Importing the reuried libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing the data set

In [2]:
```
credit = pd.read_csv("D:/Datasets/creditcard.csv")
```

Exploratory Data Analysis

In [3]:
```
## 1.) Checking the first 10 cloumns and rows of data set ##
```

In [4]:
```
credit.head(10)
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Am
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	1
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	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	3
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	1
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	
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	

10 rows × 31 columns

In [5]:
```
## 2.) Checking the last 10 columns and rows of the data set ##
```

In [6]:
```
credit.tail(10)
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	
284797	172782.0	-0.241923	0.712247	0.399806	-0.463406	0.244531	-1.343668	0.929369	-0.206210	0.106234	...	-0.228876	-0.514376	0.279598	0.371441	-0.559238	0.113144	0.131507	0.0
284798	172782.0	0.219529	0.881246	-0.635891	0.960928	-0.152971	-1.014307	0.427126	0.121340	-0.285670	...	0.099936	0.337120	0.251791	0.057688	-1.508368	0.144023	0.181205	0.2
284799	172783.0	-1.775135	-0.004235	1.189786	0.331096	1.196063	5.519980	-1.518185	2.080825	1.159498	...	0.103302	0.654850	-0.348929	0.745323	0.704545	-0.127579	0.454379	0.1
284800	172784.0	2.039560	-0.175233	-1.196825	0.234580	-0.008713	-0.726571	0.017050	-0.118228	0.435402	...	-0.268048	-0.717211	0.297930	-0.359769	-0.315610	0.201114	-0.080826	-0.0
284801	172785.0	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722	0.115093	-0.204064	...	-0.314205	-0.808520	0.050343	0.102800	-0.435870	0.124079	0.217940	0.0
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	0.8
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	-0.0
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	-0.0
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	0.1
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	0.0

10 rows × 31 columns

In [7]:
```
## 3.) Checking the number of columns and rows in the data set ##
```

In [8]:
```
credit.shape
```

Out[8]:
```
(284807, 31)
```

In [9]:
```
## 4.) Checking the data set ##
```

In [10]:
```
print('Count of columns in the data is: ', len(credit.columns))
print('Count of columns in the data is: ', len(credit))
```

```
Count of columns in the data is:  31
Count of columns in the data is:  284807
```

In [11]:
```
## 5.) Checking the Columns Varible of the data set ##
```

In [12]:
```
credit.columns
```

Out[12]:
```
Index(['Time', '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',
       'Class'],
      dtype='object')
```

In [13]:
```
## 6.) Checking the data types of the data set ##
```

In [14]:
```
credit.dtypes
```

Out[14]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

In [15]:
```
## 7.) Checking the uinue variable of the data set ##
```

In [16]:
```
credit.nunique()
```

Out[16]:
```
Time      124592
V1        275663
V2        275663
V3        275663
V4        275663
V5        275663
V6        275663
V7        275663
V8        275663
V9        275663
V10       275663
V11       275663
V12       275663
V13       275663
V14       275663
V15       275663
V16       275663
V17       275663
V18       275663
V19       275663
V20       275663
V21       275663
V22       275663
V23       275663
V24       275663
V25       275663
V26       275663
V27       275663
V28       275663
Amount     32767
Class          2
dtype: int64
```

In [17]:
```
## 8.) Checking the unique variable of Class Variable ##
```

In [18]:
```
credit['Class'].unique()
```

Out[18]:
```
array([0, 1], dtype=int64)
```

In [19]:
```
## 9.) Checking the number of counts in Class Variable ##
```

In [20]:
```
credit.Class.value_counts()
```

Out[20]:
```
0    284315
1       492
Name: Class, dtype: int64
```

Seprating Fraud and Normal Data

In [21]:
```
## 10.) Get the Fraud and Nirmal from the data set ##
```

In [22]:
```
Fraud = credit[credit['Class'] == 1]
Normal = credit[credit['Class'] == 0]
```

In [23]:
```
print(Fraud.shape)
```

```
(492, 31)
```

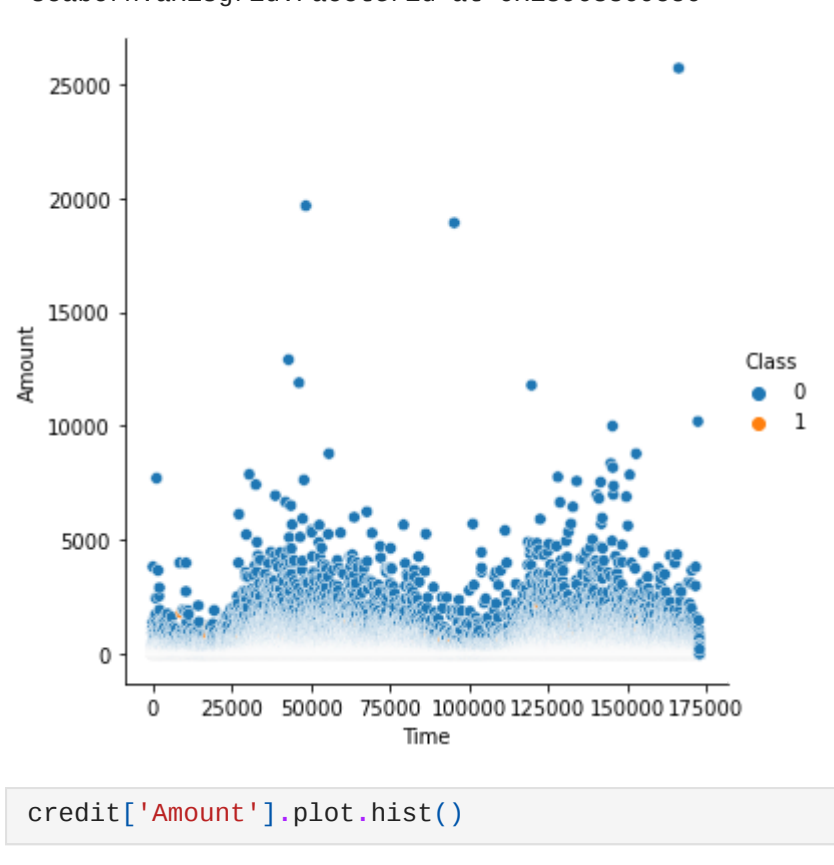
In [24]:
```
print(Normal.shape)
```

```
(284315, 31)
```

Data Visualisation

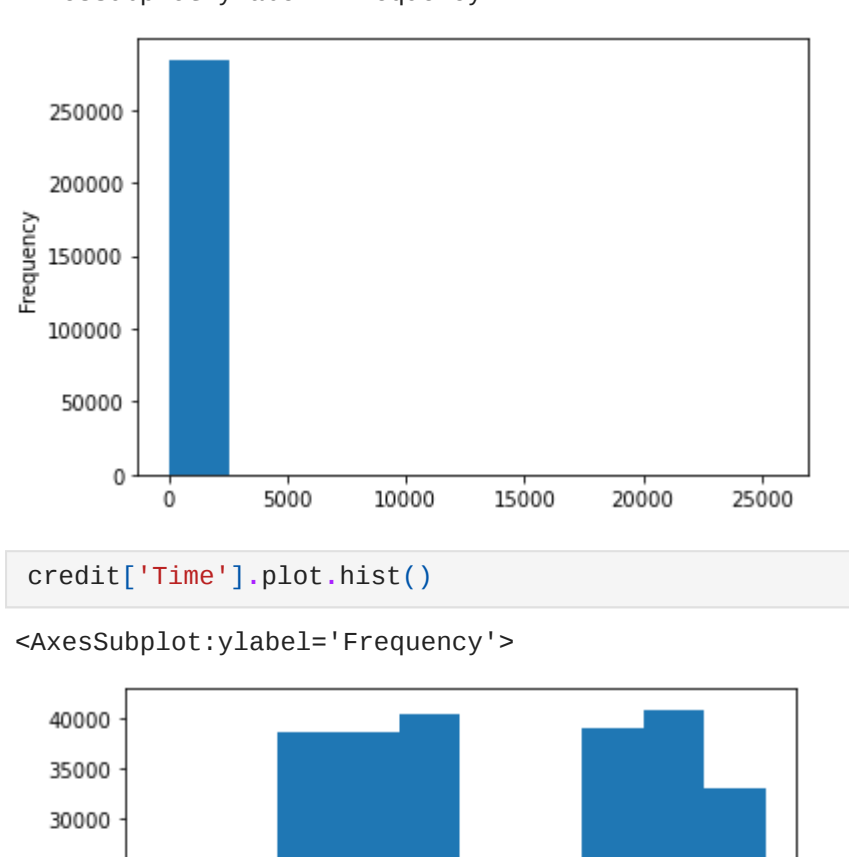
In [25]:
```
sns.relplot(x = 'Amount', y = 'Time', hue = 'Class', data = credit)
```

Out[25]:
```
<seaborn.axisgrid.FacetGrid at 0x13963360370>
```



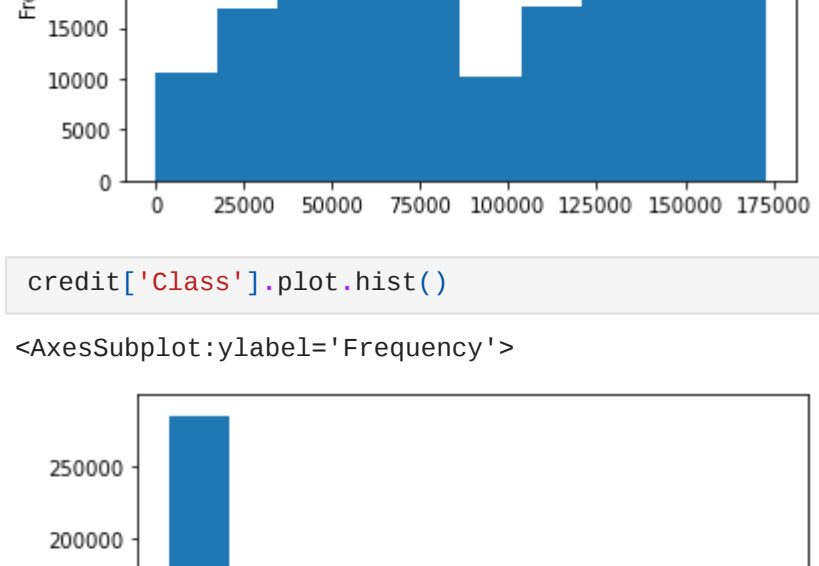
In [26]:
```
sns.relplot(x = 'Time', y = 'Amount', hue = 'Class', data = credit)
```

Out[26]:
```
<seaborn.axisgrid.FacetGrid at 0x13963360e50>
```



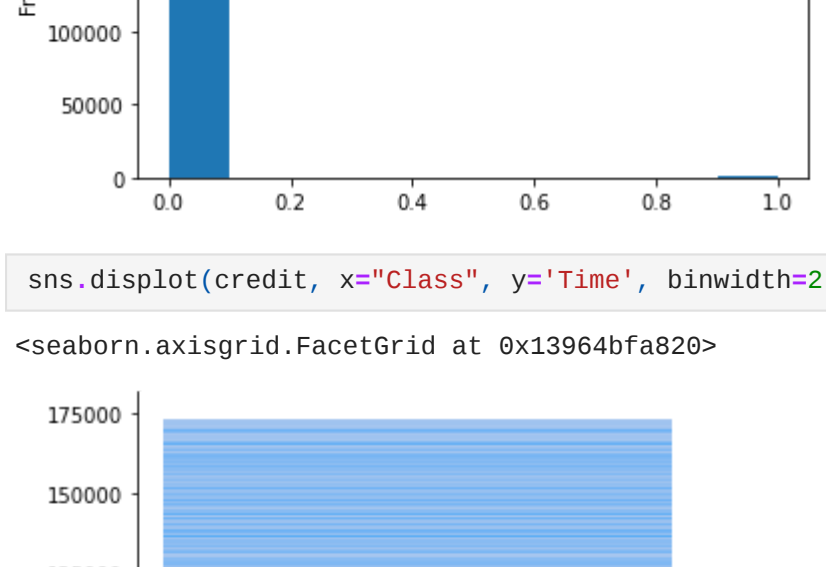
In [27]:
```
credit['Amount'].plot.hist()
```

Out[27]:
```
<AxesSubplot:ylabel='Frequency'>
```



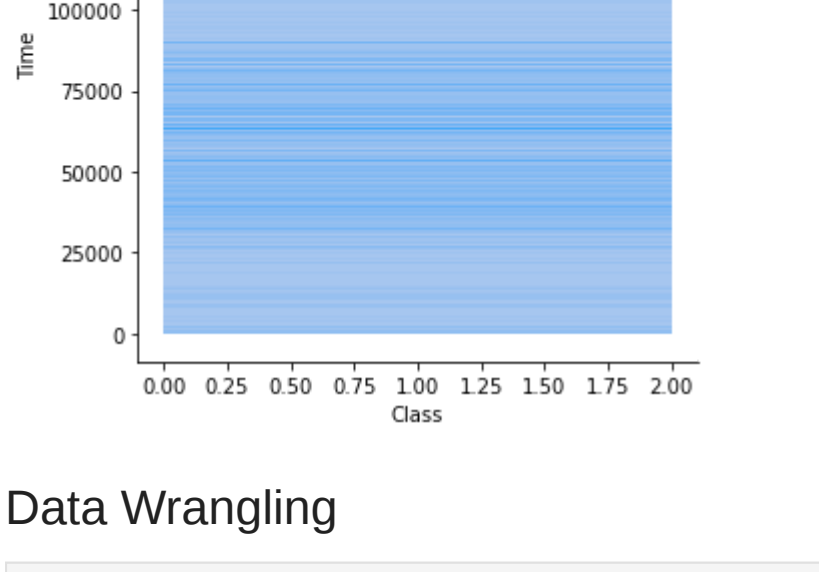
In [28]:
```
credit['Time'].plot.hist()
```

Out[28]:
```
<AxesSubplot:ylabel='Frequency'>
```



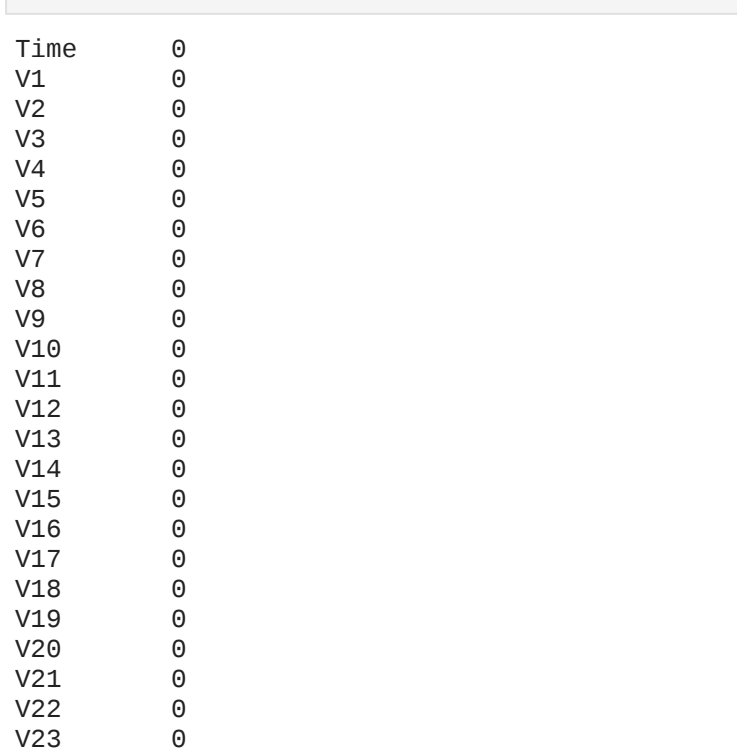
In [29]:
```
credit['Class'].plot.hist()
```

Out[29]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [30]:
```
sns.displot(credit, x="Class", y='Time', binwidth=2)
```

Out[30]:
```
<seaborn.axisgrid.FacetGrid at 0x13964bfa820>
```



Data Wrangling

In [31]:
```
## 1.) Checking for the null values ##
```

In [32]:
```
credit.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
```

Splitting the Data Set

In [33]:
```
## Splitting the dataset into Train & Test data set ##
```

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
x = credit.iloc[:,:-1]
y = credit['Class']
```

In [36]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25)
```

Model Building

1.) Logistic Regression

In [37]:
```
## Importing the Logistic Regresion Libraries
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
```

In [39]:
```
LR = LogisticRegression()
LR.fit(x_train,y_train)
```

```
C:\Users\Acer\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[39]:
```
LogisticRegression()
```

In [40]:
```
## Predicting the data set ##
```

In [41]:
```
y_pred = LR.predict(x_test)
```

In [42]:
```
## Checking for accuracy score ##
```

In [43]:
```
LR.score(x_test,y_test)
```

Out[43]:
```
0.9989185697030982
```

2.) Support Vector Machine

In [44]:
```
## Importing the SVM Regresion Libraries
```

In [45]:
```
from sklearn import model_selection,svm
from sklearn.metrics import accuracy_score
```

In [46]:
```
## Classification Algorithm ##
```

In [47]:
```
SVM = svm.SVC(C = 1.0, kernel = 'linear', degree = 3, gamma = 'auto')
```

In [48]:
```
## Fiting the training data set on classifier ##
```

In [49]:
```
SVM.fit(x_train,y_train)
```

Out[49]:
```
SVC(gamma='auto', kernel='linear')
```

In [50]:
```
## Predict the labels on validation data set ##
```

In [51]:
```
predictions_SVM = SVM.predict(x_test)
```

In [52]:
```
## Use accuracy function to get accuracy score ##
```

In [53]:
```
print("SVM Accuracy Score ->",  round(accuracy_score(predictions_SVM,y_test)*100,2),"%")
```

```
SVM Accuracy Score -> 99.87 %
```

3.) XG Boosting

In [54]:
```
## Importing the XG Boosting Regresion Libraries ##
```

In [55]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\acer\anaconda3\lib\site-packages (1.5.0)
Requirement already satisfied: scipy in c:\users\acer\anaconda3\lib\site-packages (from xgboost) (1.5.2)
Requirement already satisfied: numpy in c:\users\acer\anaconda3\lib\site-packages (from xgboost) (1.19.2)
```

In [56]:
```
import xgboost
from xgboost import XGBClassifier
```

In [57]:
```
## Prediction of the XG Boosting Regression ##
```

In [58]:
```
model = XGBClassifier(use_label_encoder = False)
model.fit(x_train,y_train)
```

```
[19:06:05] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[58]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=4,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', use_label_encoder=False,
              validate_parameters=1, verbosity=None)
```

In [59]:
```
## Predicting the X_test ##
```

In [60]:
```
prediction = model.predict(x_test)
print(prediction)
```

```
[0 0 0 ... 0 0 0]
```

In [61]:
```
## Evaluating Predictions ##
```

In [62]:
```
accuracy = accuracy_score(y_test, prediction)
print("Accuracy: %.2f%%" % (accuracy*100.0))
```

```
Accuracy: 99.96%
```

Model Choosen

In [63]:
```
## Importing the Logistic Regresion Libraries
```

In [64]:
```
from sklearn.linear_model import LogisticRegression
```

In [65]:
```
LR = LogisticRegression()
LR.fit(x_train,y_train)
```

```
C:\Users\Acer\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[65]:
```
LogisticRegression()
```

In [66]:
```
## Predicting the data set ##
```

In [67]:
```
y_pred = LR.predict(x_test)
```

In [68]:
```
## Checking for accuracy score ##
```

In [69]:
```
LR.score(x_test,y_test)
```

Out[69]:
```
0.9989185697030982
```

Model Evaluation

In [70]:
```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71126
           1       0.50      0.78      0.61        76

    accuracy                           1.00     71202
   macro avg       0.75      0.89      0.80     71202
weighted avg       1.00      1.00      1.00     71202
```