Project Overview: Credit Card Fraud Prediction

Credit card fraud is a significant concern in the financial industry due to its potential to cause substantial losses for both cardholders and institutions. Fraudulent credit card transactions refer to unauthorized usage of someone's card or card information to make purchases or withdraw cash. Identifying such transactions promptly is crucial to prevent customers from being unfairly charged and to minimize financial damage. In this project, we aim to build a classification model that can accurately predict whether a credit card transaction is fraudulent.

The dataset used consists of transactions made by European cardholders in September 2013, spanning two days. Out of 284,807 transactions, only 492 are fraudulent, making the dataset highly imbalanced, with frauds accounting for just 0.172% of all transactions. This imbalance poses a challenge, as traditional machine learning algorithms may struggle to detect the minority class. The goal is to build a robust classification model to predict fraud, considering steps such as exploratory data analysis, data cleaning, handling class imbalance, feature engineering, model selection, and deployment.

1. **Importing required libraries:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

2.**Loading and inspecting the dataset:**

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

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25
0	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
1	0.0	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
2	1.0	-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
3	1.0	-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
4	2.0	-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

5 rows × 31 columns

```python
df.Class.unique()
```

```
array([0, 1], dtype=int64)
```

```python
df.Class.value_counts()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

```python
df.shape
```

```
(284807, 31)
```

```python
df.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
```

```
df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01

8 rows × 31 columns

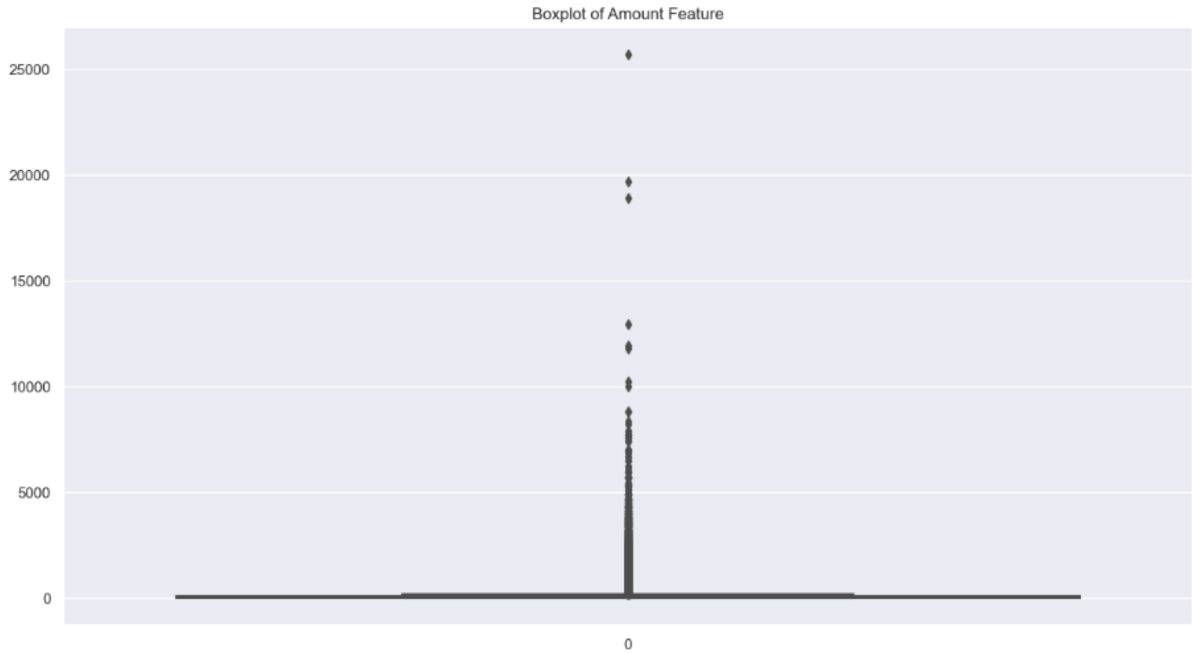
3.**Checking for Missing Values**

```
# Check for missing values
print(df.isnull().sum())

# Display the percentage of missing values for each column
print((df.isnull().sum() / len(df)) * 100)
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

4. **Outliers Detection**

```
sns.boxplot(df['Amount'])
plt.title('Boxplot of Amount Feature')
plt.show()
```



5. **Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['Time']
df[columns_to_scale] = standardScaler.fit_transform(df[columns_to_scale])
```

```
df
```

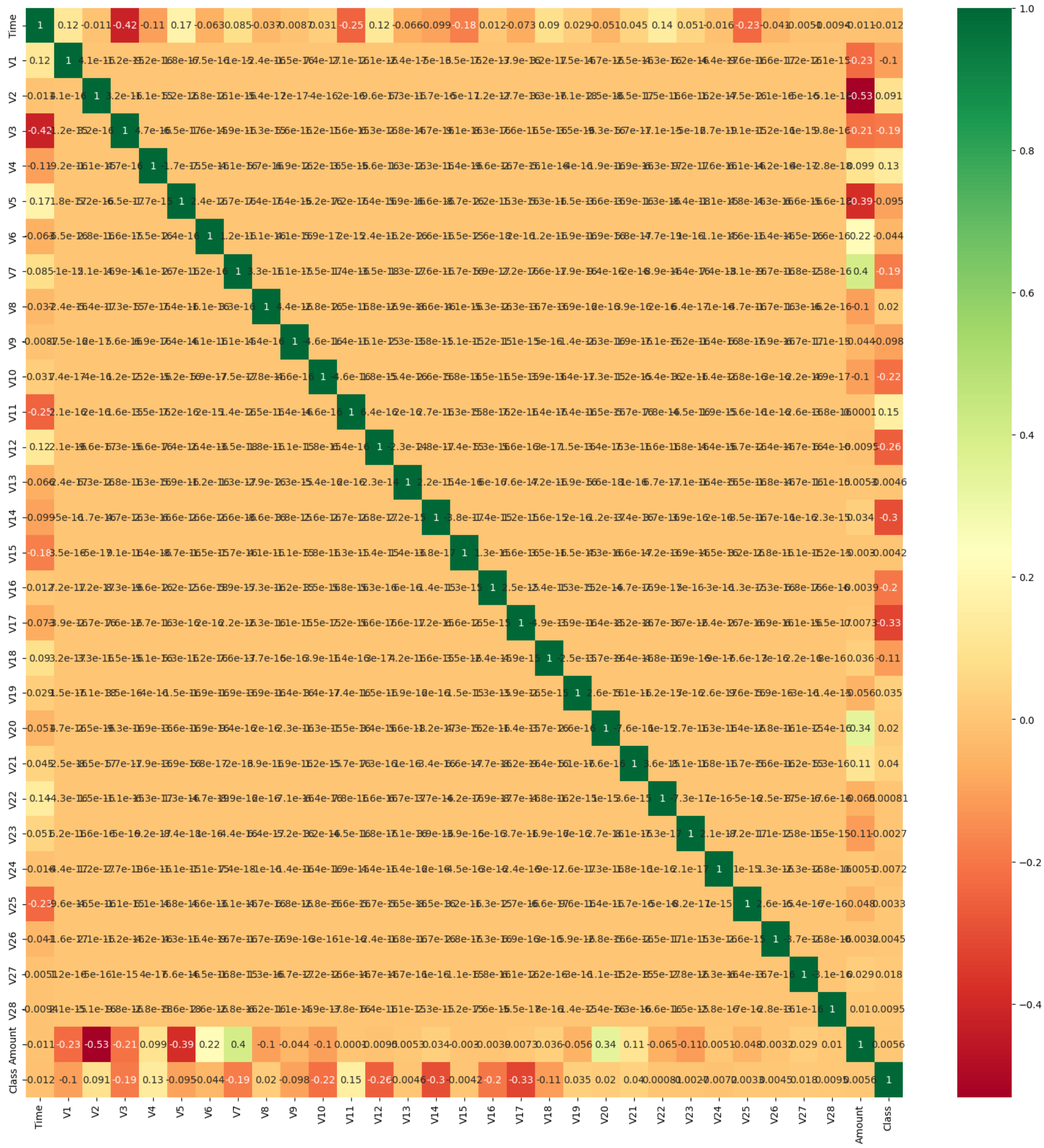
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	-1.996583	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	-1.996583	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	-1.996562	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	-1.996562	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	-1.996541	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	1.641931	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480
284803	1.641952	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463
284804	1.641974	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501
284805	1.641974	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298
284806	1.642058	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777

284807 rows × 31 columns

6. **Exploratory Data Analysis (EDA)**

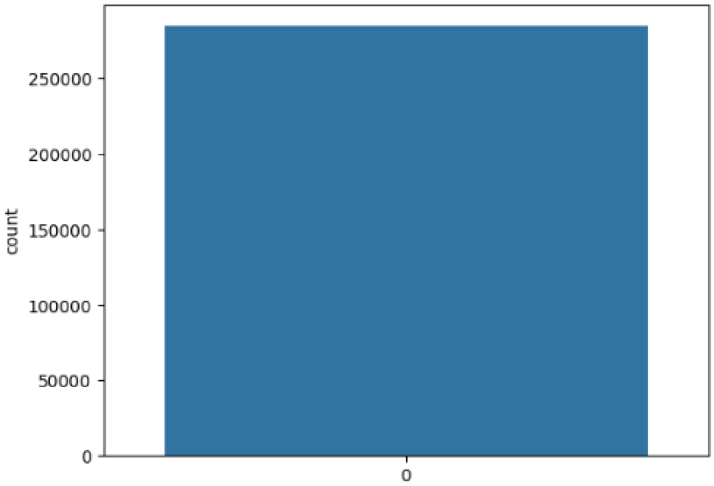
```
cor = df.corr()
top_cor_features = cor.index
plt.figure(figsize=(20,20))
#plot heat map
sns.heatmap(df[top_cor_features].corr(),annot=True,cmap="RdYlGn")
```

```
<AxesSubplot:>
```



```
y = df["Class"]

sns.countplot(y)


target_temp = df.Class.value_counts()

print(target_temp)
```

```
0    284315
1       492
Name: Class, dtype: int64
```



7.**Balancing the Data**

```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\hp\appdata\local\programs\python\python37\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\hp\appdata\local\programs\python\python37\lib\site-packages (from i
mblearn) (0.6.2)
Requirement already satisfied: numpy>=1.11 in c:\users\hp\appdata\local\programs\python\python37\lib\site-packages (from imbala
nced-learn->imblearn) (1.21.6)
Requirement already satisfied: scipy>=0.17 in c:\users\hp\appdata\local\programs\python\python37\lib\site-packages (from imbala
nced-learn->imblearn) (1.7.3)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\hp\appdata\local\programs\python\python37\lib\site-packages (from
imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: joblib>=0.11 in c:\users\hp\appdata\local\programs\python\python37\lib\site-packages (from imbal
anced-learn->imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\appdata\local\programs\python\python37\lib\site-packages (fr
om scikit-learn>=0.22->imbalanced-learn->imblearn) (3.1.0)
```

```
#independent variable and dependent variable
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

```
from imblearn.over_sampling import RandomOverSampler

nm = RandomOverSampler()
X_res,y_res=nm.fit_resample(X,y)

X_res.shape,y_res.shape
```

```
((568630, 30), (568630,))
```

8.**Train/Test Split**

```
from sklearn.model_selection import train_test_split
#from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
X_train , X_test , Y_train , Y_test = train_test_split(X_res,y_res,test_size=0.20,random_state=100)
```

```
X_train.shape , X_test.shape , Y_train.shape , Y_test.shape
```

```
((454904, 30), (113726, 30), (454904,), (113726,))
```

```
print(Y_test)
```

```
309550    1
418331    1
313225    1
354977    1
43690     0
         ..
481040    1
92883     0
269959    0
276875    0
497148    1
Name: Class, Length: 113726, dtype: int64
```

8.**Model Selection, Training, and Evaluation**

Description of Design Choices

In developing the fraud detection model for our credit card company, several key design choices were made:

- **Model Selection**: We chose a combination of models, including Random Forest, Logistic Regression, and Decision Trees, based on their ability to handle classification tasks effectively, especially with imbalanced datasets. Random Forest was prioritized due to its robustness and effectiveness in detecting fraud patterns.
- **Handling Imbalanced Data**: Given that fraudulent transactions represent a very small percentage of total transactions, techniques like Random Oversampling were employed to balance the dataset. This choice was crucial to ensure that the model could learn from both classes effectively without bias.
- **Feature Engineering**: Features such as transaction amount, time of day, and user behavior were carefully selected. We standardized certain features (like Time) to improve model performance.
- **Model Evaluation Metrics**: Metrics such as accuracy, precision, recall, and F1-score were chosen to evaluate the models. Given the imbalanced nature of the dataset, precision and recall were emphasized to ensure that we minimize false negatives (missed fraud cases).

```
#Naive Baiyes
```

```
from sklearn.naive_bayes import GaussianNB
clf_NB = GaussianNB()
clf_NB.fit(X_train, Y_train)
```

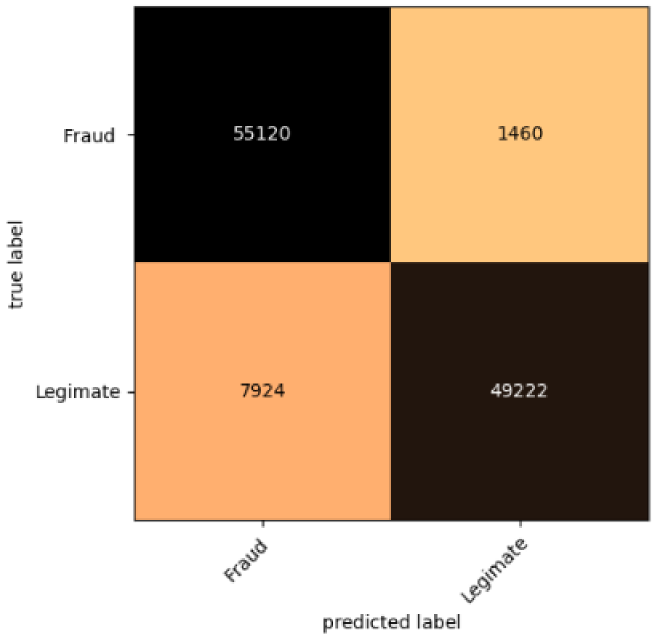
```
▾ GaussianNB
GaussianNB()
```

```
y_pred_NB = clf_NB.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_pred_NB,Y_test))
score_nb = accuracy_score(y_pred_NB,Y_test)*100
```

```
0.9174858871322301
```

```python
# Confusion matrix.
from sklearn import metrics
from mlxtend.plotting import plot_confusion_matrix
conf_mat = metrics.confusion_matrix(Y_test,y_pred_NB)
plot_confusion_matrix(conf_mat,class_names=["Fraud ","Legimate"],figsize=(12,5),cmap='copper_r');
plt.savefig("NB_con.png")
```



```python
#ranmdon forest
```

```python
from sklearn.ensemble import RandomForestClassifier
```

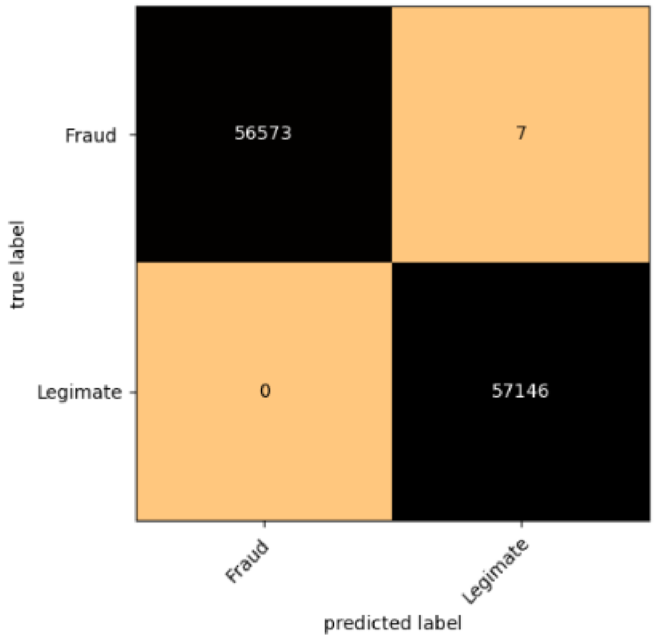
```python
clf=RandomForestClassifier(n_estimators=15)
```

```python
clf.fit(X_train,Y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=15)
```

```python
y_pred=clf.predict(X_test)
import pickle
with open('rf.pkl', 'wb') as file:
   pickle.dump(clf,file)
```

```python
# Confusion matrix.
from sklearn import metrics
from mlxtend.plotting import plot_confusion_matrix
conf_mat = metrics.confusion_matrix(Y_test,y_pred)
plot_confusion_matrix(conf_mat,class_names=["Fraud ","Legimate"],figsize=(12,5),cmap='copper_r');
plt.savefig("NB_con.png")
```




```python
score_rf = round(accuracy_score(y_pred,Y_test)*100,2)

print("The accuracy score achieved using Random Forest is: "+str(score_rf)+" %")
```

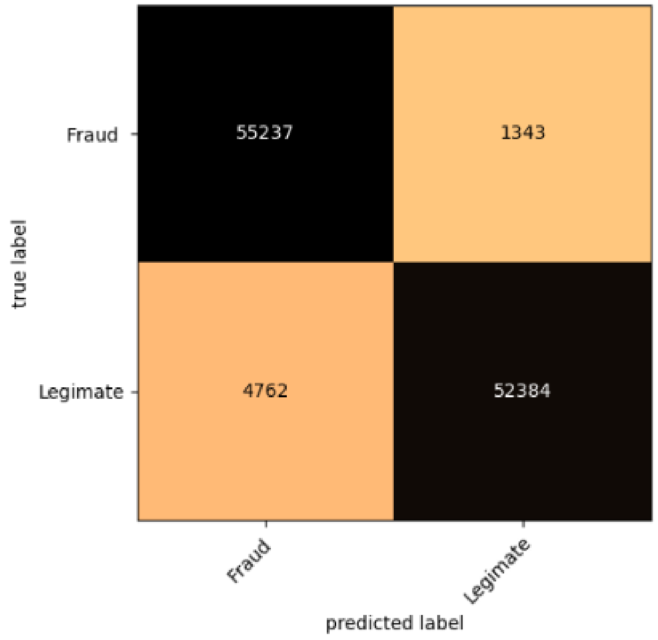
```
The accuracy score achieved using Random Forest is: 99.99 %
```

```python
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,Y_train)

#
y_pred=logreg.predict(X_test)
```

```python
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
score_lr = metrics.accuracy_score(Y_test, y_pred) * 100
import pickle
with open('ll.pkl', 'wb') as file:
   pickle.dump(logreg,file)
```

```
Accuracy: 0.9463183440901817
```

```python
# Confusion matrix.
from sklearn import metrics
from mlxtend.plotting import plot_confusion_matrix
conf_mat = metrics.confusion_matrix(Y_test,y_pred)
plot_confusion_matrix(conf_mat,class_names=["Fraud ","Legimate"],figsize=(12,5),cmap='copper_r');
plt.savefig("LR_con.png")
```




```python
#desicion tree
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

```python
clf_entropy = DecisionTreeClassifier(criterion="entropy",random_state=100,max_depth=5,min_samples_leaf=3)
clf_entropy.fit(X_train,Y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=3,
                       random_state=100)
```

```python
y_pred_en=clf_entropy.predict(X_test)
y_pred_en
score_dt = accuracy_score(Y_test,y_pred_en)*100
print("The accuracy score achieved using Decision Tree is: "+str(score_dt)+" %")
import pickle
with open('dt.pkl', 'wb') as file:
   pickle.dump(clf_entropy,file)
```

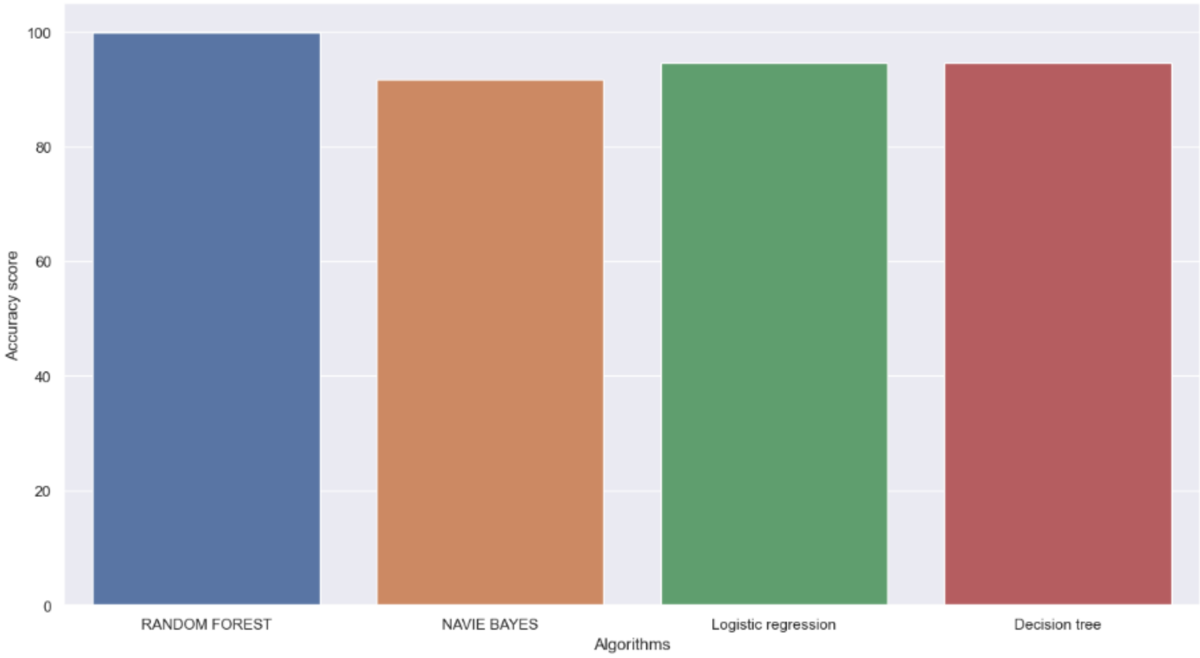
```
The accuracy score achieved using Decision Tree is: 94.57555879921917 %
```

9.**Model Comparison**

```
scores = [score_rf,score_nb,score_lr,score_dt]
algorithms = ["RANDOM FOREST","NAVIE BAYES","Logistic regression", "Decision tree"]

for i in range(len(algorithms)):
    print("The accuracy score achieved using "+algorithms[i]+" is: "+str(scores[i])+" %")
```

```
The accuracy score achieved using RANDOM FOREST is: 99.99 %
The accuracy score achieved using NAVIE BAYES is: 91.74858871322301 %
The accuracy score achieved using Logistic regression is: 94.63183440901817 %
The accuracy score achieved using Decision tree is: 94.57555879921917 %
```

```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(x=algorithms,y=scores)
```



10. **Model Deployment Plan**

Once the model is trained and evaluated, we need to make it available for use. One way to do this is by deploying the model on **AWS** (Amazon Web Services) using **SageMaker**. This allows us to create an API endpoint where the model can make predictions on new data.

Steps for Deployment on AWS SageMaker:

1. **Upload the Model to AWS**: First, the trained model (saved as a .pkl file) is uploaded to **AWS S3**, which is a storage service. This ensures the model is accessible for deployment.
2. **Create a SageMaker Endpoint**: After uploading the model, we can use **AWS SageMaker** to create an endpoint. This endpoint will listen for requests, and whenever new transaction data is sent, it will predict whether the transaction is fraudulent or legitimate.

This plan outlines how the model would be deployed, making it usable for real-time fraud detection in the future.

How the Fraud Detection Model Will Benefit Our Credit Card Company

I've been working on this fraud detection project for our credit card company, and I really believe it's going to make a big difference for us. Here's how:

1. **Less Financial Loss**:
   - The main goal here is to stop fraud before it happens. With this model, we can catch suspicious transactions quickly, which means less money lost to fraud.
   - For example, if the model flags a transaction that seems out of the ordinary—like a huge purchase from a different country—we can investigate before any money actually leaves our customers' accounts.
2. **Building Customer Trust**:
   - When customers know we have a reliable system to protect them from fraud, they feel safer using our credit cards. This builds trust and loyalty.
   - Imagine a customer getting an alert that their card was used for a suspicious transaction. If we can act fast and resolve the issue, that customer will appreciate our quick response and feel more secure with us.
3. **Making Our Team More Efficient**:
   - With the model doing the heavy lifting, our team can focus on the more complex cases instead of going through every single transaction manually.
   - This means our fraud analysts can spend their time investigating only the transactions that really need attention, making our operations smoother and more efficient.
4. **Data-Driven Decisions**:
   - The insights we get from the model will help us make better decisions about managing risks and preventing fraud in the future.
   - For instance, if we see patterns in the types of transactions that are often flagged, we can adjust our strategies and target areas that need more attention.
5. **Easily Scalable**:
   - As our customer base grows, the model can grow with us. This means we won't have to completely revamp our system every time we get more transactions.
   - By deploying it on a cloud platform like AWS, we can ensure it handles increased transaction volumes without skipping a beat.
6. **Staying Compliant**:
   - Having a solid fraud detection system helps us meet regulations and keep our customers' data safe, which is crucial in our industry.
   - Plus, we can show regulators that we're taking proactive steps to prevent fraud, which keeps us on the right side of the law.
7. **Gaining a Competitive Edge**:

- With this advanced fraud detection system, we can set ourselves apart from competitors.
- We can market our strong fraud protection as a key feature of our credit cards, attracting customers who are concerned about security.

Discussion of Future Work:

1. **Incorporating More Features**: Additional features, such as customer transaction history or geographic location, could be explored to improve model performance. This may help capture more subtle patterns associated with fraudulent behavior.

2. **Real-Time Monitoring and Retraining**: Implementing a system for real-time monitoring of model performance will be essential. As new transaction data becomes available, we will set up processes for regular retraining of the model to adapt to evolving fraud patterns.

3. **Deployment Enhancements**: Once deployed, we'll consider integrating the fraud detection system with our existing transaction processing systems to ensure seamless real-time predictions.