In this code I use certain imports like sklearn where it helps me to preprocess the data sets, and there is sklearn.metrics where I want to use the function of classification report to see like the f1 score accuracy and so on, I also use sklearn.model_selection to import train_test_split to train the dataset and test it for the like logic regressin, svm and random forest tree.

```python
import seaborn as sb
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import metrics, preprocessing
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

Python

```python
data = pd.read_csv("Assignment3-Credit-Card-Interest.csv")
```

Python

```python
data.head()
```

Python

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	SCVAQHXI	Female	29	RG277	Salaried	X1	26	No	302875	No	0
1	BHYZRTQT	Female	47	RG276	Self_Employed	X1	15	No	645552	Yes	0
2	6STHBLXU	Male	44	RG283	Self_Employed	X3	85	No	725777	Yes	0
3	KVFKFAM7	Male	77	RG268	Other	X3	49	No	2297704	Yes	0
4	RMCZIYYW	Female	84	RG279	Other	X2	93	No	555199	No	1

```python
data.info()
```

Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 196580 entries, 0 to 196579
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ID                   196580 non-null  object
 1   Gender               196580 non-null  object
 2   Age                  196580 non-null  int64
 3   Region_Code          196580 non-null  object
 4   Occupation           196580 non-null  object
 5   Channel_Code         196580 non-null  object
 6   Vintage              196580 non-null  int64
 7   Credit_Product       173216 non-null  object
 8   Avg_Account_Balance  196580 non-null  int64
 9   Is_Active            196580 non-null  object
 10  Is_Lead              196580 non-null  int64
dtypes: int64(4), object(7)
memory usage: 16.5+ MB
```

In the dataset, I've use data.head to see the info of the datasets, where we can see there are some datasets that uses strings, not number and when we are about to use like for logic regression, svm and random forest tree, usually they use numeric values, not strings, so I've decide to change all the strings into numeric values./

```python
enc = preprocessing.LabelEncoder()
data['Gender'] = enc.fit_transform(data['Gender'])
data['Occupation'] = enc.fit_transform(data['Occupation'])
data['Region_Code'] = enc.fit_transform(data['Region_Code'])
data['Channel_Code'] = enc.fit_transform(data['Channel_Code'])
data['Credit_Product'] = enc.fit_transform(data['Credit_Product'])
data['Is_Active'] = enc.fit_transform(data['Is_Active'])
```

Python

```python
data.head()
```

Python

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	SCVAQHXI	0	29	27	2	0	26	0	302875	0	0
1	BHYZRTQT	0	47	26	3	0	15	0	645552	1	0
2	6STHBLXU	1	44	33	3	2	85	0	725777	1	0
3	KVFKFAM7	1	77	18	1	2	49	0	2297704	1	0
4	RMCZIYYW	0	84	29	1	1	93	0	555199	0	1

We can see that every strings become numeric values, next I went to see if there is any null or outliers, and outliers there is quite a few where our main target the Is_lead have at least 150003outliers, which is a lot, so later in the code, I will normalize the data, but before I normalize the data, I want to see the accuracy before and after normalize the dataset.

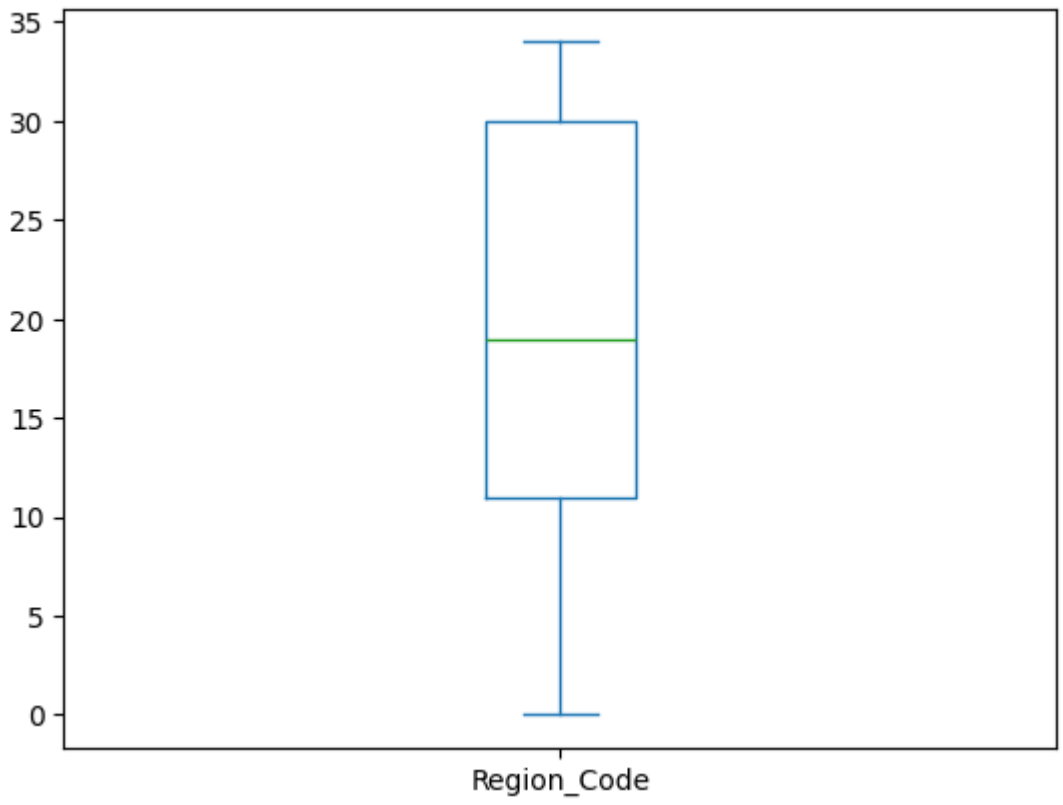
Next I've done like some visualization like boxplot, datas and others

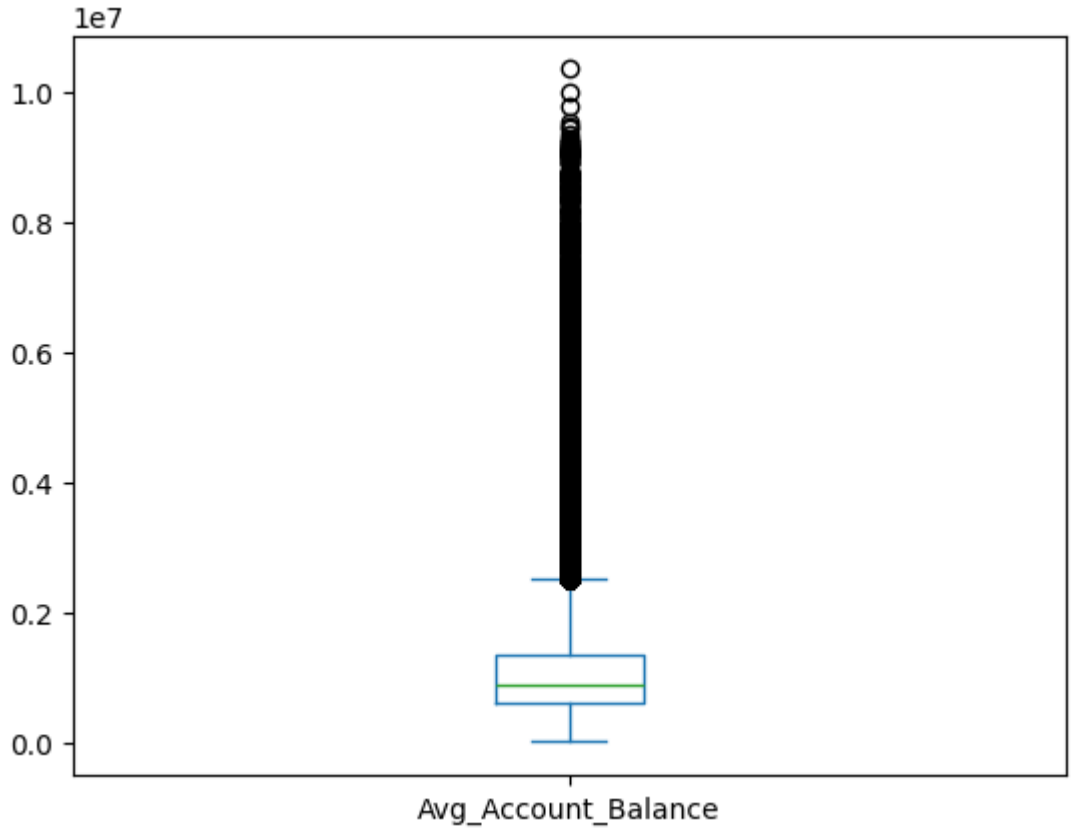
```python
numerical = ['Region_Code', 'Avg_Account_Balance', 'Is_Active', 'Is_Lead']
```

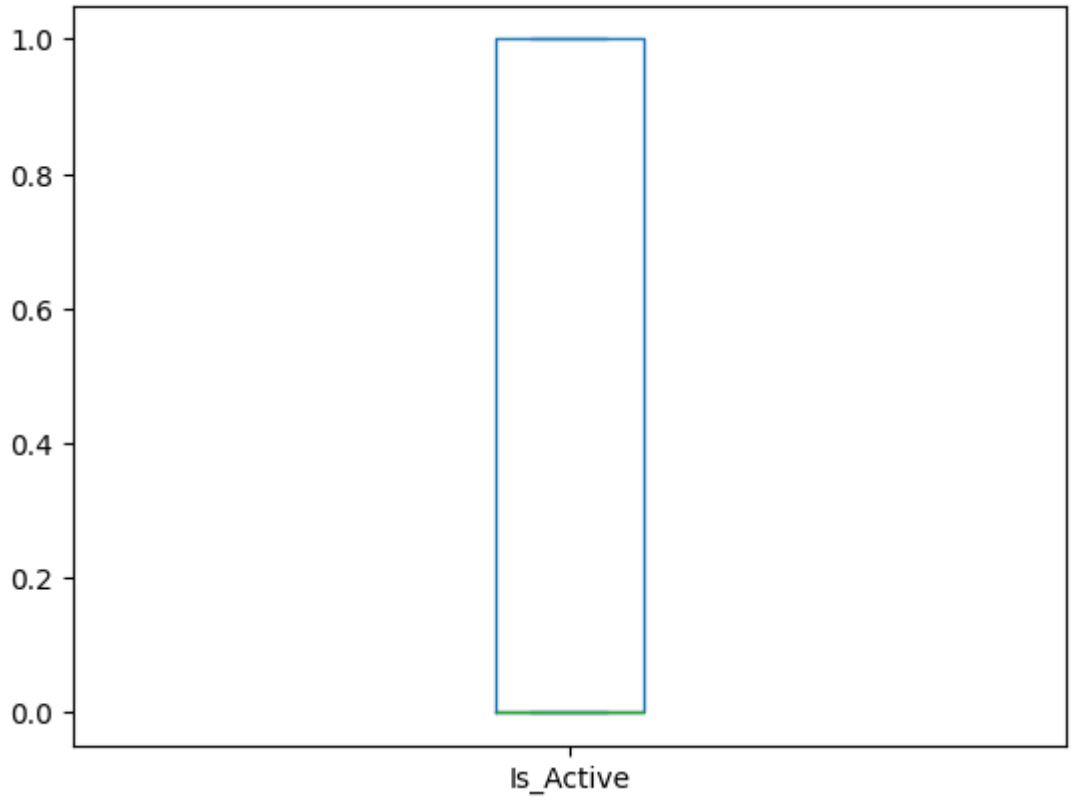
Python

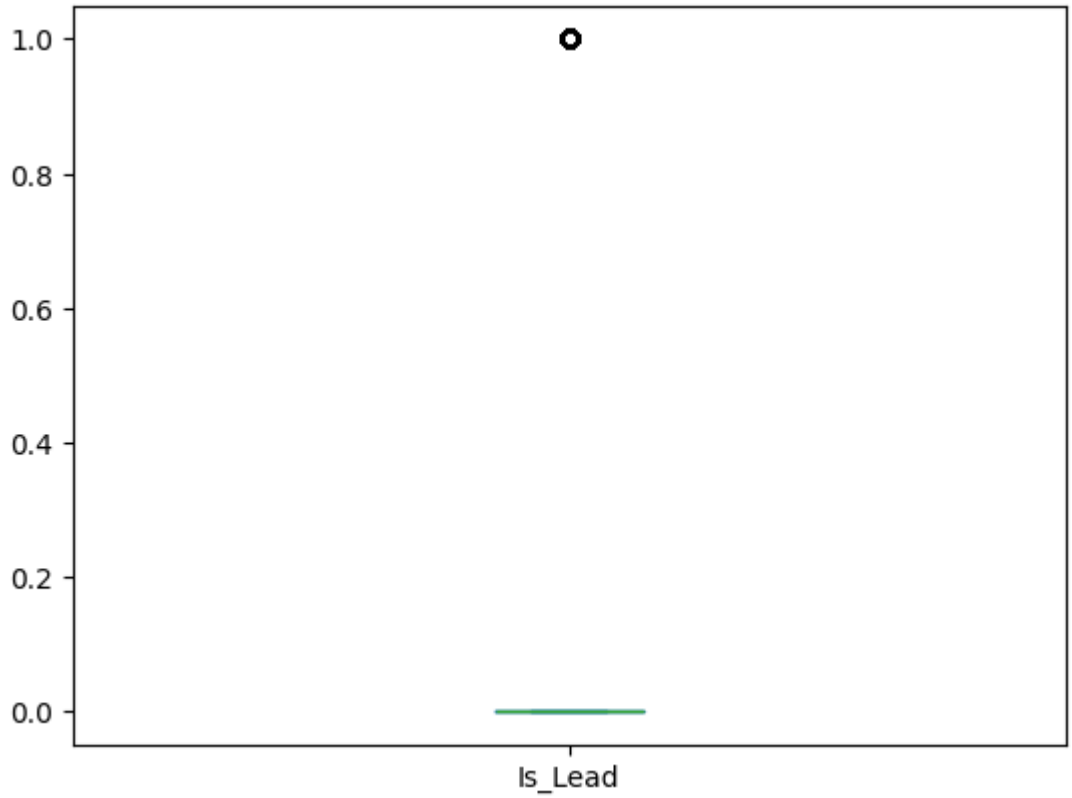
```python
for num in numerical:
    ax = data[num].plot(kind = 'box')
    plt.show()
```

Python




1e7
1.0
0.8
0.6
0.4
0.2
0.0
Avg_Account_Balance

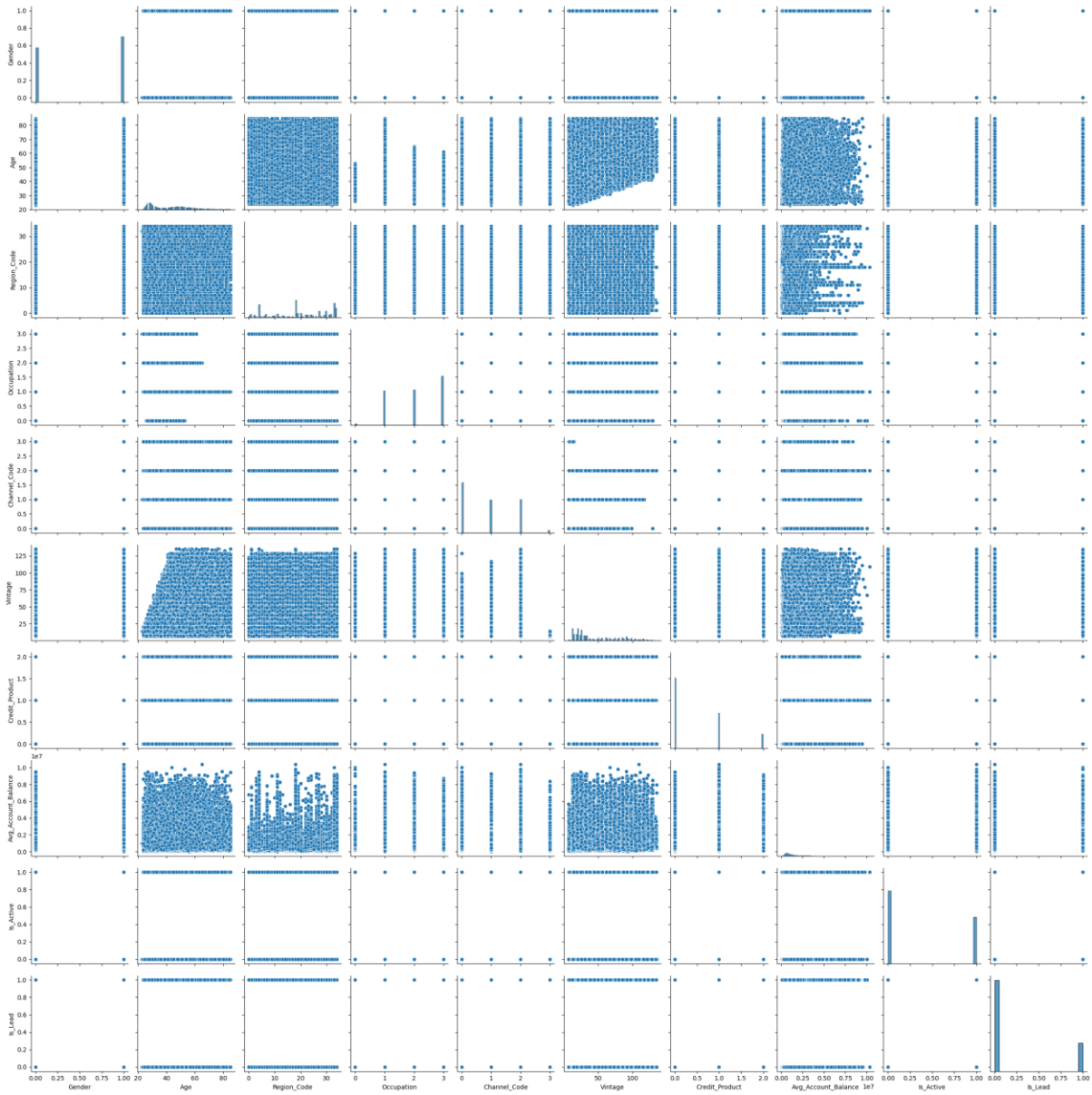

1.0
0.8
0.6
0.4
0.2
0.0
Is_Active




```
sb.pairplot(data)
```

Python

```
<seaborn.axisgrid.PairGrid at 0x1d245756e60>
```



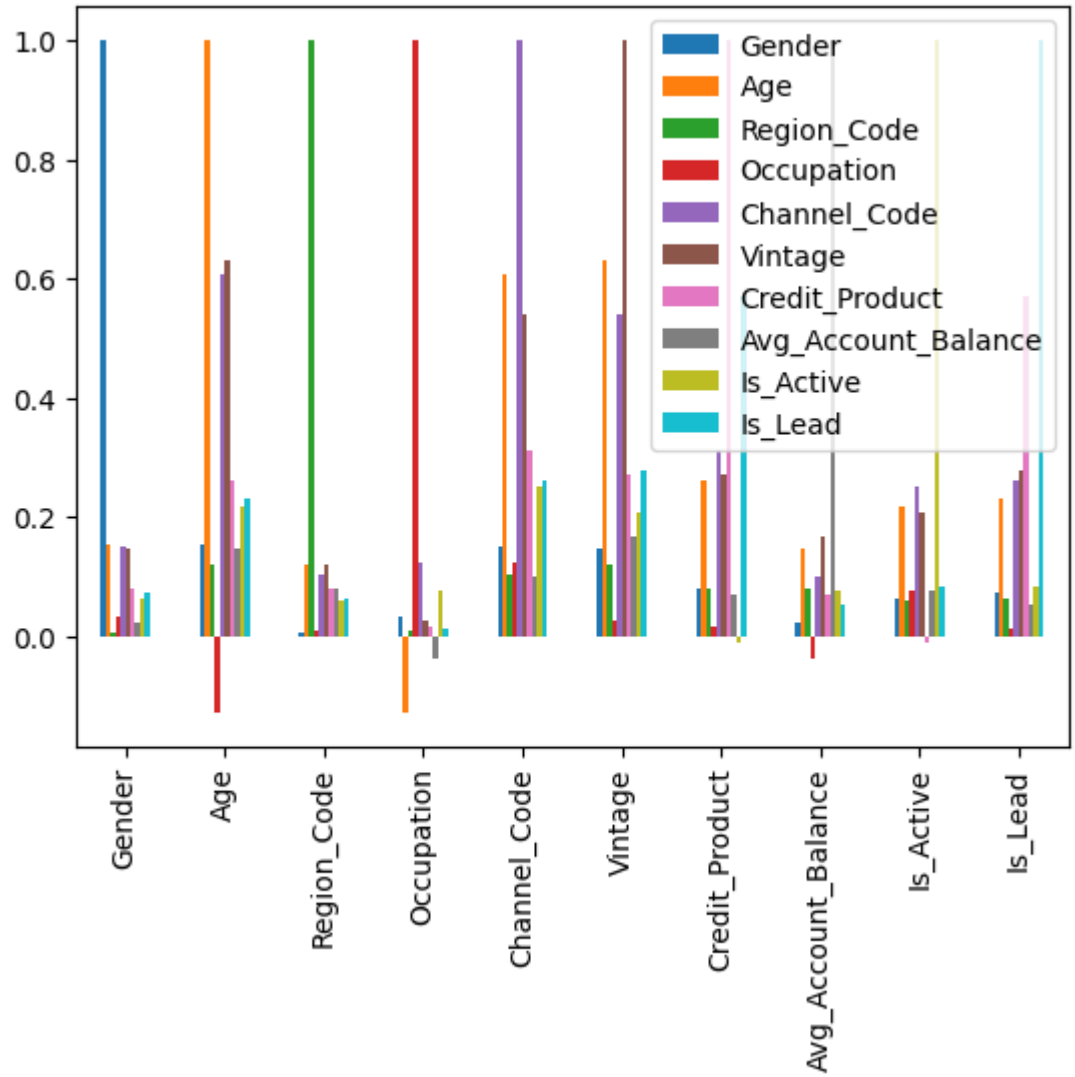

```python
data = data.drop(columns = 'ID')
```

Python

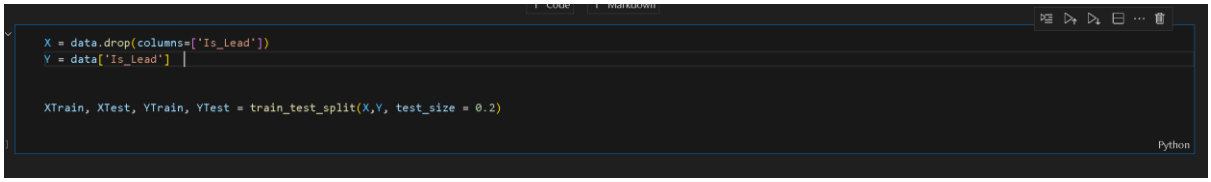
```python
corr_data = data.corr()
corr_data.plot.bar()
```

Python

```
<Axes: >
```



And then I've initiate X and Y where target is is_lead because is_lead already give out like the types of people that want to have the credit score or not



```
X = data.drop(columns=['Is_Lead'])
Y = data['Is_Lead']


XTrain, XTest, YTrain, YTest = train_test_split(X,Y, test_size = 0.2)
```

BEFORE NORMALIZATION:

So I use logicregression, svm and random forest tree

Logic regression:

```python
lgr = LogisticRegression().fit(XTrain, YTrain)
predictLGR = lgr.predict(XTest)

print(classification_report(YTest, predictLGR))
print(classification_report(YTrain, lgr.predict(XTrain)))
```

Python

```
c:\Users\gil\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linea
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(
              precision    recall  f1-score   support

           0       0.79      0.96      0.86     29885
           1       0.57      0.18      0.27      9431

    accuracy                           0.77     39316
   macro avg       0.68      0.57      0.57     39316
weighted avg       0.74      0.77      0.72     39316

              precision    recall  f1-score   support

           0       0.79      0.96      0.87    120118
           1       0.57      0.18      0.28     37146

    accuracy                           0.77    157264
   macro avg       0.68      0.57      0.57    157264
weighted avg       0.74      0.77      0.73    157264
```

Is 0.77 accuracy

SVM:

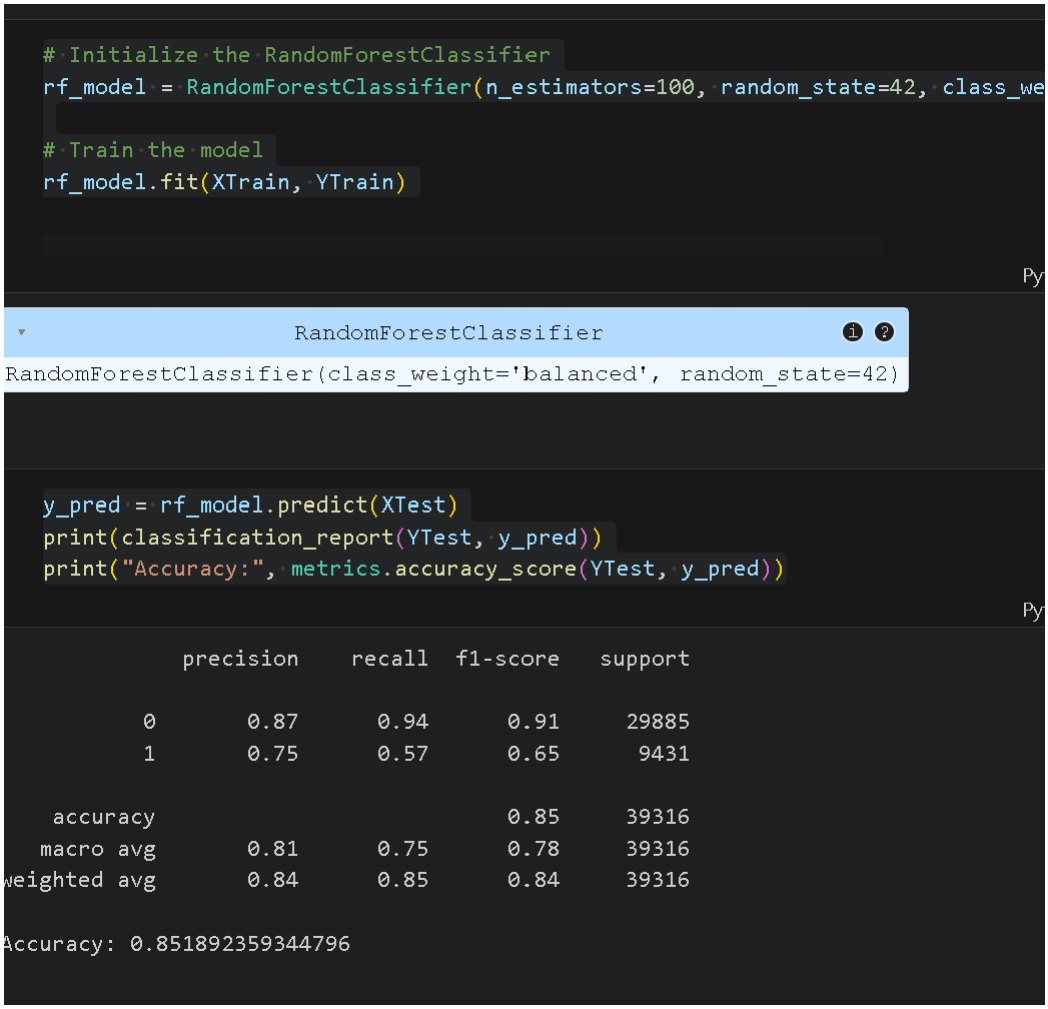
```python
SVM = SVC().fit(XTrain, YTrain)
CTest = SVM.predict(XTest)
CTrain = SVM.predict(XTrain)
print("Train set acc: ", metrics.accuracy_score(YTrain, CTrain))
print("Test set acc: ", metrics.accuracy_score(YTest, CTest))
```

Pyth

```
Train set acc:  0.7637984535558042
Test set acc:  0.7601231050971614
```

Is 0.76 accuracy

And randomforest tree:



```
# Initialize the RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42, class_we

# Train the model
rf_model.fit(XTrain, YTrain)
```

```
RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
y_pred = rf_model.predict(XTest)
print(classification_report(YTest, y_pred))
print("Accuracy:", metrics.accuracy_score(YTest, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.91     29885
           1       0.75      0.57      0.65      9431

    accuracy                           0.85     39316
   macro avg       0.81      0.75      0.78     39316
weighted avg       0.84      0.85      0.84     39316

Accuracy: 0.851892359344796
```

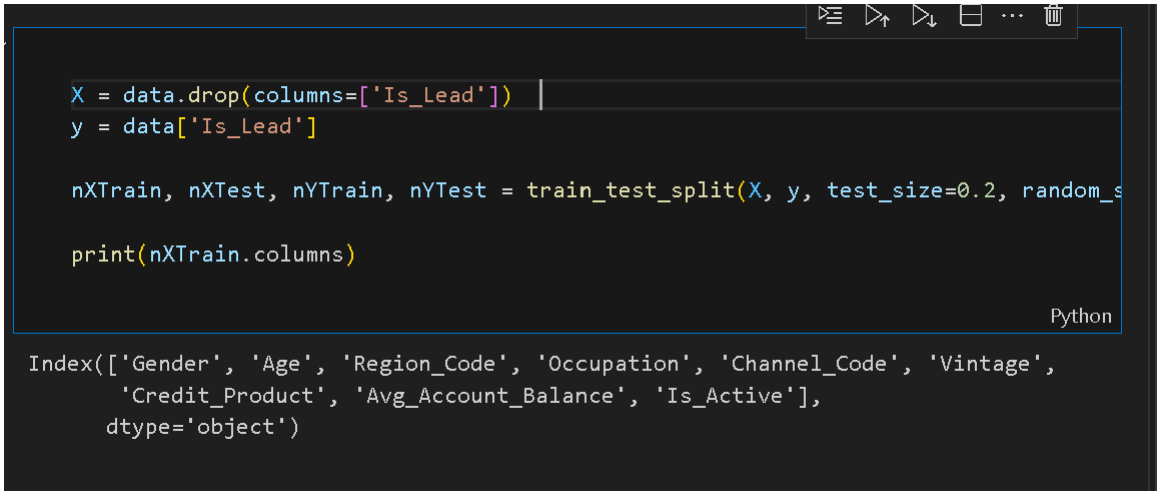
Is 0.85

And now is NORMALIZING THE DATASET

normalized = data.copy()

normalized["Is_Lead"] = (normalized["Is_Lead"] - normalized["Is_Lead"].min() / normalized["Is_Lead"].max() - normalized["Is_Lead"].min())

normalized["Is_Lead"] = (normalized["Is_Lead"] - normalized["Is_Lead"].min() / normalized["Is_Lead"].max() - normalized["Is_Lead"].min())



Where I reset like the dataset data.drop because if I didn't do this, the code already knows the previous answer, so it will be cheating and before I didn't do this, everything is 100% accuracy which is wrong because there isn't any 100% accuracy for training models

Results:

Logistic regression:

```
    lgr = LogisticRegression().fit(nXTrain, nYTrain)
    npredictLGR = lgr.predict(nXTest)

    print(classification_report(nYTest, npredictLGR))
    print(classification_report(nYTrain, lgr.predict(nXTrain)))
```

```
c:\Users\gil\AppData\Local\Programs\Python\Python310\lib\site-packages\sklear
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-regres
  n_iter_i = _check_optimize_result(
              precision    recall  f1-score   support

           0       0.78      0.96      0.86     29908
           1       0.56      0.15      0.23      9408

    accuracy                           0.77     39316
   macro avg       0.67      0.55      0.55     39316
weighted avg       0.73      0.77      0.71     39316

              precision    recall  f1-score   support

           0       0.79      0.96      0.87    120095
           1       0.55      0.15      0.24     37169

    accuracy                           0.77    157264
   macro avg       0.67      0.56      0.55    157264
weighted avg       0.73      0.77      0.72    157264
```

0.77 accuracy

Svm:

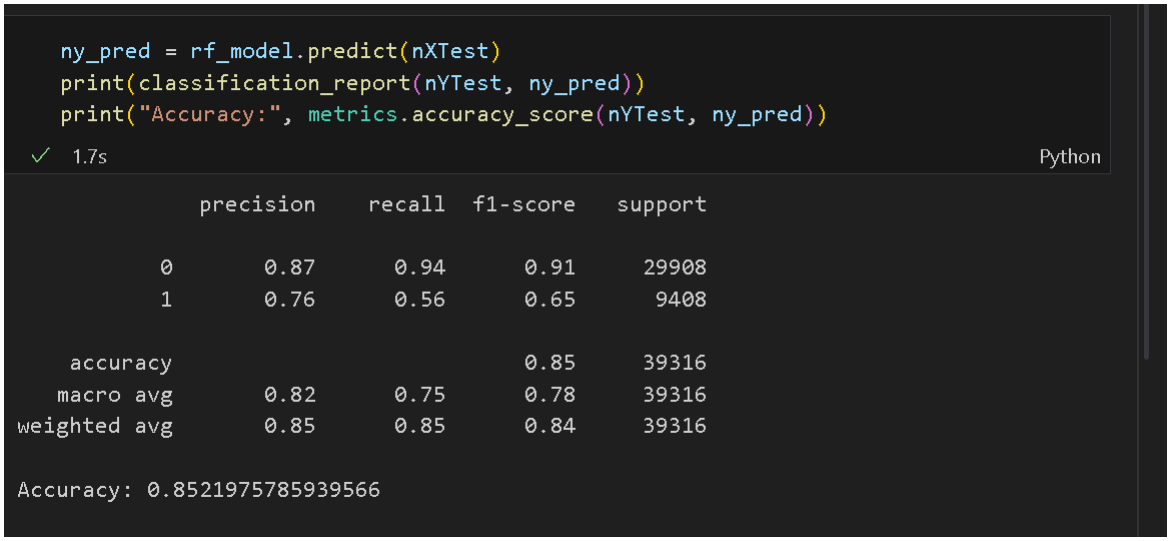
```
    SVM = SVC().fit(nXTrain, nYTrain)
    nCTest = SVM.predict(nXTest)
    nCTrain = SVM.predict(nXTrain)
    print("Train set acc: ", metrics.accuracy_score(nYTrain, nCTrain))
    print("Test set acc: ", metrics.accuracy_score(nYTest, nCTest))
```

✓ 31m 50.9s

```
Train set acc:  0.7636522026655814
Test set acc:  0.7607081086580527
```

0.76 accuracy

Random forest tree:



0.85 accuracy

As you can see between unnormalize and normalize, there is a tiny bit difference where normalize have a little of increase of accuracy then the unnormalize dataset

We can see from 3 different classifiers or models, the lowest accuracy is svm with 0.76, second is logistic regression with 0.77 and first is random forest tree with 0.85 accuracy this means that random forest tree model is more suitable and accurate.