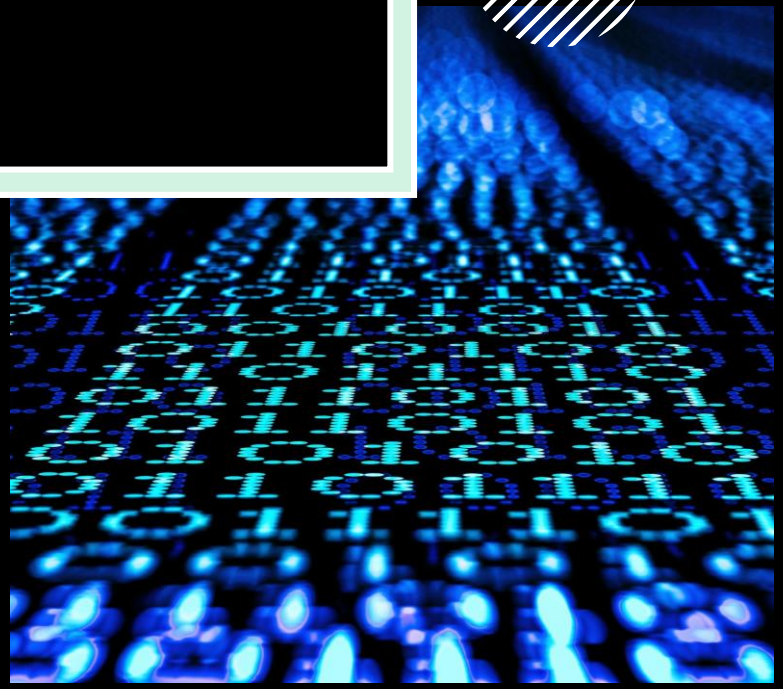
FRAUD DETECTION MODEL

MOBILE MONEY TRANSACTION

SEGMENT'S IN APPROACHING A MODEL

INITIAL STAGE

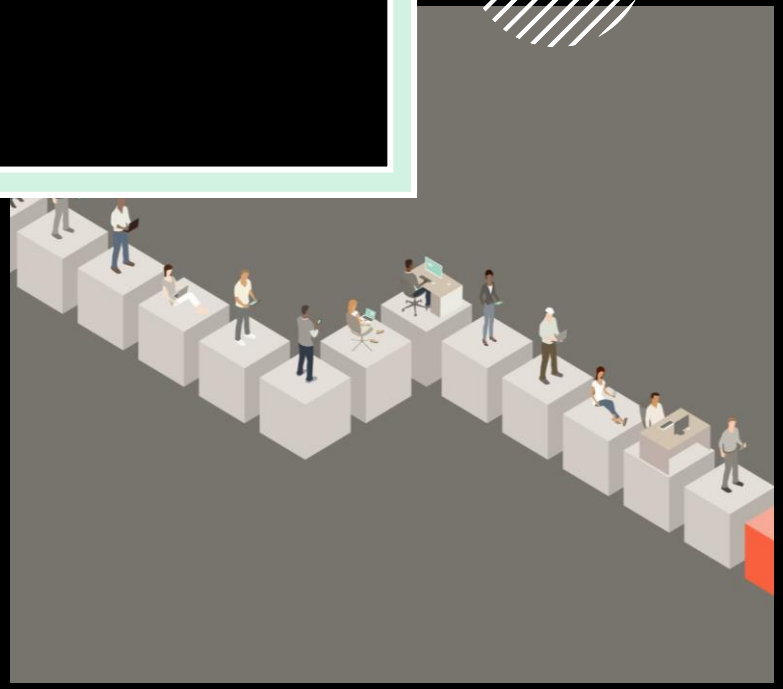


- GATHERING OF DATA.
- UNDERSTANDING AND ANALYSIS.
- IMPLEMENTATION PLANNING

MIDDLE STAGE

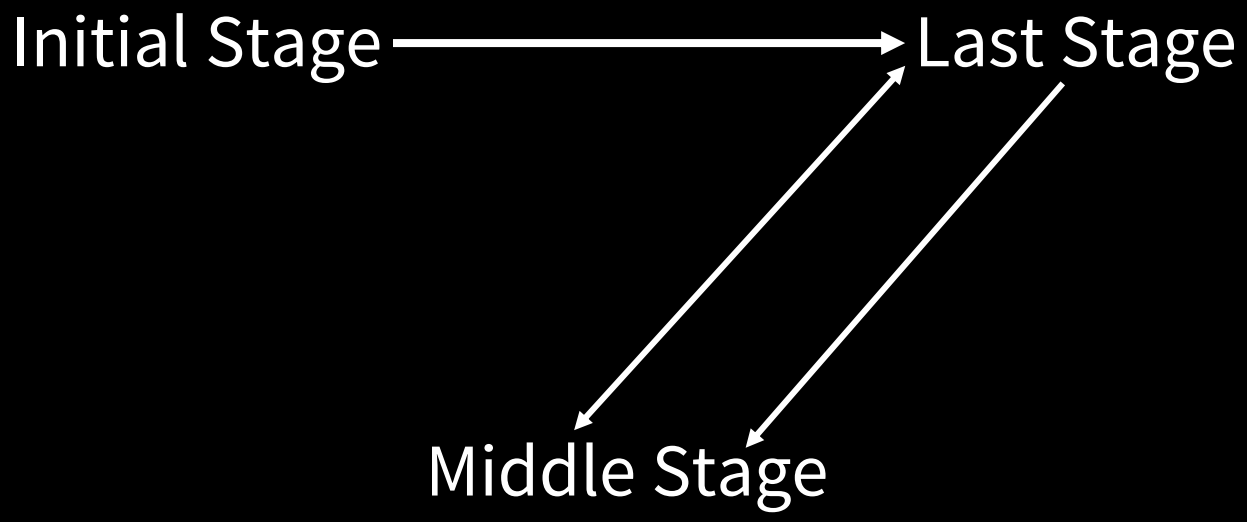
- Explorative Data Analysis
- Feature Engineering
- Feature Selection
- Physical Design

LAST STAGE

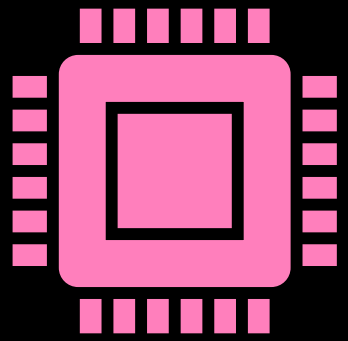


- Development
- Evaluation
- Monitoring and performance tuning
- Deployment

Different Approach



Dataset Used (Kaggle)



Synthetic Financial Datasets for fraud detection
(Generated by PaySim mobile money generator).

https://www.kaggle.com/datasets/ealaxi/paysim1

SOFTWARE USED

JUPYTER NOTEBOOK

RULE OF THUMB (COMMON LIBRARIES)

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import confusion_matrix
```

Importing the data and Validating the shape

```
df=pd.read_csv(r"C:\Users\adity\OneDrive\Desktop\Python\PS_20174392719_1491204439457_log.csv")
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0

```
df.shape  #checking the total number of rows and columns
```

```
(6362620, 11)
```

pd.read_csv = Used for reading the data or giving a source file extension.

df.head () (Appointed into the pd.read_csv input) = User for importing the data.

df. shape = Used for checking the total number of rows and columns (Rows = 6362620
Columns = 11)

Dropping Character Variables

```
df3 = df.drop(['nameOrig','nameDest','isFlaggedFraud'],axis = 1)
```

```
df3.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	170136.0	160296.36	0.0	0.0	0
1	1	PAYMENT	1864.28	21249.0	19384.72	0.0	0.0	0
2	1	TRANSFER	181.00	181.0	0.00	0.0	0.0	1
3	1	CASH_OUT	181.00	181.0	0.00	21182.0	0.0	1
4	1	PAYMENT	11668.14	41554.0	29885.86	0.0	0.0	0

- Dropping of the unwanted variables which are not required in terms of logistic regression.
- Putting the axis as 1 in terms of column.
- Storing of the data in df3.
- Using df3.head to show the output
- Ignoring the dropping of TYPE column as the dropped columns were mostly id variables whereas the type column is not.
- The TYPE column defines a category of the data.

Treating the TYPE Variable

```
# Getting in touch with the TYPE column and checking all the unique values
df3['type'].unique()
```

```
array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
      dtype=object)
```

Finding the Unique Values of the TYPE variable.

Five Unique Values mentioned in the TYPE variable [Payment , Transfer , CASH_OUT , Debit , CASH_IN]

Every element having it’s own specification and data as mentioned in the dataset provided.

One Hot Encoding

```
dummies = pd.get_dummies(df3['type']).drop(['CASH_IN'],axis = 1)
dummies
```

	CASH_OUT	DEBIT	PAYMENT	TRANSFER
0	0	0	1	0
1	0	0	1	0
2	0	0	0	1
3	1	0	0	0
4	0	0	1	0
...	...	...	...	...
6362615	1	0	0	0
6362616	0	0	0	1
6362617	1	0	0	0
6362618	0	0	0	1
6362619	1	0	0	0

6362620 rows × 4 columns

- Alloting Numbers to every unique value mentioned in the TYPE column.
- Separating the amount's from these unique values help in getting a clear view.
- Wherever there is an intersection in these columns 1 is added to that column.
- Rest all the columns remain 0.

Concatenation

```
df4 = pd.concat([df3,dummies],axis = 1).drop(['type'],axis = 1)
df4
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	CASH_OUT	DEBIT	PAYMENT	TRANSFER
0	1	9839.64	170136.00	160296.36	0.00	0.00	0	0	0	1	0
1	1	1864.28	21249.00	19384.72	0.00	0.00	0	0	0	1	0
2	1	181.00	181.00	0.00	0.00	0.00	1	0	0	0	1
3	1	181.00	181.00	0.00	21182.00	0.00	1	1	0	0	0
4	1	11668.14	41554.00	29885.86	0.00	0.00	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...
6362615	743	339682.13	339682.13	0.00	0.00	339682.13	1	1	0	0	0
6362616	743	6311409.28	6311409.28	0.00	0.00	0.00	1	0	0	0	1
6362617	743	6311409.28	6311409.28	0.00	68488.84	6379898.11	1	1	0	0	0
6362618	743	850002.52	850002.52	0.00	0.00	0.00	1	0	0	0	1
6362619	743	850002.52	850002.52	0.00	6510099.11	7360101.63	1	1	0	0	0

6362620 rows × 11 columns

- Concatenation help’s in giving a brief look of the entire data mentioned.
- Mainly carried out after deletion of CASH_IN column.
- Help’s in joining the table directly to the dataframe.

Model Development

Step 1) Getting in touch with the dependent and independent variables.

Step 2) All the variables mentioned in the above slide are independent except the **isFraud** variable as mentioned in the picture.

Step 3) Separating the **isFraud** variable and putting the rest of the variables together will help in getting a brief look of the data mentioned.

Step 4) Using the **train_test_split** function in this case to give a starting point in development of the particular model.

Step 5) Checking the shape of the training data as well as the df4 data mentioned.

```
X_train, X_test, y_train, y_test = train_test_split(df4.drop(['isFraud'],axis = 1),df4.isFraud,test_size = 0.2,random_state = Fal
```

```
X_train.shape
```

```
(5090096, 10)
```

```
df4.shape
```

```
(6362620, 11)
```

```
X_train.head()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	CASH_OUT	DEBIT	PAYMENT	TRANSFER
1657090	158	99986.73	7845.07	0.00	914025.09	1014011.83	1	0	0	0
3056545	234	62405.63	16603.00	0.00	466525.55	528931.18	1	0	0	0
224722	14	2291393.07	0.00	0.00	3729395.93	6020789.00	0	0	0	1
495935	20	14173.77	11447430.20	11461603.97	3047711.43	3033537.66	0	0	0	0
4412383	322	301264.36	0.00	0.00	806211.40	1107475.76	1	0	0	0

Test Method View

```
y_test.head()
```

```
4644207    0
3800666    0
4426240    0
5788765    0
2010701    0
Name: isFraud, dtype: int64
```

Dependent Variable View

Creation of the Model

```
model = LogisticRegression()
```

```
model.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
model.score(X_test,y_test)
```

```
0.9982577931732526
```

Creating the logistic regression model and getting the prediction percent of the model showing 99% perfection.

Fitting the model and checking of the score in such case.

99% prediction is mainly happening due to the unbalanced data mentioned in the dataset.

Unbalanced Dataset, Why?

```
df4['isFraud'].unique()
```

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

```
df4.groupby('isFraud').sum()
```

isFraud	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	CASH_OUT	DEBIT	PAYMENT	TRANSFER
0	1545618400	1.132337e+12	5.292133e+12	5.439183e+12	6.998877e+12	7.783676e+12	2233384	41432	2151495	528812
1	3025783	1.205642e+10	1.354872e+10	1.580121e+09	4.469922e+09	1.051024e+10	4116	0	0	4097

Finding out all the unique values from the isFraud variable column and grouping them before finding the sum.

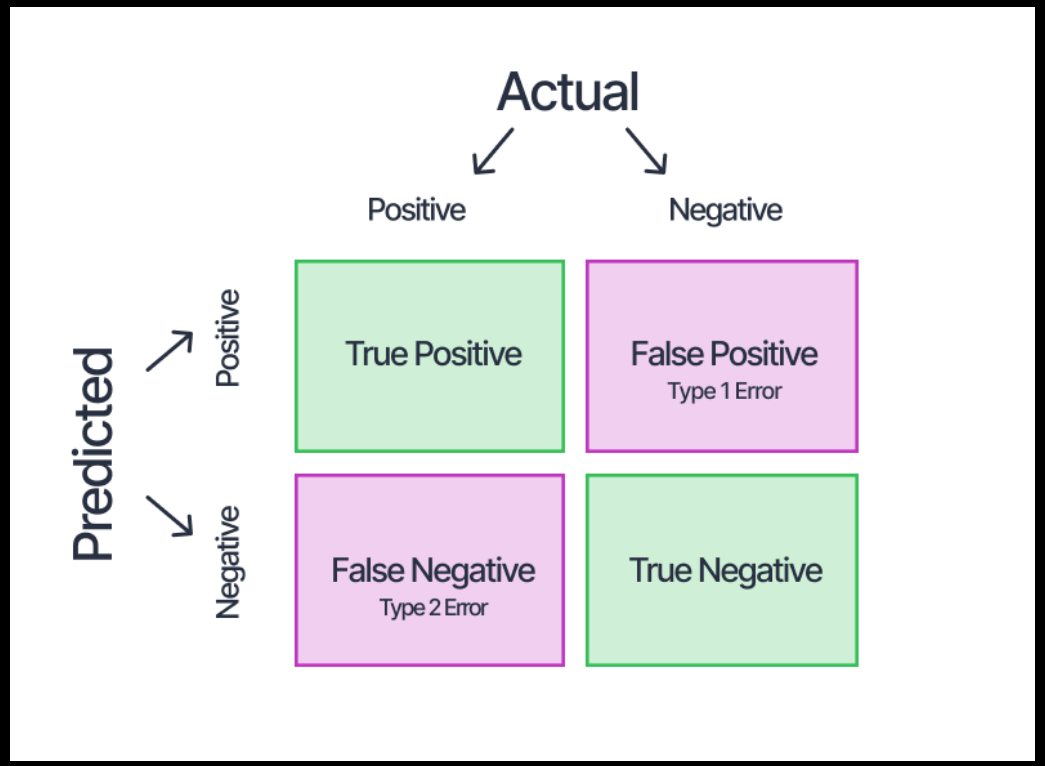
After checking the values mentioned in the STEP column we can see how disturbed and imbalanced the data is.

This show's that the prortion is too low and gives us a brief of the possibility level of a fraud taking place.

Thus the prediction of the data is 99%

Confusion Matrix

- A 2x2 table where at X axis we put the **predicted** methods whereas on the Y axis we put the **actual's** .



```
predict = model.predict(X_test)
predict
```

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

```
cm = confusion_matrix(y_test,predict)
```

Predicting the model and implementing the confusion matrix as mentioned in the above picture.

X_test dataset is stored in a variable named PREDICT.

Prediction in this case is done to use in the confusion matrix mentioned above.

Storing it in a variable named cm.

HEATMAP PLOT CREATION

```
sns.heatmap(cm,cmap='Oranges',annot=True,fmt='d',cbar=False,linecolor='Black',linewidths=5)
plt.xticks(np.arange(2)+.5,['No Fraud','Fraud'])
plt.yticks(np.arange(2)+.5,['No Fraud','Fraud'])
plt.xlabel("predicted")
plt.ylabel("actuals")
```

Creation of a heatmap plot named sns.heatmap and adding a variable cm to the particular heatmap.

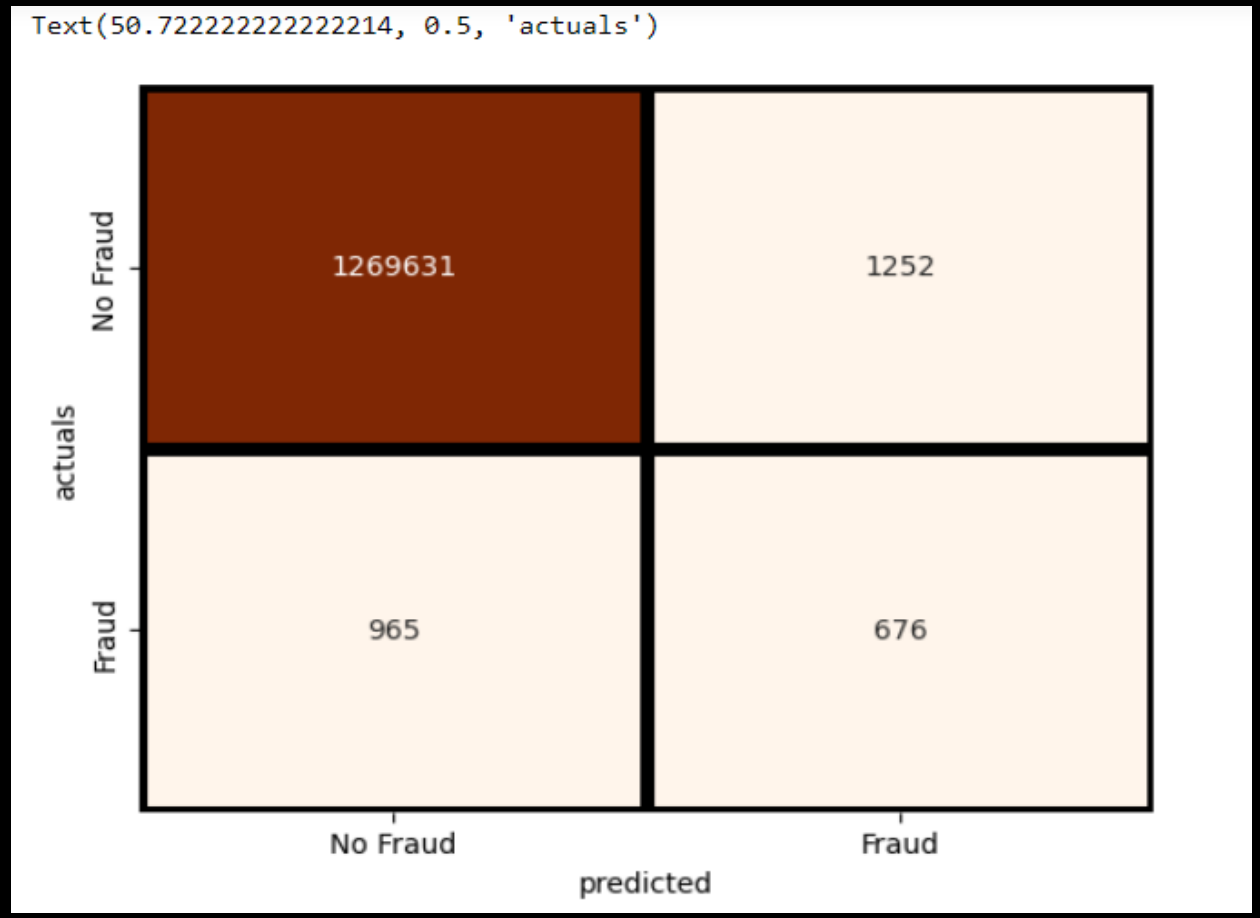
Addition of colours to the particular heatmap for it to look more accurate using cmap = ‘oranges’.

Adding number’s to the heatmap by using the annot.

To add readable numbers the fmt = ‘d’ parameter is used, in this ‘d’ stands for DIGIT.

Dropping of the column to make the heatmap look better using the cbar = false method.

Addition of the rest palettes to make the heatmap look readable.



- In the diagram generated, mostly less than 50% of the frauds are TRUE NEGATIVE.
- Whereas the rest are FALSE NEGATIVE.
- The 99% prediction is because of the highest numbered area shown in DARK BROWN colour.
- This model is mostly predicting the Data to be NO FRAUD.
- This is mainly creating the largest volume in our data thus showing the prediction of 99%.

THANK YOU

NAME – ADITYA TIKONE