Credit Card Fraud Detection

Agenda

1
Introduction

2
Data Set

3
Preprocessing

4
Handle imbalanced class problem

5
Applying Different Models Confusion Matrix

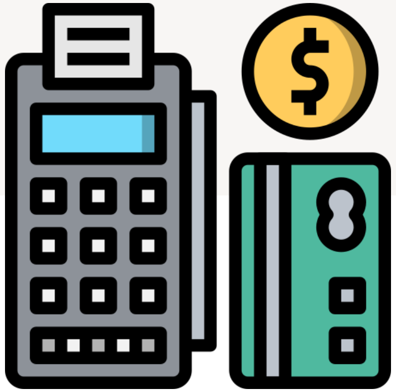
6
Thank You

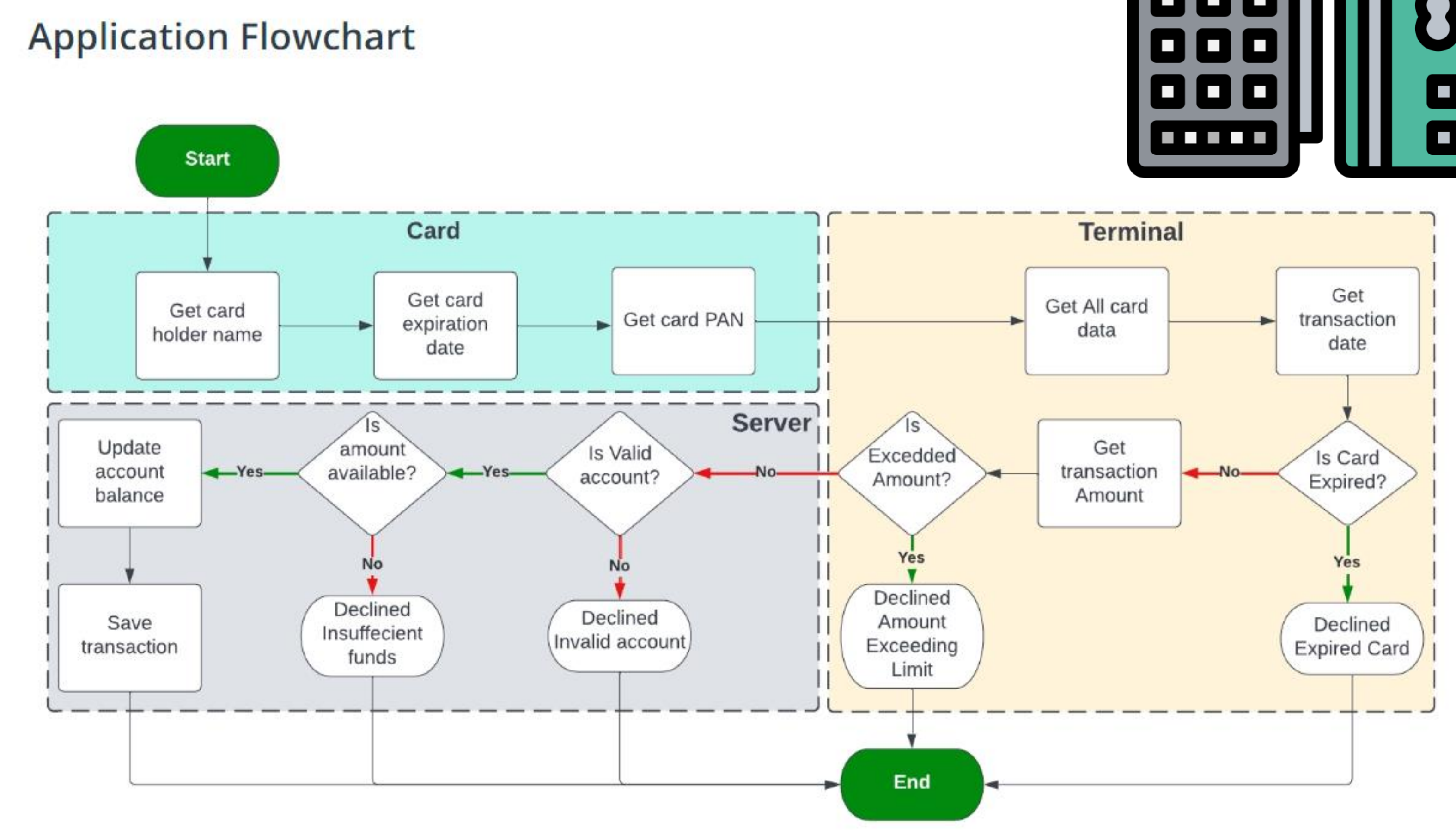
Introduction

It is important that credit card companies are able to recognize fraud credit card transactions so that customers are not charged for items that they did not purchase.

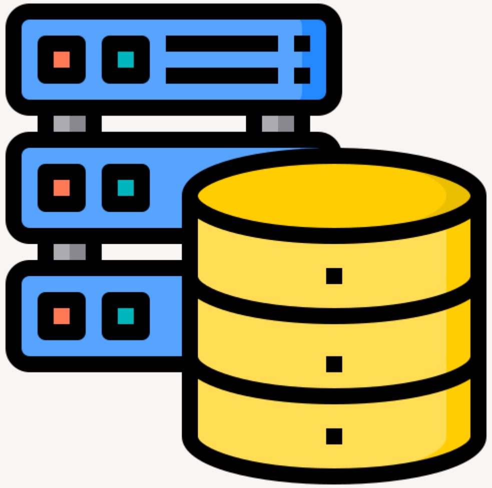
Introduction

What is fraud?





Data Set



Data Set

This dataset presents transactions that occurred in **two days**, where we have **284,807** transactions :

- **492** frauds
- **284315** legits

The dataset is highly unbalanced, the positive class (frauds) accounts for 0.172% of all transactions.

- It contains only numerical input variables which are the result of a **PCA** transformation.
- **Confidentiality issues**: we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are **'Time**' and **'Amount**'.
- Feature 'Class' is the response variable, and it takes value:
  - ❑ 1 in case of fraud
  - ❑ 0 in case of legit
- Dealing with class imbalance ratio as frauds are much less than legits

Data Set

[1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:

```
df=pd.read_csv("/kaggle/input/creditcardfraud/creditcard.csv")
```

```
df.head()
```

[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	-0.189115	0.133558	-0.021053	149.62	0
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	0.125895	-0.008983	0.014724	2.69	0
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	-0.139097	-0.055353	-0.059752	378.66	0
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	-0.221929	0.062723	0.061458	123.50	0
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	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

+ Code + Markdown

[4]:

```
df.tail()
```

[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

5 rows × 31 columns

Data Set

```
[5]: 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
```

```
[6]: #here we check for null so that we don't have any
     #missing values because if we have we will have to do more processing
     df.isnull().sum()
```

```
[6]: 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
```

```
[8]: # distribution of legit & fraud transaction
     # 0 --> Normal Transaction
     # 1 --> Fraud Transaction
     legit = df[df.Class == 0]
     fraud = df[df.Class == 1]
```

```
[9]: print(legit.shape)
     print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

Preprocessing

Preprocessing

Dropping time as it’s useless in our calculations and doesn’t fit other features.

[12]:

```
# compare the values for both transactions
df.groupby('Class').mean()
```

[12]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000024	0.000070	0.000182	-0.000072	-0.000089	-0.000295	-0.000131	88.291022
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575	0.075667	122.211321

2 rows × 30 columns

+ Code + Markdown

[13]:

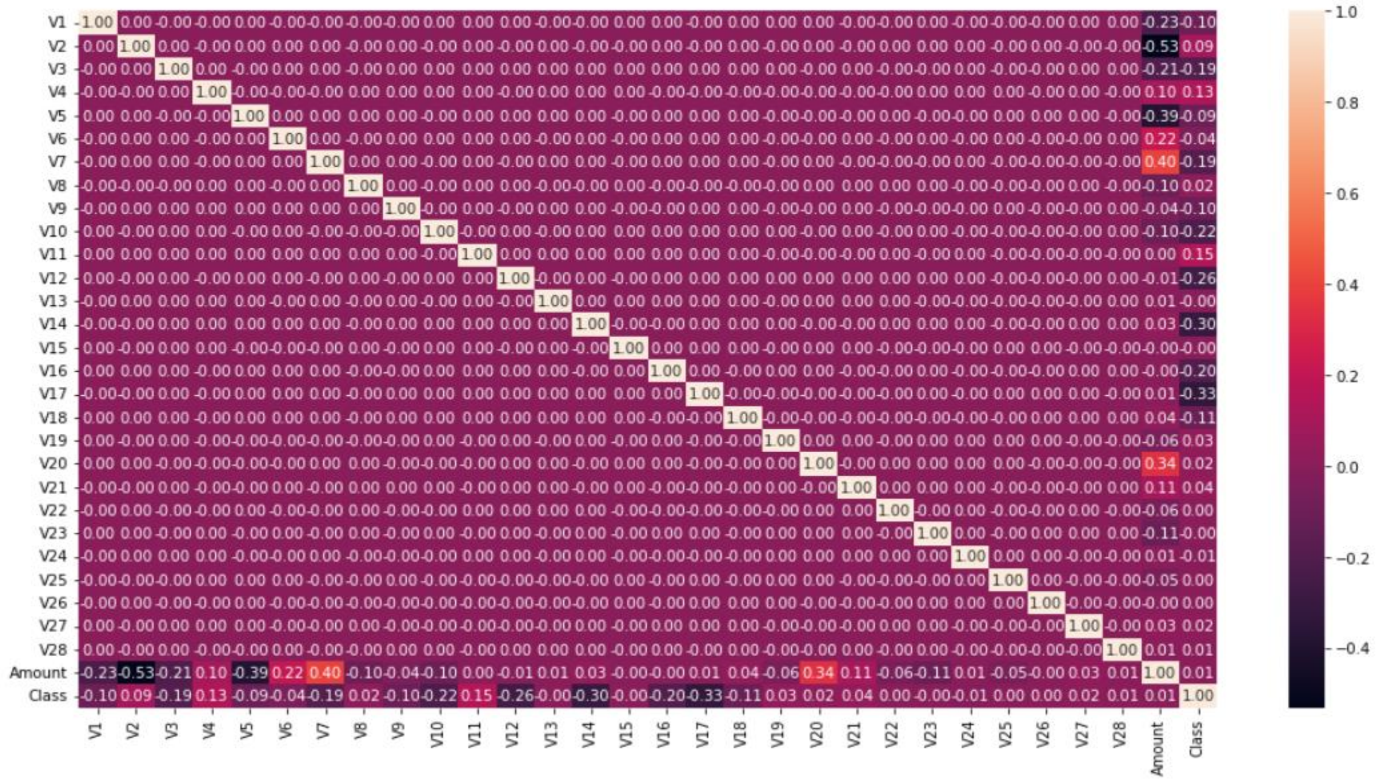
```
df.drop(['Time'], axis=1,inplace=True)
```

Preprocessing

Heat map to describe the co-relation between features so we can see if some features are connected so we can drop one of them.

```
[15]: plt.figure(figsize= (17,9))
      sns.heatmap(df.corr(),annot=True,fmt='0.2f')
```

```
[15]: <AxesSubplot:>
```



Handle imbalanced class problem

Handle an imbalanced class problem

Problems with imbalanced data classification

- How accurately are we actually predicting both majority and minority class.

  - Assume we are going to predict disease from an existing dataset where for every 100 records only 5 patients are diagnosed with the disease. So, the majority class is 95% with no disease and the minority class is only 5% with the disease. Now, assume our model predicts that all 100 out of 100 patients have no disease.
  - In the above case, it is (0+95)/(0+95+0+5)=0.95 or 95%. It means that the model fails to identify the minority class yet the accuracy score of the model will be 95%. Thus, our traditional approach of classification and model accuracy calculation is not useful in the case of the imbalanced dataset.

	Original	
Prediction	Positive	Negative
Positive	True Positive	False Positive
Negative	False Negative	True Negative

ACCURACY = TP + TN / TP+TN+FP+FN

Handle imbalanced class problem

Imbalanced classess detection

[16]:
```
# distribution of legit & fraud transaction
# 0 --> Normal Transaction
# 1 --> Fraud Transaction
df['Class'].value_counts()
```

[16]:
```
0    284315
1       492
Name: Class, dtype: int64
```

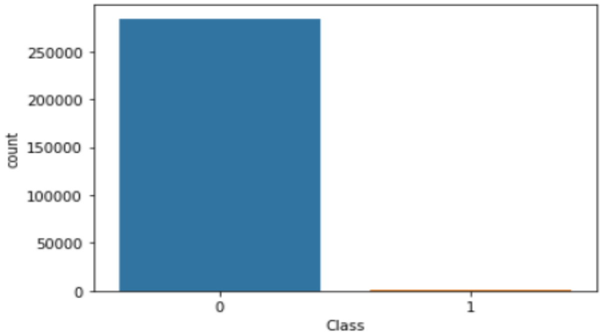
[17]:
```
df['Class'].value_counts()  / len(df)* 100
```

[17]:
```
0    99.827251
1     0.172749
Name: Class, dtype: float64
```

+ Code + Markdown

[18]:
```
sns.countplot(x='Class', data=df)
```

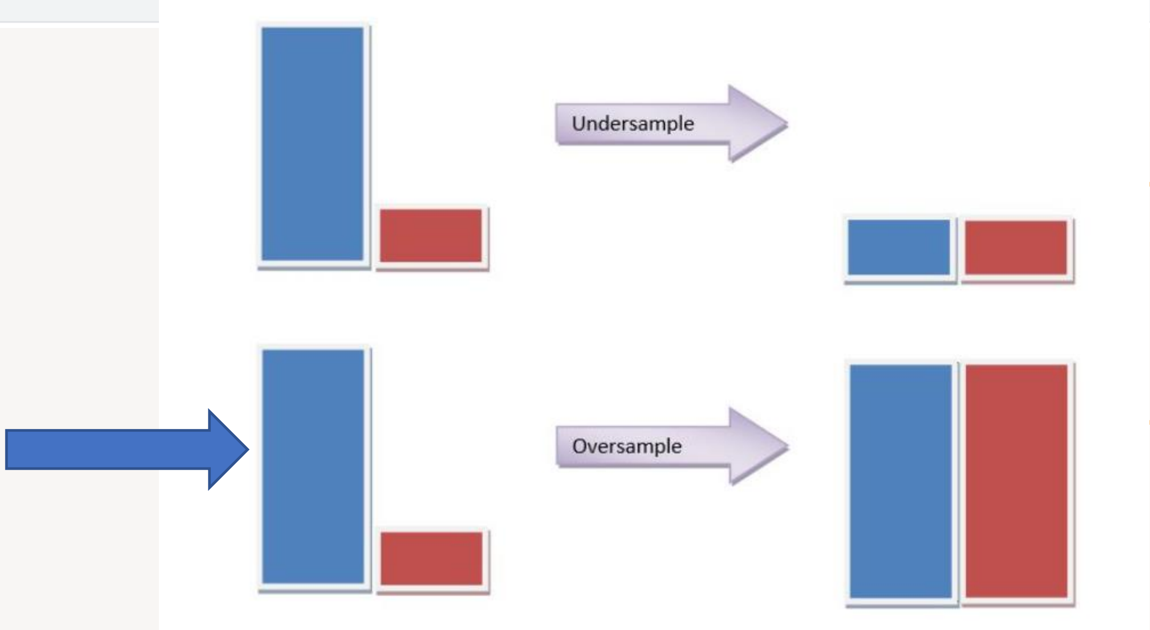
[18]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



Handle imbalanced class problem

Oversampling to handle imbalanced class problem Using **SMOTE**

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
x = df.drop('Class', axis=1) #input data
y = df['Class'] #output data

# setting up testing and training sets
#splitting data to test=25% , training=75%
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
sm = SMOTE(random_state=27)
x_train, y_train = sm.fit_resample(x_train, y_train)
x_test, y_test = sm.fit_resample(x_test, y_test)
```



Handle imbalanced class problem

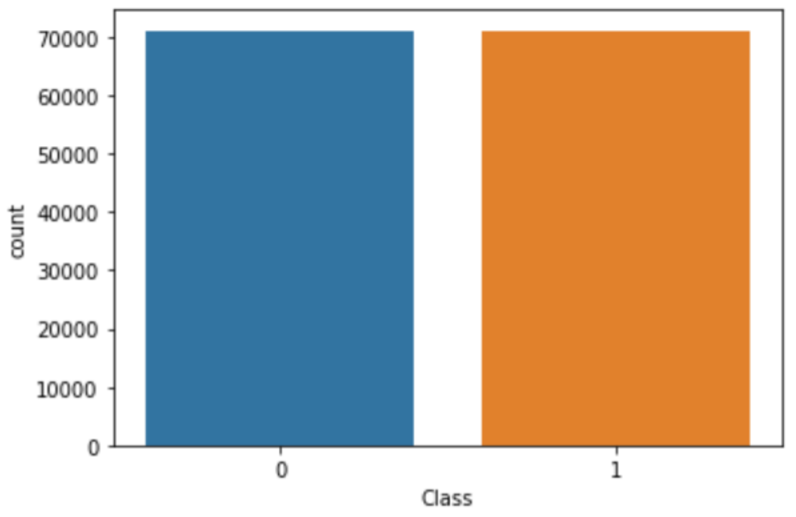
Oversampling to handle imbalanced class problem Using **SMOTE**

```
y_test.value_counts()
```

```
[392… 1    71089
      0    71089
      Name: Class, dtype: int64
```

```
[393]: sns.countplot(y_test)
```

```
[393… <AxesSubplot:xlabel='Class', ylabel='count'>
```



Applying Different Models Confusion Matrix

Applying Different Models & Confusion Matrix

1.Logistic Regression

[706]:
```
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=950)
model.fit(x_train, y_train)
```
[706… LogisticRegression(max_iter=950)

[707]:
```
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```
```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96    213226
           1       0.98      0.94      0.96    213226

    accuracy                           0.96    426452
   macro avg       0.96      0.96      0.96    426452
weighted avg       0.96      0.96      0.96    426452
```

[708]:
```
#evaluation on test data
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```
```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     71089
           1       0.98      0.95      0.96     71089

    accuracy                           0.96    142178
   macro avg       0.97      0.96      0.96    142178
weighted avg       0.97      0.96      0.96    142178
```

+ Code + Markdown

[709]:
```
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = LogisticRegression(max_iter=950)
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

[710]:
```
print(confusion_matrix(y_test,y_pre))
```
```
[[69724  1365]
 [ 3630 67459]]
```

1.Logistic Regression

```
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train, y_train)
```
[750… LogisticRegression()

[751]:
```
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```
```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96    213226
           1       0.98      0.94      0.96    213226

    accuracy                           0.96    426452
   macro avg       0.96      0.96      0.96    426452
weighted avg       0.96      0.96      0.96    426452
```

[752]:
```
#evaluation on test data
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```
```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     71089
           1       0.98      0.95      0.96     71089

    accuracy                           0.97    142178
   macro avg       0.97      0.97      0.97    142178
weighted avg       0.97      0.97      0.97    142178
```

[753]:
```
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = LogisticRegression()
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

[754]:
```
print(confusion_matrix(y_test,y_pre))
```
```
[[69690  1399]
 [ 3548 67541]]
```

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Applying Different Models & Confusion Matrix

2.Passive Aggressive Classifier

[755]:
```
from sklearn.linear_model import PassiveAggressiveClassifier
model = PassiveAggressiveClassifier()
model.fit(x_train, y_train)
```

[755… PassiveAggressiveClassifier()

[756]:
```
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91    213226
           1       0.95      0.85      0.90    213226

    accuracy                           0.90    426452
   macro avg       0.91      0.90      0.90    426452
weighted avg       0.91      0.90      0.90    426452
```

[757]:
```
#evaluation on test
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.83      0.95      0.89     71089
           1       0.94      0.81      0.87     71089

    accuracy                           0.88    142178
   macro avg       0.89      0.88      0.88    142178
weighted avg       0.89      0.88      0.88    142178
```

[758]:
```
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = PassiveAggressiveClassifier()
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

[759]:
```
print(confusion_matrix(y_test,y_pre))
```

```
[[69127  1962]
 [26764 44325]]
```

2.Passive Aggressive Classifier

[711]:
```
from sklearn.linear_model import PassiveAggressiveClassifier
model = PassiveAggressiveClassifier(max_iter=950)
model.fit(x_train, y_train)
```

[711… PassiveAggressiveClassifier(max_iter=950)

[712]:
```
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.83      0.99      0.90    213226
           1       0.98      0.80      0.88    213226

    accuracy                           0.89    426452
   macro avg       0.91      0.89      0.89    426452
weighted avg       0.91      0.89      0.89    426452
```

[713]:
```
#evaluation on test
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.82      0.99      0.89     71089
           1       0.98      0.78      0.87     71089

    accuracy                           0.88    142178
   macro avg       0.90      0.88      0.88    142178
weighted avg       0.90      0.88      0.88    142178
```

[714]:
```
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = PassiveAggressiveClassifier(max_iter=950)
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

[715]:
```
print(confusion_matrix(y_test,y_pre))
```

```
[[45967 25122]
 [ 2207 68882]]
```

Applying Different Models & Confusion Matrix

3.Perceptron

+ Code + Markdown

```
[760]:
from sklearn.linear_model import Perceptron
model = Perceptron()
model.fit(x_train, y_train)
```

```
[760… Perceptron()
```

+ Code + Markdown

```
[761]:
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96    213226
           1       0.96      0.96      0.96    213226

    accuracy                           0.96    426452
   macro avg       0.96      0.96      0.96    426452
weighted avg       0.96      0.96      0.96    426452
```

```
[762]:
#evaluation on test
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     71089
           1       0.96      0.96      0.96     71089

    accuracy                           0.96    142178
   macro avg       0.96      0.96      0.96    142178
weighted avg       0.96      0.96      0.96    142178
```

```
[763]:
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = Perceptron()
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

```
print(confusion_matrix(y_test,y_pre))
```

```
[[68347  2742]
 [ 3188 67901]]
```

3.Perceptron

+ Code + Markdown

```
from sklearn.linear_model import Perceptron
model = Perceptron(alpha=1.0,max_iter=950)
model.fit(x_train, y_train)
```

```
[716… Perceptron(alpha=1.0, max_iter=950)
```

```
[717]:
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96    213226
           1       0.96      0.96      0.96    213226

    accuracy                           0.96    426452
   macro avg       0.96      0.96      0.96    426452
weighted avg       0.96      0.96      0.96    426452
```

```
[718]:
#evaluation on test
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     71089
           1       0.96      0.96      0.96     71089

    accuracy                           0.96    142178
   macro avg       0.96      0.96      0.96    142178
weighted avg       0.96      0.96      0.96    142178
```

```
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = Perceptron(alpha=1.0,max_iter=950)
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

```
[720]:
print(confusion_matrix(y_test,y_pre))
```

```
[[68347  2742]
 [ 3188 67901]]
```

Applying Different Models & Confusion Matrix

4.RidgeClassifier

[765]:
```
from sklearn.linear_model import RidgeClassifier
model = RidgeClassifier()
model.fit(x_train, y_train)
```

[765… RidgeClassifier()

[766]:
```
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.93    213226
           1       0.99      0.85      0.92    213226

    accuracy                           0.92    426452
   macro avg       0.93      0.92      0.92    426452
weighted avg       0.93      0.92      0.92    426452
```

[767]:
```
#evaluation on test
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     71089
           1       0.99      0.88      0.93     71089

    accuracy                           0.94    142178
   macro avg       0.94      0.94      0.94    142178
weighted avg       0.94      0.94      0.94    142178
```

[768]:
```
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = RidgeClassifier()
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

[769]:
```
print(confusion_matrix(y_test,y_pre))
```

```
[[70273   816]
 [ 8300 62789]]
```

4.RidgeClassifier

```
from sklearn.linear_model import RidgeClassifier
model = RidgeClassifier(tol=1e-12,alpha=0.9,max_iter=950)
model.fit(x_train, y_train)
```

[721… RidgeClassifier(alpha=0.9, max_iter=950, tol=1e-12)

[722]:
```
#evaluation on train data
y_pre=model.predict(x_train)
print(classification_report(y_train, y_pre))
```

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.93    213226
           1       0.99      0.85      0.92    213226

    accuracy                           0.92    426452
   macro avg       0.93      0.92      0.92    426452
weighted avg       0.93      0.92      0.92    426452
```

[723]:
```
#evaluation on test
y_pre=model.predict(x_test)
print(classification_report(y_test, y_pre))
```

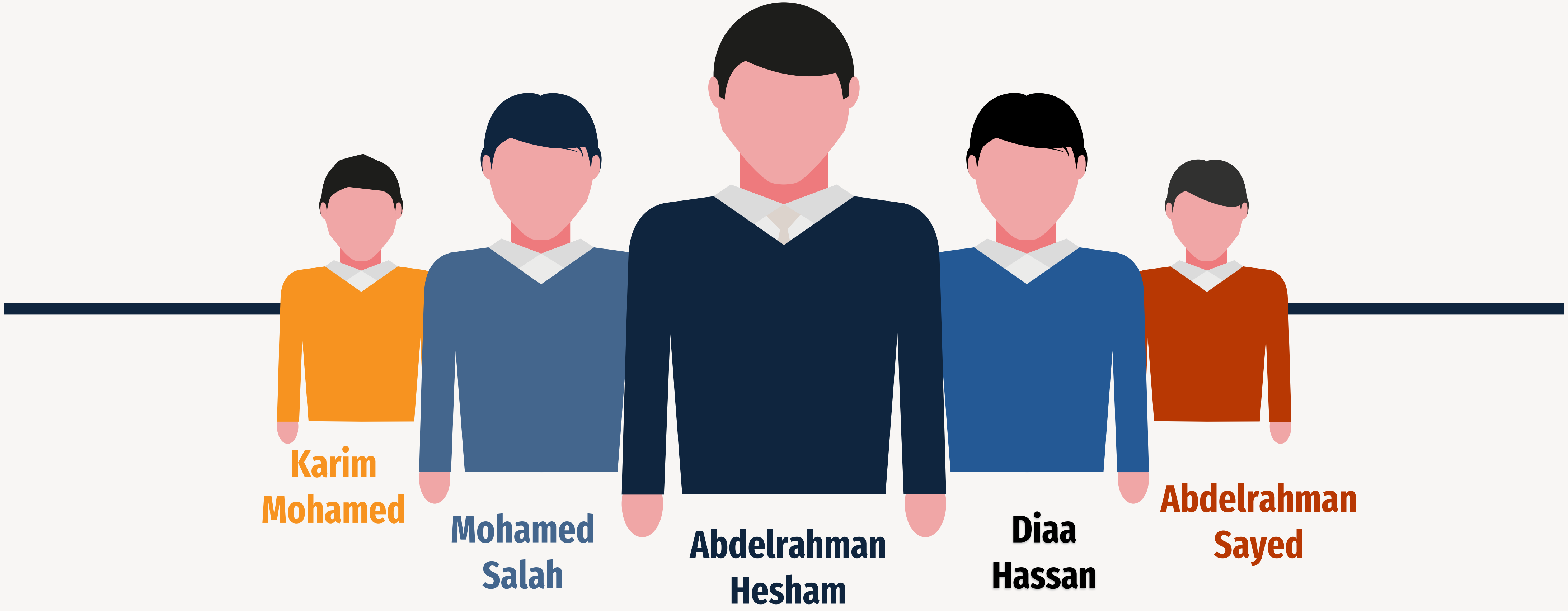
```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     71089
           1       0.99      0.88      0.93     71089

    accuracy                           0.94    142178
   macro avg       0.94      0.94      0.94    142178
weighted avg       0.94      0.94      0.94    142178
```

[724]:
```
#confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
model = RidgeClassifier(tol=1e-12,alpha=0.9,max_iter=950)
model.fit(x_train, y_train)
y_pre=model.predict(x_test)
```

```
print(confusion_matrix(y_test,y_pre))
```

```
[[70273   816]
 [ 8300 62789]]
```

Our Team


Karim Mohamed
Mohamed Salah
Abdelrahman Hesham
Diaa Hassan
Abdelrahman Sayed

Thank you