```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as wr
wr.filterwarnings('ignore')
```

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

```
Mounted at /content/drive
```

```
df=pd.read_csv('/content/drive/MyDrive/ML Project/creditcard.csv')
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

```
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
```

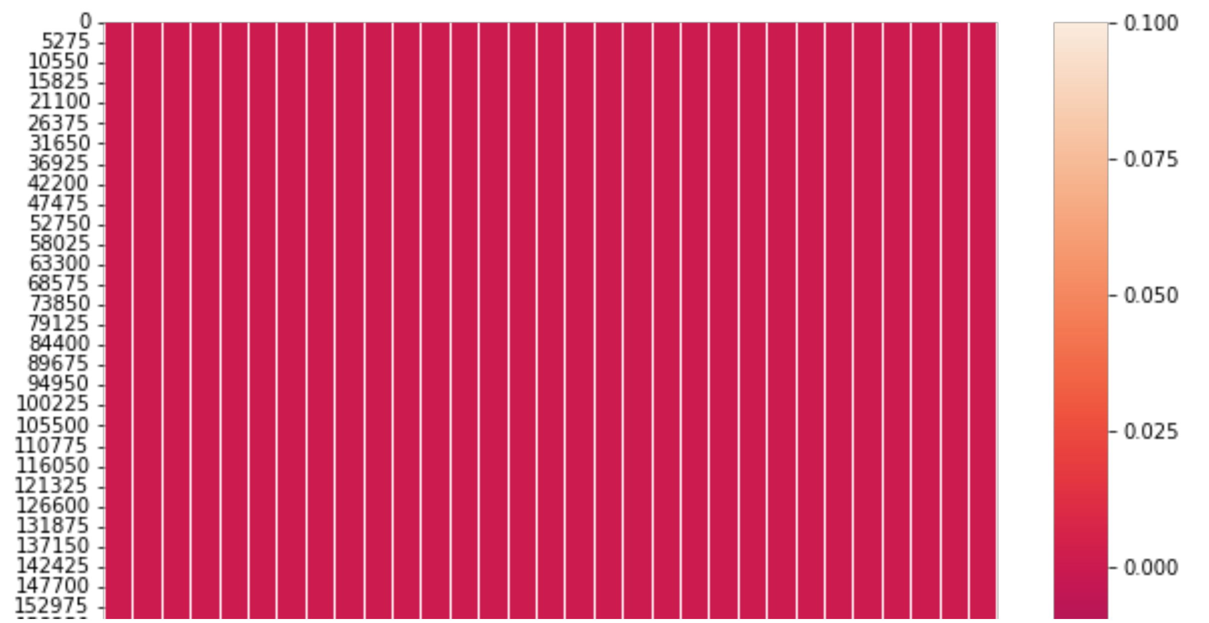
Executing (1h 15m 42s) Cell > train_model() > fit() > _dense_fit()

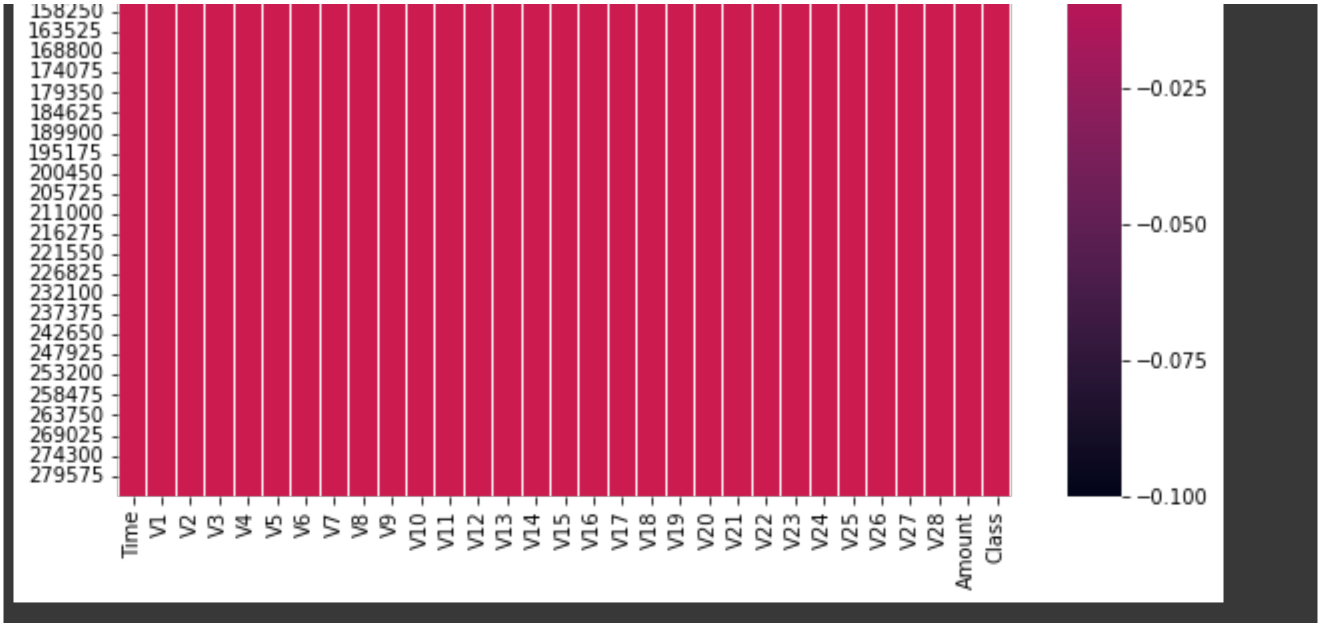
```
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 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
```

```
# There are no Null Value in DataFrame
# There are 31 columns in Dataset.
# There are 2 types of data types in Dataset float64 and int64.
# float64 contains 30 columns and 1 columns of int64.
```

```
# Creating a heatmap to visualize the null values.
plt.figure(figsize=(10,10))
sns.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a7c0fe1d0>
```



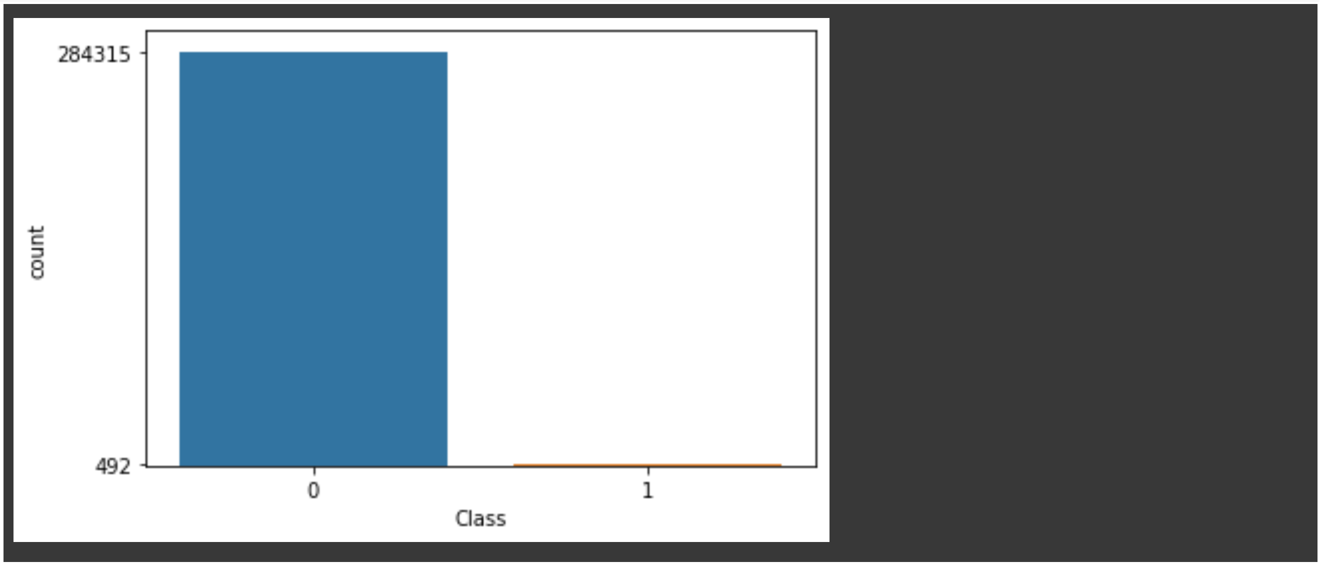


Checking if Output is balance .

```
df['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

```
# Visualizing the balance.
sns.countplot(data=df,x='Class')
f=df['Class'].value_counts()
plt.yticks(f)
plt.show()
```



Spliting DataFrame in x(input) and y(output)

```
# Split Data in
x=df.drop('Class',axis=1) # Input
y=df['Class'] # Output
```

```
# importing train test split
from sklearn.model_selection import train_test_split
# Spliting Data for train and test
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=1,test_size=0.3)
```

Applying Scaling .

```
from sklearn.preprocessing import StandardScaler
# creating object for StandardScaler
ss=StandardScaler()
# Appling Scaling
x_train=ss.fit_transform(x_train)
x_test=ss.transform(x_test)
```

```
y_train.value_counts()
```

```
0    199007
1       357
Name: Class, dtype: int64
```

```
y_test.value_counts()
```

```
0    85308
1      135
Name: Class, dtype: int64
```

```
# Apply RandomOverSampler inbuild class
from imblearn.over_sampling import RandomOverSampler
# create object for Random Over Sampler
ros=RandomOverSampler(random_state=1)
```

```
# Applying RandomOverSampler on training data(70%)
x_train,y_train=ros.fit_resample(x_train,y_train)
```

```
# Check after apply RandomOverSampler
y_train.value_counts()
```

```
0    199007
1    199007
Name: Class, dtype: int64
```

```
     Name: Class, dtype: int64
```

```
# Applying RandomOverSampler on testing data(30%)
x_test,y_test=ros.fit_resample(x_test,y_test)
```

```
# Check after apply RandomOverSampler
y_test.value_counts()
```

```
     0    85308
     1    85308
     Name: Class, dtype: int64
```

```
# Importing some other essential modules
from sklearn.metrics import classification_report,confusion_matrix
```

```
# User Define function After Equalizing Dataset:
def train_model(model):
  model.fit(x_train,y_train)
  y_pred=model.predict(x_test)
  print(classification_report(y_test,y_pred))
  print(confusion_matrix(y_test,y_pred))
  return model
```

Logistic Regression

```
# Importing Logistic Regression.
from sklearn.linear_model import LogisticRegression
# Creating an object for LogisticRegression.
lr=LogisticRegression(random_state=1)
```

```
lr=train_model(lr)
```

```
                   precision    recall  f1-score   support

                0       0.90      0.98      0.94     85308
                1       0.98      0.90      0.94     85308

         accuracy                           0.94    170616
        macro avg       0.94      0.94      0.94    170616
     weighted avg       0.94      0.94      0.94    170616

     [[83724  1584]
      [ 8874 76434]]
```

Decision Tree Classifier

Decision Tree Classifier

Decision Tree Classifier Gini index

```
# import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
# Creating object
dtc=DecisionTreeClassifier(random_state=1)
```

```
#Calling the function.
dtc=train_model(dtc)
```

```
              precision    recall  f1-score   support

           0       0.74      1.00      0.85     85308
           1       1.00      0.65      0.79     85308

    accuracy                           0.83    170616
   macro avg       0.87      0.83      0.82    170616
weighted avg       0.87      0.83      0.82    170616

[[85275    33]
 [29819 55489]]
```

Decision Tree Classifier with Entropy

```
# Creating object with Entropy
dtc0=DecisionTreeClassifier(random_state=1,criterion='entropy')
#Calling the function.
dtc0=train_model(dtc0)
```

```
              precision    recall  f1-score   support

           0       0.77      1.00      0.87     85308
           1       1.00      0.70      0.82     85308

    accuracy                           0.85    170616
   macro avg       0.88      0.85      0.85    170616
weighted avg       0.88      0.85      0.85    170616

[[85272    36]
 [25398 59910]]
```

Decision Tree Classifier with Max Depth

```
# Creating object with MAx Depth
dtc1=DecisionTreeClassifier(random_state=1,max_depth=3)
```

```
dtc1=DecisionTreeClassifier(random_state=1,max_depth=5)
#calling the function
dtc1=train_model(dtc1)
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92     85308
           1       0.98      0.85      0.91     85308

    accuracy                           0.92    170616
   macro avg       0.92      0.92      0.92    170616
weighted avg       0.92      0.92      0.92    170616

[[83524  1784]
 [12622 72686]]
```

Decision Tree Classifier with Min Sample

```
# Creating object with Min Sample Leaf
dtc2=DecisionTreeClassifier(random_state=1,min_samples_leaf=40)
# calling object
dtc2=train_model(dtc2)
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91     85308
           1       1.00      0.81      0.89     85308

    accuracy                           0.90    170616
   macro avg       0.92      0.90      0.90    170616
weighted avg       0.92      0.90      0.90    170616

[[85153   155]
 [16491 68817]]
```

Random Forest Classifier

```
# import Random Forest Library
from sklearn.ensemble import RandomForestClassifier
# creating object
rfc=RandomForestClassifier(random_state=1,n_estimators=100,max_features=4)
# calling function
rfc=train_model(rfc)
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.90     85308
           1       1.00      0.77      0.87     85308
```

```
    accuracy                           0.88    170616
   macro avg       0.91      0.88      0.88    170616
weighted avg       0.91      0.88      0.88    170616

[[85300     8]
 [19663 65645]]
```

Random Forest Classifier with Entropy

```
# Creating object with Entropy
rfc1= RandomForestClassifier(criterion='entropy',random_state=1,n_estimators=100,max_featur
# Calling the object
rfc1=train_model(rfc1)
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.90     85308
           1       1.00      0.77      0.87     85308

    accuracy                           0.88    170616
   macro avg       0.91      0.88      0.88    170616
weighted avg       0.91      0.88      0.88    170616

[[85299     9]
 [19691 65617]]
```

Boosting Tenhnique

ADA Boosting (Adaptor Boosting)

```
from sklearn.ensemble import AdaBoostClassifier
```

```
# creting object
ada=AdaBoostClassifier(random_state=1,n_estimators=10)
# calling object
ada=train_model(ada)
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93     85308
           1       0.96      0.90      0.93     85308

    accuracy                           0.93    170616
   macro avg       0.93      0.93      0.93    170616
weighted avg       0.93      0.93      0.93    170616
```

```
[[81925  3383]
 [ 8199 77109]]
```

Gradient Boosting Classifier

```
# import Gradinet Boost Library
from sklearn.ensemble import GradientBoostingClassifier
```

```
# Create Library
gbc=GradientBoostingClassifier(random_state=1,n_estimators=40)
# caling function
gbc=train_model(gbc)
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93     85308
           1       0.99      0.87      0.92     85308

    accuracy                           0.93    170616
   macro avg       0.94      0.93      0.93    170616
weighted avg       0.94      0.93      0.93    170616

[[84454   854]
 [11329 73979]]
```

Xetreme Gradient Boost

```
# importing Xetreme Gradient Boost library
from xgboost import XGBClassifier
```

```
# Creating th eobject
xgb=XGBClassifier(random_state=1,reg_alpha=1,n_estimators=40)
# calling the object
xgb=train_model(xgb)
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93     85308
           1       0.99      0.86      0.92     85308

    accuracy                           0.93    170616
   macro avg       0.93      0.93      0.93    170616
weighted avg       0.93      0.93      0.93    170616
```

```
[[84714   594]
 [11961 73347]]
```

Support Vector Machine

Linear SVC

```
from sklearn.svm import LinearSVC
```

```
# creating the object
svc=LinearSVC(random_state=1,C=0.99)
# calling the object
svc=train_model(svc)
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94     85308
           1       0.98      0.90      0.94     85308

    accuracy                           0.94    170616
   macro avg       0.94      0.94      0.94    170616
weighted avg       0.94      0.94      0.94    170616

[[83834  1474]
 [ 8874 76434]]
```

SVC Polynomial Kernel Function

```
from sklearn.svm import SVC
# creating object
svc1=SVC(random_state=1,kernel='poly')
# calling the function
svc1=train_model(svc1)
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92     85308
           1       1.00      0.84      0.91     85308

    accuracy                           0.92    170616
   macro avg       0.93      0.92      0.92    170616
weighted avg       0.93      0.92      0.92    170616

[[85082   226]
 [13871 71437]]
```

SVC Radial Basis Kernel Function

```
svc2=SVC(random_state=1,kernel='rbf')
svc2=train_model(svc2)
```

K-NN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knc=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
knc=train_model(knc)
```

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90     85308
           1       1.00      0.78      0.87     85308

    accuracy                           0.89    170616
   macro avg       0.91      0.89      0.89    170616
weighted avg       0.91      0.89      0.89    170616

[[85247    61]
 [19106 66202]]
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb=GaussianNB()
gnb=train_model(gnb)
```

```
              precision    recall  f1-score   support
```

```
           0       0.85      0.98      0.91     85308
           1       0.97      0.82      0.89     85308

    accuracy                           0.90    170616
   macro avg       0.91      0.90      0.90    170616
weighted avg       0.91      0.90      0.90    170616

[[83203  2105]
 [15152 70156]]
```

Stacking Classifier

```
import six
import sys
sys.modules['sklearn.externals.six'] = six
```

```
!pip install mlrose
```

```
Collecting mlrose
  Downloading mlrose-1.3.0-py3-none-any.whl (27 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Installing collected packages: mlrose
Successfully installed mlrose-1.3.0
```

```
from mlxtend.classifier import StackingClassifier
lr=LogisticRegression(random_state=1)
dtc=DecisionTreeClassifier(random_state=1,max_depth=3)
rfc=RandomForestClassifier(random_state=1,n_estimators=100,max_features=4)
```

```
model_list=[lr,dtc,rfc]
meta=LogisticRegression()
```

```
sc=StackingClassifier(classifiers=model_list,meta_classifier=meta)
sc=train_model(sc)
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.90     85308
           1       1.00      0.77      0.87     85308

    accuracy                           0.88    170616
   macro avg       0.91      0.88      0.88    170616
```

```
weighted avg       0.91      0.88      0.88    170616

[[85300     8]
 [19663 65645]]
```

Step Perform

- Import Basic Library
- Allocating Credit Card csv to df variable
- Check for Null Values
- Check if Output Value is Balanced
- Split Dataset in x and y
- Split train and test Data
- Apply Standard Sacler to Convert all colunm in same unit
- Apply Random Over Sampler for Inbalanced Data
- Create User defined function for y_pred,Confussion Matrix and Classification report
- Traning and Testing data with different types of Machine Learning Algorithm on Dataset

conclusion

- We have tried different type of Machine Learning Algorithm on Dataset
- The Best Result we got is from Logistic Regression 0 - 0.98% & 1 - 0.90%
- But After Traning Dataset with different Algorithm we got 0 - 0.98% & 1 - 0.90% from Linear Linear Support Vector Machine After Adding Error.