Leitfaden für nachvollziehbare Schritte

1. Kurze Darstellung des Problembereichs / Aufriss des Themas

1.1 Inhaltlich

Kern der Untersuchung: Data Analysis and creating a model for detecting credit card fraud detection

Grobziele der Arbeit : Credit card fraud detection is presently the most frequently occurring problem in the present world,as there is increase in the online transactions of money .When such fraudelent activities happen everyone involved will suffer ,the account holder the bank or any finacial sector who issued the card and the merchant who is finalizing the transaction with purchase.Hence this makes it extremely essential to identify the fraudulent transactions at the onset. This project aims to focus mainly on machine learning algorithms for detection of fraud.

1.2 Begründung desThemas

Darstellung der Relevanz des Themas?

Warum ist das Thema wichtig und interessant und daher bearbeitungs- und förderungswürdig?

Credit cards have become an attractive spending options these days , as we can spend the money but technically pay later for it. With the buy now and pay later phenomenon gaining attention ,the number of customer adopting this has increased now days .Also these spending options come with attractive offers like points system and converting them to cash . Moreover the competition in this segment has also increased . But these option comes with many resposibilities and whole lot of scenarios where your payment cards become susceptible to frauds. Thus, fraud detection systems have become essential for banks and financial institution, to minimize their losses.

Darstellung eines persönlichen Erkenntnisinteresses.

Dieser Abschnitt soll ein prägnanter Einstieg in die Projektarbeit / Seminararbeit sein.

Er soll beim Leser Interesse für das Thema und die Bereitschaft wecken oder verstärken, die Arbeit zu betreuen bzw. zu fördern und dient der Eigenmotivation.

Credit card fraud costs consumers and the financial company billions of dollar annually.Hence there is a urge to creat a system with the help of machine learning models to identify and track such frauds. Hence this is an intresting Machine Learning Case study to detect the frauds and to analyse what triggers for it. This dataset contains the real bank transactions made by European cardholders in the year 2013. As a security concern, the actual variables are not being shared but they have been transformed versions of PCA.

2. Nachvollziehbare Schritte

2.1 Der Stand der Forschung / Auswertung der vorhandenen Literatur / Tutorials ...

Wurde das Problem früher bereits untersucht?

Welche Aspekte wurden untersucht und welche nicht?

Yes the problem has been investigated pervious with applying different algorithm of machine Learning. The challenges involved in credit card fraud detection project is primarily the data itself. The data is heavily imbalanced, i.e., the count of data labeled as fraudulent is way less than the data labeled as non-fraudulent data. This makes it extremely tricky to train the model as it tends to overfit for the majority class and underfit for the minority class.

Welche Kontroversen gab es und welche Methoden standen bis jetzt im Vordergrund?

Lösungswege strukturieren!

Importing important libraries

Load the dataset into a data frame using Pandas

Explore the number of rows & columns, ranges of values etc.

The data had no missing values hence there was no need for replacing any values

Wichtigste (verwendete) wissenschaftliche Positionen zum ausgewählten Thema?

(Z.B. **Tutorials …**)

2.2 Fragestellung

Can a fraud transaction be prevent before it occurs And protect consumers' interests and reduce the heavy annual financial losses caused by fraud around the world?

2.4 Wissenslücke

The data being highly imbalance, if we train our model without taking care of the imbalance issues, it predicts the label with higher importance given to genuine transactions (as there is majority of non-fraud transaction) and hence obtains more accuracy.

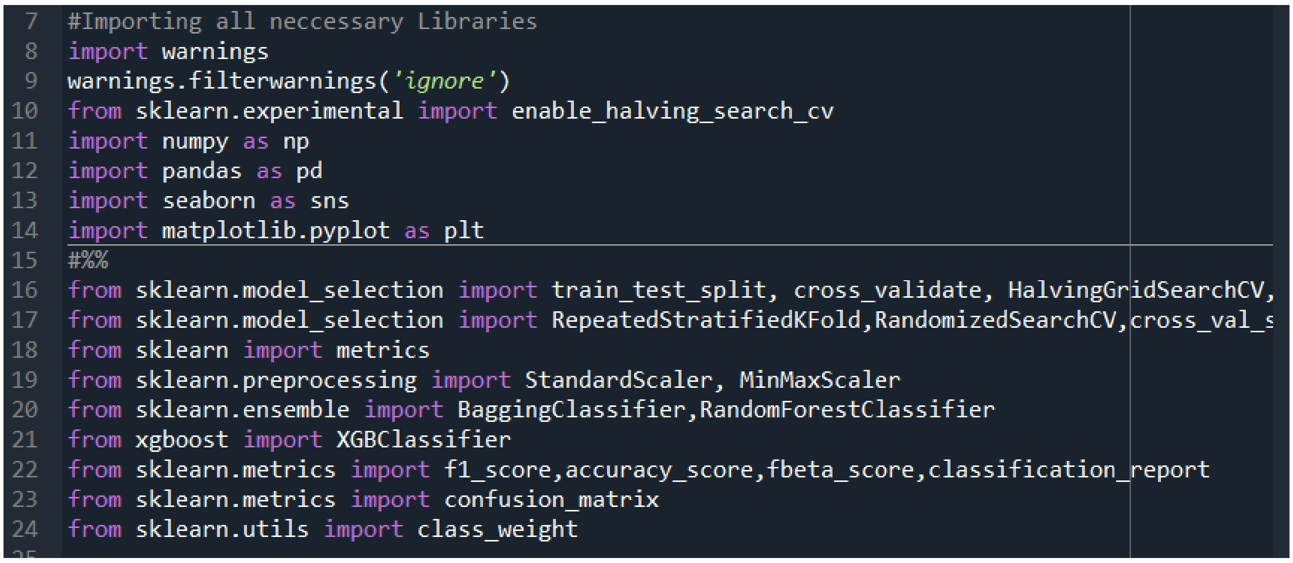
2.5 Methode

Detaillierte nachvollziehbare Beschreibung der Vorgehensweise !!

Vgl. MUSTER-PROJEKTE in den Tutorials !!

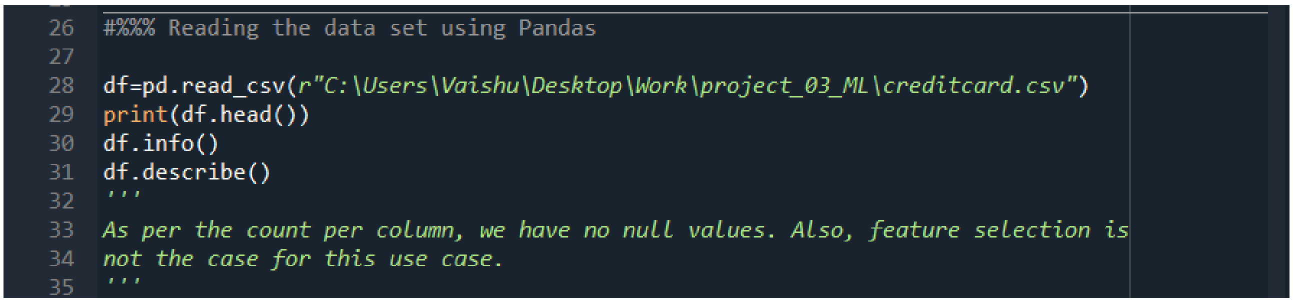
- Pandas for reading and analysing the data
- Seaborn and plotly for plotting the graphs
- Modelbuilding
- Hyperparameters Tuning

Importing important Libraries

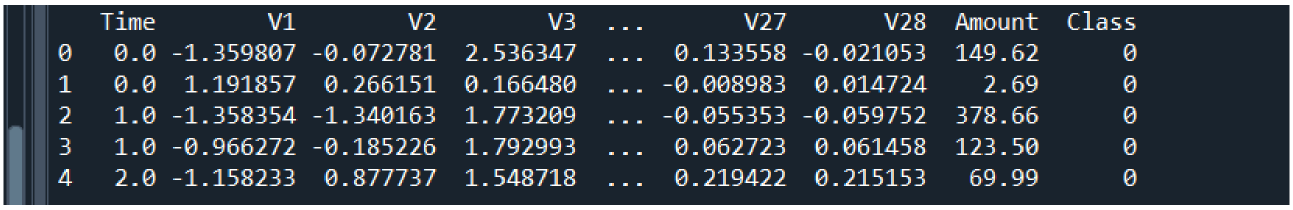


```
#Importing all neccessary Libraries
import warnings
warnings.filterwarnings('ignore')
from sklearn.experimental import enable_halving_search_cv
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#%%
from sklearn.model_selection import train_test_split, cross_validate, HalvingGridSearchCV,
from sklearn.model_selection import RepeatedStratifiedKFold,RandomizedSearchCV,cross_val_s
from sklearn import metrics
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import BaggingClassifier,RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import f1_score,accuracy_score,fbeta_score,classification_report
from sklearn.metrics import confusion_matrix
from sklearn.utils import class_weight
```

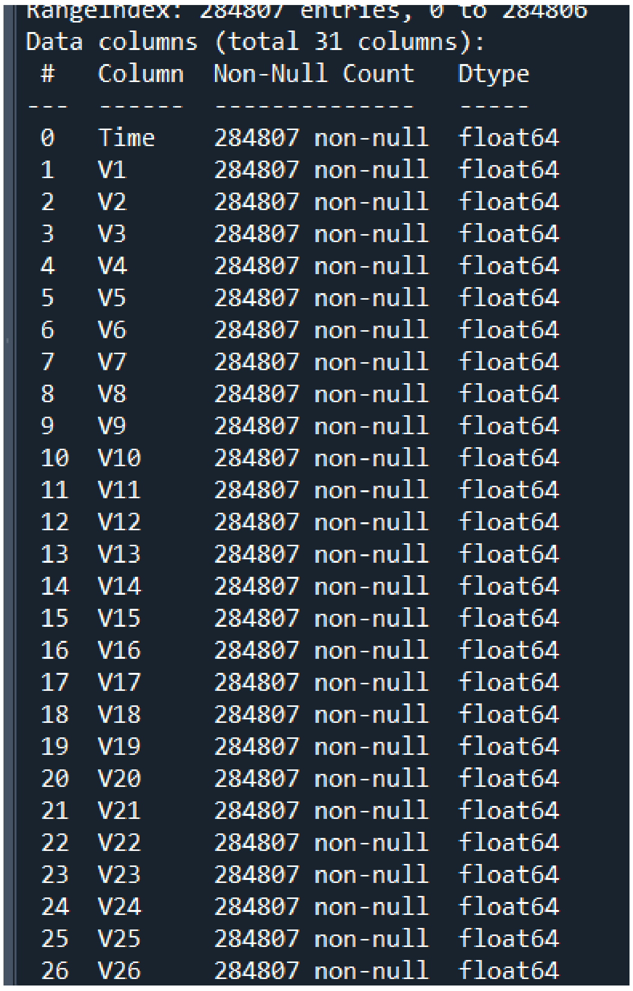
Reading the data using pandas and exploring the columns



```
#%%% Reading the data set using Pandas

df=pd.read_csv(r"C:\Users\Vaishu\Desktop\Work\project_03_ML\creditcard.csv")
print(df.head())
df.info()
df.describe()
'''
As per the count per column, we have no null values. Also, feature selection is
not the case for this use case.
'''
```



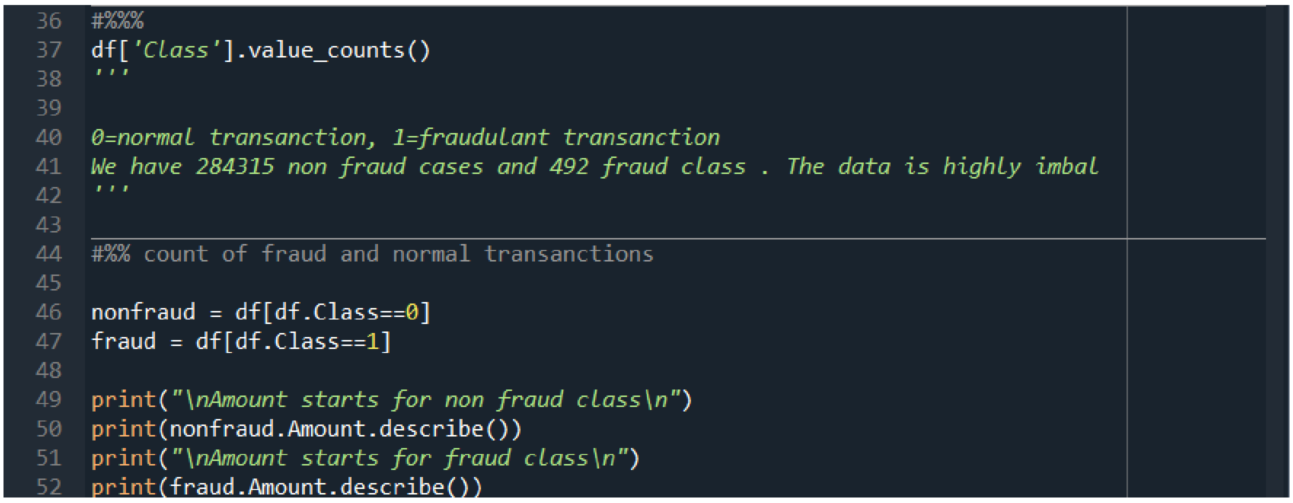
```
   Time        V1        V2        V3  ...       V27       V28  Amount  Class
0   0.0 -1.359807 -0.072781  2.536347  ...  0.133558 -0.021053  149.62      0
1   0.0  1.191857  0.266151  0.166480  ... -0.008983  0.014724    2.69      0
2   1.0 -1.358354 -1.340163  1.773209  ... -0.055353 -0.059752  378.66      0
3   1.0 -0.966272 -0.185226  1.792993  ...  0.062723  0.061458  123.50      0
4   2.0 -1.158233  0.877737  1.548718  ...  0.219422  0.215153   69.99      0
```



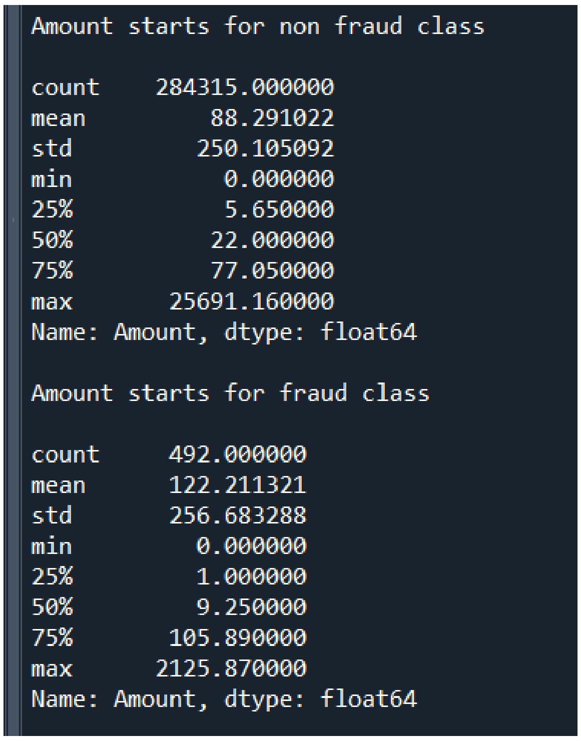
```
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 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
```

We can see here we have 31 columns and 284807 rows .Also the data set is clean, meaning there are no missing values

Count of Fraud and Non-fraud Transactions



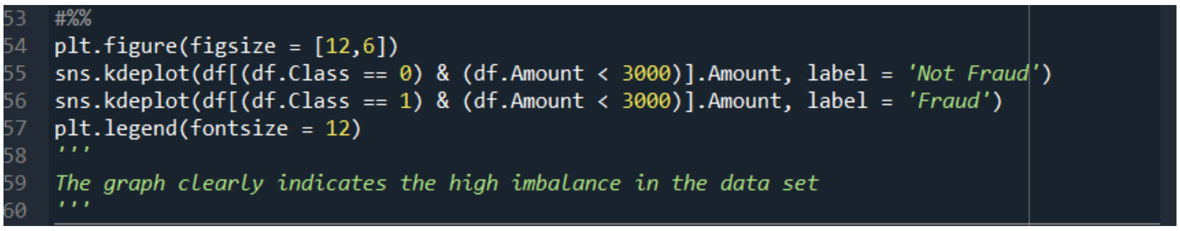
```
#%%
df['Class'].value_counts()
'''

0=normal transanction, 1=fraudulant transanction
We have 284315 non fraud cases and 492 fraud class . The data is highly imbal
'''

#%% count of fraud and normal transanctions

nonfraud = df[df.Class==0]
fraud = df[df.Class==1]

print("\nAmount starts for non fraud class\n")
print(nonfraud.Amount.describe())
print("\nAmount starts for fraud class\n")
print(fraud.Amount.describe())
```



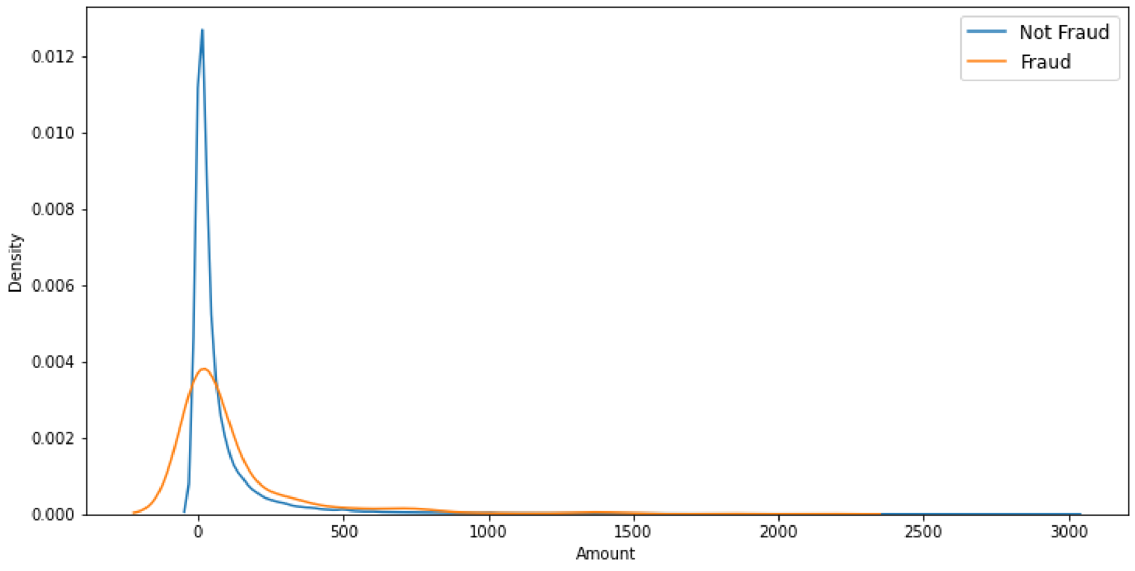
```
Amount starts for non fraud class

count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64

Amount starts for fraud class

count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

Its Very much clear from the above result that we have 284315 Non-fraud transaction and only 492 fraud transaction . This means our data is highly Imbalance

Plotting the Graph of amount with respect to number of transactions

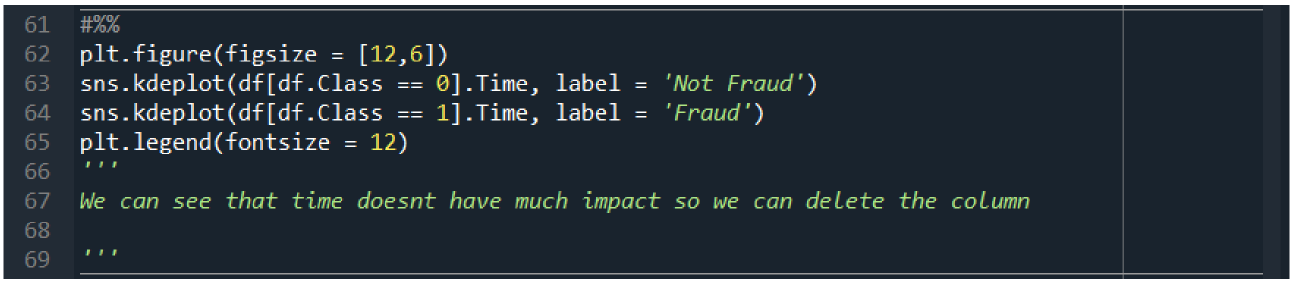


```
#%%
plt.figure(figsize = [12,6])
sns.kdeplot(df[(df.Class == 0) & (df.Amount < 3000)].Amount, label = 'Not Fraud')
sns.kdeplot(df[(df.Class == 1) & (df.Amount < 3000)].Amount, label = 'Fraud')
plt.legend(fontsize = 12)
'''
The graph clearly indicates the high imbalance in the data set
'''
```

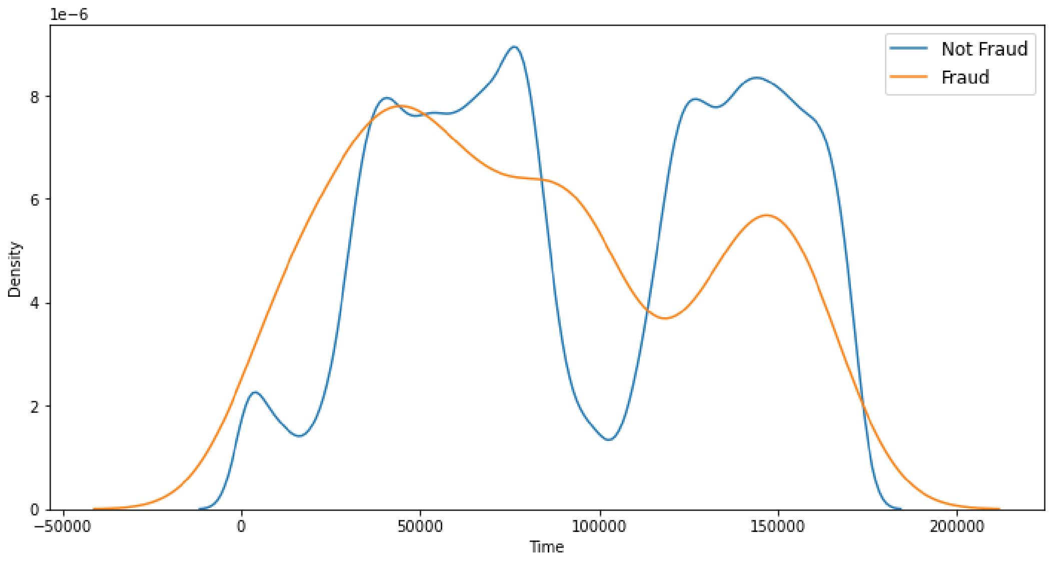


This graph represents the amount that was transcated. A majority of transactions are relatively small and only a handful of them come close to the maximum transacted amount.

Plottins the Graph with respect to time of transactions



```
#%%
plt.figure(figsize = [12,6])
sns.kdeplot(df[df.Class == 0].Time, label = 'Not Fraud')
sns.kdeplot(df[df.Class == 1].Time, label = 'Fraud')
plt.legend(fontsize = 12)
'''
We can see that time doesnt have much impact so we can delete the column

'''
```



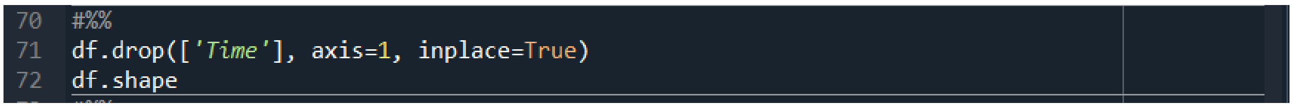
This graph shows the times at which transactions were done

within two days. It can be seen that the least number of

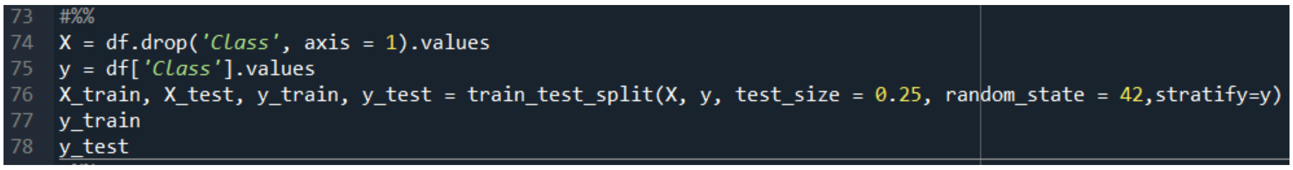
transactions were made during night time in non-fraudulent transactions and highest during

the days. Also we see that for fraud transaction there isn't much impact of time .

Data Preprocessing and Splitting the data into train_test_split Method



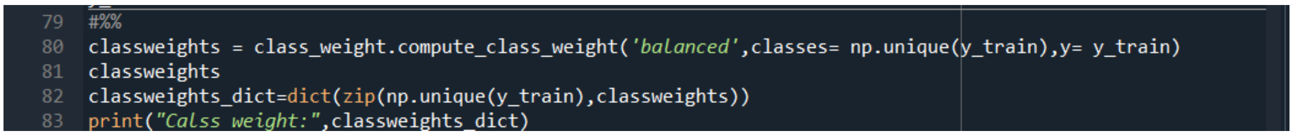
```
#%%
df.drop(['Time'], axis=1, inplace=True)
df.shape
```



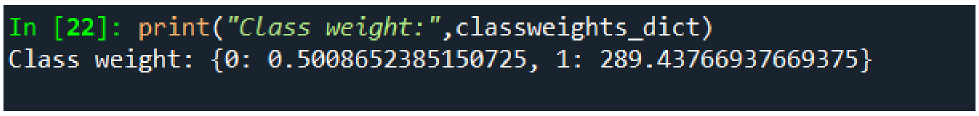
```
#%%
X = df.drop('Class', axis = 1).values
y = df['Class'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 42,stratify=y)
y_train
y_test
```

Now, it's time to split credit card data with a split of 75-25 using stratified train_test_split().As our data is highly imbalance it is desirable to split the dataset into train and test sets in a way that preserves the same proportions of examples in each class as observed in the original dataset.This is called a stratified train-test split. We can achieve this by setting the "stratify" argument to the y component of the original dataset.

Handling the Imbalance in Data



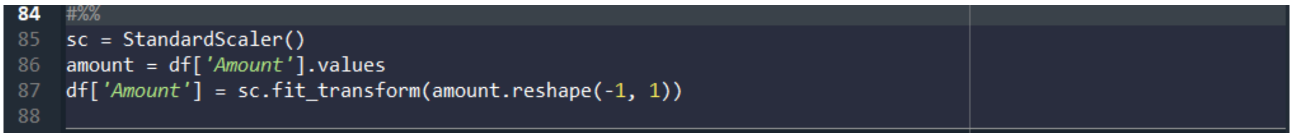
```
#%%
classweights = class_weight.compute_class_weight('balanced',classes= np.unique(y_train),y= y_train)
classweights
classweights_dict=dict(zip(np.unique(y_train),classweights))
print("Calss weight:",classweights_dict)
```



```
In [22]: print("Class weight:",classweights_dict)
Class weight: {0: 0.5008652385150725, 1: 289.43766937669375}
```

Before creating machine learning algorithms which are not very useful with biased class data. we can modify the current training algorithm to take into account the skewed distribution of the classes. This can be achieved by giving different weights to both the majority and minority classes. The difference in weights will influence the classification of the classes during the training phase. The whole purpose is to penalize the misclassification made by the minority class by setting a higher class weight and at the same time reducing weight for the majority class.Hence I have used Class weights from Scikitlearn.

Using Standarad Scaler for Amount column



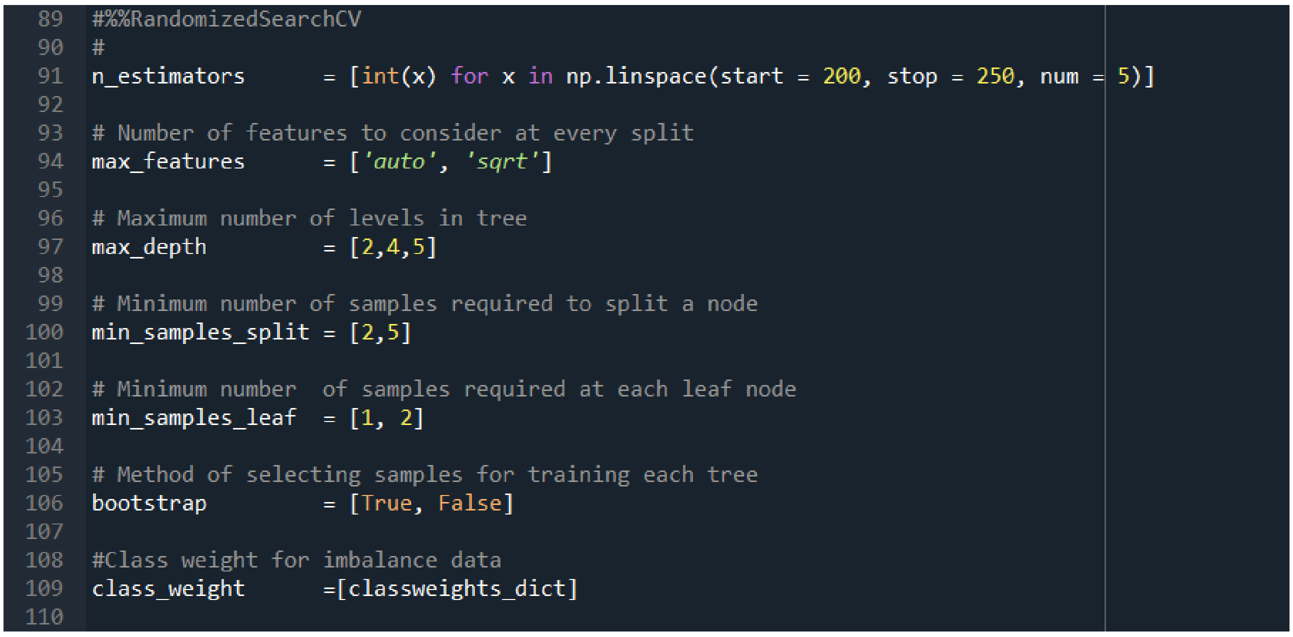
```
#%%
sc = StandardScaler()
amount = df['Amount'].values
df['Amount'] = sc.fit_transform(amount.reshape(-1, 1))

```

As the amount ranges from 0 to 25691.15 , it is a good practice to scale this variable. We can use a standard scaler to make it fix.

Building The Models

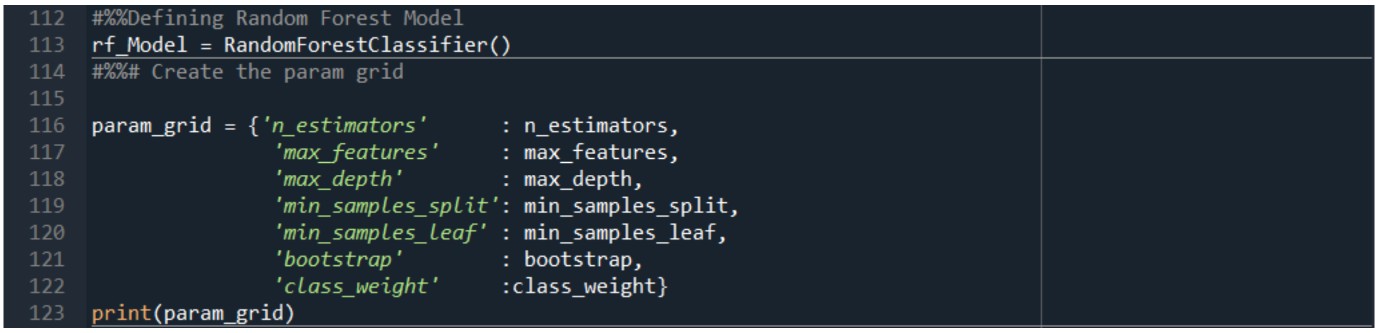
Let's train different models on our dataset and observe which algorithm works better for our problem. This is actually a binary classification problem as we have to predict only one of the two class labels. We can apply many algorithms but I'm using Random Forest and XGBoost which are Ensembling techniques. To get the best Parameters I have used Hyperparameter tuning methods such as Randomized Search CV and GridSearchCV

Randomized Search CV



```
#%%RandomizedSearchCV
#
n_estimators      = [int(x) for x in np.linspace(start = 200, stop = 250, num = 5)]

# Number of features to consider at every split
max_features      = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth         = [2,4,5]

# Minimum number of samples required to split a node
min_samples_split = [2,5]

# Minimum number  of samples required at each leaf node
min_samples_leaf  = [1, 2]

# Method of selecting samples for training each tree
bootstrap         = [True, False]

#Class weight for imbalance data
class_weight      =[classweights_dict]

```

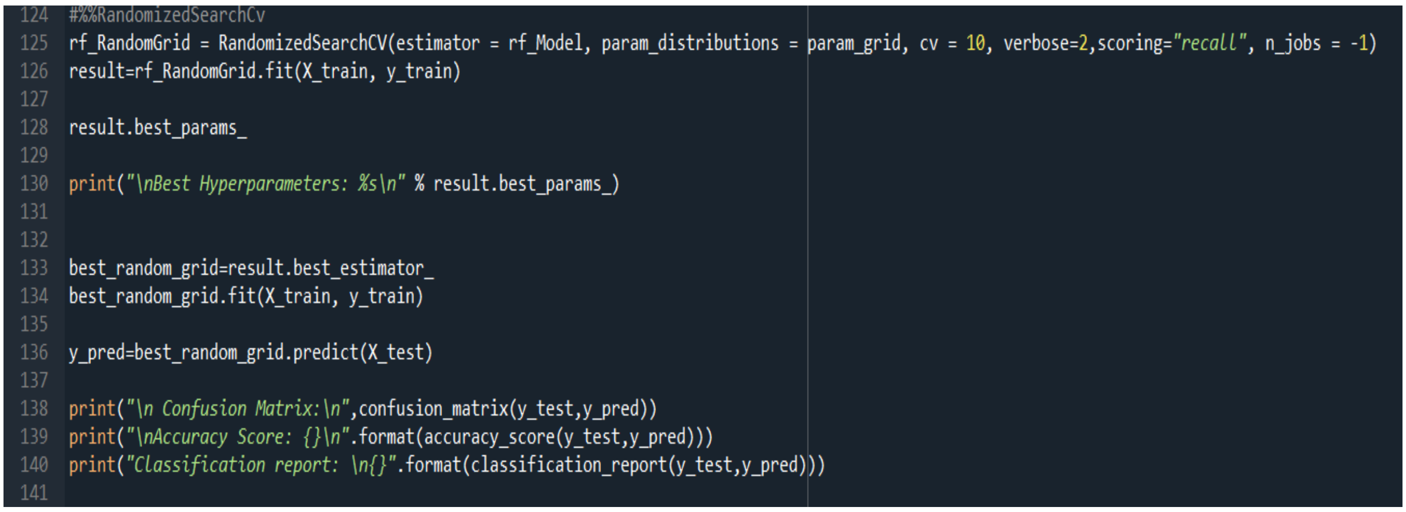
Defining the Model and Param Grid



```
#%%Defining Random Forest Model
rf_Model = RandomForestClassifier()
#%%# Create the param grid

param_grid = {'n_estimators'      : n_estimators,
              'max_features'      : max_features,
              'max_depth'         : max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf' : min_samples_leaf,
              'bootstrap'         : bootstrap,
              'class_weight'      :class_weight}
print(param_grid)
```

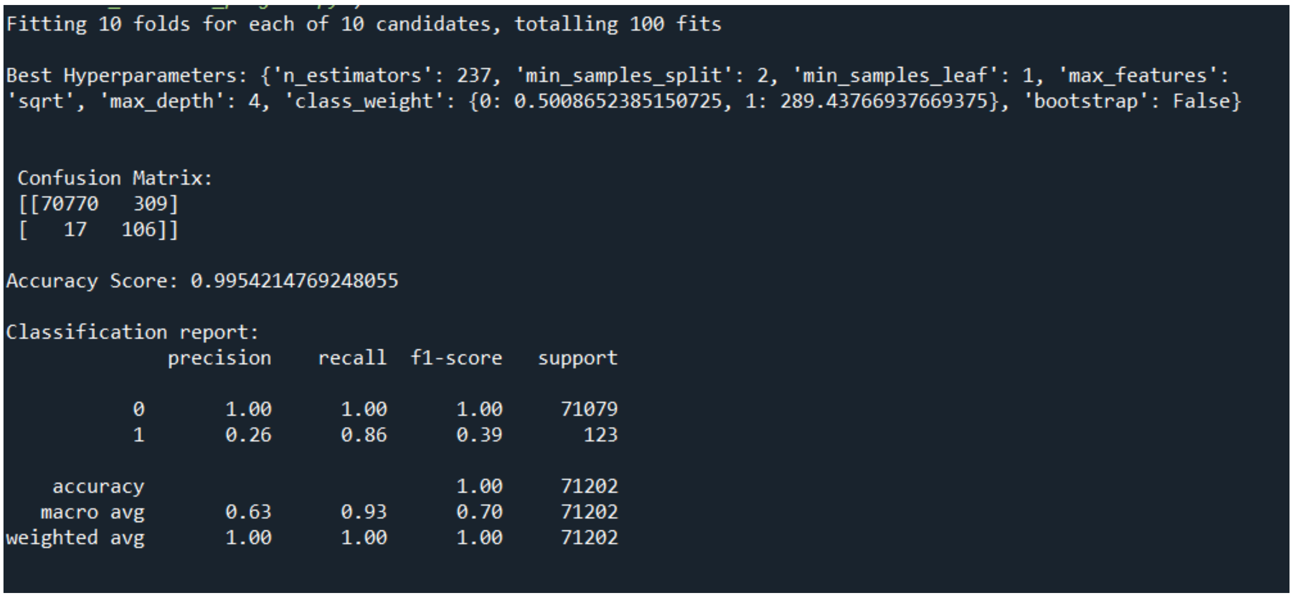
Defining the parameter for RandomizedSearchCV and creating Param Grid for Radom Forest classifier The random forest randomly selects the features that is independent variables and also randomly selects the rows by row sampling and the number of decision tree can be determined by using hyper parameter optimization. This is one is the widely used machine learning algorithm in real word scenarios and in deployed models.

Evaluation and fitting the model



```
#%%RandomizedSearchCv
rf_RandomGrid = RandomizedSearchCV(estimator = rf_Model, param_distributions = param_grid, cv = 10, verbose=2,scoring="recall", n_jobs = -1)
result=rf_RandomGrid.fit(X_train, y_train)

result.best_params_

print("\nBest Hyperparameters: %s\n" % result.best_params_)


best_random_grid=result.best_estimator_
best_random_grid.fit(X_train, y_train)

y_pred=best_random_grid.predict(X_test)

print("\n Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("\nAccuracy Score: {}\n".format(accuracy_score(y_test,y_pred)))
print("Classification report: \n{}".format(classification_report(y_test,y_pred)))
```

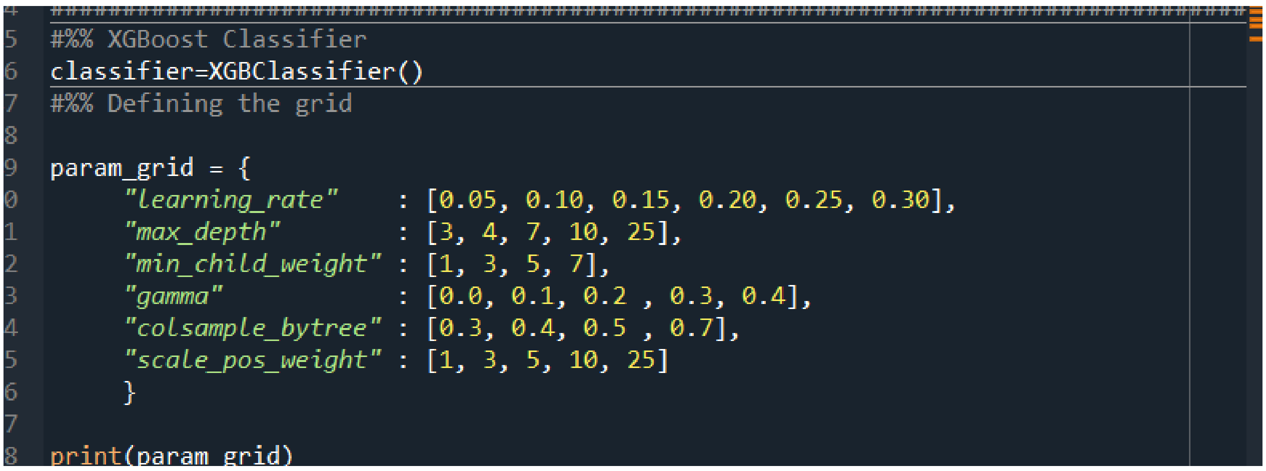
Result



```
Fitting 10 folds for each of 10 candidates, totalling 100 fits

Best Hyperparameters: {'n_estimators': 237, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features':
'sqrt', 'max_depth': 4, 'class_weight': {0: 0.5008652385150725, 1: 289.43766937669375}, 'bootstrap': False}


 Confusion Matrix:
 [[70770   309]
 [   17   106]]

Accuracy Score: 0.9954214769248055

Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.26      0.86      0.39       123

    accuracy                           1.00     71202
   macro avg       0.63      0.93      0.70     71202
weighted avg       1.00      1.00      1.00     71202
```

The above result shows the best parameters, confusion matrix , accuracy score and classification report. Keeping in mind that we can't use 'accuracy' as a metric in our credit card fraud detection project. That's because it would reflect all the transactions as normal with 99% accuracy, rendering our algorithm useless. Typically, a confusion matrix is a visualization of a classification model that shows how well the model has predicted the outcomes when compared to the original ones.Here in this case our model has predict 309 false negatives and 17 false positives . We can say that our model has classified the non-fraud transactions pretty well. The scoring parameter used here is recall . Recall is a metric that quantifies the number of correct positive predictions made out of all positive predictions that could have been made. Also for imbalanced learning, recall is typically used to measure the coverage of the minority class.

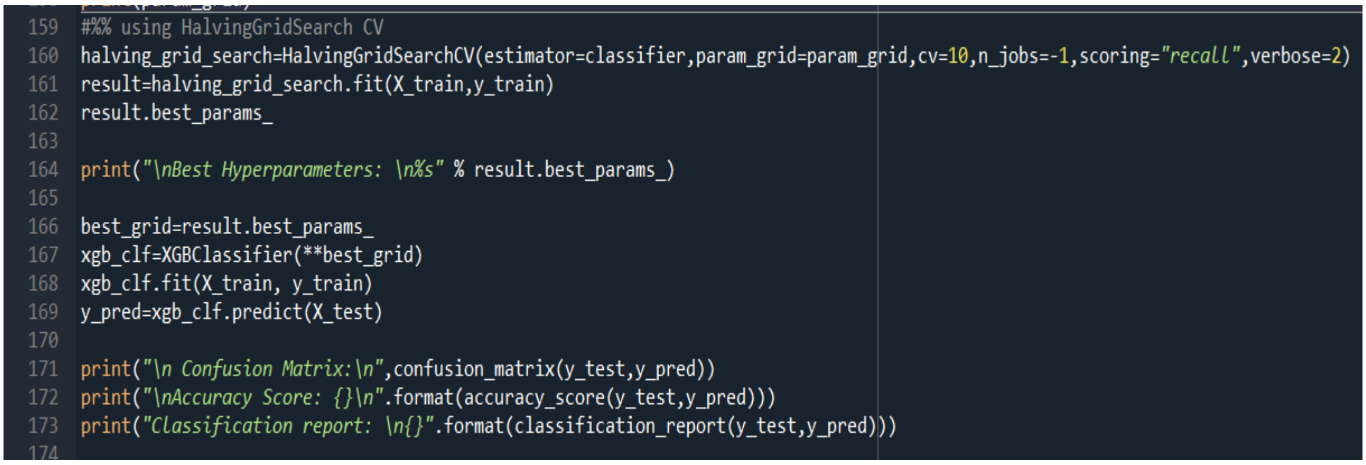
Let's build another model which is XGBoost . XGBoost has many hyper-parameters which make it powerful and flexible, but also very difficult to tune due to the high-dimensional parameter space. Hyper-parameter tuning can considerably improve the performance of learning algorithms Hence I have used the HalvingGridSearchCv which is 20 times faster than GridSearchCV .The HalvingGridSearchCV trains a subset of data to all the combinations of parameters. Also Top-performing candidates or combinations are selected and a larger subset of training data is trained on the top-performing candidates. The above steps are repeated until the best set of hyperparameters are obtained. Scikit-Learn library comes with the implementation of the Halving Grid Search CV algorithm.



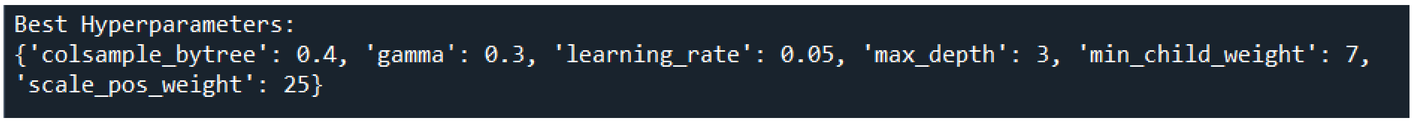
```
#%% XGBoost Classifier
classifier=XGBClassifier()
#%% Defining the grid

param_grid = {
     "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
     "max_depth"        : [3, 4, 7, 10, 25],
     "min_child_weight" : [1, 3, 5, 7],
     "gamma"            : [0.0, 0.1, 0.2 , 0.3, 0.4],
     "colsample_bytree" : [0.3, 0.4, 0.5 , 0.7],
     "scale_pos_weight" : [1, 3, 5, 10, 25]
     }

print(param_grid)
```

.

Halving Grid Search CV

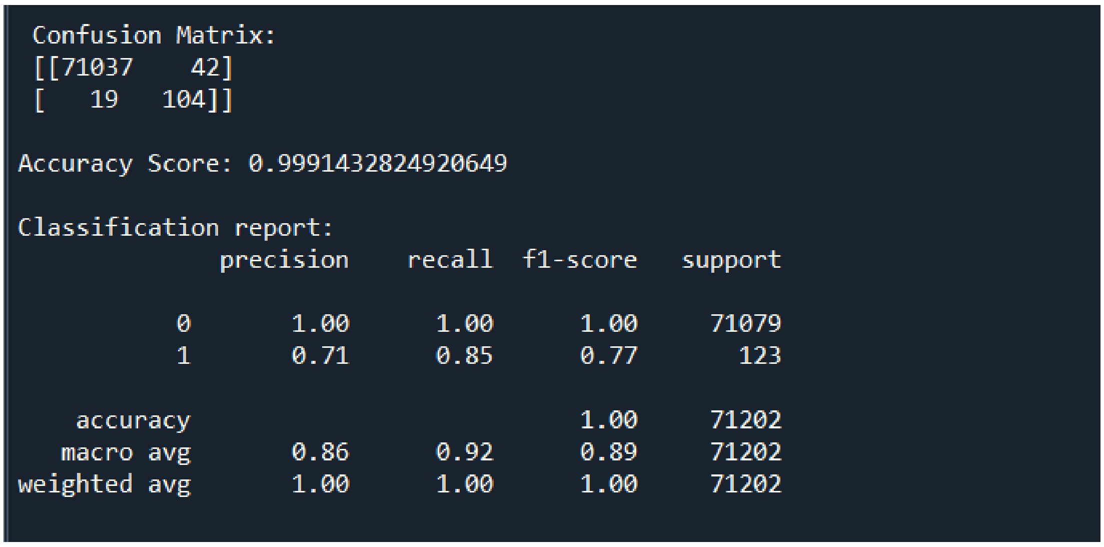


```
#%% using HalvingGridSearch CV
halving_grid_search=HalvingGridSearchCV(estimator=classifier,param_grid=param_grid,cv=10,n_jobs=-1,scoring="recall",verbose=2)
result=halving_grid_search.fit(X_train,y_train)
result.best_params_

print("\nBest Hyperparameters: \n%s" % result.best_params_)

best_grid=result.best_params_
xgb_clf=XGBClassifier(**best_grid)
xgb_clf.fit(X_train, y_train)
y_pred=xgb_clf.predict(X_test)

print("\n Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("\nAccuracy Score: {}\n".format(accuracy_score(y_test,y_pred)))
print("Classification report: \n{}".format(classification_report(y_test,y_pred)))
```



```
Best Hyperparameters:
{'colsample_bytree': 0.4, 'gamma': 0.3, 'learning_rate': 0.05, 'max_depth': 3, 'min_child_weight': 7,
'scale_pos_weight': 25}
```

Result



```
 Confusion Matrix:
 [[71037    42]
 [   19   104]]

Accuracy Score: 0.9991432824920649

Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.71      0.85      0.77       123

    accuracy                           1.00     71202
   macro avg       0.86      0.92      0.89     71202
weighted avg       1.00      1.00      1.00     71202
```

Let’s Understanding the confusion matrix of the XGBoost model. In the first row the number of true negatives is 71037and the number of false positives is 42 . In the second row number of false negatives 19 and the number of true positives is 104. We can say that our model has classified the non-fraud transactions pretty well. The recall score was about 0.85 and f1 score of 0.77 which is quite good. This high percentage of accuracy is to be expected due to the huge imbalance between the number of invalid and number of valid transactions.

2.7 Ausblick

Credit card fraud is no doubt a serious problem encountered by people who use them and who isuue them. This project aimed to solve the problem by building two models , but we can use many other algorithms too. But due to time and resourse restriction i have focused on only two ensembling tecniques.