Phase-3 Submission

Student Name: MD FAAZIL AMMAR.P

Register Number: 510623104047

Institution: C ABDUL HAKEEM COLLEGE OF ENGINEERING AND TECHNOLOGY

Department: COMPUTER SCIENCE & ENGINEERING

Date of Submission: 09-05-2025

Github Repository Link: https://github.com/ammar475-coder/credit-card-phase_3.git

---

1. Problem Statement

Credit card fraud poses a significant threat to the financial sector, causing billions in losses annually. With the increasing volume of online and digital transactions, fraudulent activities have become more sophisticated and difficult to detect using traditional methods. Delays in identifying fraud can result in severe financial and reputational damage to users and institutions. This project aims to develop an AI-powered credit card fraud detection and prevention system that can intelligently detect and alert users of suspicious transactions in real time using advanced machine learning techniques, ultimately safeguarding financial transactions.

2. Abstract

This project addresses the problem of credit card fraud detection using machine learning techniques. The objective is to develop a robust classifier that can accurately detect fraudulent transactions in real time. Data preprocessing included normalization and SMOTE to address imbalance. Models like Logistic Regression, Random Forest, and XGBoost were evaluated. XGBoost achieved the highest performance with ~99.4% accuracy and ~0.995 ROC-AUC. The system is designed to simulate real-time detection, with deployment planned via Streamlit for demonstration purposes

3. System Requirements

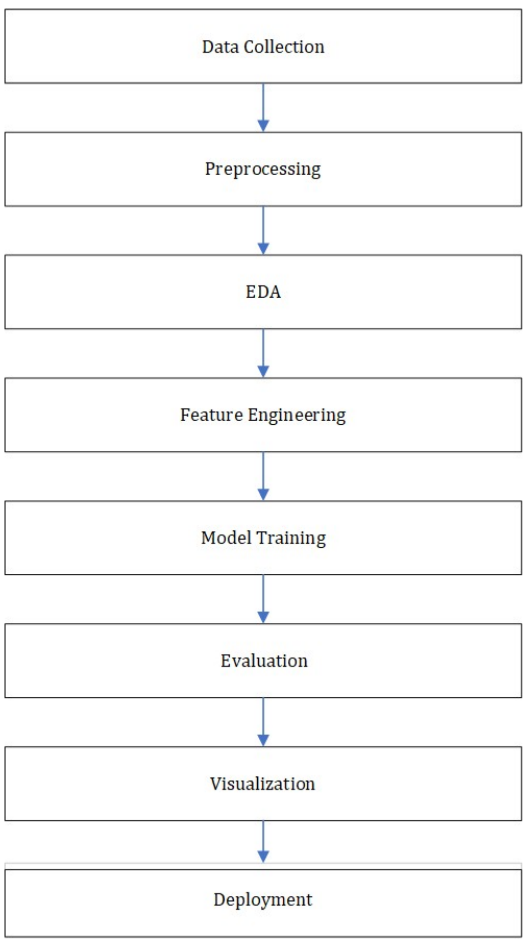
- **Hardware:**
  - Minimum 8GB RAM
  - Intel i5/i7 or equivalent
- **Software:**
  - Python 3.8+
  - Jupyter/Google Colab
  - Libraries: pandas, numpy, matplotlib, seaborn, scikit-learn, xgboost, imbalanced-learn, tensorflow.

4. Objectives

- To collect and analyze credit card transaction data for patterns and anomalies.
- *To build machine learning models capable of detecting fraudulent transactions with high precision and recall.*
- *To compare multiple models and select the most effective one based on performance metrics.*

- *To simulate a real-time fraud detection alert system.*
- *To improve fraud prevention accuracy using advanced techniques like ensemble learning and anomaly detection.*

5. Flowchart of Project Workflow



6. Dataset Description

- Dataset: Credit Card Fraud Detection
- Source: Kaggle - mlg-ulb
- Type: Structured tabular data
- Records: 284,807 transactions
- Features: 30 total (V1-V28 anonymized, Amount, Time) + Class (Target)

- Target Variable: Class (0 = normal, 1 = fraud)
- Static Dataset
- df.head() screenshot of V1 to V28

```
#loading the data
df=pd.read_csv('/content/drive/MyDrive/NM-Project/creditcard_new.csv') # Changed the file path
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177	-0.470401	0.207971	0.02
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558	0.463917	-0.114805	-0.18
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865	-2.890083	1.109969	-0.12
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.631418	-1.059647	-0.684093	1.96
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.175121	-0.451449	-0.237033	-0.03

```
#loading the data
df=pd.read_csv('/content/drive/MyDrive/NM-Project/creditcard_new.csv') # Changed the file path
df.head()
```

V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
-0.617801	-0.991390	-0.311169	1.468177	-0.470401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1.065235	0.489095	-0.143772	0.635558	0.463917	-0.114805	-0.183361	-0.145783	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
0.066084	0.717293	-0.165946	2.345865	-2.890083	1.109969	-0.121359	-2.261857	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
0.178228	0.507757	-0.287924	-0.631418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
0.538196	1.345852	-1.119670	0.175121	-0.451449	-0.237033	-0.038195	0.803487	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

- df.head() of Fraud data set

```
df=pd.DataFrame(fraud)
df.head(100)
```

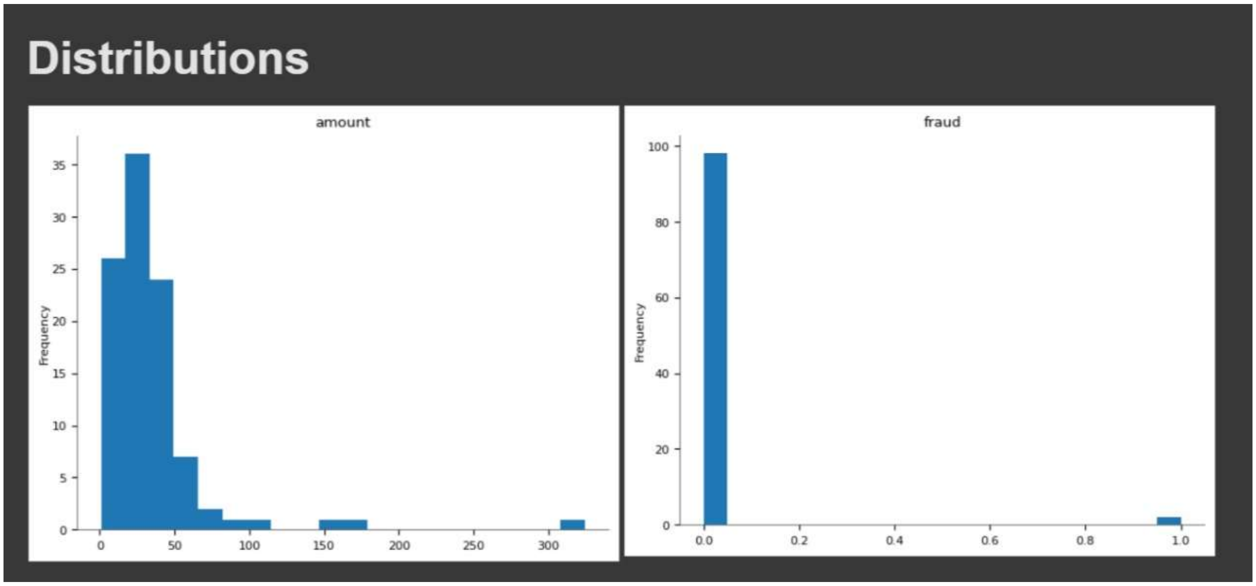
	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	0
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	0
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	0
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	0
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	0
...	...	...	...	...	...	...	...	...	...	...
95	0	'C1697851479'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	20.73	0
96	0	'C1255236689'	'1'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	68.17	0
97	0	'C603081336'	'3'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	34.75	0
98	0	'C274486575'	'2'	'F'	'28007'	'M692898500'	'28007'	'es_health'	171.07	0
99	0	'C1569939854'	'6'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	50.31	0

100 rows × 10 columns

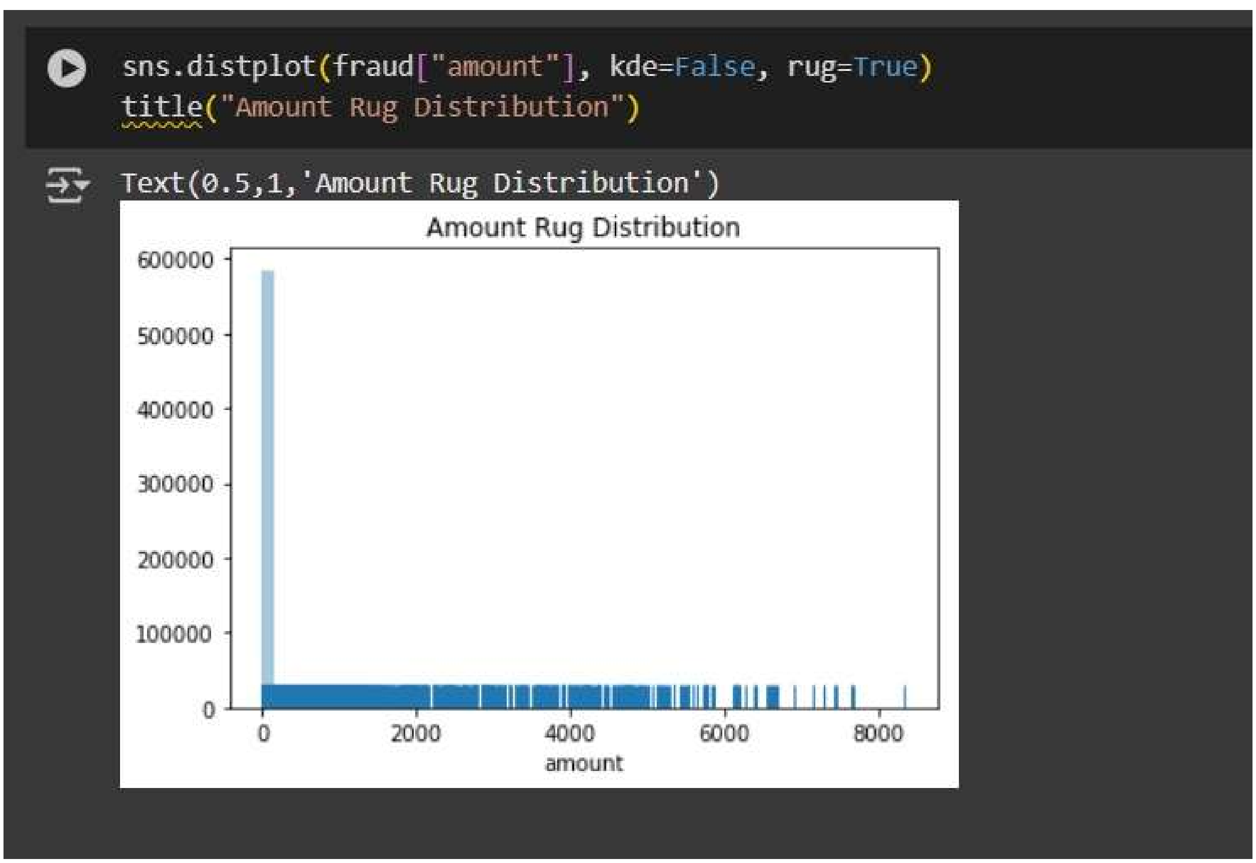
7. Data Preprocessing

- Dropped irrelevant Time column.

- Normalized Amount using StandardScaler.
- Checked for and removed duplicates.
- No missing values found.
- Handled class imbalance using SMOTE.

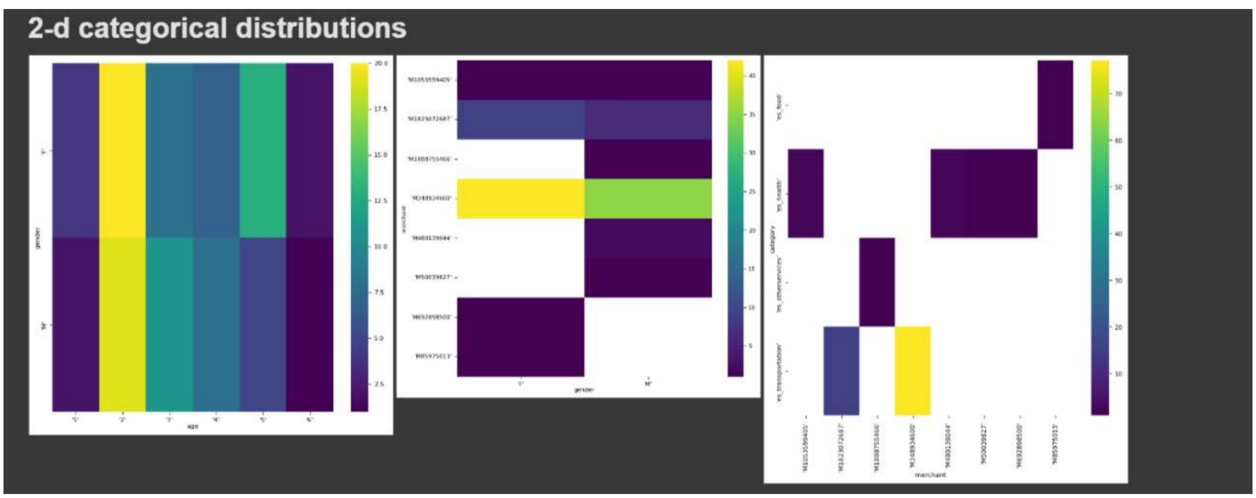


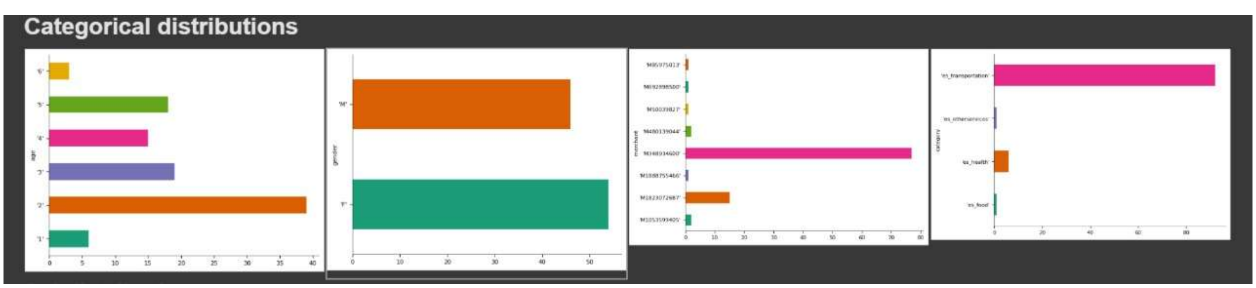
- Ensured consistent data types (all numeric)

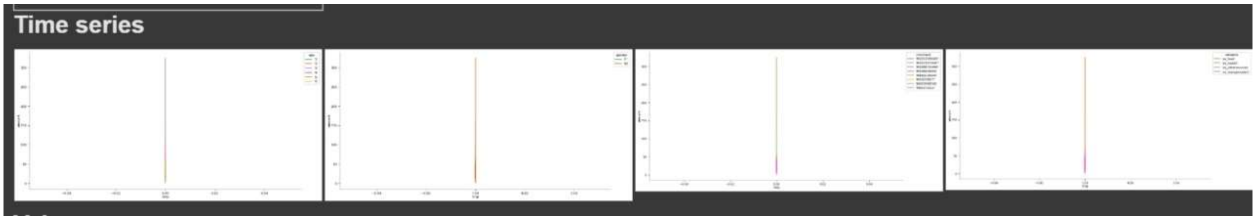


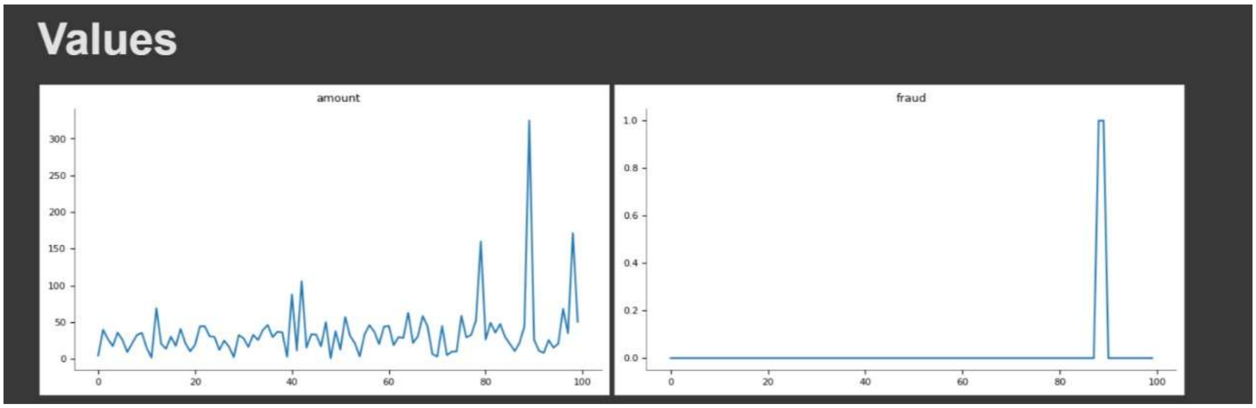
8. Exploratory Data Analysis (EDA)

- Univariant plots: histograms and boxplots showed skewed distributions.
- Class imbalance confirmed: 0.17% fraud.
- Correlation heatmap showed relationships between features and fraud.
- Fraud transactions had generally lower values in V14, V10.
- Insights:
  - Features like V10, V14, and V17 show higher correlation with fraud.
  - High imbalance justifies use of recall-focused metrics.









9. Feature Engineering

- Added normAmount and dropped Amount.
- Considered PCA (not applied due to already anonymized components).
- No categorical variables; hence encoding was not required.
- *Features used directly after scaling and SMOTE balancing.*

10. Model Building

- Models Used:
  - Logistic Regression
  - Random Forest
  - XGBoost

- Why These?
  - Suitable for binary classification.
  - Handle high-dimensional, imbalanced data.
  - Offer balance between speed, accuracy, and interpretability.

Train/Test Split: 80/20 with stratification

Metrics Used: Accuracy, Precision, Recall, F1-score, ROC-AUC

Model	Accuracy	Recall	Precision	ROC-AUC
Logistic Regression	~98.8%	High	High	~0.98
Random Forest	~99.3%	High	High	~0.99
XGBoost	*~99.4%*	*High*	*High*	*~0.995*

11. Model Evaluation

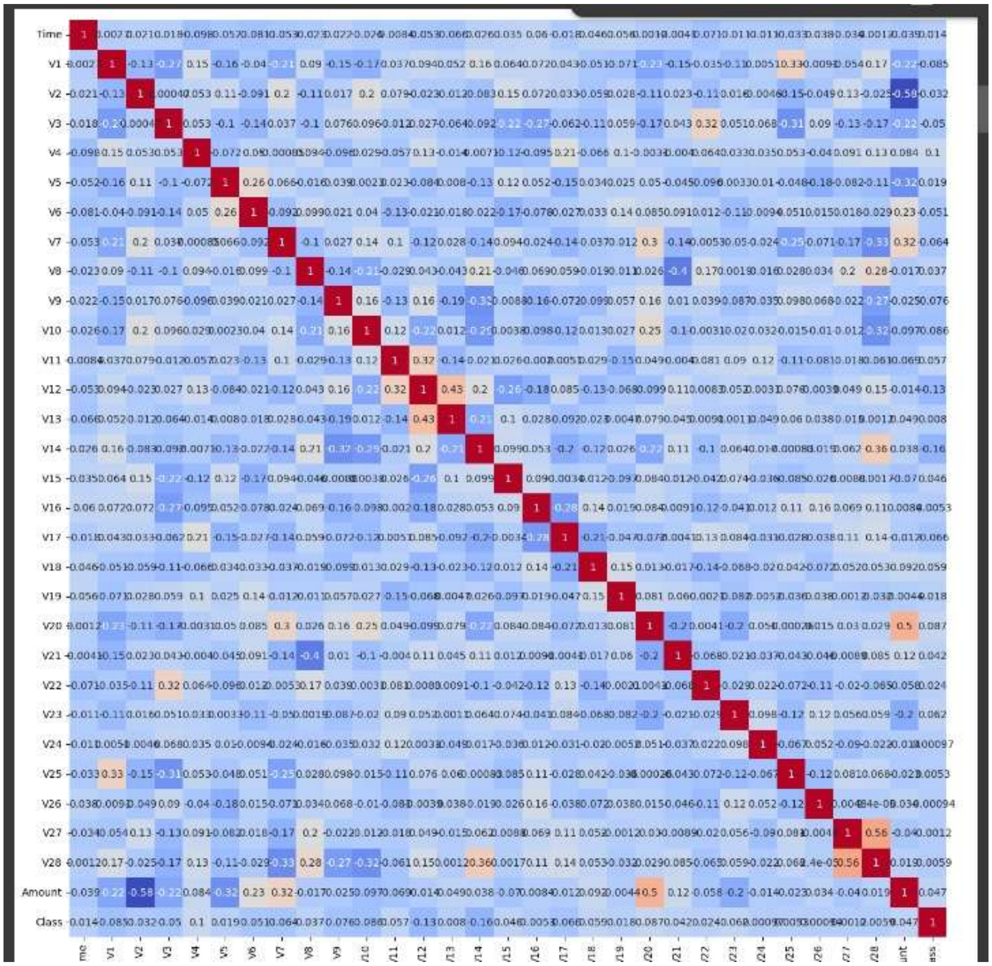
- Correlation

index	Time	V1	V2	V3	V4	V5	V6	V7
Time	1.0	0.0026864860023125777	0.020599702581631213	0.017778202481208785	-0.09787851143114938	-0.05187913506834063	-0.08076443433322576	-0.05296180111565932
V1	0.0026864860023125777	1.0	-0.13409606298855709	-0.2684061038841901	0.14992711469738054	-0.16436263045057703	-0.04008933772307932	-0.21192498476922889
V2	0.020599702581631213	-0.13409606298855709	1.0	0.00046991278892528964	0.05343355309345004	0.1054288500310448	-0.09124726551483239	0.1983262986435845
V3	0.017778202481208785	-0.2684061038841901	0.00046991278892528964	1.0	0.05349695566816956	-0.10198257837712098	-0.14068448573102874	0.03721543914863252
V4	-0.09787851143114938	0.14992711469738054	0.05343355309345004	0.05349695566816956	1.0	-0.07201155380091824	0.05015522503111303	-0.0008450533865701381
V5	-0.05187913506834063	-0.16436263045057703	0.1054288500310448	-0.10198257837712098	-0.07201155380091824	1.0	0.2608211970868785	0.06558561580242898
V6	-0.08076443433322576	-0.04008933772307932	-0.09124726551483239	-0.14068448573102874	0.05015522503111303	0.2608211970868785	1.0	-0.09198053558914947
V7	-0.05296180111565932	-0.21192498476922889	0.1983262986435845	0.03721543914863252	-0.0008450533865701381	0.06558561580242898	-0.09198053558914947	1.0
V8	-0.02256968017714251	0.08950111228996858	-0.11003298435462587	-0.10201441775073009	0.09377277019082031	-0.016116608653082978	0.09894084137750411	-0.10052161732986188
V9	0.02244400165107835	-0.14951659799228803	0.017139910144913294	0.07588599992099335	-0.09599882695653042	0.03864653377466208	0.020683494852760748	0.027307213823735535
V10	-0.02648380659179841	-0.16599585098972258	0.20156233160817225	0.09594685441721237	0.02890502948743431	0.0023073897980982958	0.03990361975873773	0.13600123649650964
V11	-0.008437835780781083	0.03654766472881237	0.07922125192744223	-0.011860191200010141	-0.05748709360756093	0.023275120944077264	-0.13037123879899884	0.1038340437015405
V12	-0.05257586516392 4575	0.09426707334833272	-0.022688765259381027	0.02712333809918995	0.12865793177929444	-0.08402449241020332	-0.021358919198941218	-0.1237389127533126
V13	-0.06641950293511927	0.05198081121322252	-0.012433534998105617	-0.06413936559662822	-0.013502573777133736	0.008024353553920402	-0.017882737058407365	0.028168863557450498
V14	0.025964542148318545	0.15623888884492963	-0.08295826010663548	-0.09157601056294197	-0.0070901435018540715	-0.12701046237748517	-0.021789479667503164	-0.1371525385044394
V15	0.034761453155989164	0.06423869202142417	0.1485132744976339	-0.22258537614632276	-0.12164096157552506	0.12199209790956206	-0.16647797623433316	0.09401130768383535
V16	0.05979178691573992	0.07164467241823783	0.0716073476262239	-0.27125804045737095	-0.09538121597248693	0.05151015774203243	-0.07817375121743182	-0.02441562021885164
V17	-0.017921915795272046	0.04291897947446053	0.03264518174918471	-0.06213623625045226	0.2071548314231877	-0.15359168958460392	-0.02727247917659616	-0.14283279308329072
V18	0.04624944345765195	-0.05106968901979239	-0.058953242530119736	-0.11012326593571237	-0.06616718843335262	0.034479778857581196	0.03274855398764016	-0.037417031137226504
V19	0.056121730987790365	-0.07102115077260378	0.028251148519771552	0.059460868755991346	0.10405902421907098	0.025449314035575564	0.13522682610573514	-0.011916056928539328
V20	0.001161473614743883	-0.2307977046982375	-0.11450126521858241	-0.16568826797634575	-0.0030689427864153836	0.04975253136198062	0.0849787166419713	0.300909378859616
V21	-0.00407538659908738	-0.1492363134145112	0.022959612095881345	0.04346131980058751	-0.004024655263661871	-0.04513025594497108	0.09069395998005438	-0.14117442302260233
V22	-0.07140874079034337	-0.034600457689975446	-0.10681557856572328	0.316771319968308	0.06397112760662656	-0.09571927879586078	0.011694575712194207	-0.005271629547251453
V23	-0.010547880307655088	-0.10956712377592483	0.015567387261071597	0.05087745541029701	0.03264652565425102	0.0032758687856546285	-0.11229637229698336	-0.04959278543318667
V24	-0.011013239162890905	0.0051276352685189095	-0.004639386088415938	0.06820640521550446	0.03461241993037018	0.009952817117739546	-0.009361064910644798	-0.02357951874247128 7

V8	V9	V10	V11	V12	V13	V14	V15	V16
-0.02256968017714251	0.02244400165107835	-0.02648380659179841	-0.008437835780781083	-0.05257586516392 4575	-0.06641950293511927	0.025964542148318545	0.034761453155989164	0.0597917869
0.08950111228996858	-0.14951659799228803	-0.16599585098972258	0.03654766472881237	0.09426707334833272	0.05198081121322252	0.15623888884492963	0.06423869202142417	0.0716446724
-0.11003298435462587	0.017139910144913294	0.20156233160817225	0.07922125192744223	-0.022688765259381027	-0.012433534998105617	-0.08295826010663548	0.1485132744976339	0.0716073476
-0.10201441775073009	0.07588599992099335	0.09594685441721237	-0.011860191200010141	0.02712333809918995	-0.06413936559662822	-0.09157601056294197	-0.22258537614632276	-0.2712580404
0.09377277019082031	-0.09599882695653042	0.02890502948743431	-0.05748709360756093	0.12865793177929444	-0.013502573777133736	-0.0070901435018540715	-0.12164096157552506	-0.0953812159
0.016116608653082978	0.03864653377466208	0.0023073897980982958	0.023275120944077264	-0.08402449241020332	0.008024353553920402	-0.12701046237748517	0.12199209790956206	0.0515101577
0.09894084137750411	0.020683494852760748	0.03990361975873773	-0.13037123879899884	-0.021358919198941218	-0.017882737058407365	-0.021789479667503164	-0.16647797623433316	-0.0781737512
-0.10052161732986188	0.027307213823735535	0.13600123649650964	0.1038340437015405	-0.1237389127533126	0.028168863557450498	-0.1371525385044394	0.09401130768383535	-0.0244156202
1.0	-0.14372561545926082	-0.2117442859509841	-0.028751597633427543	0.042776208139140515	-0.042546003567187386	0.21106113066563886	-0.04554745821287145	0.0690147884
-0.14372561545926082	1.0	0.16481586662562286	-0.12530934348670555	0.1631940728336848	-0.18533603945754465	-0.31734383666900445	-0.008778201842439791	-0.1594714635
-0.2117442859509841	0.16481586662562286	1.0	0.12273976228930346	-0.22314919961394414	0.012231979070942249	-0.2897049532019623	0.003773063113769182	-0.0982543712
0.028751597633427543	-0.12530934348670555	0.12273976228930346	1.0	0.3242600544001402	-0.13935926033255175	-0.02107317133595728	0.026409604289295924	-0.0019813109
0.042776208139140515	0.1631940728336848	-0.22314919961394414	0.3242600544001402	1.0	0.43176493735916455	0.19579824728674475	-0.26057021294792854	-0.1832333941
0.042546003567187386	-0.18533603945754465	0.012231979070942249	-0.13935926033255175	0.43176493735916455	1.0	-0.20634096333381025	0.09961072686167115	0.0277264722
0.21106113066563886	-0.31734383666900445	-0.2897049532019623	-0.02107317133595728	0.19579824728674475	-0.20634096333381025	1.0	0.0993469629252637	0.0532076760
-0.04554745821287145	-0.008778201842439791	0.003773063113769182	0.026409604289295924	-0.26057021294792854	0.09961072686167115	0.0993469629252637	1.0	0.0899018555
0.06901478844867924	-0.15947146356761496	-0.09825437127335278	-0.001981310969731231	-0.18323339414605105	0.027726472236011938	0.05320767600803998	0.08990185559599198	
0.05895648754535658	-0.07175138481668297	-0.12349958927973635	0.0051321155182635384	0.08512141166786617	-0.09178227150640092	-0.19612874199337108	-0.003439086971153879	-0.2751721406
-0.01892313869461621	-0.09856624234894759	0.013476636456132705	0.02949286895120329	-0.12771106616577563	-0.022750317733714366	-0.12422829803207894	0.011821863928408624	0.1436133978
0.011378448387180978	0.05733198642724949	0.026735862516434162	-0.14829287482708423	-0.06785939127909227	-0.004685302100674802	0.02556348971953643	-0.09668312057333205	0.0192316971
0.02584835619173381	0.16399204445024282	0.2521220616161963	0.04872634407462445	-0.09942517581938 97	0.07925103188033811	-0.2205955000149275	0.0839192453520951	0.0835930645
-0.39827087557579066	0.010298619664844404	-0.10078700657694252	-0.004046093130136478	0.10985292888103 59	0.04499512936672348	0.11274642059652253	0.011688002601482225	0.0091493764
0.1747842402634572	0.03893267291756799	-0.0031039832006079407	0.08122283598427922	0.008292519065747366	0.009108774363747824	-0.10176283501218654	-0.04232615802377164	-0.1200071835
.0018980161359543834	-0.08720080454851634	-0.02046167924703101	0.08962406223521946	0.05165309175615279	0.001134948150927253	0.06372415235131809	0.0744857090369862	-0.0412237972
0.016076291477067704	-0.03532074207893957	0.03222841619538773	0.12438156752530173	0.003123107526134647	-0.0487036695032196	0.01663622552945568	-0.03593423743112617	0.0123856968

V16	V17	V18	V19
0.05979178691573992	-0.017921915795272046	0.04624944345765195	0.056121730987790365
0.07164467241823783	0.04291897947446053	-0.05106968901979239	-0.07102115077260378
0.0716073476262239	0.03264518174918471	-0.058953242530119736	0.028251148519771552
-0.27125804045737095	-0.06213623625045226	-0.11012326593571237	0.059460868755991346
-0.09538121597248693	0.2071548314231877	-0.06616718843335262	0.10405902421907098
0.05151015774203243	-0.15359168958460392	0.034479778857581196	0.025449314035575564
-0.07817375121743182	-0.02727247917659616	0.03274855398764016	0.13522682610573514
-0.02441562021885164	-0.14283279308329072	-0.037417031137226504	-0.011916056928539328
0.06901478844867924	0.05895648754535658	-0.01892313869461621	-0.011378448387180978
-0.15947146356761496	-0.07175138481668297	-0.09856624234894759	0.05733198642724949
-0.09825437127335278	-0.12349958927973635	0.013476636456132705	0.026735862516434162
-0.001981310969731231	0.0051321155182635384	0.02949286895120329	-0.14829287482708423
-0.18323339414605105	0.08512141166786617	-0.12771106616577563	-0.06785939127909227
0.027726472236011938	-0.09178227150640092	-0.022750317733714366	-0.004685302100674802
0.05320767600803998	-0.19612874199337108	-0.12422829803207894	0.02556348971953643
0.08990185559599198	-0.003439086971153879	0.011821863928408624	-0.09668312057333205
1.0	-0.27517214068823315	0.14361339780325927	0.01923169716774418
-0.27517214068823315	1.0	-0.20602705683296488	-0.04705955701502714
0.14361339780325927	-0.20602705683296488	1.0	0.15056876014802886
0.01923169716774418	-0.04705955701502714	0.15056876014802886	1.0
0.08359306457102018	-0.07203765496993027	0.013479424330907985	0.08103091383564011
0.009149376425033964	-0.004137089200848911	-0.017434694453420725	0.06035899484759624
-0.12000718350134588	0.1269671133375839	-0.140445499562902	-0.0020743274059004738
-0.04122379720659809	0.08376189212560853	-0.06752937275321542	-0.08151385522587036
0.012385696856879238	-0.03131757798508211	-0.019848411408168704	-0.005217081524483134

- Heat map



- **UI Screenshot:**

```
Customer Group 0: 1.0 not fraud
Customer Group 1: 0.9791281589923939 not fraud | 0.020871841007606l2 fraud
Customer Group 2: 0.9748987854251012 not fraud | 0.025101214574898785 fraud
Customer Group 3: 0.2097902097902098 not fraud | 0.7902097902097902 fraud
Customer Group 4: 1.0 not fraud
```

Naive Bayes

```
gnb = GaussianNB()
nb = cross_val_score(gnb, X_train, y_train, cv = 10)
print("Train Data:", numpy.mean(nb))

gnb = GaussianNB()
nb = cross_val_score(gnb, X_test, y_test, cv = 10)
print("Test Data:", numpy.mean(nb))
```

```
Train Data: 0.9507042088251954
Test Data: 0.9501341129450445
```

- **Sample Prediction Output:**
  After inputting test transaction data in the notebook, the model generated outputs like:

  Prediction: 1 (FRAUD)
  or
  Prediction: 0 (NORMAL)

12. Source code

import pandas as pd
import numpy
import string
from sklearn.model_selection import train_test_split
import sklearn.decomposition
import matplotlib.pyplot as plt

```
from sklearn.cluster import KMeans
%pylab inline
import math
import statistics
import sklearn
from sklearn import neighbors
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, completeness_score
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn import tree
import graphviz
import pydot
import pydotplus
from sklearn import svm
import os
os.environ["PATH"] += os.pathsep + "C:\\Program Files
(x86)\\Graphviz2.38\\bin\\"
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')


#Original Data
fraud = pd.read_csv('/content/drive/My Drive/fraudData.csv')
#Upsampled Data – Training

df2=pd.read_csv('/content/drive/My Drive/creditcard_new.csv')
df2.head()

#checking correlation
corr=df2.corr()
corr

#checking the correlation in hetmap
plt.figure(figsize=(19,16))
sns.heatmap(corr, cmap="coolwarm",annot=True)
plt.show()

df=pd.DataFrame(fraud)
```

```
df.head(100)

fraud["customer"] = fraud["customer"].str.replace('[^\w\s]','')
fraud["age"] = fraud["age"].str.replace('[^\w\s]','')
fraud["gender"] = fraud["gender"].str.replace('[^\w\s]','')
fraud["zipcodeOri"] = fraud["zipcodeOri"].str.replace('[^\w\s]','')
fraud["merchant"] = fraud["merchant"].str.replace('[^\w\s]','')
fraud["zipMerchant"] = fraud["zipMerchant"].str.replace('[^\w\s]','')
fraud["category"] = fraud["category"].str.replace('[^\w\s]','')

fraud["step"] = fraud["step"].astype("category")
fraud["customer"] = fraud["customer"].astype("category")
fraud["age"] = fraud["age"].astype("category")
fraud["gender"] = fraud["gender"].astype("category")
fraud["zipcodeOri"] = fraud["zipcodeOri"].astype("category")
fraud["merchant"] = fraud["merchant"].astype("category")
fraud["zipMerchant"] = fraud["zipMerchant"].astype("category")
fraud["category"] = fraud["category"].astype("category")
fraud["amount"] = fraud["amount"].astype(float)
fraud["fraud"] = fraud["fraud"].astype("category")

fraud = fraud.drop(["zipcodeOri"], axis = 1)
fraud = fraud.drop(["zipMerchant"], axis = 1)

fraudBinaryCols = pandas.get_dummies(fraud,columns =
["age","gender","merchant","category"])
fraudBinaryCols = fraudBinaryCols.drop(["customer"], axis = 1)
sns.distplot(fraud["amount"], kde=False, rug=True)
title("Amount Rug Distribution")

scaler = StandardScaler()
fraudElim = fraudBinaryCols
fraudElim = fraudElim.drop(["step", "fraud"], axis = 1)
fraudStandScaler = scaler.fit_transform(fraudBinaryCols)
fraudStand = pandas.DataFrame(fraudStandScaler)
fraudStand.columns = list(fraudBinaryCols)
dataStandardized = fraudStand
dataStandardized["fraud"] = fraudBinaryCols["fraud"]

scaler = StandardScaler()
fraudStandScaler = scaler.fit_transform(fraudUp)
fraudStand = pandas.DataFrame(fraudStandScaler)
fraudStand.columns = list(fraudBinaryCols)
dataStandardizedUP = fraudStand
```

```
error = list()
kList = list()
for k in range(2, 11):
    kmeans_model = KMeans(n_clusters = k, random_state =
891).fit(dataStandardized)
    labels = kmeans_model.labels_
    labels = labels.tolist()
    cost = kmeans_model.inertia_
    error.append(cost)
    kList.append(k)
    print("k:", k, " cost:", cost)
plt.plot(kList, error)
plt.title("K Values And Error Score - Original Data")
plt.xlabel("K Value")
plt.ylabel("Error Score")
plt.grid()
plt.show()

kmeans = KMeans(n_clusters = 4, random_state = 891).fit(dataStandardized)
labs = kmeans.labels_
labsList = labs.tolist()
dataStandardized["customerGroup"] = labsList
dataStandardized["fraud"] = fraud["fraud"]

counts = dataStandardized.groupby("customerGroup").count()
counts = counts["step"].values.tolist()

dataStandardized.groupby(['customerGroup',
'fraud']).size().unstack().plot(kind='bar', stacked=True)
annotate(counts[0], [-0.21, 325000])
annotate(counts[1], [0.9, 50000])
annotate(counts[2], [1.76, 280000])
annotate(counts[3], [2.9, 50000])

ylabel("Number of Transactions")
xlabel("Customer Group")
grid()
title("Customer Group Breakdown - Original Data")

cust0 = dataStandardized.loc[dataStandardized['customerGroup'] == 0]
cust0 = cust0.reset_index(drop = True)
cust1 = dataStandardized.loc[dataStandardized['customerGroup'] == 1]
cust1 = cust1.reset_index(drop = True)
```

```
cust2 = dataStandardized.loc[dataStandardized['customerGroup'] == 2]
cust2 = cust2.reset_index(drop = True)
cust3 = dataStandardized.loc[dataStandardized['customerGroup'] == 3]
cust3 = cust3.reset_index(drop = True)

counts0 = cust0.groupby("fraud").count()
counts0 = counts0["step"].values.tolist()
nofraud0 = counts0[0]/sum(counts0)
fraud0 = counts0[1]/sum(counts0)
print("Customer Group 0:", "%s not fraud" % nofraud0, "| %s fraud" % fraud0)

counts1 = cust1.groupby("fraud").count()
counts1 = counts1["step"].values.tolist()
nofraud1 = counts1[0]/sum(counts1)
fraud1 = counts1[1]/sum(counts1)
print("Customer Group 1:", "%s not fraud" % nofraud1, "| %s fraud" % fraud1)

counts2 = cust2.groupby("fraud").count()
counts2 = counts2["step"].values.tolist()
nofraud2 = counts2[0]/sum(counts2)
fraud2 = counts2[1]/sum(counts2)
print("Customer Group 2:", "%s not fraud" % nofraud2, "| %s fraud" % fraud2)

counts3 = cust3.groupby("fraud").count()
counts3 = counts3["step"].values.tolist()
nofraud3 = counts3[0]/sum(counts3)
fraud3 = counts3[1]/sum(counts3)
print("Customer Group 3:", "%s not fraud" % nofraud3, "| %s fraud" % fraud3)

error = list()
kList = list()
for k in range(2, 11):
   kmeans_model = KMeans(n_clusters = k, random_state =
891).fit(dataStandardizedUP)
   labels = kmeans_model.labels_
   labels = labels.tolist()
   cost = kmeans_model.inertia_
   error.append(cost)
   kList.append(k)
   print("k:", k, " cost:", cost)
plot(kList, error)
title("K Values And Error Score - Upsampled Data")
xlabel("K Value")
ylabel("Error Score")
```

```
grid()
show()

kmeans = KMeans(n_clusters = 5, random_state = 891).fit(dataStandardizedUP)
labs = kmeans.labels_
labsList = labs.tolist()
dataStandardizedUP["customerGroup"] = labsList
dataStandardizedUP["fraud"] = fraudUp["fraud"]

counts = dataStandardizedUP.groupby("customerGroup").count()
counts = counts["amount"].values.tolist()

dataStandardizedUP.groupby(['customerGroup',
'fraud']).size().unstack().plot(kind='bar', stacked=True)
annotate(counts[0], [-0.21, 50000])
annotate(counts[1], [0.73, 439000])
annotate(counts[2], [1.76, 50000])
annotate(counts[3], [2.74, 50000])
annotate(counts[4], [3.73, 410000])

ylabel("Number of Transactions")
xlabel("Customer Group")
grid()
title("Customer Group Breakdown - Upsampled Data")

cust0 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 0]
cust0 = cust0.reset_index(drop = True)
cust1 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 1]
cust1 = cust1.reset_index(drop = True)
cust2 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 2]
cust2 = cust2.reset_index(drop = True)
cust3 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 3]
cust3 = cust3.reset_index(drop = True)
cust4 = dataStandardizedUP.loc[dataStandardizedUP['customerGroup'] == 4]
cust4 = cust4.reset_index(drop = True)

counts0 = cust0.groupby("fraud").count()
counts0 = counts0["amount"].values.tolist()
nofraud0 = counts0[0]/sum(counts0)
fraud0 = counts0[1]/sum(counts0)
print("Customer Group 0:", "%s not fraud" % nofraud0, "| %s fraud" % fraud0)

counts1 = cust1.groupby("fraud").count()
counts1 = counts1["amount"].values.tolist()
```

```
nofraud1 = counts1[0]/sum(counts1)
fraud1 = counts1[1]/sum(counts1)
print("Customer Group 1:", "%s not fraud" % nofraud1, "| %s fraud" % fraud1)

counts2 = cust2.groupby("fraud").count()
counts2 = counts2["amount"].values.tolist()
nofraud2 = counts2[0]/sum(counts2)
fraud2 = counts2[1]/sum(counts2)
print("Customer Group 2:", "%s not fraud" % nofraud2, "| %s fraud" % fraud2)

counts3 = cust3.groupby("fraud").count()
counts3 = counts3["amount"].values.tolist()
nofraud3 = counts3[0]/sum(counts3)
print("Customer Group 3:", "%s not fraud" % nofraud3, "| %s fraud" % 0)

counts4 = cust4.groupby("fraud").count()
counts4 = counts4["amount"].values.tolist()
nofraud4 = counts4[0]/sum(counts4)
print("Customer Group 4:", "%s not fraud" % nofraud4, "| %s fraud" % 0)

error = list()
kList = list()
for k in range(2, 11):
    kmeans_model = KMeans(n_clusters = k, random_state =
891).fit(dataStandardizedTest)
    labels = kmeans_model.labels_
    labels = labels.tolist()
    cost = kmeans_model.inertia_
    error.append(cost)
    kList.append(k)
    print("k:", k, " cost:", cost)
plot(kList, error)
title("K Values And Error Score - Test Data")
xlabel("K Value")
ylabel("Error Score")
grid()
show()

kmeans = KMeans(n_clusters = 5, random_state = 891).fit(dataStandardizedTest)
labs = kmeans.labels_
labsList = labs.tolist()
dataStandardizedTest["customerGroup"] = labsList
dataStandardizedTest["fraud"] = fraudTest["fraud"]
```

```
counts = dataStandardizedTest.groupby("customerGroup").count()
counts = counts["amount"].values.tolist()

dataStandardizedTest.groupby(['customerGroup',
'fraud']).size().unstack().plot(kind='bar', stacked=True)
annotate(counts[0], [-0.21, 52000])
annotate(counts[1], [0.73, 60000])
annotate(counts[2], [1.76, 3000])
annotate(counts[3], [2.74, 1000])
annotate(counts[4], [3.73, 8200])

ylabel("Number of Transactions")
xlabel("Customer Group")
grid()
title("Customer Group Breakdown - Test Data")

cust0 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 0]
cust0 = cust0.reset_index(drop = True)
cust1 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 1]
cust1 = cust1.reset_index(drop = True)
cust2 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 2]
cust2 = cust2.reset_index(drop = True)
cust3 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 3]
cust3 = cust3.reset_index(drop = True)
cust4 = dataStandardizedTest.loc[dataStandardizedTest['customerGroup'] == 4]
cust4 = cust4.reset_index(drop = True)

counts0 = cust0.groupby("fraud").count()
counts0 = counts0["amount"].values.tolist()
nofraud0 = counts0[0]/sum(counts0)
print("Customer Group 0:", "%s not fraud" % nofraud0)#, "%s fraud" % fraud0)

counts1 = cust1.groupby("fraud").count()
counts1 = counts1["amount"].values.tolist()
nofraud1 = counts1[0]/sum(counts1)
fraud1 = counts1[1]/sum(counts1)
print("Customer Group 1:", "%s not fraud" % nofraud1, "| %s fraud" % fraud1)

counts2 = cust2.groupby("fraud").count()
counts2 = counts2["amount"].values.tolist()
nofraud2 = counts2[0]/sum(counts2)
fraud2 = counts2[1]/sum(counts2)
print("Customer Group 2:", "%s not fraud" % nofraud2, "| %s fraud" % fraud2)
```

```
counts3 = cust3.groupby("fraud").count()
counts3 = counts3["amount"].values.tolist()
nofraud3 = counts3[0]/sum(counts3)
fraud3 = counts3[1]/sum(counts3)
print("Customer Group 3:", "%s not fraud" % nofraud3, "| %s fraud" % fraud3)

counts4 = cust4.groupby("fraud").count()
counts4 = counts4["amount"].values.tolist()
nofraud4 = counts4[0]/sum(counts4)
print("Customer Group 4:", "%s not fraud" % nofraud4)#, "%s fraud" % fraud4)

dataStandardized1 = dataStandardized
dataStandardized1 = dataStandardized1.drop("fraud", axis = 1)
X_train, X_test, y_train, y_test = train_test_split(dataStandardized1,
dataStandardized['fraud'], test_size = 0.40, random_state = 10, stratify =
dataStandardized['fraud'])

gnb = GaussianNB()
nb = cross_val_score(gnb, X_train, y_train, cv = 10)
print("Train Data:", numpy.mean(nb))

gnb = GaussianNB()
nb = cross_val_score(gnb, X_test, y_test, cv = 10)
print("Test Data:", numpy.mean(nb))

reg = sklearn.linear_model.LogisticRegression()
reg.fit(X_train, y_train)
print(reg.coef_)
y_pred = reg.predict(X_test)
confMat = sklearn.metrics.confusion_matrix(y_test, y_pred)
confMatList = confMat.tolist()
TN = confMatList[0][0]
TP = confMatList[1][1]
FN = confMatList[1][0]
FP = confMatList[0][1]

precision = (TP) / (TP + FP)
recall = (TP) / (TP + FN)
print("Precision:", precision)
print("Recall:", recall)
confMat

reg = MLPRegressor()
reg.fit(X_train, y_train)
```

```
y_pred = reg.predict(X_test)
y_pred = numpy.rint(y_pred)

confMat = sklearn.metrics.confusion_matrix(y_test, y_pred)
confMatList = confMat.tolist()
TN = confMatList[0][0]
TP = confMatList[1][1]
FN = confMatList[1][0]
FP = confMatList[0][1]

precision = (TP) / (TP + FP)
recall = (TP) / (TP + FN)
print("Precision:", precision)
print("Recall:", recall)
confMat

accuracy = []
for x in range(2, 101):
    print(x)
    clf = tree.DecisionTreeClassifier(max_depth = x)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    confMat = sklearn.metrics.confusion_matrix(y_test, y_pred)
    confMatList = confMat.tolist()
    TN = confMatList[0][0]
    TP = confMatList[1][1]
    FN = confMatList[1][0]
    FP = confMatList[0][1]

    precision = (TP) / (TP + FP)
    recall = (TP) / (TP + FN)
    f1 = 2*((precision * recall) / (precision + recall))
    accuracy.append([x, precision, recall, f1])

accuracyDF = pandas.DataFrame(data = accuracy, columns = ["k", "precision",
"recall", "f-measure"])
accuracyDF = accuracyDF.sort_values(by = ["precision", "recall"], ascending =
False)
accuracyDF

maxVals = list(accuracyDF["k"])
best = maxVals[0]
clf = tree.DecisionTreeClassifier(max_depth = best)
clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)
accuracy.append([x, sklearn.metrics.accuracy_score(y_test, y_pred)])
dot_data = tree.export_graphviz(clf, out_file = None)
graph = graphviz.Source(dot_data)
graph.render("fraudDT")

#print("Accuracy:", sklearn.metrics.accuracy_score(y_test, y_pred))
confMat = sklearn.metrics.confusion_matrix(y_test, y_pred)
confMatList = confMat.tolist()
TN = confMatList[0][0]
TP = confMatList[1][1]
FN = confMatList[1][0]
FP = confMatList[0][1]

precision = (TP) / (TP + FP)
recall = (TP) / (TP + FN)
print("Precision:", precision)
print("Recall:", recall)
confMat
```

13. Future scope

- Integrate real-time data pipeline with streaming APIs
- Use deep learning models for improved anomaly detection
- Incorporate user feedback loop to reduce false positives

13.Team Members and Roles

- **Mohammed Ayaz. A [510623104056] – Team Lead & Model Building**
  Leads the project, implements and evaluates machine learning models, oversees integration of AI-based detection techniques.

- **Mohammed Azhan. U [510623104057] – Data Collection & Preprocessing**

Responsible for sourcing the dataset, handling missing values, class imbalance, and preparing data for modeling.

- **Md Faazil Ammar. P [510623104047] – Exploratory Data Analysis & Visualization**
  Performs data analysis, identifies patterns in fraudulent transactions, and visualizes important insights.

- **Kashif Ulhaq. K [510623104040] – Feature Engineering & Dimensionality Reduction**
  Designs new features to enhance detection accuracy and applies PCA/feature selection where necessary.

- **Ashfaq Ahmed. M [510623104009] – Model Evaluation & Validation**
  Compares multiple models, evaluates them using classification metrics, and ensures robust performance.

- **Abrar Ul Haque. R [510623104004] – Report Writing & Presentation**
  Compiles project outcomes into a well-structured report and creates a visual presentation for submission.