Credit Card Fraud Detection

Problem Statement: Build a machine learning model to identify fraudulent credit card transactions. Preprocess and normalize the transaction data, handle class imbalance issues, and split the dataset into training and testing sets. Train a classification algorithm, such as logistic regression or random forests, to classify transactions as fraudulent or genuine. Evaluate the model'

s performance using metrics like precision, recall,

and F1-score, and consider techniques like oversampling or undersampling for improving results.

In [1]:
```
# import the necessary packages
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

2. The Data Set

In the following cells, we will import our dataset from a .csv file as a Pandas DataFrame. Furthermore, we will begin exploring the dataset to gain an understanding of the type, quantity, and distribution of data in our dataset. For this purpose, we will use Pandas' built-in describe feature, as well as parameter histograms and a correlation matrix.

In [2]:
```
# Load the dataset from the csv file using pandas
data = pd.read_csv(r'D:\Datasets\creditcard.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [4]:
```
# Start exploring the dataset
data.columns
```

Out[4]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1
0',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
0',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [5]:
```
# Print the shape of the data
data.shape
```

Out[5]:
```
(284807, 31)
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [7]:
```
data.describe()
# V1 - V28 are the results of a PCA Dimensionality reduction to protect use
```

Out[7]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e

8 rows × 31 columns

In [8]:
```
data['Class'].value_counts()
```

Out[8]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [9]:
```
legit = data[data.Class==0]
fraud = data[data.Class==1]
```

In [10]:
```
legit.shape
```

Out[10]:
```
(284315, 31)
```

In [11]:
```
fraud.shape
```

Out[11]:
```
(492, 31)
```

In [12]:
```
legit.Amount.describe()
```

Out[12]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

In [13]:
```
fraud.Amount.describe()
```

Out[13]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

In [14]:
```
data.groupby('Class').mean()
```

Out[14]:

Class	Time	V1	V2	V3	V4	V5	V6	V
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0096
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5687

2 rows × 30 columns

Undersampling

Building a sample dataset which contain similar distribution of Legit transactions and Fraud transactions.

In [15]:
```
legit_sample = legit.sample(n=492)
```

Now will concatinate the two dataframe into legit sample and fraud

In [16]:
```
new_data = pd.concat([legit_sample, fraud], axis=0)
```

In [17]:
```
new_data.head()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7
237802	149400.0	2.161543	0.068915	-2.465194	-0.051840	1.251262	-0.251903	0.578921
111700	72331.0	-5.467393	4.520035	-0.956856	-2.375816	-1.511242	-1.500989	0.122696
79415	58017.0	-0.743646	0.723658	1.820321	0.474289	-0.072016	-0.278892	0.639433
269434	163657.0	-0.562317	0.704013	2.144150	-0.338884	-0.275852	-0.460350	0.481065
282038	170620.0	2.022222	-0.224775	-0.944111	0.529741	-0.530831	-1.423159	0.014193

5 rows × 31 columns

In [18]:
```
new_data.tail()
```

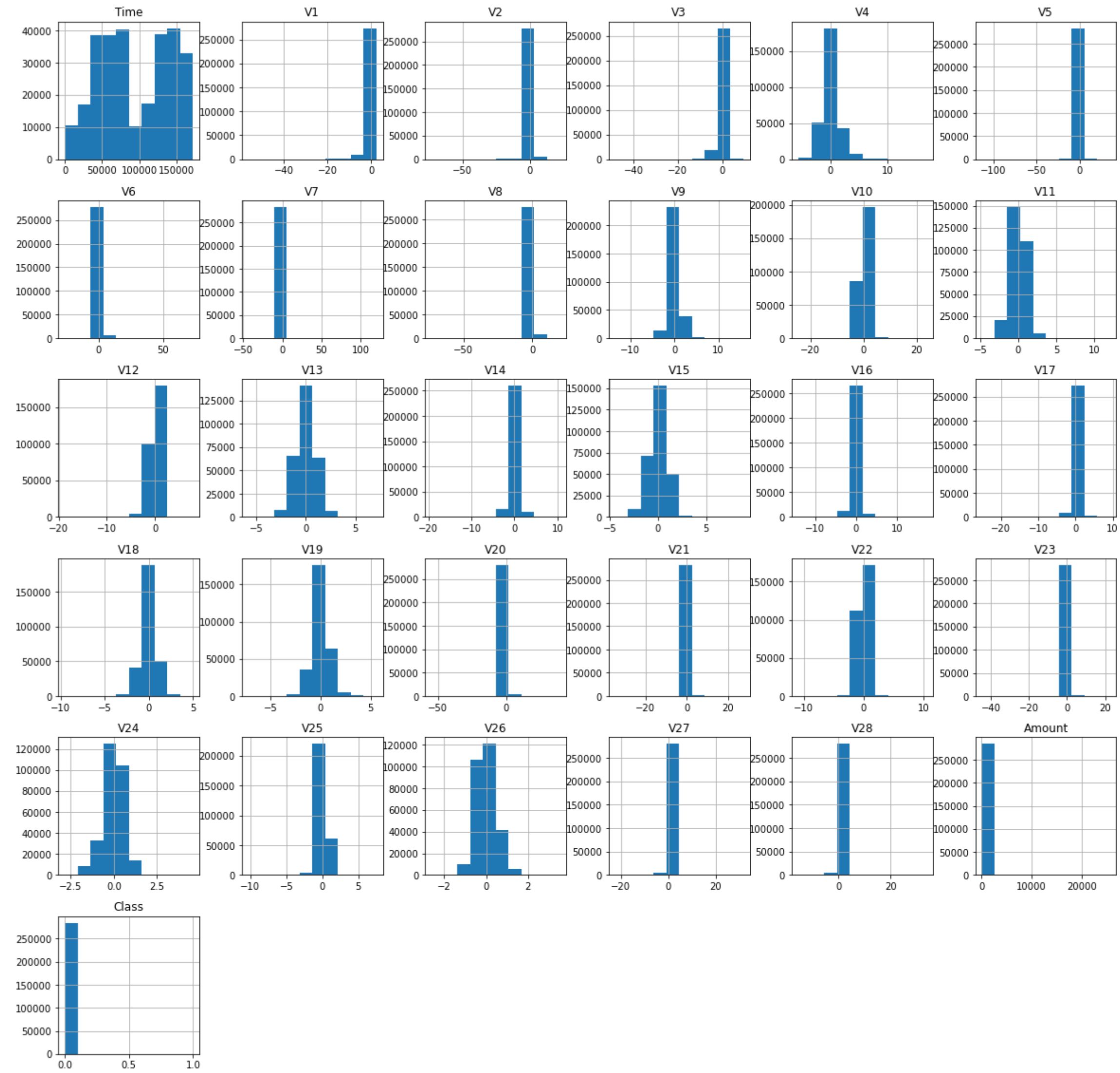
Out[18]:

	Time	V1	V2	V3	V4	V5	V6	V7
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050

5 rows × 31 columns

EDA

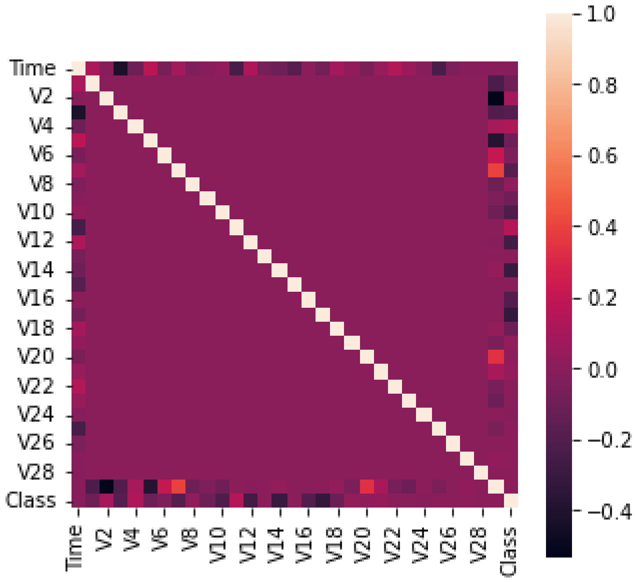
In [19]:
```
# Plot histograms of each parameter
data.hist(figsize = (20, 20))
plt.show()
```



In [20]:
```
# Determine number of fraud cases in dataset

Fraud = data[data['Class'] == 1]
Valid = data[data['Class'] == 0]

outlier_fraction = len(Fraud)/float(len(Valid))
print(outlier_fraction)

print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

```
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
```

In [21]:
```
# Correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (5,5))

sns.heatmap(corrmat,square = True)
plt.show()
```



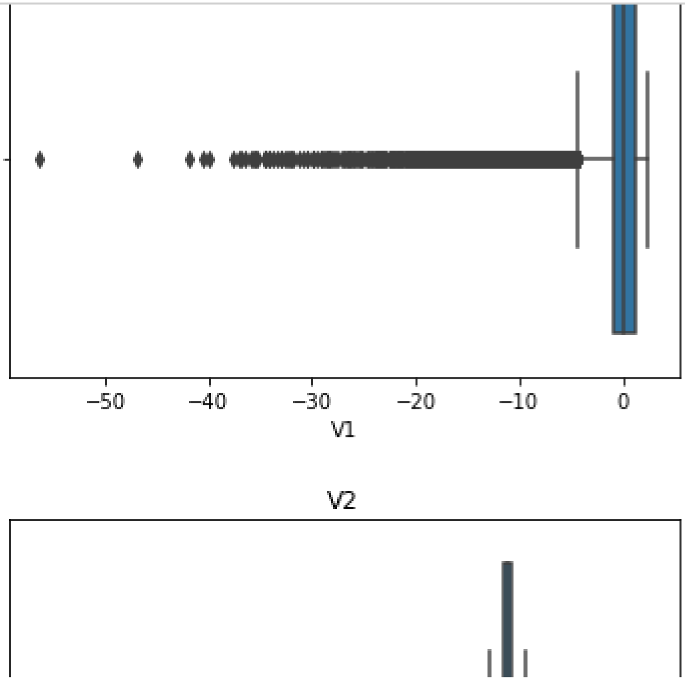
In [22]:
```
# Get all the columns from the dataFrame
columns = data.columns.tolist()

# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]

# Store the variable we'll be predicting on
target = "Class"

X = data[columns]
Y = data[target]

# Print shapes
print(X.shape)
print(Y.shape)
```

```
(284807, 30)
(284807,)
```

Checking outliers

In [23]:
```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

# define random states
state = 1

# define outlier detection tools to be compared
classifiers = {
    "Isolation Forest": IsolationForest(max_samples=len(X),
                                        contamination=outlier_fraction,
                                        random_state=state),
    "Local Outlier Factor": LocalOutlierFactor(
        n_neighbors=20,
        contamination=outlier_fraction)}
```

In [24]:
```
num_cols=data.columns
for i in num_cols:
    sns.boxplot(data[i])
    plt.title(i)
    plt.show()
```



In [ ]:
```

```

In [26]:
```
from sklearn.ensemble import IsolationForest

# Create an Isolation Forest instance
clf = IsolationForest(contamination=0.1)

# Fit the data and tag outliers
clf.fit(X)
scores_pred = clf.decision_function(X)
```

In [27]:
```
!pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\chait\anaconda3\li
b\site-packages (1.3.1)
Requirement already satisfied: scipy>=1.5.0 in c:\users\chait\anaconda3\li
b\site-packages (from scikit-learn) (1.7.3)
Requirement already satisfied: numpy<2.0,>=1.17.3 in c:\users\chait\anacon
da3\lib\site-packages (from scikit-learn) (1.22.4)
Requirement already satisfied: joblib>=1.1.1 in c:\users\chait\anaconda3\l
ib\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\chait\anac
onda3\lib\site-packages (from scikit-learn) (2.2.0)
```

In [28]:
```
new_data['Class'].value_counts()
```

Out[28]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [29]:
```
new_data.groupby('Class').mean()
```

Out[29]:

Class	Time	V1	V2	V3	V4	V5	V6	V7
0	98413.619919	-0.063513	0.149684	-0.068118	0.060168	0.113367	-0.039915	0.049392
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns

Splitting in features and targets

In [30]:
```
X = new_data.drop(columns='Class', axis=1)
```

In [31]:
```
Y = new_data['Class']
```

In [32]:
```
X.head()
```

Out[32]:

	Time	V1	V2	V3	V4	V5	V6	V7
237802	149400.0	2.161543	0.068915	-2.465194	-0.051840	1.251262	-0.251903	0.578921
111700	72331.0	-5.467393	4.520035	-0.956856	-2.375816	-1.511242	-1.500989	0.122696
79415	58017.0	-0.743646	0.723658	1.820321	0.474289	-0.072016	-0.278892	0.639433
269434	163657.0	-0.562317	0.704013	2.144150	-0.338884	-0.275852	-0.460350	0.481065
282038	170620.0	2.022222	-0.224775	-0.944111	0.529741	-0.530831	-1.423159	0.014193

5 rows × 30 columns

In [33]:
```
Y
```

Out[33]:
```
237802    0
111700    0
79415     0
269434    0
282038    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

now will split into train_test_split

In [34]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2, stra
```

In [35]:
```
X.shape, X_train.shape, X_test.shape
```

Out[35]:
```
((984, 30), (787, 30), (197, 30))
```

In [36]:
```
Y.shape, Y_train.shape, Y_test.shape
```

Out[36]:
```
((984,), (787,), (197,))
```

Will train the model

In [37]:
```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Logistic Regression

In [38]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X_train, Y_train)

X_train_prediction = model.predict(X_train)
X_test_prediction = model.predict(X_test)

training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)

test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score on Test Data : ', test_data_accuracy)

all_accuracy = {}
all_accuracy['Logistic']=test_data_accuracy
```

```
Accuracy on Training data :  0.9466327827191868
Accuracy score on Test Data :  0.934010152284264
```

RandomForest Classifier

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()

# training the Randomforestclassifier Model with Training Data

model.fit(X_train, Y_train)
RandomForestClassifier()


X_train_prediction = model.predict(X_train)
X_test_prediction = model.predict(X_test)

training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)


test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score on Test Data : ', test_data_accuracy)

all_accuracy['RandomForest']=test_data_accuracy
```

```
Accuracy on Training data :  1.0
Accuracy score on Test Data :  0.949238578680203
```

SVM(Support Vectore Machine)

In [40]:
```
from sklearn.svm import SVC
model = SVC(kernel="linear")
model.fit(X_train, Y_train)


X_train_prediction = model.predict(X_train)
X_test_prediction = model.predict(X_test)

training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)


test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score on Test Data : ', test_data_accuracy)

all_accuracy['SVM']=test_data_accuracy
```

```
Accuracy on Training data :  0.8919949174078781
Accuracy score on Test Data :  0.9187817258883249
```

Bagging

In [41]:
```
from sklearn.ensemble import BaggingClassifier
model = BaggingClassifier(random_state = 100)
model.fit(X_train, Y_train)

X_train_prediction = model.predict(X_train)
X_test_prediction = model.predict(X_test)

test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print(f'Accuracy score on Test Data  : ', test_data_accuracy)

all_accuracy['Bagging']=test_data_accuracy
```

```
Accuracy score on Test Data  :  0.9441624365482234
```

XGBoost

In [42]:
```
Y_train = Y_train.astype('category').cat.codes
Y_train.value_counts()
Y_train.astype('category')
Y_test = Y_test.astype('category').cat.codes
Y_test.astype('category')
```

Out[42]:
```
157871    1
249239    1
122479    1
27749     1
149942    0
         ..
214548    0
96994     1
163586    1
100850    0
141258    1
Length: 197, dtype: category
Categories (2, int64): [0, 1]
```

In [43]:
```
import xgboost
from xgboost import XGBClassifier
model_Xgboost = XGBClassifier()
model_Xgboost.fit(X_train, Y_train)
y_pred_XGBoost = model_Xgboost.predict(X_test)

print("Accuracy is:",accuracy_score(Y_test,y_pred_XGBoost))

all_accuracy['XGBoost']=test_data_accuracy
```

```
Accuracy is: 0.949238578680203
```

In [44]:
```
Accuracy = pd.DataFrame([all_accuracy]).T
Accuracy = Accuracy.rename(columns={0:'Accuracy'})
```

In [45]:
```
Accuracy
```

Out[45]:

	Accuracy
Logistic	0.934010
RandomForest	0.949239
SVM	0.918782
Bagging	0.944162
XGBoost	0.944162

We are getting the most accuracy score for RandomForest Algorithm i.e. 94.923%

In [ ]:

In [ ]: