Problem Statement:

In the banking industry, credit card fraud detection using machine learning is not only a trend but a necessity for them to put proactive monitoring and fraud prevention mechanisms in place. Machine learning is helping these institutions to reduce time-consuming manual reviews, costly chargebacks and fees as well as denials of legitimate transactions. In this project we will detect fraudulent credit card transactions with the help of Machine learning models. We will analyse customer-level data that has been collected and analysed. the main objective of this project is detect fraudulent transactions with the help of credit card details.

import libraries

In [28]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
```

Data Gathering

In [29]:
```
df= pd.read_csv("creditcard.csv")
df.head()
```

Out[29]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3637
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2554
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5146
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3870
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8177

In [30]:
```
pd.options.display.max_columns = None # it will show all columns
```

EDA

In [31]:
```
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
```

In [32]:

```
df.isnull().sum()
```

Out[32]:
```
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
```

In [33]:
```python
df.describe()
```

Out[33]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

In [34]:
```python
df.shape
```

Out[34]:
```
(284807, 31)
```

In [35]:
```python
print("Number of rows are >>",df.shape[0])
print("Number of columns are>>",df.shape[1])
```

```
Number of rows are >> 284807
Number of columns are>> 31
```

Data engineering

In [36]:
```
from sklearn.preprocessing import StandardScaler
```

In [37]:
```
sc = StandardScaler()
df["Amount"]=sc.fit_transform(pd.DataFrame(df["Amount"]))
```

In [38]:
```
df.head()
```

Out[38]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3637
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2554
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5146
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3870
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8177

In [39]:
```
df=df.drop("Time",axis=1)
```

In [40]:
```
df.head()
```

Out[40]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.

In [41]:
```
# lets check our dataset contain duplicate values
df.duplicated().any()
```

Out[41]:
```
True
```

In [42]:
```
df=df.drop_duplicates()
```

In [43]:
```
df.duplicated().any()
```

Out[43]:
```
False
```

In [44]:
```
df.shape
```

Out[44]:
```
(275663, 30)
```

In [ ]:

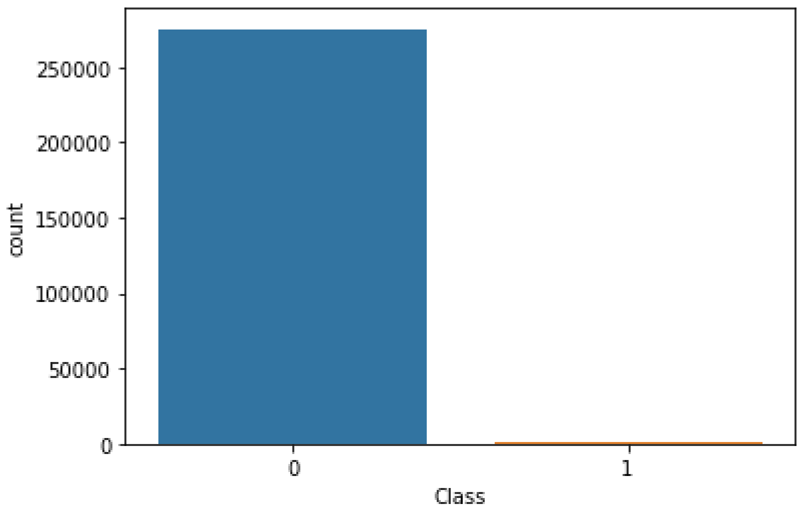
In [45]:
```
# lets check whether our data is balanced or not
```

In [46]:
```
df["Class"].value_counts()
```

Out[46]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [47]:
```
sns.countplot(df["Class"])
```

Out[47]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



Training and testing of data

In [48]:
```
x=df.drop("Class",axis=1)
y=df["Class"]
```

In [49]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,stratify=y)
```

Handling imbalanced data

In [50]:
```
from imblearn.over_sampling import SMOTE
```

In [51]:
```
smt = SMOTE(k_neighbors=5,random_state=45)
x_sampled,y_sampled=smt.fit_resample(x,y)
y_sampled.value_counts()
```

Out[51]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

In [55]:
```
y_sampled.value_counts()
```

Out[55]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

In [56]:
```
x_sampled.value_counts()
```

Out[56]:
```
V1          V2          V3          V4          V5          V6          V7          V8
V9          V10         V11         V12         V13         V14         V15         V16
V17         V18         V19         V20         V21         V22         V23         V24
V25         V26         V27         V28         Amount
-56.407510  -72.715728  -6.605265   16.491217   34.801666  -26.160506  -19.399981  -
1.501300    6.967698    9.537780    3.089395    1.776452    3.732744   -2.530792    5.78451
4   3.903988  -1.929314   0.206699    2.805883  -12.360962  -6.266878  -1.272167   7.8
93082   0.767805    5.376595   0.163672  -8.358317   33.847808    4.451791    1
 0.210065    3.673594   -5.939243    6.069659    1.357794   -2.333811   -1.008120
0.426847  -3.778337  -4.638134   5.195455  -5.874979  -2.496669  -10.541567   0.04370
4  -0.876309   0.328050   0.911202  -1.566508   0.347403    0.378945  -0.348357  -0.4
51519  -0.497239   1.047773   0.547369   0.555570   0.377787   -0.350447    1
 0.210190    0.376752   -0.628059    1.610832    1.154689   -0.509063    0.277630   -
0.092298  -0.880588   0.858676  -0.065022  -0.256592  -0.136804  -0.151135    0.05668
5  -0.226301   0.555460  -0.090705   1.443113   0.162963    0.053548   0.230794   0.2
24975   0.727468  -0.940203   1.612213  -0.074546   0.096378   -0.329432    1
 0.210189    1.182290    0.320918    2.879797    1.418419   -0.485918    1.434163   -
0.387686  -1.559430   1.077658  -1.519936  -0.951400  -0.674375   0.141278   -1.48313
6   0.416870  -0.827278  -0.314274  -1.754266  -0.208021    0.238386   0.826897  -0.3
57316  -0.012981   0.454867   0.221242  -0.268452  -0.318921   -0.323809    1
 0.210179    1.089803   -1.957672  -0.433683    0.992292   -1.093528    0.990245
0.199385  -0.690934  -0.884654   0.731512   0.198398  -1.020869  -0.088691   -1.36881
4   0.171193   0.600599   0.510005  -0.088766  -0.302418    0.232839   0.568465  -0.0
02169   0.711447  -0.348031   0.493986  -0.167354  -0.055372   -0.260674    1

..
-1.673558   -1.368263    0.975668  -1.900789    2.129729    3.614421   -1.116419
1.081276  -0.747162   0.008210  -0.917064  -0.329076   0.185192  -0.592586   -0.24006
1  -0.819964  -0.634553   1.620631  -0.850185  -0.428966   -0.477222  -0.979069  -0.1
94489   1.015702   0.295645   0.352492   0.034362   0.075749    0.102472    1
-1.673574    3.359726   -3.885016    2.704382   -1.984990   -2.728860   -3.450300
1.391066  -0.750002  -6.147363   4.380344  -8.239907   0.358676  -6.598172    0.31106
6  -3.063004  -4.194180  -1.206355  -0.185158   0.366676    0.458317  -0.552484   0.0
75773   0.462209  -0.044226   0.306312   0.534047   0.238698   -0.349231    1
-1.673577    3.354635   -3.888617    2.685783   -1.965664   -2.688006   -3.454074
1.397930  -0.764230  -6.131050   4.420122  -8.195404   0.404094  -6.573929    0.27561
5  -3.053412  -4.241402  -1.188935  -0.155479   0.370232    0.459960  -0.547497   0.0
70226   0.425582  -0.038336   0.307724   0.533759   0.237436   -0.349231    1
-1.673590    0.572325    0.621793  -1.932534   -0.431970   -0.923449   -0.322573
1.008947   0.577738  -1.931596   0.921290   0.935435  -1.730317   1.175463   -0.29424
8  -0.463194   0.218518  -0.031859  -0.110530  -0.578605    0.078727  -0.042385  -0.1
64784   0.181623  -0.084221  -1.047195  -0.165178  -0.118881   -0.349231    1
 2.454930   -0.989065   -2.512114  -1.877104    0.081287   -0.831825   -0.240601   -
0.467361  -1.949390   1.737065  -1.553888  -1.361226   0.249681   0.210405   -0.04982
0  -0.857286   0.362569  -0.126316   0.143420  -0.437697   -0.074210   0.247125  -0.0
37124  -0.075137   0.445896   0.102445  -0.056362  -0.079442   -0.321245    1
Length: 550380, dtype: int64
```

In [59]:
```
x_train,x_test,y_train,y_test=train_test_split(x_sampled,y_sampled,test_size=0.20,ranc
```

Model evaluation

1) logistic regression

In [60]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import f1_score,recall_score,accuracy_score,precision_score,confu
```

In [61]:
```
log_model = LogisticRegression()
log_model.fit(x_train,y_train)
```

Out[61]:
```
▾ LogisticRegression
LogisticRegression()
```

In [62]:
```
# Training
y_pred_train = log_model.predict(x_train)
cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("confusion matrix\n",cnf_matrix)
print("*"*20)
accuracy = accuracy_score(y_train,y_pred_train)
print("accuracy",accuracy)
print("*"*20)
clf_report = classification_report(y_train,y_pred_train)
print("classificatio report\n",clf_report)
```

```
confusion matrix
 [[214713   5556]
 [ 18579 201456]]
********************
accuracy 0.9451855990406628
********************
classificatio report
               precision    recall  f1-score   support

           0       0.92      0.97      0.95    220269
           1       0.97      0.92      0.94    220035

    accuracy                           0.95    440304
   macro avg       0.95      0.95      0.95    440304
weighted avg       0.95      0.95      0.95    440304
```

In [63]:
```
# Testing
y_pred = log_model.predict(x_test)
cnf_matrix = confusion_matrix(y_test,y_pred)
print("confusion matrix\n",cnf_matrix)
print("*"*20)
accuracy = accuracy_score(y_test,y_pred)
print("accuracy",accuracy)
print("*"*20)
clf_report = classification_report(y_test,y_pred)
print("classificatio report\n",clf_report)
```

```
confusion matrix
 [[53463  1458]
 [ 4685 50470]]
********************
accuracy 0.9441931029470547
********************
classificatio report
               precision    recall  f1-score   support

           0       0.92      0.97      0.95     54921
           1       0.97      0.92      0.94     55155

    accuracy                           0.94    110076
   macro avg       0.95      0.94      0.94    110076
weighted avg       0.95      0.94      0.94    110076
```

In [ ]:

Decision Tree Classifier

In [64]:
```
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
```

Model Selection

In [65]:
```
dt_clf=DecisionTreeClassifier()
dt_clf.fit(x_train,y_train)
```

Out[65]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Model Evaluation

In [66]:
```
# training
y_pred_train = dt_clf.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion Matrix\n",cnf_matrix)

Accuracy = accuracy_score(y_train,y_pred_train)
print("ACCURACY",Accuracy*100)

cls_report = classification_report(y_train,y_pred_train)
print("CLASSIFICATION REPORT\n",cls_report)
```

```
Confusion Matrix
 [[220269      0]
 [     0 220035]]
ACCURACY 100.0
CLASSIFICATION REPORT
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    220269
           1       1.00      1.00      1.00    220035

    accuracy                           1.00    440304
   macro avg       1.00      1.00      1.00    440304
weighted avg       1.00      1.00      1.00    440304
```

In [ ]:

In [67]:

```python
# testing
y_pred_test = dt_clf.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred_test)
print("Confusion Matrix\n",cnf_matrix)

Accuracy = accuracy_score(y_test,y_pred_test)
print("ACCURACY",Accuracy*100)

clf_report = classification_report(y_test,y_pred_test)
print("CLASSIFICATION REPORT\n",clf_report)
```

```
Confusion Matrix
 [[54775   146]
 [   47 55108]]
ACCURACY 99.8246665939896
CLASSIFICATION REPORT
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     54921
           1       1.00      1.00      1.00     55155

    accuracy                           1.00    110076
   macro avg       1.00      1.00      1.00    110076
weighted avg       1.00      1.00      1.00    110076
```

In [ ]:

Hyper parameter Tuning

In [68]:

```python
dt_model = DecisionTreeClassifier()
hyper_para = {"criterion" :['gini',"entropy"],
"max_depth":np.arange(2,8),
"min_samples_split":np.arange(3,20),
"min_samples_leaf":np.arange(3,15),
}
rscv_dt_clf = RandomizedSearchCV(dt_model,hyper_para,cv=5)
rscv_dt_clf.fit(x_train,y_train)
```

Out[68]:

```
RandomizedSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [71]:

```
dt_tuning=rscv_dt_clf.best_estimator_
```

In [72]:

```
dt_tuning
```

Out[72]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7, min_samples_leaf=4, min_samples_split=1
0)
```

In [75]:

```
dt_clf = DecisionTreeClassifier(max_depth=7, min_samples_leaf=4, min_samples_split=10)
dt_clf.fit(x_train,y_train)
```

Out[75]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7, min_samples_leaf=4, min_samples_split=1
0)
```

In [76]:

```
#training
y_pred_train = dt_clf.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)

Accuaracy = accuracy_score(y_train,y_pred_train)
print("Accuracy",Accuracy)

clf_report = classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[215460   4809]
 [  7659 212376]]
Accuracy 0.9982466659398961
Classification report
               precision    recall  f1-score   support

           0       0.97      0.98      0.97    220269
           1       0.98      0.97      0.97    220035

    accuracy                           0.97    440304
   macro avg       0.97      0.97      0.97    440304
weighted avg       0.97      0.97      0.97    440304
```

In [ ]:

In [77]:

```
#testing
y_pred_test = dt_clf.predict(x_test)
```

```
cnf_matrix = confusion_matrix(y_test,y_pred_test)
print("Confusion matrix\n",cnf_matrix)

Accuaracy = accuracy_score(y_test,y_pred_test)
print("Accuracy",Accuracy)

clf_report = classification_report(y_test,y_pred_test)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[53678  1243]
 [ 1954 53201]]
Accuracy 0.9982466659398961
Classification report
               precision    recall  f1-score   support

           0       0.96      0.98      0.97     54921
           1       0.98      0.96      0.97     55155

    accuracy                           0.97    110076
   macro avg       0.97      0.97      0.97    110076
weighted avg       0.97      0.97      0.97    110076
```

In [ ]:

Random Forest classifier

In [78]:

```
from sklearn.ensemble import RandomForestClassifier
```

Model training

In [79]:

```
rf_clf= RandomForestClassifier()
rf_clf.fit(x_train,y_train)
```

Out[79]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [80]:

```
#Training
y_pred_train = rf_clf.predict(x_train)

cnf_metrics = confusion_matrix(y_train,y_pred_train)
print("confusion metrics\n",cnf_metrics)

accuracy = accuracy_score(y_train,y_pred_train)
print("accuracy",accuracy*100)

clf_report = classification_report(y_train,y_pred_train)
print("clf_report\n",clf_report)
```

```
confusion metrics
 [[220269      0]
 [     0 220035]]
accuracy 100.0
clf_report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    220269
           1       1.00      1.00      1.00    220035

    accuracy                           1.00    440304
   macro avg       1.00      1.00      1.00    440304
weighted avg       1.00      1.00      1.00    440304
```

In [ ]:

In [81]:

```
#Testing
y_pred_test = rf_clf.predict(x_test)

cnf_metrics = confusion_matrix(y_test,y_pred_test)
print("confusion metrics\n",cnf_metrics)

accuracy = accuracy_score(y_test,y_pred_test)
print("accuracy",accuracy*100)

clf_report = classification_report(y_test,y_pred_test)
print("clf_report\n",clf_report)
```

```
confusion metrics
 [[54901    20]
 [    0 55155]]
accuracy 99.98183073512845
clf_report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     54921
           1       1.00      1.00      1.00     55155

    accuracy                           1.00    110076
   macro avg       1.00      1.00      1.00    110076
weighted avg       1.00      1.00      1.00    110076
```

HyperParamter Tuning

In [82]:

```
hyperparamter = {"n_estimators":np.arange(10,20),
"criterion":["gini","entropy"],
"max_depth" :np.arange(4,10),
"min_samples_split":np.arange(3,10),
"min_samples_leaf":np.arange(3,10),
"max_features":["sqrt", "log2"],
"random_state":[41,42,43,44,45],
"oob_score":[True]}
rdscv = RandomizedSearchCV(rf_clf,hyperparamter,cv=4)
rdscv.fit(x_train,y_train)
```

Out[82]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

In [83]:

```
r_tuning=rdscv.best_estimator_
```

In [84]:

```
r_tuning
```

Out[84]:

```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=9, max_features='log
2',
                       min_samples_leaf=6, min_samples_split=4, n_estimators
=19,
                       oob_score=True, random_state=44)
```

In [85]:

```
rf_clf= RandomForestClassifier(criterion='entropy', max_depth=9, max_features='log2',
                             min_samples_leaf=6, min_samples_split=4, n_estimators=19,
                             oob_score=True, random_state=44)
```

In [86]:

```
rf_clf.fit(x_train,y_train)
```

Out[86]:

```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=9, max_features='log
2',
                       min_samples_leaf=6, min_samples_split=4, n_estimators
=19,
                       oob_score=True, random_state=44)
```

In [87]:

```
# training
y_pred_train = rf_clf.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusioin Matrix\n",cnf_matrix)

Accuracy=accuracy_score(y_train,y_pred_train)
print("Accuracy",Accuracy*100)

clf_report=classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusioin Matrix
 [[219861    408]
 [  8044 211991]]
Accuracy 98.08041716632145
Classification report
               precision    recall  f1-score   support

           0       0.96      1.00      0.98    220269
           1       1.00      0.96      0.98    220035

    accuracy                           0.98    440304
   macro avg       0.98      0.98      0.98    440304
weighted avg       0.98      0.98      0.98    440304
```

In [ ]:

In [88]:

```
# testing
y_pred_test = rf_clf.predict(x_test)

cnf_matrix=confusion_matrix(y_test,y_pred_test)
print("Confusion Matrix\n",cnf_matrix)

Accuracy=accuracy_score(y_test,y_pred_test)
print("Accuracy",Accuracy*100)

clf_report=classification_report(y_test,y_pred_test)
print("Classification report\n",clf_report)
```

```
Confusion Matrix
 [[54781   140]
 [ 2036 53119]]
Accuracy 98.02318398197609
Classification report
               precision    recall  f1-score   support

           0       0.96      1.00      0.98     54921
           1       1.00      0.96      0.98     55155

    accuracy                           0.98    110076
   macro avg       0.98      0.98      0.98    110076
weighted avg       0.98      0.98      0.98    110076
```

In [ ]:

SVM

In [89]:

```
from sklearn.svm import SVC
```

In [90]:

```
svc_model = SVC()
svc_model.fit(x_train,y_train)
```

Out[90]:

```
▾ SVC
SVC()
```

In [ ]:

Evaluation

In [91]:

```
#Training Data
y_pred_train = svc_model.predict(x_train)

cnf_metrix = confusion_matrix(y_train,y_pred_train)
print("confusion matrix\n",cnf_metrix)

accuracy = accuracy_score(y_train,y_pred_train)
print("accuarcy",accuracy*100)

clf_report = classification_report(y_train,y_pred_train)
print("classification report",clf_report)
```

```
confusion matrix
 [[216609   3660]
 [  4927 215108]]
accuarcy 98.04975653185072
classification report               precision    recall  f1-score   support

           0       0.98      0.98      0.98    220269
           1       0.98      0.98      0.98    220035

    accuracy                           0.98    440304
   macro avg       0.98      0.98      0.98    440304
weighted avg       0.98      0.98      0.98    440304
```

In [92]:

```
#Testing Data
y_pred = svc_model.predict(x_test)

cnf_metrix = confusion_matrix(y_test,y_pred)
print("confusion matrix\n",cnf_metrix)

accuracy = accuracy_score(y_test,y_pred)
print("accuarcy",accuracy*100)

clf_report = classification_report(y_test,y_pred)
print("classification report",clf_report)
```

```
confusion matrix
 [[53958   963]
 [ 1268 53887]]
accuarcy 97.97321850357935
classification report               precision    recall  f1-score   support

           0       0.98      0.98      0.98     54921
           1       0.98      0.98      0.98     55155

    accuracy                           0.98    110076
   macro avg       0.98      0.98      0.98    110076
weighted avg       0.98      0.98      0.98    110076
```

Adaboost classifier

In [93]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [94]:
```
ada_clf = AdaBoostClassifier()
ada_clf.fit(x_train,y_train)
```

Out[94]:
```
▾ AdaBoostClassifier
AdaBoostClassifier()
```

In [95]:
```
#training
y_pred_train = ada_clf.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion Matrix\n",cnf_matrix)

clf_report = classification_report(y_train,y_pred_train)
print("Classification Report\n",clf_report)

Accuracy = accuracy_score(y_train,y_pred_train)
print("ACCURACY",Accuracy*100)
```

```
Confusion Matrix
 [[215064   5205]
 [ 11295 208740]]
Classification Report
               precision    recall  f1-score   support

           0       0.95      0.98      0.96    220269
           1       0.98      0.95      0.96    220035

    accuracy                           0.96    440304
   macro avg       0.96      0.96      0.96    440304
weighted avg       0.96      0.96      0.96    440304

ACCURACY 96.2525891202442
```

In [ ]:

In [96]:
```
# testing
y_pred_test =ada_clf.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred_test)
print("Confusion Matrix\n",cnf_matrix)

clf_report = classification_report(y_test,y_pred_test)
print("Classification Report\n",clf_report)

Accuracy = accuracy_score(y_test,y_pred_test)
print("ACCURACY",Accuracy*100)
```

```
Confusion Matrix
 [[53555  1366]
 [ 2884 52271]]
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.98      0.96     54921
           1       0.97      0.95      0.96     55155

    accuracy                           0.96    110076
   macro avg       0.96      0.96      0.96    110076
weighted avg       0.96      0.96      0.96    110076

ACCURACY 96.13903121479704
```

In [ ]:

Lets see the accuracy we've got by the models

In [98]:

```
ACCURACY_df = pd.DataFrame({"MODEL":["Logostic Regression","Decision tree","Decision t
                            "Random Forest","Random Forest with Hyperparameter","SVM",
                            "Adaboost classifier"],
                   "Training Accuracy":[94.51,100.0,99.82,100.0,98.08,98.04,96.25],
                   "Testing Accuracy":[94.41,99.82,99.82,99.98,98.02,97.97,96.13],
                   })
ACCURACY_df
```

Out[98]:

	MODEL	Training Accuracy	Testing Accuracy
0	Logostic Regression	94.51	94.41
1	Decision tree	100.00	99.82
2	Decision tree with Hyperparameter	99.82	99.82
3	Random Forest	100.00	99.98
4	Random Forest with Hyperparameter	98.08	98.02
5	SVM	98.04	97.97
6	Adaboost classifier	96.25	96.13

In [117...

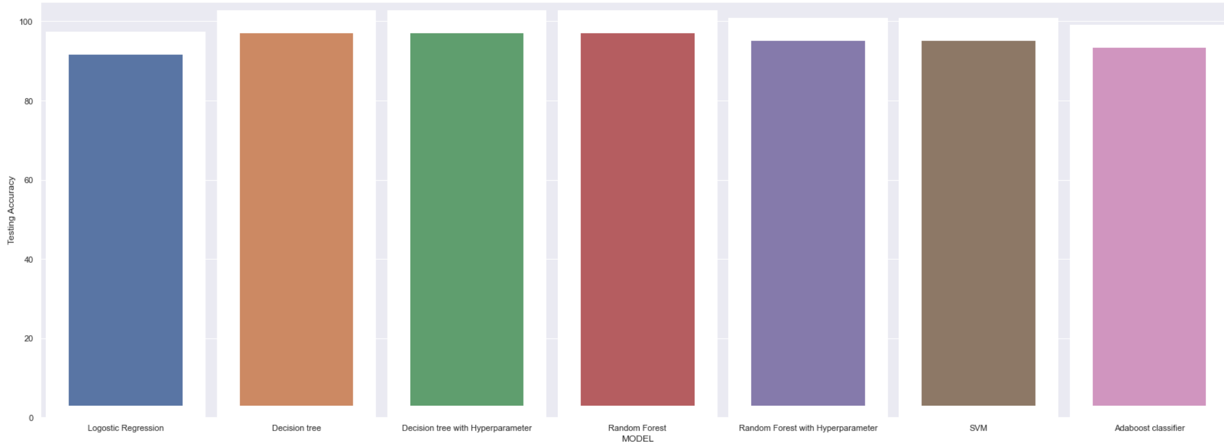
```
sns.set(rc={"figure.figsize":(28, 10)})
```

In [118...

```
# lets see thye graphical representation of our models accuracy

sns.barplot(ACCURACY_df["MODEL"],ACCURACY_df["Testing Accuracy"],capsize=50,linewidth=
```

Out[118]: <AxesSubplot:xlabel='MODEL', ylabel='Testing Accuracy'>


Testing Accuracy
100
80
60
40
20
0
Logostic Regression
Decision tree
Decision tree with Hyperparameter
Random Forest
Random Forest with Hyperparameter
SVM
Adaboost classifier
MODEL

In [ ]: