Experiment No. 6
Apply XGBOOST for credit fraud detection
Date of Performance:
Date of Submission:

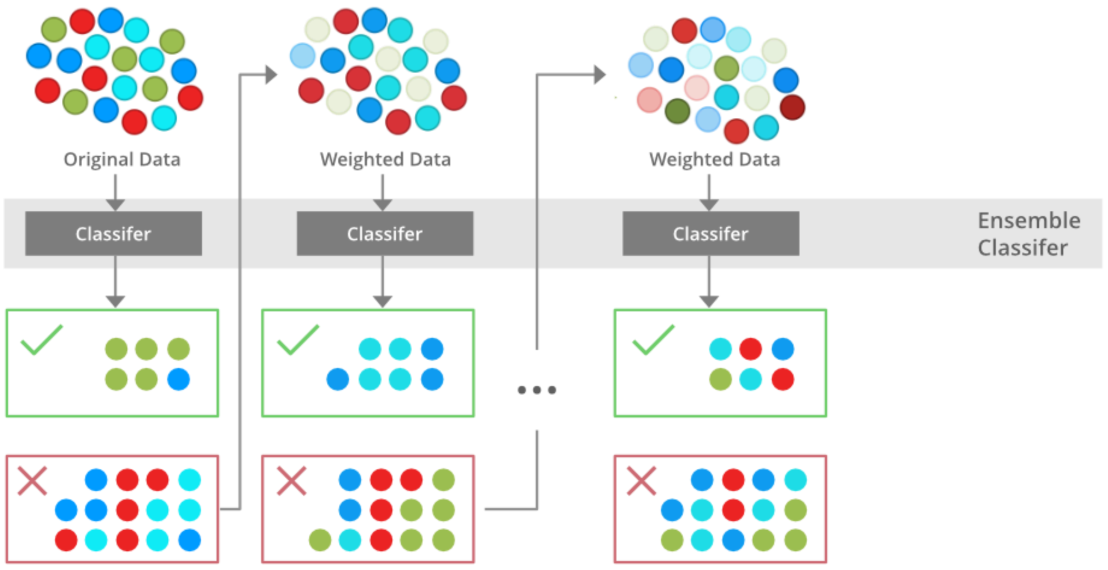
Aim: Apply XGBOOST for credit fraud detection.

Objective: Ability to implement ensemble learning algorithms.

Theory:

Boosting:

Boosting is an ensemble modelling, technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

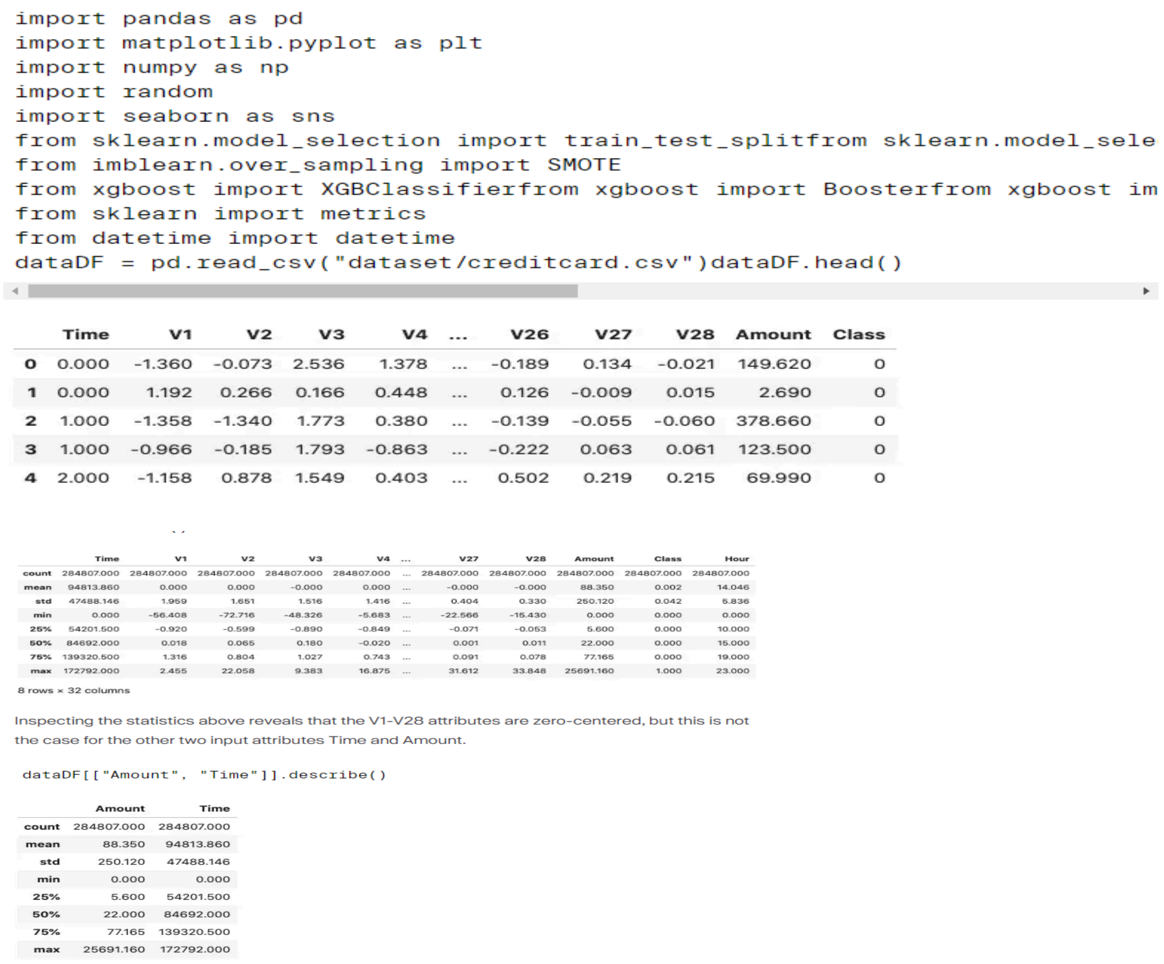


XGBoost:

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for "Extreme Gradient Boosting" and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression.

One of the key features of XGBoost is its efficient handling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing. Additionally, XGBoost has built-in support for parallel processing, making it possible to train models on large datasets in a reasonable amount of time.

Results:



```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import random
import seaborn as sns
from sklearn.model_selection import train_test_splitfrom sklearn.model_sele
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifierfrom xgboost import Boosterfrom xgboost im
from sklearn import metrics
from datetime import datetime
dataDF = pd.read_csv("dataset/creditcard.csv")dataDF.head()
```

	Time	V1	V2	V3	V4	...	V26	V27	V28	Amount	Class
0	0.000	-1.360	-0.073	2.536	1.378	...	-0.189	0.134	-0.021	149.620	0
1	0.000	1.192	0.266	0.166	0.448	...	0.126	-0.009	0.015	2.690	0
2	1.000	-1.358	-1.340	1.773	0.380	...	-0.139	-0.055	-0.060	378.660	0
3	1.000	-0.966	-0.185	1.793	-0.863	...	-0.222	0.063	0.061	123.500	0
4	2.000	-1.158	0.878	1.549	0.403	...	0.502	0.219	0.215	69.990	0

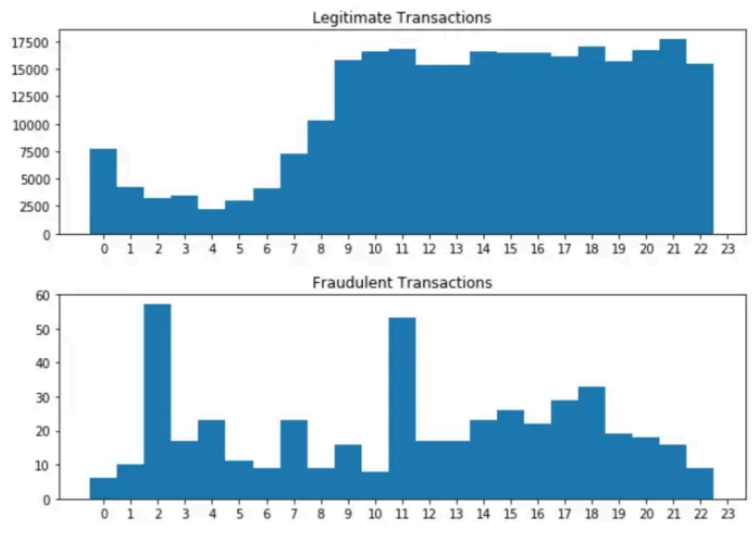
	Time	V1	V2	V3	V4	...	V27	V28	Amount	Class	Hour
count	284807.000	284807.000	284807.000	284807.000	284807.000	...	284807.000	284807.000	284807.000	284807.000	284807.000
mean	94813.860	0.000	0.000	-0.000	0.000	...	-0.000	-0.000	88.350	0.002	14.046
std	47488.146	1.959	1.651	1.516	1.416	...	0.404	0.330	250.120	0.042	5.836
min	0.000	-56.408	-72.716	-48.326	-5.683	...	-22.566	-15.430	0.000	0.000	0.000
25%	54201.500	-0.920	-0.599	-0.890	-0.849	...	-0.071	-0.053	5.600	0.000	10.000
50%	84692.000	0.018	0.065	0.180	-0.020	...	0.001	0.011	22.000	0.000	15.000
75%	139320.500	1.316	0.804	1.027	0.743	...	0.091	0.078	77.165	0.000	19.000
max	172792.000	2.455	22.058	9.383	16.875	...	31.612	33.848	25691.160	1.000	23.000

8 rows × 32 columns

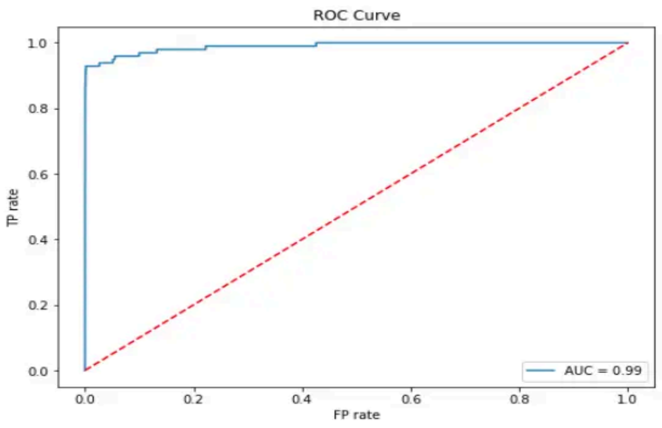
Inspecting the statistics above reveals that the V1-V28 attributes are zero-centered, but this is not the case for the other two input attributes Time and Amount.

```
dataDF[["Amount", "Time"]].describe()
```

	Amount	Time
count	284807.000	284807.000
mean	88.350	94813.860
std	250.120	47488.146
min	0.000	0.000
25%	5.600	54201.500
50%	22.000	84692.000
75%	77.165	139320.500
max	25691.160	172792.000



```
y_pred = model.predict_proba(X_test)[:,1]
fp_r, tp_r, t = metrics.roc_curve(y_test, y_pred)
auc = metrics.auc(fp_r, tp_r)
plt.figure(figsize=(8, 6))
plt.plot(fp_r, tp_r, label="AUC = %.2f" % auc)plt.plot([0,1],[0,1],"r--")
plt.ylabel("TP rate")plt.xlabel("FP rate")
plt.legend(loc=4)plt.title("ROC Curve")plt.show()
```



Conclusion:

In this blog post we looked at a real-world credit card transaction dataset and demonstrated how machine learning can be used to automate the detection of fraudulent transactions. The model can be automatically updated as new data comes in, and the re-training process doesn't require human intervention.The imbalance in the dataset can be addressed by using under/oversampling techniques, and the interpretation of the probabilities can be fine-tuned to produce a better balance between false alarms and missed fraudulent transactions.