```
# import csv file
import pandas as pd
credit_card_data = pd.read_csv('credit_card_data.csv')
print(credit_card_data)
```

```
            Time         V1         V2        V3        V4
V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321

1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018

2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198

3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309

4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193

...          ...        ...        ...       ...       ...       ...

284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473

284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229

284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515

284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961

284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546


              V6        V7        V8        V9  ...       V21
V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307
0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -
0.638672
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998
0.771679
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300
0.005274
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431
0.798278
...          ...       ...       ...       ...  ...       ...       ..
.
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454
0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205
0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045
0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245
```

```
0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057
0.643078

              V23       V24       V25       V26       V27       V28
Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
69.99
...          ...       ...       ...       ...       ...       ...
...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649
217.00

        Class
0           0
1           0
2           0
3           0
4           0
...       ...
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

```python
import pandas as pd

# Assuming you have already read the CSV file into credit_card_data
credit_card_data = pd.read_csv('credit_card_data.csv')

# Split the data into features (X) and target (Y)
```

```python
X = credit_card_data.drop(columns='Class', axis=1)
Y = credit_card_data['Class']

# Print the first few rows of X and Y to check
print("Features (X):")
print(X.head())

print("\nTarget (Y):")
print(Y.head())
```

```
Features (X):
   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  ...       V20       V21       V22       V23
V24  \
0  0.098698  0.363787  ...  0.251412 -0.018307  0.277838 -0.110474
0.066928
1  0.085102 -0.255425  ... -0.069083 -0.225775 -0.638672  0.101288 -
0.339846
2  0.247676 -1.514654  ...  0.524980  0.247998  0.771679  0.909412 -
0.689281
3  0.377436 -1.387024  ... -0.208038 -0.108300  0.005274 -0.190321 -
1.175575
4 -0.270533  0.817739  ...  0.408542 -0.009431  0.798278 -0.137458
0.141267

        V25       V26       V27       V28  Amount
0  0.128539 -0.189115  0.133558 -0.021053  149.62
1  0.167170  0.125895 -0.008983  0.014724    2.69
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3  0.647376 -0.221929  0.062723  0.061458  123.50
4 -0.206010  0.502292  0.219422  0.215153   69.99

[5 rows x 30 columns]

Target (Y):
0    0
1    0
2    0
```

```
3    0
4    0
Name: Class, dtype: int64

import pandas as pd
from imblearn.over_sampling import RandomOverSampler

# Assuming X and Y are already defined and contain your data
# Split the data into features (X) and target (Y)
# X = credit_card_data.drop(columns='Class', axis=1)
# Y = credit_card_data['Class']

# Handle the class imbalance issue using RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, Y_resampled = ros.fit_resample(X, Y)

# Print the resampled data
print("Resampled Features (X):")
print(X_resampled.head())

print("\nResampled Target (Y):")
print(Y_resampled.head())

Resampled Features (X):
   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  ...       V20       V21       V22       V23       
V24  \
0  0.098698  0.363787  ...  0.251412 -0.018307  0.277838 -0.110474  
0.066928
1  0.085102 -0.255425  ... -0.069083 -0.225775 -0.638672  0.101288 -
0.339846
2  0.247676 -1.514654  ...  0.524980  0.247998  0.771679  0.909412 -
0.689281
3  0.377436 -1.387024  ... -0.208038 -0.108300  0.005274 -0.190321 -
1.175575
4 -0.270533  0.817739  ...  0.408542 -0.009431  0.798278 -0.137458  
0.141267
```

```
        V25       V26       V27       V28  Amount
0  0.128539 -0.189115  0.133558 -0.021053  149.62
1  0.167170  0.125895 -0.008983  0.014724    2.69
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3  0.647376 -0.221929  0.062723  0.061458  123.50
4 -0.206010  0.502292  0.219422  0.215153   69.99

[5 rows x 30 columns]

Resampled Target (Y):
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler

# Assuming X_resampled and Y_resampled are already defined and contain
your resampled data
# Handle the class imbalance issue using RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, Y_resampled = ros.fit_resample(X, Y)

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_resampled,
Y_resampled, test_size=0.2, random_state=42)

# Print the shape of training and testing sets to see the output
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: (454904, 30)
X_test shape: (113726, 30)
Y_train shape: (454904,)
Y_test shape: (113726,)
```

```python
from sklearn.linear_model import LogisticRegression

# Create a Logistic Regression model
model = LogisticRegression()

# Print the model to see the output
print(model)
```

```
LogisticRegression()
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler

# Assuming X_resampled and Y_resampled are already defined and contain
your resampled data
# Handle the class imbalance issue using RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, Y_resampled = ros.fit_resample(X, Y)

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_resampled,
Y_resampled, test_size=0.2, random_state=42)

# Scale the input features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Logistic Regression model with increased max_iter
model = LogisticRegression(max_iter=1000)  # Increase max_iter to 1000
or a higher value

# Train the model on the scaled training data
model.fit(X_train_scaled, Y_train)

# Print a message to indicate that the training is complete
print("Model training completed.")
```

```
Model training completed.
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import RandomOverSampler

# Assuming X_resampled and Y_resampled are already defined and contain
your resampled data
# Handle the class imbalance issue using RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, Y_resampled = ros.fit_resample(X, Y)

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_resampled,
Y_resampled, test_size=0.2, random_state=42)

# Scale the input features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)

# Create a Logistic Regression model with increased max_iter
model = LogisticRegression(max_iter=1000)  # Increase max_iter to 1000
or a higher value

# Train the model on the scaled training data
model.fit(X_train_scaled, Y_train)

# Evaluate the model on the training data
X_train_prediction = model.predict(X_train_scaled)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data:', training_data_accuracy)
```

```
Accuracy on Training data: 0.9493849251710251
```

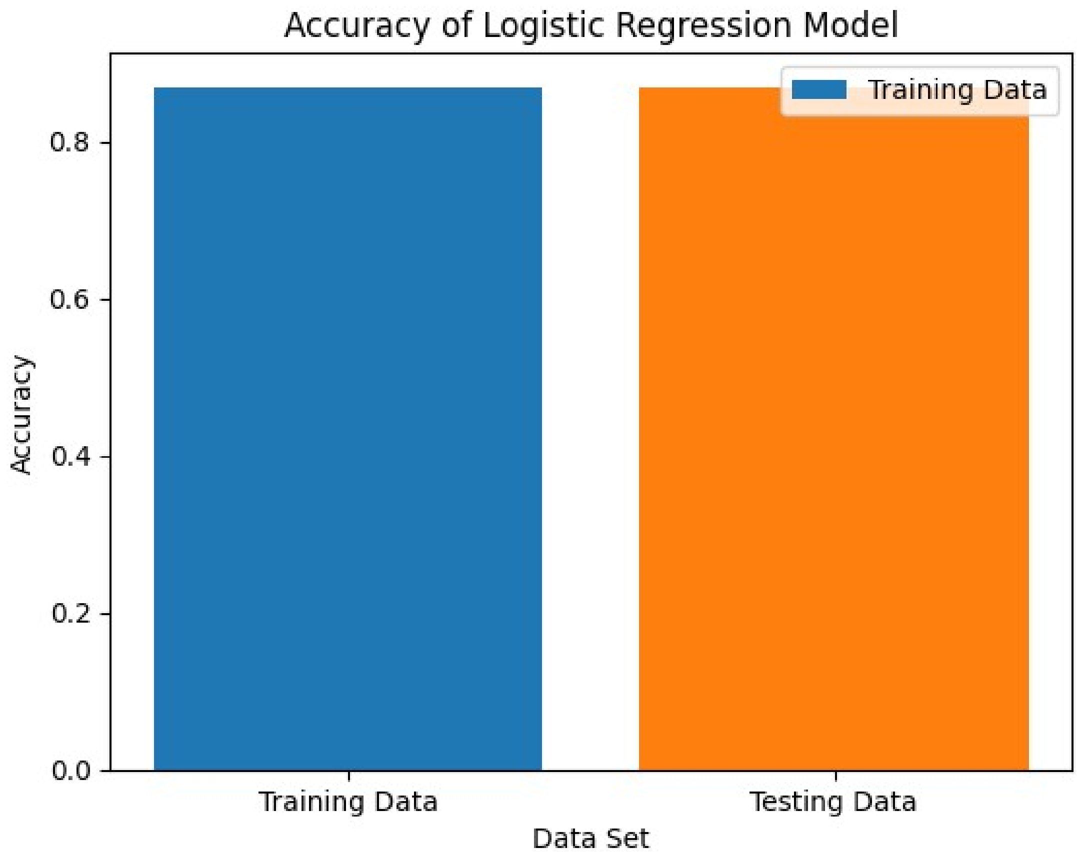
```
from sklearn.metrics import accuracy_score

# Assuming X_test_scaled, Y_test, and model are already defined and
contain your scaled testing data and trained model, respectively
# Evaluate the model on the testing data
X_test_prediction = model.predict(X_test_scaled)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score on Test Data:', test_data_accuracy)
```

```
Accuracy score on Test Data: 0.9495541916536236
```

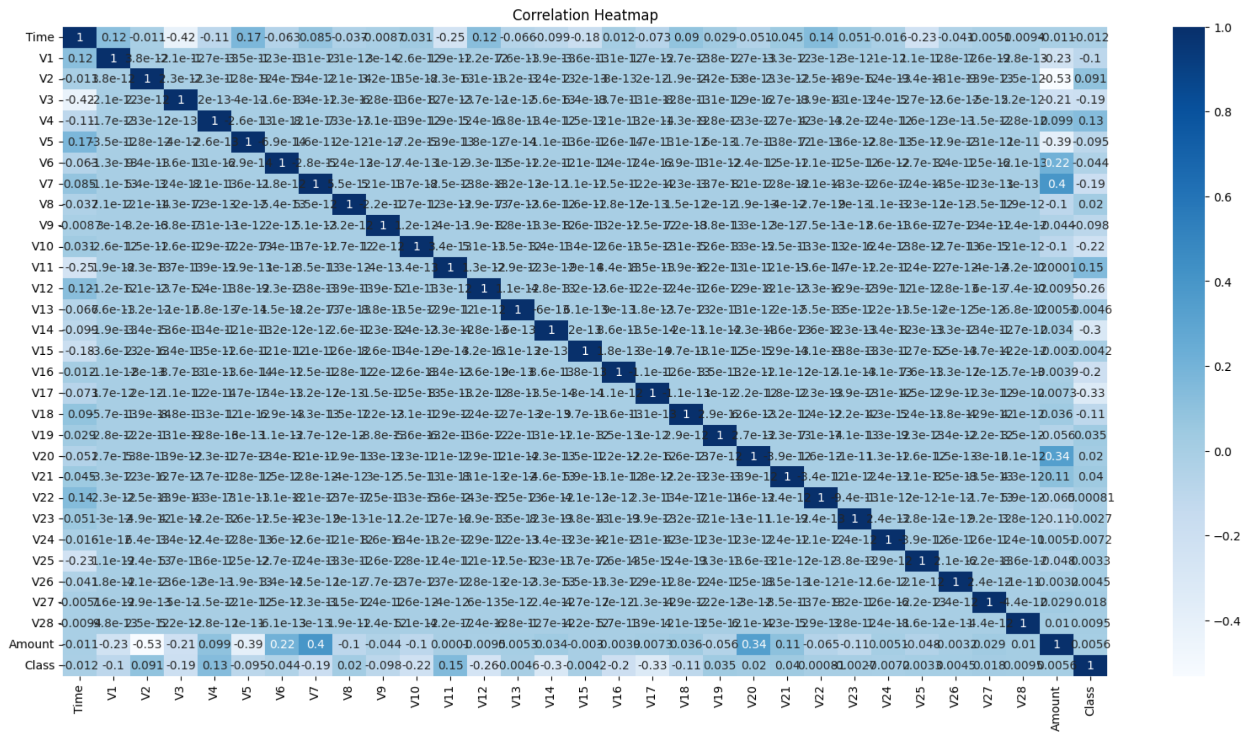
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import RandomOverSampler
import matplotlib.pyplot as plt

# Load the dataset
credit_card_data = pd.read_csv('credit_card_data.csv')

# Split the data into features (X) and target (Y)
X = credit_card_data.drop(columns='Class', axis=1)
Y = credit_card_data['Class']

# Handle the class imbalance issue using RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, Y_resampled = ros.fit_resample(X, Y)

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_resampled,
Y_resampled, test_size=0.2, random_state=42)

# Create a Logistic Regression model with increased max_iter and a
different solver
```

```
model = LogisticRegression(max_iter=1000, solver='saga')  # Increase
max_iter and try a different solver like 'saga'

# Train the model on the training data
model.fit(X_train, Y_train)

# Evaluate the model on the training data
training_data_accuracy = accuracy_score(Y_train,
model.predict(X_train))

# Evaluate the model on the testing data
test_data_accuracy = accuracy_score(Y_test, model.predict(X_test))

# Plot the accuracy
plt.bar(['Training Data', 'Testing Data'], [training_data_accuracy,
test_data_accuracy], color=['#1f77b4', '#ff7f0e'])
plt.xlabel('Data Set')
plt.ylabel('Accuracy')
plt.title('Accuracy of Logistic Regression Model')
plt.legend(['Training Data', 'Testing Data'])
plt.show()
```

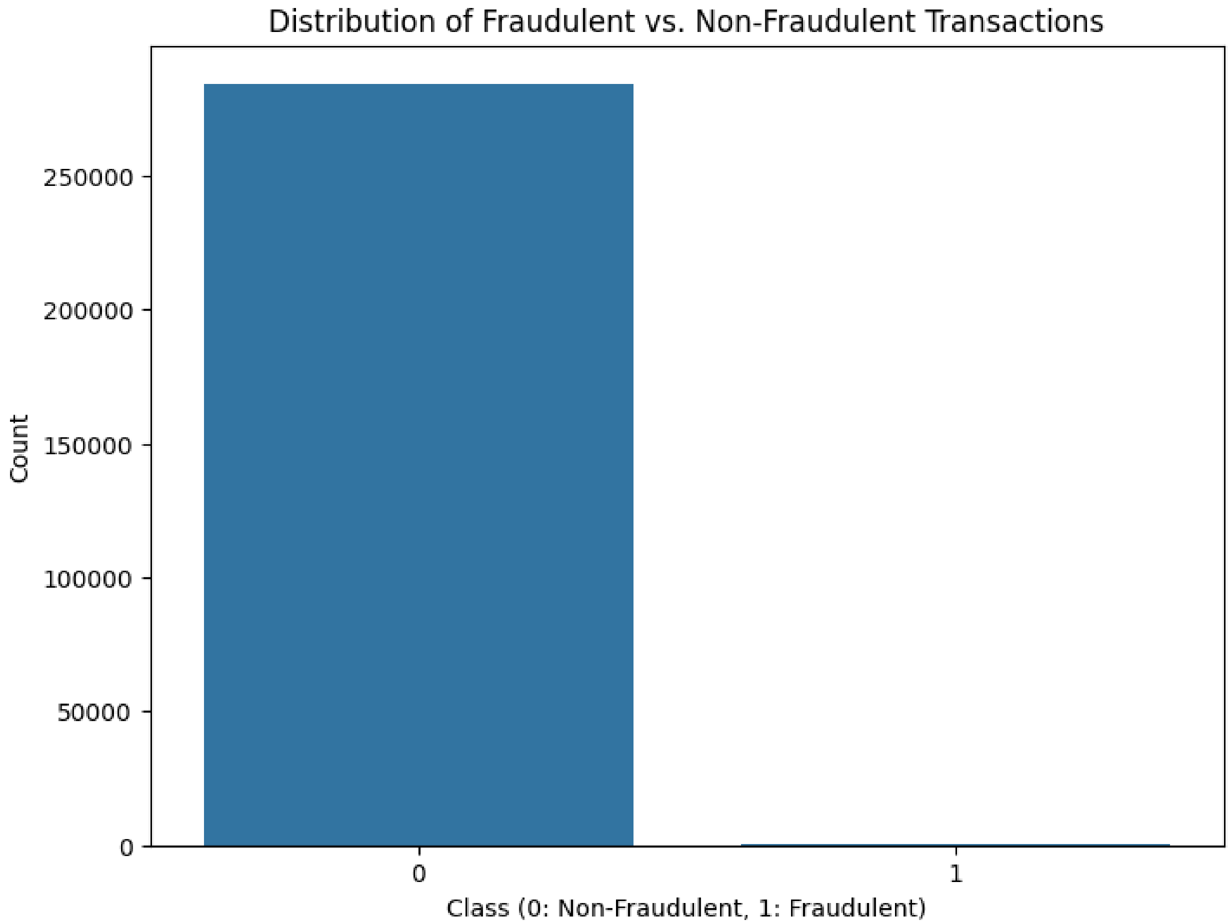
```
C:\Users\dell\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  warnings.warn(
```




```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'credit_card_data' is your DataFrame containing the data
# For example, if your DataFrame is named 'credit_card_data':

# Plotting the correlation heatmap
plt.figure(figsize=(20, 10))
sns.heatmap(credit_card_data.corr(), annot=True, cmap='Blues')
plt.title('Correlation Heatmap')
plt.show()
```




```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
credit_card_data = pd.read_csv('credit_card_data.csv')

# Plot the distribution of fraudulent vs. non-fraudulent transactions
plt.figure(figsize=(8, 6))
sns.countplot(data=credit_card_data, x='Class')
plt.title('Distribution of Fraudulent vs. Non-Fraudulent
Transactions')
plt.xlabel('Class (0: Non-Fraudulent, 1: Fraudulent)')
plt.ylabel('Count')
plt.show()
```


Distribution of Fraudulent vs. Non-Fraudulent Transactions
Count
250000
200000
150000
100000
50000
0
0
1
Class (0: Non-Fraudulent, 1: Fraudulent)