129954, Intercept: -25.343700528067235
```

Compute the log odds for each entry and merge the results with the data as a new column:

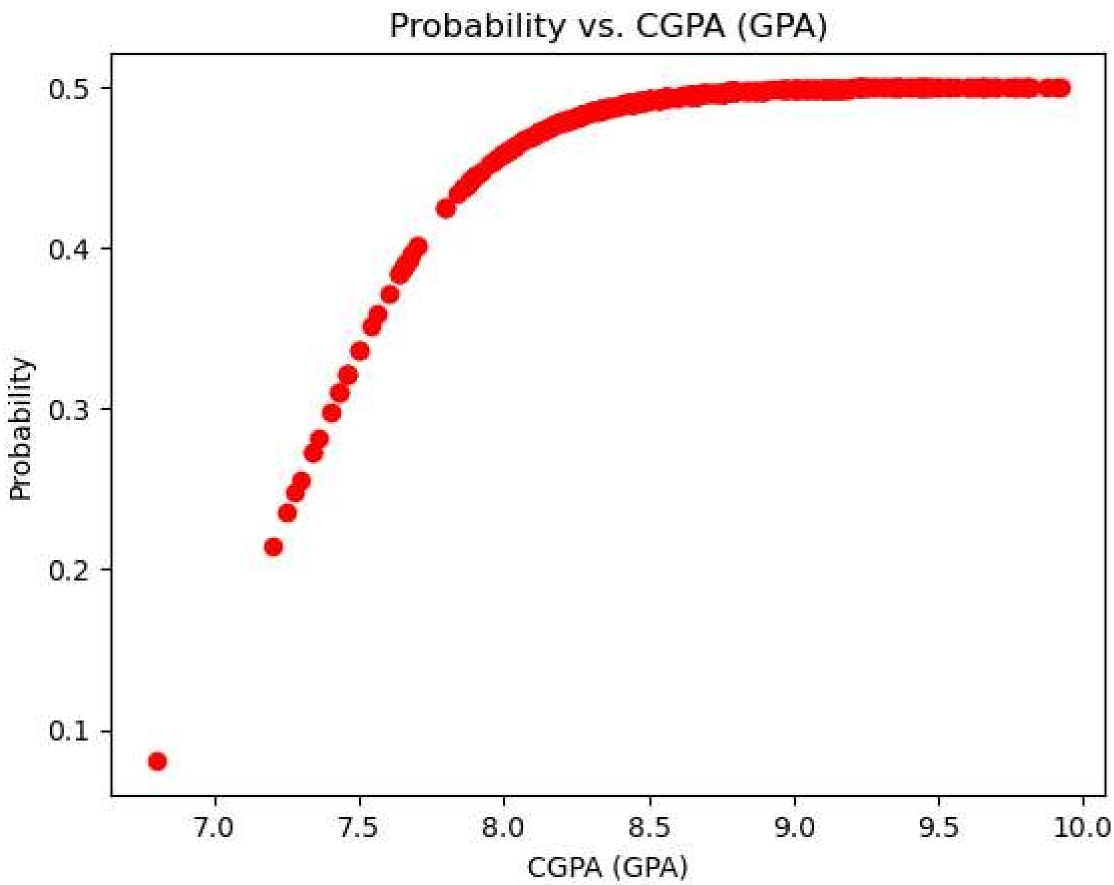
In [ ]:
```
# Compute log odds for each entry
log_odds = model.predict_log_proba(X)[:, 1]

# Merge log odds as a new column in the DataFrame
data['LogOdds'] = log_odds
```

e) Using the log odds compute the probability for each entry.

In [ ]:
```
data['Probability'] = 1 / (1 + np.exp(-log_odds))
data.to_csv('Admission_Predict (1).csv', index=False)
```

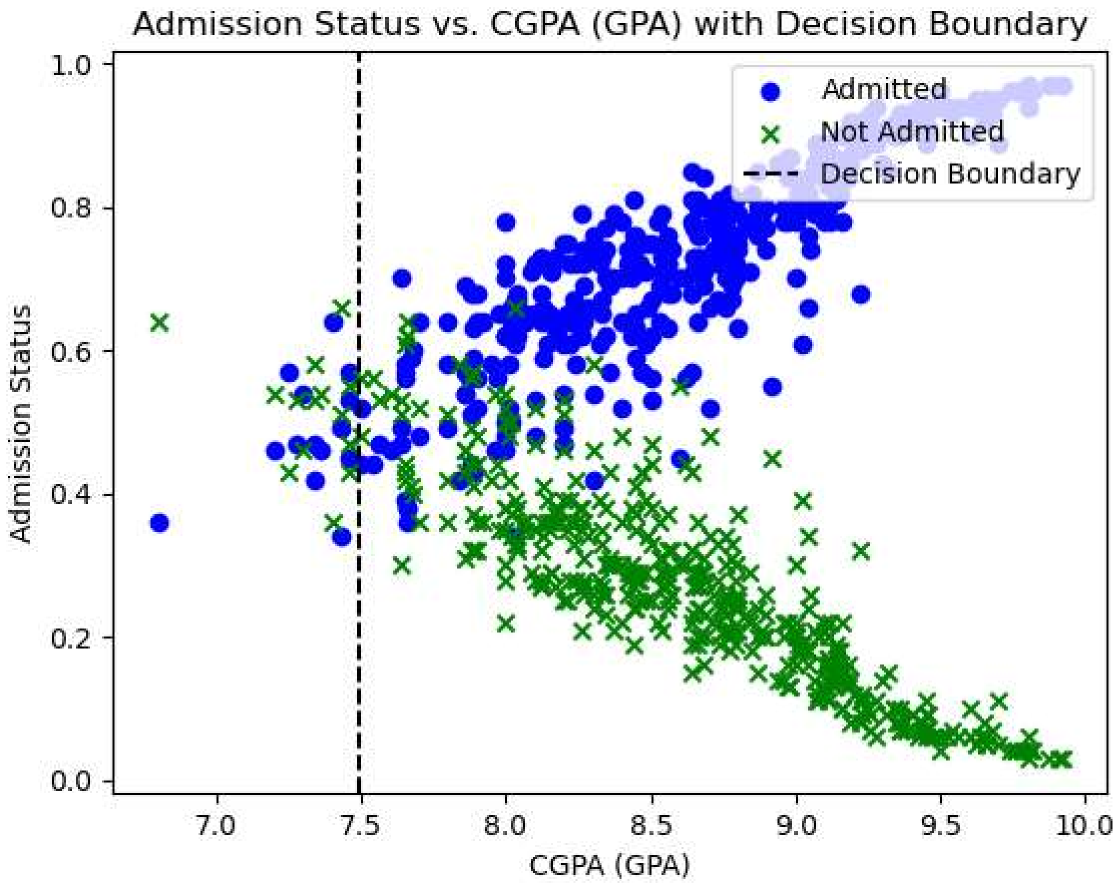
f) Plot the probabilities vs gpa graph.

In [ ]:
```
plt.scatter(data['CGPA'], data['Probability'], marker='o', c='r', label='Probability o
plt.xlabel('CGPA (GPA)')
plt.ylabel('Probability')
plt.title('Probability vs. CGPA (GPA)')
plt.show()
```



g) Show the decision boundary of the regression model.

In [ ]:
```
X_values = np.linspace(data['CGPA'].min(), data['CGPA'].max(), 100)
decision_boundary = -intercept / slope  # Decision boundary where log odds = 0
plt.scatter(data['CGPA'], data['Admit'], marker='o', c='b', label='Admitted')
plt.scatter(data['CGPA'], 1 - data['Admit'], marker='x', c='g', label='Not Admitted')
plt.axvline(decision_boundary, color='k', linestyle='--', label='Decision Boundary')
plt.xlabel('CGPA (GPA)')
plt.ylabel('Admission Status')
plt.legend(loc='upper right')
plt.title('Admission Status vs. CGPA (GPA) with Decision Boundary')
plt.show()
```



h) Show the accuracy of the regressor model.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression Model: {accuracy * 100:.2f}%")
```

```
Accuracy of the Logistic Regression Model: 89.17%
```

In [ ]:

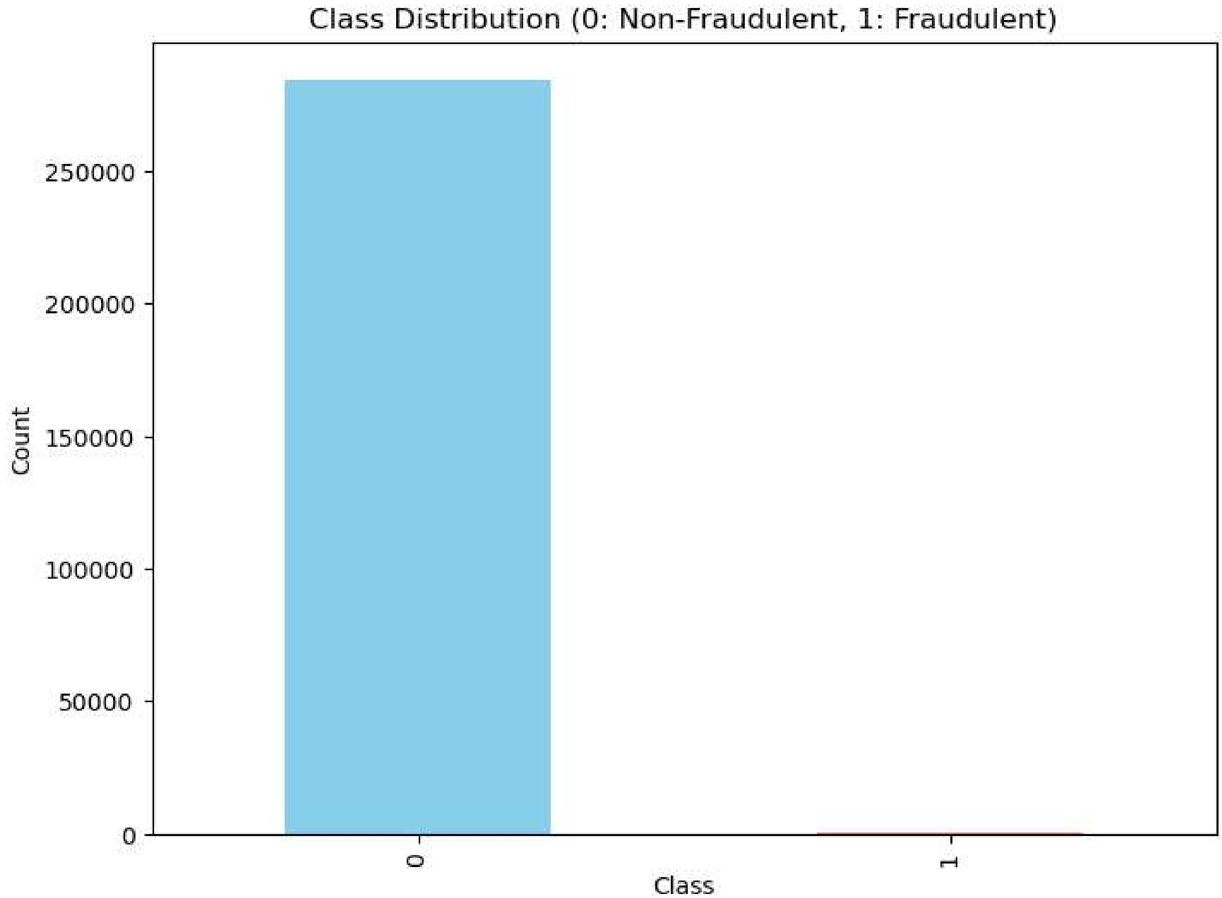
2. Perform logistic regression on the credit card dataset

Load the dataset, visualize it, show the data headers

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, precision_score, recall_score

# Load the dataset
df = pd.read_csv('creditcard.csv')
```

```
# Show the first few rows of the dataset
print(df.head())

# Visualize the class distribution (fraudulent vs. non-fraudulent)
plt.figure(figsize=(8, 6))
df['Class'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Class Distribution (0: Non-Fraudulent, 1: Fraudulent)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```



c) Preprocess the dataset if required

In [ ]:
```
# i. Check for duplicate data and remove it:
df = df.drop_duplicates()
print("Number of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 0
```

In [ ]:
```
# Remove unnecessary columns like 'Time':
df = df.drop(columns=['Time'])
```

In [ ]:
```
# Separate the dataset into feature and target columns:
X = df.drop(columns=['Class'])  # Features
y = df['Class']  # Target
```

In [ ]:
```
# Scale the dataset using standard scaling:
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

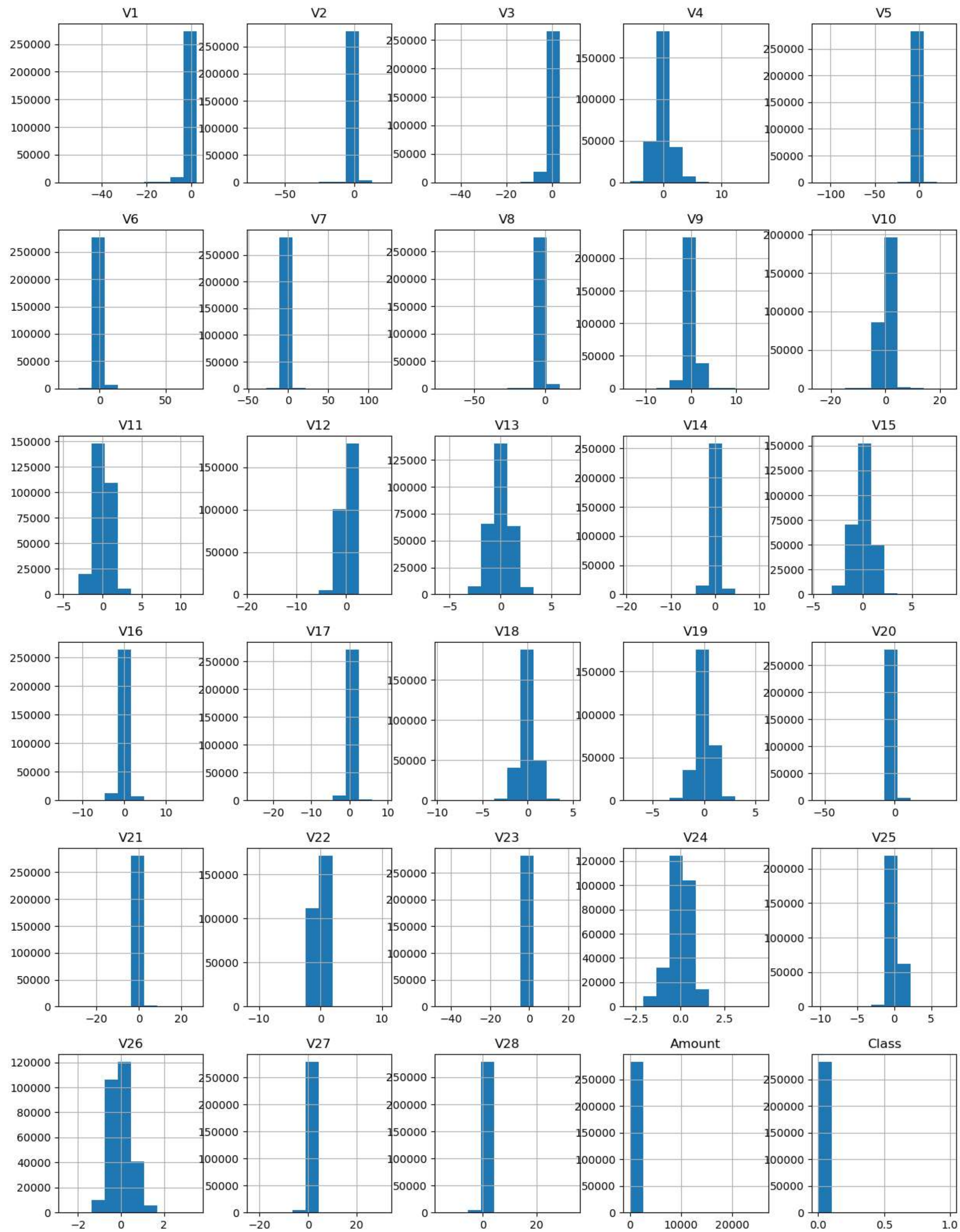
In [ ]:
```
# Partition the dataset into training and testing sets (80%-20%):

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, randon

# Check the shapes of the resulting datasets
```

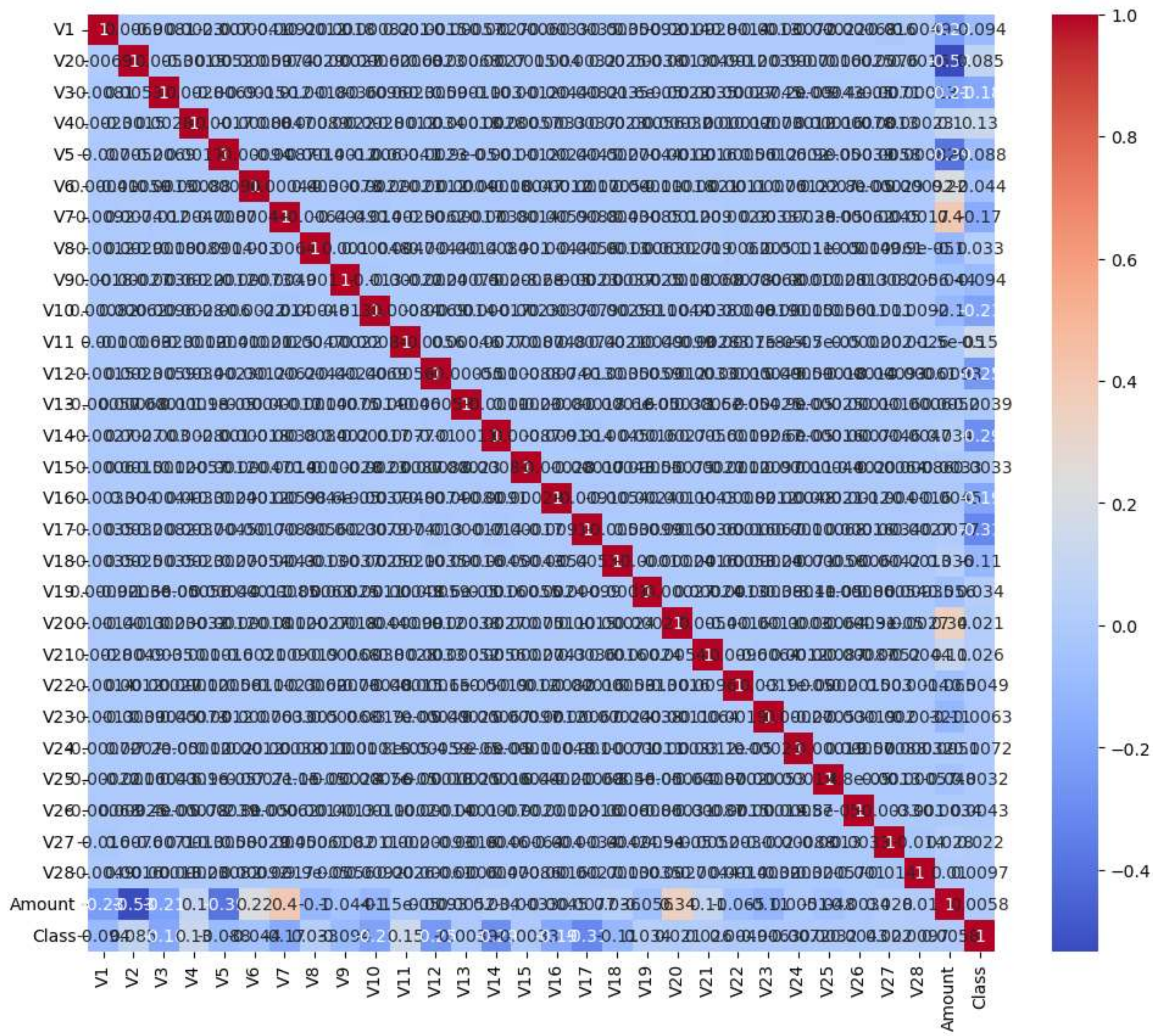
```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (226980, 29)
X_test shape: (56746, 29)
y_train shape: (226980,)
y_test shape: (56746,)
```

Step d: Plot histograms/heatmaps

In [ ]:
```
df.hist(figsize=(15, 20))
plt.show()
corr = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```


V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class



Step e: Train the model using logistic regression

In [ ]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[ ]:
```
LogisticRegression()
```

Step f: Obtain the training accuracy

In [ ]:
```
train_accuracy = model.score(X_train, y_train)
print(f"Training Accuracy: {train_accuracy}")
```

```
Training Accuracy: 0.9991937615648956
```

Step g: Test the model and obtain the testing accuracy

In [ ]:
```
test_accuracy = model.score(X_test, y_test)
print(f"Testing Accuracy: {test_accuracy}")
```

```
Testing Accuracy: 0.9991717477883904
```

Step h: Generate confusion matrix, precision and recall

In [ ]:
```
y_pred = model.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

In [ ]:
```
print(f"Confusion Matrix:\n{confusion}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Confusion Matrix:
[[56650     6]
 [   41    49]]
Precision: 0.8909090909090909
Recall: 0.5444444444444444
```

In [ ]: