

Importing the required Libraries

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import datetime as dt
```

Importing the Data.

```
In [2]: 1 train_df = pd.read_csv('fraudTrain.csv')
2 test_df = pd.read_csv('fraudTest.csv')
```

```
In [3]: 1 train_df.head()
```

Out[3]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	J
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Ste
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	E
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	J
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	

5 rows × 23 columns

```
In [4]: 1 test_df.head()
```

Out[4]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	.
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	
4	4	2020-06-21 12:15:17	3626826139003947	fraud_Johnston-Casper	travel	3.19	

```
In [5]: 1 train_df.info()
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

In [5]:

1train\_df.info()

2020-06-21 12:14:533598215285024754

fraud Swaniawski,  
Nitzsche and  
Welch

health\_fitness 41.28

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1296675 entries, 0 to 1296674  
Data columns (total 23 columns):  
# Column Non-Null Count Dtype  
---  
0 Unnamed: 0 1296675 non-null int64  
1 trans\_date\_trans\_time 1296675 non-null object  
2 cc\_num 1296675 non-null int64  
3 merchant 1296675 non-null object  
4 category 1296675 non-null object  
5 amt 1296675 non-null float64  
6 first 1296675 non-null object  
7 last 1296675 non-null object  
8 gender 1296675 non-null object  
9 street 1296675 non-null object  
10 city 1296675 non-null object  
11 state 1296675 non-null object  
12 zip 1296675 non-null int64  
13 lat 1296675 non-null float64  
14 long 1296675 non-null float64  
15 city\_pop 1296675 non-null int64  
16 job 1296675 non-null object  
17 dob 1296675 non-null object  
18 trans\_num 1296675 non-null object  
19 unix\_time 1296675 non-null int64  
20 merch\_lat 1296675 non-null float64  
21 merch\_long 1296675 non-null float64  
22 is\_fraud 1296675 non-null int64  
dtypes: float64(5), int64(6), object(12)  
memory usage: 227.5+ MB

In [6]:

1test\_df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 555719 entries, 0 to 555718  
Data columns (total 23 columns):  
# Column Non-Null Count Dtype  
---  
0 Unnamed: 0 555719 non-null int64  
1 trans\_date\_trans\_time 555719 non-null object  
2 cc\_num 555719 non-null int64  
3 merchant 555719 non-null object  
4 category 555719 non-null object  
5 amt 555719 non-null float64  
6 first 555719 non-null object  
7 last 555719 non-null object  
8 gender 555719 non-null object  
9 street 555719 non-null object  
10 city 555719 non-null object  
11 state 555719 non-null object  
12 zip 555719 non-null int64  
13 lat 555719 non-null float64  
14 long 555719 non-null float64  
15 city\_pop 555719 non-null int64  
16 job 555719 non-null object  
17 dob 555719 non-null object  
18 trans\_num 555719 non-null object  
19 unix\_time 555719 non-null int64  
20 merch\_lat 555719 non-null float64  
21 merch\_long 555719 non-null float64  
22 is\_fraud 555719 non-null int64  
dtypes: float64(5), int64(6), object(12)  
memory usage: 227.5+ MB

In [9]:

1test\_data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 555719 entries, 0 to 555718  
Data columns (total 23 columns):  
# Column Non-Null Count Dtype  
---  
0 Unnamed: 0 555719 non-null int64  
1 trans\_date\_trans\_time 555719 non-null object  
2 cc\_num 555719 non-null int64  
3 merchant 555719 non-null object  
4 category 555719 non-null object  
5 amt 555719 non-null float64  
6 first 555719 non-null object  
7 last 555719 non-null object  
8 gender 555719 non-null object  
9 street 555719 non-null object  
10 city 555719 non-null object  
11 state 555719 non-null object  
12 zip 555719 non-null int64  
13 lat 555719 non-null float64  
14 long 555719 non-null float64  
15 city\_pop 555719 non-null int64  
16 job 555719 non-null object  
17 dob 555719 non-null object  
18 trans\_num 555719 non-null object  
19 unix\_time 555719 non-null int64  
20 merch\_lat 555719 non-null float64  
21 merch\_long 555719 non-null float64  
22 is\_fraud 555719 non-null int64  
dtypes: float64(5), int64(6), object(12)  
memory usage: 227.5+ MB

In [7]:

1## WE have established that both train and test data have same columns with

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 555719 entries, 0 to 555718  
Data columns (total 23 columns):  
# Column Non-Null Count Dtype  
---  
0 Unnamed: 0 555719 non-null int64  
1 trans\_date\_trans\_time 555719 non-null object  
2 cc\_num 555719 non-null int64  
3 merchant 555719 non-null object  
4 category 555719 non-null object  
5 amt 555719 non-null float64  
6 first 555719 non-null object  
7 last 555719 non-null object  
8 gender 555719 non-null object  
9 street 555719 non-null object  
10 city 555719 non-null object  
11 state 555719 non-null object  
12 zip 555719 non-null int64  
13 lat 555719 non-null float64  
14 long 555719 non-null float64  
15 city\_pop 555719 non-null int64  
16 job 555719 non-null object  
17 dob 555719 non-null object  
18 trans\_num 555719 non-null object  
19 unix\_time 555719 non-null int64  
20 merch\_lat 555719 non-null float64  
21 merch\_long 555719 non-null float64  
22 is\_fraud 555719 non-null int64

## Conducting EDA

In [8]:

1train\_data = train\_df  
2test\_data = test\_df

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 555719 entries, 0 to 555718  
Data columns (total 23 columns):  
# Column Non-Null Count Dtype  
---  
0 Unnamed: 0 555719 non-null int64  
1 trans\_date\_trans\_time 555719 non-null object  
2 cc\_num 555719 non-null int64  
3 merchant 555719 non-null object  
4 category 555719 non-null object  
5 amt 555719 non-null float64  
6 first 555719 non-null object  
7 last 555719 non-null object  
8 gender 555719 non-null object  
9 street 555719 non-null object  
10 city 555719 non-null object  
11 state 555719 non-null object  
12 zip 555719 non-null int64  
13 lat 555719 non-null float64  
14 long 555719 non-null float64  
15 city\_pop 555719 non-null int64  
16 job 555719 non-null object  
17 dob 555719 non-null object  
18 trans\_num 555719 non-null object  
19 unix\_time 555719 non-null int64  
20 merch\_lat 555719 non-null float64  
21 merch\_long 555719 non-null float64  
22 is\_fraud 555719 non-null int64

localhost:8888/notebooks/Downloads/Credit Card CapStone Project/Credit\_Card\_Fraud\_Detection.ipynb

2/10

In [9]:

```
20 merch_lat 555719 non-null float64
21 test_data.info() 555719 non-null float64
22 is_fraud 555719 non-null int64
dtypes: float64(5), int64(6), object(12)
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
```

In [7]:

```
1 # WE have established that both train and test data have same columns wi
---
0 Unnamed: 0 555719 non-null int64
1 trans_date_trans_time 555719 non-null object
2 cc_num 555719 non-null int64
3 merchant 555719 non-null object
4 category 555719 non-null object
5 amt 555719 non-null float64
6 first 555719 non-null object
7 last 555719 non-null object
8 gender 555719 non-null object
9 street 555719 non-null object
10 city 555719 non-null object
11 state 555719 non-null object
12 zip 555719 non-null int64
13 lat 555719 non-null float64
14 long 555719 non-null float64
15 city_pop 555719 non-null int64
16 job 555719 non-null object
17 dob 555719 non-null object
18 trans_num 555719 non-null object
19 unix_time 555719 non-null int64
20 merch_lat 555719 non-null float64
21 merch_long 555719 non-null float64
22 is_fraud 555719 non-null int64
dtypes: float64(5), int64(6), object(12)
memory usage: 97.5+ MB
```

In [8]:

```
1 train_data = train_df
2 test_data = test_df
3 last
```

In [10]:

```
1 train_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Unnamed: 0            1296675 non-null int64
1   trans_date_trans_time 1296675 non-null object
2   cc_num                1296675 non-null int64
3   merchant              1296675 non-null object
4   category              1296675 non-null object
5   amt                   1296675 non-null float64
6   first                 1296675 non-null object
7   last                  1296675 non-null object
8   gender                1296675 non-null object
9   street                1296675 non-null object
10  city                  1296675 non-null object
11  state                 1296675 non-null object
12  zip                   1296675 non-null int64
13  lat                   1296675 non-null float64
14  long                  1296675 non-null float64
15  city_pop              1296675 non-null int64
16  job                   1296675 non-null object
17  dob                   1296675 non-null object
18  trans_num             1296675 non-null object
19  unix_time             1296675 non-null int64
20  merch_lat             1296675 non-null float64
21  merch_long            1296675 non-null float64
22  is_fraud              1296675 non-null int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

In [11]:

```
1 train_data.shape
2 train_data['hour'] = pd.to_datetime(train_data['trans_date_trans_time']).dt
3 test_data['hour'] = pd.to_datetime(test_data['trans_date_trans_time']).dt
```

In [12]:

```
1 #Card usage or transaction frequency also might also play a role in relat
2 test_data.shape
3 # So we will make a new column of CC_frequency.
```

Out[12]:

```
1 freq_enc = (train_data.groupby("cc_num").size())
2 freq_enc.sort_values(ascending=True)
3 train_data["cc_frequency"] = train_data["cc_num"].apply(lambda x: freq_en
4
5 freq_enc = (test_data.groupby("cc_num").size())
6 freq_enc.sort_values(ascending=True)
7 test_data["cc_frequency"] = test_data["cc_num"].apply(lambda x: freq_enc[
8
9 # Age group might also play an important role in fraud transaction
10 # So we will make a new column of age.
11
12 train_data['age'] = dt.date.today().year-pd.to_datetime(train_data['dob']).
13 test_data['age'] = dt.date.today().year-pd.to_datetime(test_data['dob']).
```

100%

100/100 [00:10<00:00, 9.84it/s]

```
20 merch_lat 1296675 non-null float64
21 merch_long 1296675 non-null float64
22 trans_date 1296675 non-null int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB

In [13]: 1 #Transaction time seemingly plays a important role in respect to fraud.
2 #So we will make a new column of hour.

In [11]: 3 train_data.shape
4 train_data['hour'] = pd.to_datetime(train_data['trans_date_trans_time']).dt
Out[11]: (1296675, 23)
5 test_data['hour'] = pd.to_datetime(test_data['trans_date_trans_time']).dt
6

In [12]: 7 #Card usage or transaction frequency also might also play a role in relat
8 #ed to this we will make a new column of CC_frequency.
9

Out[12]: (555719, 23)
10 freq_enc = (train_data.groupby("cc_num").size())
11 freq_enc.sort_values(ascending=True)
12 train_data["cc_frequency"] = train_data["cc_num"].apply(lambda x: freq_en
13
14 freq_enc = (test_data.groupby("cc_num").size())
15 freq_enc.sort_values(ascending=True)
16 test_data["cc_frequency"] = test_data["cc_num"].apply(lambda x: freq_enc[
17
18 # Age group might also play an important role in fraud transaction
19 # So we will make a new column of age.
20
21 train_data['age'] = dt.date.today().year-pd.to_datetime(train_data['dob'])
22 test_data['age'] = dt.date.today().year-pd.to_datetime(test_data['dob']).dt
100%|████████████████████████████████████████████████████████████████████████████████| 100/100 [00:10<00:00, 9.84it/s]
```

```
In [14]: 1 #Calculate distance between merchant and home location
2 train_data['latitudinal_distance'] = abs(round(train_data['merch_lat']-train_data['home_lat'])
3 train_data['longitudinal_distance'] = abs(round(train_data['merch_long']-train_data['home_long'])
4
5 test_data['latitudinal_distance'] = abs(round(test_data['merch_lat']-test_data['home_lat'])
6 test_data['longitudinal_distance'] = abs(round(test_data['merch_long']-test_data['home_long'])
```

```
In [15]: 1 # Converting Male/ Female into 0/1 for the machine to read easily.
2 gender_mapping = {"F": 0, "M": 1}
3
4 train_data["gender_binary"] = train_data["gender"].map(gender_mapping)
5 test_data["gender_binary"] = test_data["gender"].map(gender_mapping)
```

```
In [16]: 1 #Drop Columns that are not relevant to predicy fraud transaction
2 drop_columns = ['cc_num','Unnamed: 0','street','zip','city_pop','trans_num']
3 train_data.drop(columns=drop_columns,inplace=True)
4 test_data.drop(columns=drop_columns,inplace=True)
```

```
In [17]: 1 # Making dummie columns for category variable.
2
3 test_data = pd.get_dummies(test_data,columns=['category'],drop_first=True)
4 train_data = pd.get_dummies(train_data,columns=['category'],drop_first=True)
```

```
In [18]: 1 # Scaling the amount and frequency variables.
2 from sklearn.preprocessing import StandardScaler
3
```

```
In [21]: 4 # Creating function for scaling
5 def StandardScaler(df, col_names):
Out[21]: 6     features = df[col_names]
7     scaler = StandardScaler().fit(features.values)
8     df[col_names] = scaler.transform(features)
9     return df
10
11 0 -0.424483 12 0.437280 55 0.020 0.265
12 1 -0.252332 12 0.171606 33 0.870 0.476
13 2 -0.179353 0 12 0.901033 53 0.177 0.660
14 3 -0.069655 12 0.366192 36 0.243 0.064
15 4 -0.23389 12 0.33839 66 0.706 0.868
16
17 5 rows x 21 columns
```

```
In [19]: 1 train_data.shape
```

```
In [22]: 1 train_data.head()
Out[19]: (1296675, 21)
```

```
Out[22]: 1 train_data.head()
In [20]: 1 test_data.head()
Out[20]: (555719, 21)
2
3
4
```

	trans_date_trans_time	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gender
0	0.407826	0	0	0.281849	35	0.068	0.870
1	0.230039	0	0	1.631335	45	0.271	0.024
2	0.934149	0	0	-1.772008	61	0.970	0.108
3	-0.158132	0	0	-1.785476	56	0.804	0.447
4	-0.177094	0	0	0.267035	37	0.254	0.830

In [21]:

```
4 # Creating function for scaling
5 def standard_scaler(df, col_names):
6     features = df[col_names]
7     scaler = StandardScaler().fit(features.values)
8     features = scaler.transform(features.values)
9     df[col_names] = features
10
11 return df
12
13 # Creating train and test data
14 train_data = train_data[['amt', 'is_fraud', 'hour', 'cc_frequency', 'age', 'latitudinal_distance', 'longitudinal_distance', 'gender_binary', 'category_food_dining', 'category_gas_transport', 'category_grocery_net', 'category_grocery_pos', 'category_health_fitness', 'category_home', 'category_kids_pets', 'category_misc_pos', 'category_personal_care', 'category_shopping_net', 'category_shopping_pos', 'category_travel']]
15 train_data_std = StandardScaler (train_data, col_names)
16 test_data_std = StandardScaler (test_data, col_names)
17
18 5 rows x 21 columns
```

In [19]:

```
1 train_data.shape
```

Out[19]:

```
(1296675, 21)
```

Out[22]:

In [20]:

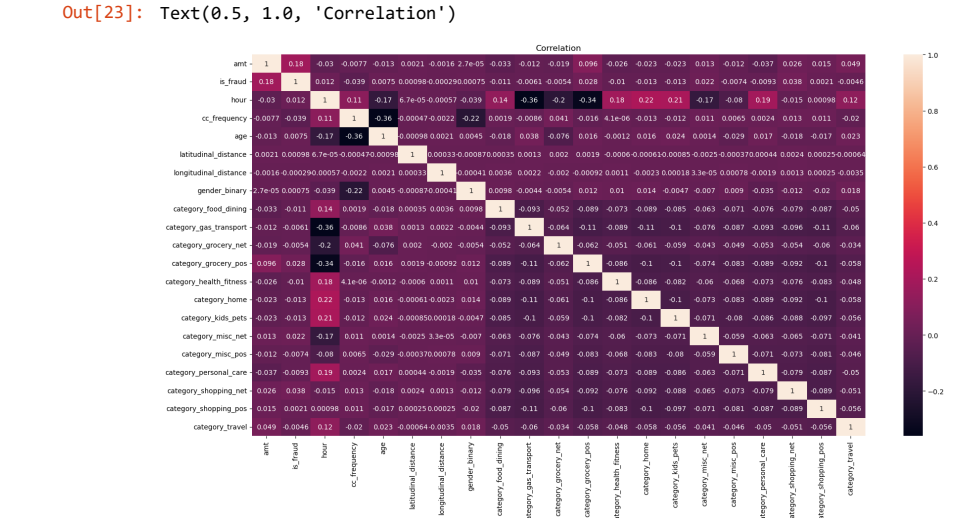
```
1 test_data_std.head()
```

	amt	is_fraud	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gender
0	-0.407826	0	0	0.281849	35	0.068	0.870	
1	0.230039	0	0	1.631335	45	0.271	0.024	
2	0.934149	0	0	-1.772008	61	0.970	0.108	
3	-0.158132	0	0	-1.785476	56	0.804	0.447	
4	-0.177094	0	0	0.267035	37	0.254	0.830	

5 rows x 21 columns

In [23]:

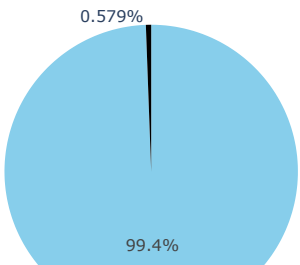
```
1 # creating a correlation heatmap using this we will check for any multicollinearity is when two variables have a correlation >0.7 with each other
2 # multicollinearity is when two variables have a correlation >0.7 with each other
3 import seaborn as sns
4 fig, ax = plt.subplots(figsize=(20,10))
5 sns.heatmap(test_data.corr(), annot=True).set_title('Correlation')
```



In [24]:

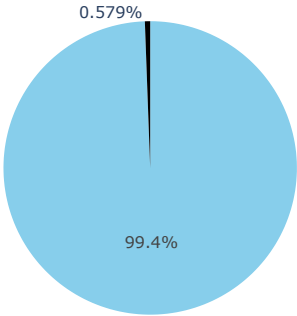
```
1 # As we know fraud transactions are very less than the genuine transactions
2 # We explore how imbalanced the data is.
3
4 import plotly.express as px
5 fig = px.pie(values=train_data['is_fraud'].value_counts(), names=["Genuine", "Fraud"], title="Fraud vs Genuine transactions")
6
7 fig.show()
```

Fraud vs Genuine transactions



```
In [24]: 1 # As we know fraud transactions are very less than the genuine transaction
2 # We explore how imbalanced the data is.
3
4 import plotly.express as px
5 fig = px.pie(values=train_data['is_fraud'].value_counts(), names=["Genuine", "Fraud"],
6              ,title="Fraud vs Genuine transactions")
7 fig.show()
```

Fraud vs Genuine transactions



```
In [25]: 1 #Splitting the data into X = train data, y = target column.
```

```
In [26]: 1 X_train_data = train_data.drop('is_fraud', axis = 1)
2 y_train_data = train_data['is_fraud']
3
4 X_test_data = test_data.drop('is_fraud', axis = 1)
5 y_test_data = test_data['is_fraud']
```

We can see that our data set is unbalanced as only 0.58% data contains frudulent transactions which we have to label.

Thus lets explore how we can manage this by using SMOTE (Synthetic Minority Oversampling Technique).

```
In [29]: 1 smote_data.head()
In [27]: 1 from imblearn.over_sampling import SMOTE
Out[29]: 2
3 # Initialize the SMOTE object
4 smote = SMOTE(random_state=42)
5 Apply SMOTE to the training data
6 X_train_data, y_train_data = smote.fit_resample(X_train_data, y_train_data)
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
In [30]: 1 y_train_data.info()

<class 'pandas.core.series.Series'>
RangeIndex: 2578338 entries, 0 to 2578337
Series name: is_fraud
Non-Null Count  Dtype
-----
2578338 non-null  int64
dtypes: int64(1)
memory usage: 19.7 MB
```

```
In [31]: 1 # Checking the change in train data after SMOTE
```

```
OverSampling Techniques,

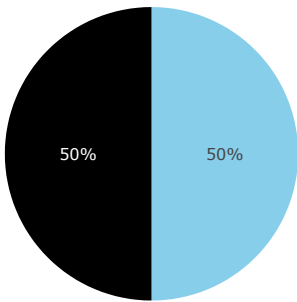
In [29]: 1 smote_data.head()
In [27]: 1 from imblearn.over_sampling import SMOTE
Out[29]: 2
3 # Initialize the SMOTE object
4 smote = SMOTE(random_state=42)
5
16 0.230089 by SMOTE to 1.68335 raising data
27 X_train_data, y_train_data = smote.fit_resample(X_train_data, y_train_data)
8
3 -0.158132 0 -1.785476 56 0.804 0.447 1
4 -0.177094 0 0.267035 37 0.254 0.830 1
2 smote_data = pd.concat(L, ignore_index=False, axis=1)
5 rows x 21 columns

In [30]: 1 y_train_data.info()

<class 'pandas.core.series.Series'>
RangeIndex: 2578338 entries, 0 to 2578337
Series name: is_fraud
Non-Null Count  Dtype
-----
2578338 non-null  int64
dtypes: int64(1)
memory usage: 19.7 MB

In [31]: 1 # Checking the change in train data after SMOTE
2
3 fig = px.pie(values=smote_data['is_fraud'].value_counts(), names=["Genuine",
4 ,title="Fraud vs Genuine transactions")
5 fig.show()
```

Fraud vs Genuine transactions



```
In [35]: 1 from sklearn.metrics import confusion_matrix, recall_score, precision_sco
In [32]: 2 # Importing following Models for testing scores.
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn.ensemble import GradientBoostingClassifier
6 rf_recall = recall_score(y_test_data, y_pred)
7 f1_score = f1_score(y_test_data, y_pred)
8 rf_f1 = f1_score(y_test_data, y_pred)
9 rf_accuracy = accuracy_score(y_test_data, y_pred)
10 rf = RandomForestClassifier(random_state=23)
11 knn = KNeighborsClassifier()
12 score = [rf_recall, rf_precision, rf_f1, rf_accuracy]
13 gboost = GradientBoostingClassifier(random_state=23)
14 lgbm = LGBMClassifier(random_state=23)
15 rf_score.insert(0, 'Model', 'RandomForestClassifier')
16 rf_score

In [33]: 1 # Model Fitting RandomForestClassifier
2 rf_fit = rf.fit(X_train_data, y_train_data)
3 rf_score

Out[33]: 1 RandomForestClassifier
2
3 Model Recall Precision F1 Score Accuracy
4 RandomForestClassifier 0.861072 0.241343 0.377016 0.989016

In [34]: 1 # Making prediction using the Model
In [36]: 2 print(cm)
3 y_pred = rf.predict(X_test_data)
4 [[547768 5806]
5 [ 298 1847]]

In [37]: 1 # Model Fitting KNeighborsClassifier
2
```

```

In [35]: 1 from sklearn.metrics import confusion_matrix, recall_score, precision_score
In [32]: 2 # Importing following Models for testing scores.
3 cm = confusion_matrix(y_test_data, y_pred)
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn.ensemble import GradientBoostingClassifier
6 from sklearn.metrics import recall_score, precision_score, f1_score
7 rf_recall = recall_score(y_test_data, y_pred)
8 rf_precision = precision_score(y_test_data, y_pred)
9 rf_f1 = f1_score(y_test_data, y_pred)
10 rf_accuracy = accuracy_score(y_test_data, y_pred)
11 rf = RandomForestClassifier(random_state=23)
12 knn = KNeighborsClassifier()
13 score = [(rf_recall, rf_precision, rf_f1, rf_accuracy)]
14 gboost = GradientBoostingClassifier(random_state=23)
15 lgbm = LGBMClassifier(random_state=23)
16 rf_score = pd.DataFrame(data = score, columns=['Recall', 'Precision', 'F1 Score', 'Accuracy'])
17 rf_score.insert(0, 'Model', 'RandomForestClassifier')

```

```

In [33]: 1 rf_score
2 # Model Fitting RandomForestClassifier
3 rf.fit(X_train_data, y_train_data)

```

```

Out[33]: 0 RandomForestClassifier
Model Recall Precision F1 Score Accuracy
RandomForestClassifier(random_state=23)
0 RandomForestClassifier 0.861072 0.241343 0.377016 0.989016

```

```

In [34]: 1 # Making prediction using the Model
In [36]: 2 print(cm)
3 y_pred = rf.predict(X_test_data)
4 [[547768 5806]
5 [ 298 1847]]

```

```

In [37]: 1 # Model Fitting KNeighborsClassifier
2
3 knn.fit(X_train_data, y_train_data)

```

```

Out[37]: 0 KNeighborsClassifier
KNeighborsClassifier()

```

```

In [38]: 1 # Making prediction using the Model KNeighborsClassifier
2
3 y_pred = knn.predict(X_test_data)

```

```

In [39]: 1 cm = confusion_matrix(y_test_data, y_pred)
2
3 knn_recall = recall_score(y_test_data, y_pred)
4 knn_precision = precision_score(y_test_data, y_pred)
5 knn_f1 = f1_score(y_test_data, y_pred)
6 knn_accuracy = accuracy_score(y_test_data, y_pred)
7
8 score = [(knn_recall, knn_precision, knn_f1, knn_accuracy)]
9
10 knn_score = pd.DataFrame(data = score, columns=['Recall', 'Precision', 'F1 Score', 'Accuracy'])
11 knn_score.insert(0, 'Model', 'KNeighborsClassifier')
12 knn_score
13
14 knn_score

```

```

Out[39]: 0 KNeighborsClassifier
Model Recall Precision F1 Score Accuracy
KNeighborsClassifier
0 KNeighborsClassifier 0.735664 0.190557 0.302705 0.986918

```

```

In [43]: 1 cm = confusion_matrix(y_test_data, y_pred)
In [40]: 2 gboost_recall = recall_score(y_test_data, y_pred)
3 gboost_precision = precision_score(y_test_data, y_pred)
4 gboost_f1 = f1_score(y_test_data, y_pred)
5 gboost_accuracy = accuracy_score(y_test_data, y_pred)
6
7 score = [(gboost_recall, gboost_precision, gboost_f1, gboost_accuracy)]
8 # Model Fitting GradientBoostingClassifier
9 gboost.fit(X_train_data, y_train_data)
10 gboost_score = pd.DataFrame(data = score, columns=['Recall', 'Precision', 'F1 Score', 'Accuracy'])
11 gboost_score.insert(0, 'Model', 'GradientBoostingClassifier')
12 gboost_score
13
14 gboost_score

```

```

Out[43]: 0 GradientBoostingClassifier
Model Recall Precision F1 Score Accuracy
GradientBoostingClassifier
0 GradientBoostingClassifier 0.940793 0.080057 0.147558 0.958044

```

```

In [44]: 1 print (cm)
2
3 [[530385 23189]
4 [ 127 2018]]

```

```

In [45]: 1 # Making prediction using the Model LGBMClassifier
2 lgbm.fit(X_train_data, y_train_data)

```

[LightGBM] [Info] Number of positive: 1289169, number of negative: 1289169  
[LightGBM] [Info] Auto-choosing row-wise multi-threading. the overhead of



```
In [43]: 1 cm = confusion_matrix(y_test_data, y_pred)
In [40]: 2 gboost_recall = recall_score(y_test_data, y_pred)
          3 gboost_precision = precision_score(y_test_data, y_pred)
          4 gboost_f1 = f1_score(y_test_data, y_pred)
          5 gboost_accy = accuracy_score(y_test_data, y_pred)
          6
In [41]: 7 score = [(gboost Recall, gboost Precision, gboost_f1, gboost_accuracy)]
          8 # Model Fitting GradientBoostingClassifier.
          9 gboost_score = pd.DataFrame(data = score, columns=['Recall','Precision','
Out[41]: 10 gboost_score.insert(0, 'Model', 'GradientBoostingClassifier')
          11 gboost_score
          12 GradientBoostingClassifier
          13 GradientBoostingClassifier(random_state=23)

Out[43]:
In [42]: 1 # Making prediction using the Model
          2 y_pred = gboost.predict(X_test_data)
          0 GradientBoostingClassifier 0.940793 0.080057 0.147558 0.958044

In [44]: 1 print (cm)

[[530385 23189]
 [ 127  2018]]

In [45]: 1 # Making prediction using the Model LGBMClassifier
          2 lgbm.fit(X_train_data, y_train_data)

[LightGBM] [Info] Number of positive: 1289169, number of negative: 1289169
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.014545 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1154
[LightGBM] [Info] Number of data points in the train set: 2578338, number of
used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.0000
00

Out[45]: ▾ LGBMClassifier
          LGBMClassifier(random_state=23)

In [46]: 1 # Making prediction using the Model LGBMClassifier
          2
          3 y_pred = lgbm.predict(X_test_data)

In [47]: 1 cm = confusion_matrix(y_test_data, y_pred)
          2
          3 lgbm_Recall = recall_score(y_test_data, y_pred)
          4 lgbm_Precision = precision_score(y_test_data, y_pred)
          5 lgbm_f1 = f1_score(y_test_data, y_pred)
          6 lgbm_accuracy = accuracy_score(y_test_data, y_pred)
          7
          8 score = [(lgbm_Recall, lgbm_Precision, lgbm_f1, lgbm_accuracy)]
          9
          10 lgbm_score = pd.DataFrame(data = score, columns=['Recall','Precision','F1
          11 lgbm_score.insert(0, 'Model', 'LGBMClassifier')
          12 lgbm_score
          13
          14 lgbm_score

Out[47]:
          Model  Recall  Precision  F1 Score  Accuracy
0  LGBMClassifier  0.951981    0.10072  0.182167  0.967007

In [48]: 1 print(cm)

[[535342 18232]
 [ 103  2042]]
```

Comparing the performance of different Models.

```
In [49]: 1 predictions = pd.concat([rf_score, knn_score, lgbm_score, gboost_score],
          2 predictions.sort_values(by='Recall', ascending=False)

Out[49]:
          Model  Recall  Precision  F1 Score  Accuracy
2  LGBMClassifier  0.951981    0.100720  0.182167  0.967007
3  GradientBoostingClassifier  0.940793    0.080057  0.147558  0.958044
0  RandomForestClassifier  0.861072    0.241343  0.377016  0.989016
1  KNeighborsClassifier  0.735664    0.190557  0.302705  0.986918
```

```
13  
14 lgbm_score
```

Out[47]:

	Model	Recall	Precision	F1 Score	Accuracy
0	LGBMClassifier	0.951981	0.10072	0.182167	0.967007

```
In [48]: 1 print(cm)
```

```
[[535342 18232]  
 [ 103   2042]]
```

Comparing the performance of different Models.

```
In [49]: 1 predictions = pd.concat([rf_score, knn_score, lgbm_score, gboost_score],  
2 predictions.sort_values(by=['Recall'], ascending=False)
```

Out[49]:

	Model	Recall	Precision	F1 Score	Accuracy
2	LGBMClassifier	0.951981	0.100720	0.182167	0.967007
3	GradientBoostingClassifier	0.940793	0.080057	0.147558	0.958044
0	RandomeForestClassifier	0.861072	0.241343	0.377016	0.989016
1	KNeighborsClassifier	0.735664	0.190557	0.302705	0.986918