Final Machine learning project

1. **Author(s)**

- **Chanho Park**
- **Dunsin Fayode**
- **Gilbert Tetteh**

2. **Project characteristics**

For the past few years, search advertising has been one of the major revenue sources for the Internet industry. One of the best search advertising is to predict the click-through rate (pCTR) of ads, as the economic model behind search advertising requires pCTR values to rank ads and price clicks.

This project aims to build a machine learning model that predicts the click-through rate (pCTR) of ads based on the log/data provided by the Tencent proprietary search engine, soso.com.

3. **Description of fields in the log/data**

- **Click** - did the user click
- **DisplayURL** - ad URL (encoded as a huge integer)
- **AdId** - identifier of specific ad (integer)
- **AdvertiserId** - identifier of specific advertiser (integer)
- **Depth** - number of ads displayed in a session (1-3)
- **Position** - position of ad in the list of displayed ads
- **Gender** - 1,2 = male/female, 0 = unknown
- **Age** - discretized into 6 intervals
- **AdKeyword** tokens - keywords for an ad
- **AdTitle** tokens - the title of an ad
- **AdDescription** tokens - description of an add
- Query tokens - user query

4. **Process of building the model and packages used.**

To work on this project, our team worked together using Google Colab, as it provides the best virtual environment to work together, and we did not have to install any packages that we used for building our model. In total, we imported ten packages which are as follows:

1.**numpy**- to find out unique elements from an array.
2.**matplotlib.pyplot**- to plot figures.
3.**seaborn**- to make statistical graphics
4.**sklearn.model_selection** import **train_test_split -** to split arrays or matrices into random train and test subsets.
5.**sklearn.model_selection** import **cross_val_score** - cross validation.
6. **sklearn.model_selection** import **KFold, StratifiedKFold**

7.**sklearn.preprocessing** import **StandardScaler** - standardizing features by removing the mean and scaling to unit variance.

8.**sklearn.linear_model** import **LogisticRegression** - for Logistic regression

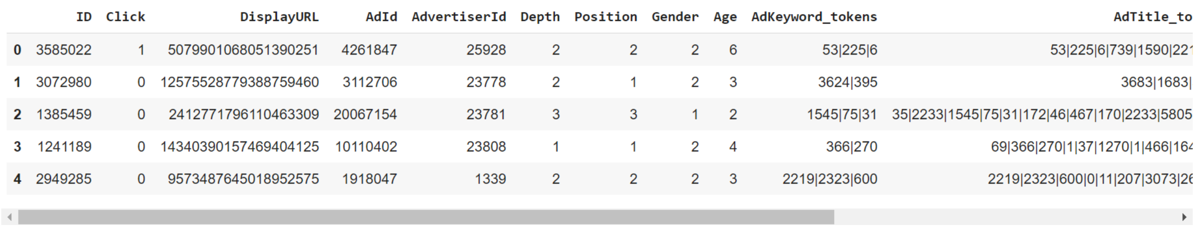
9.**imblearn.over_sampling** import **SMOTE** - to deal with imbalanced class

10.**re** -

We followed the Machine learning pipeline to build our model. Therefore, first, we extracted the data and inspected all the fields, then prepared the data and set some training models, such as Logistical regression. After training the model, we evaluated it and validated it.

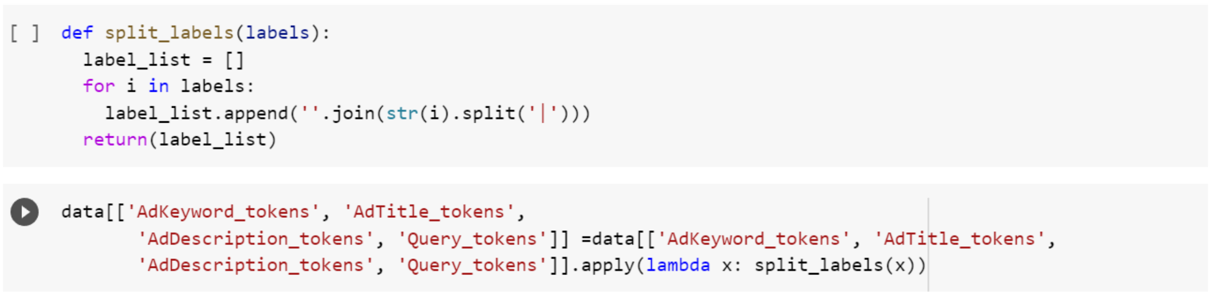
1. Detailed process of building the model

Primarily, we imported our log/data used for building the model, then imported the useful packages mentioned above. Then we began inspecting the structure of the log/data. First, we displayed the top 5 rows of our data.



	ID	Click	DisplayURL	AdId	AdvertiserId	Depth	Position	Gender	Age	AdKeyword_tokens	AdTitle_to
0	3585022	1	5079901068051390251	4261847	25928	2	2	2	6	53|225|6	53|225|6|739|1590|22
1	3072980	0	12575528779388759460	3112706	23778	2	1	2	3	3624|395	3683|1683|
2	1385459	0	2412771796110463309	20067154	23781	3	3	1	2	1545|75|31	35|2233|1545|75|31|172|46|467|170|2233|5805
3	1241189	0	14340390157469404125	10110402	23808	1	1	2	4	366|270	69|366|270|1|37|1270|1|466|16
4	2949285	0	9573487645018952575	1918047	1339	2	2	2	3	2219|2323|600	2219|2323|600|0|11|207|3073|2

We found out that there are special symbol used in AdKeyword_tokens', 'AdTitle_tokens', 'AdDescription_tokens', 'Query_tokens' (i.e. " | ") which will affect our further analysis,therefore, we split this symbol by using this function:

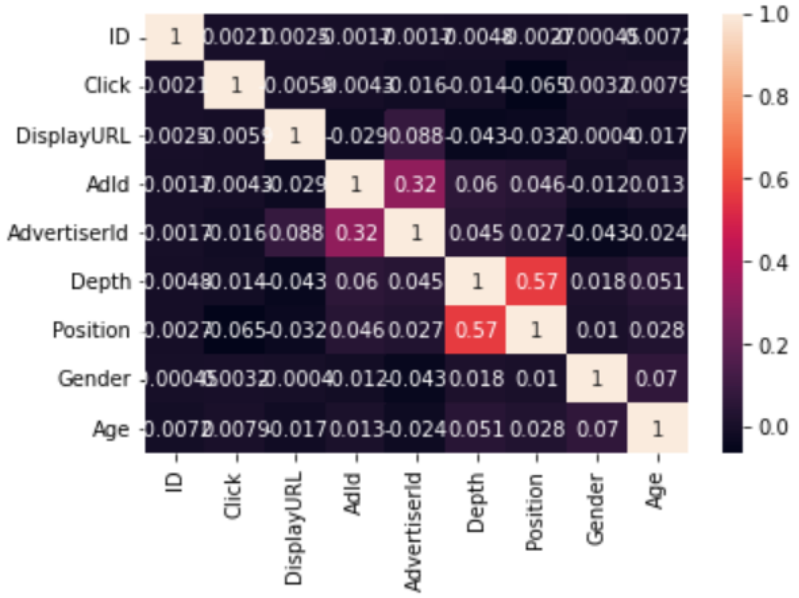


```
[ ] def split_labels(labels):
      label_list = []
      for i in labels:
        label_list.append(''.join(str(i).split('|')))
      return(label_list)
```

```
data[['AdKeyword_tokens', 'AdTitle_tokens',
       'AdDescription_tokens', 'Query_tokens']] =data[['AdKeyword_tokens', 'AdTitle_tokens',
       'AdDescription_tokens', 'Query_tokens']].apply(lambda x: split_labels(x))
```

After splitting the special symbol, we checked the correlation between different variables to find out which ones were most important(positively correlated) and plotted a heatmap using the seaborn package.

```python
a = data.corr()
sns.heatmap(a, annot=True)
plt.show()
```



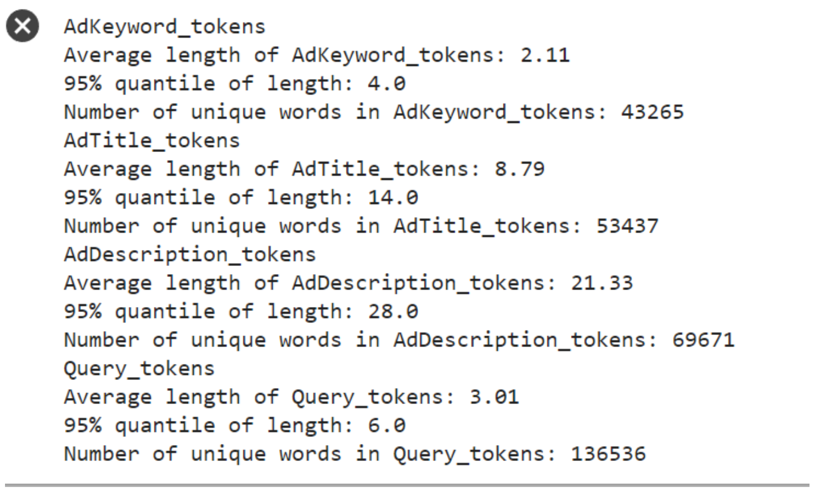
Then, we explored how many data entries were there and whether there were any null values, and we found out that there were no nulls, and there were 4 million data entries and 13 columns.

```
ID                      0
Click                   0
DisplayURL              0
AdId                    0
AdvertiserId            0
Depth                   0
Position                0
Gender                  0
Age                     0
AdKeyword_tokens        0
AdTitle_tokens          0
AdDescription_tokens    0
Query_tokens            0
dtype: int64
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000000 entries, 0 to 3999999
Data columns (total 13 columns):
 #   Column                Dtype
---  ------                -----
 0   ID                    int64
 1   Click                 int64
 2   DisplayURL            uint64
 3   AdId                  int64
 4   AdvertiserId          int64
 5   Depth                 int64
 6   Position              int64
 7   Gender                int64
 8   Age                   int64
 9   AdKeyword_tokens      object
 10  AdTitle_tokens        object
 11  AdDescription_tokens  object
 12  Query_tokens          object
dtypes: int64(8), object(4), uint64(1)
memory usage: 396.7+ MB
```

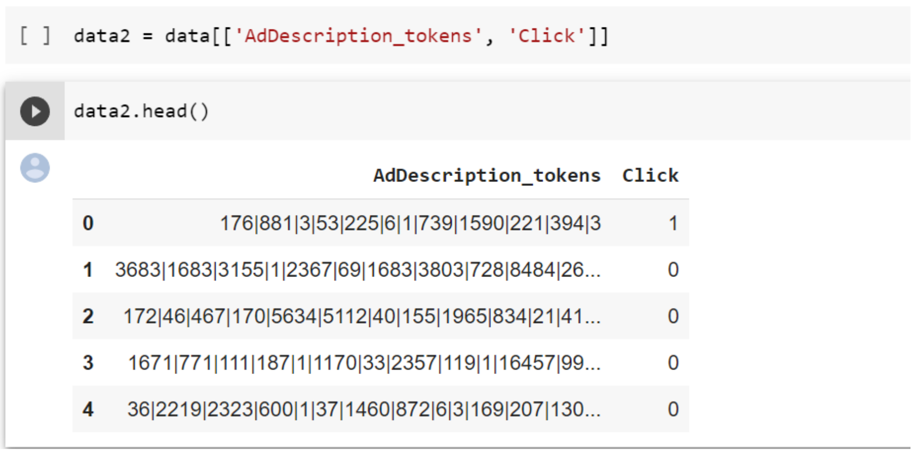
Then, we inspected our target variable “Click”, as the pCTR rate is measured by the number of clicks. We noticed that we have an Imbalance class and we have to find the best Metrics for it. 95% to 5% of data.

Then, explored the text columns, which were 'AdKeyword_tokens', 'AdTitle_tokens','AdDescription_tokens', 'Query_tokens'. And we found the following:



```
AdKeyword_tokens
Average length of AdKeyword_tokens: 2.11
95% quantile of length: 4.0
Number of unique words in AdKeyword_tokens: 43265
AdTitle_tokens
Average length of AdTitle_tokens: 8.79
95% quantile of length: 14.0
Number of unique words in AdTitle_tokens: 53437
AdDescription_tokens
Average length of AdDescription_tokens: 21.33
95% quantile of length: 28.0
Number of unique words in AdDescription_tokens: 69671
Query_tokens
Average length of Query_tokens: 3.01
95% quantile of length: 6.0
Number of unique words in Query_tokens: 136536
```

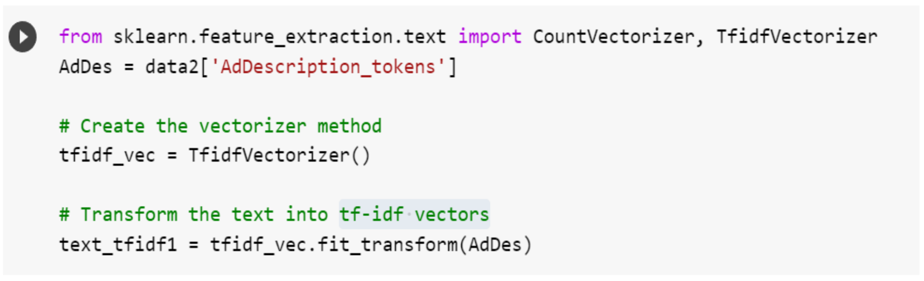
For practice purposes, we dropped the text encoded column. for Data1,
For Data2, we only included it in the data for the experiment.
Furthermore, we wanted to extract some unique features, so we created the vectorizer and transformed the text into tf-idf vectors.



```
[ ] data2 = data[['AdDescription_tokens', 'Click']]
```

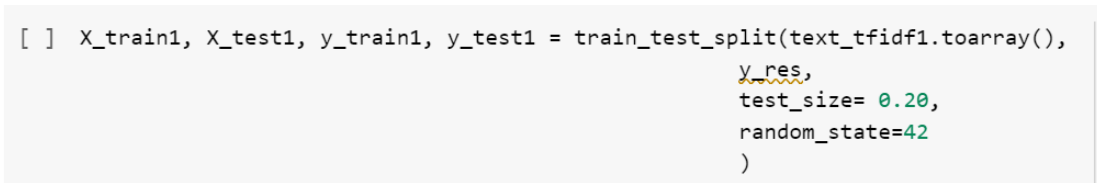
```
data2.head()
```

	AdDescription_tokens	Click
0	176|881|3|53|225|6|1|739|1590|221|394|3	1
1	3683|1683|3155|1|2367|69|1683|3803|728|8484|26...	0
2	172|46|467|170|5634|5112|40|155|1965|834|21|41...	0
3	1671|771|111|187|1|1170|33|2357|119|1|16457|99...	0
4	36|2219|2323|600|1|37|1460|872|6|3|169|207|130...	0

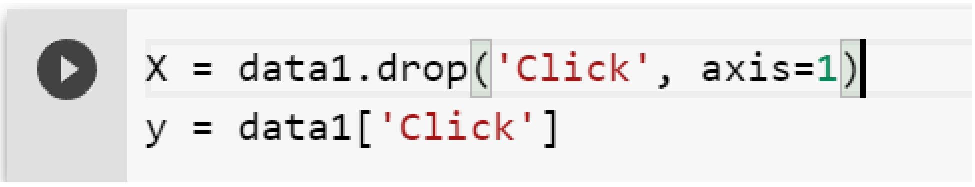


```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
AdDes = data2['AdDescription_tokens']

# Create the vectorizer method
tfidf_vec = TfidfVectorizer()

# Transform the text into tf-idf vectors
text_tfidf1 = tfidf_vec.fit_transform(AdDes)
```

Then, we tried to split the text, but we encountered a system crash after any process to pass the Text vector to Array.

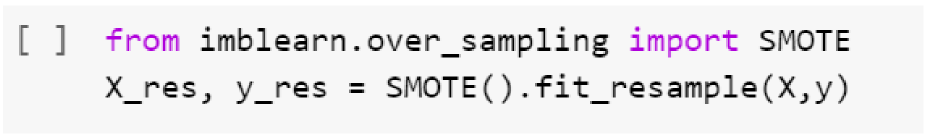


```
[ ] X_train1, X_test1, y_train1, y_test1 = train_test_split(text_tfidf1.toarray(),
                                                            y_res,
                                                            test_size= 0.20,
                                                            random_state=42
                                                            )
```

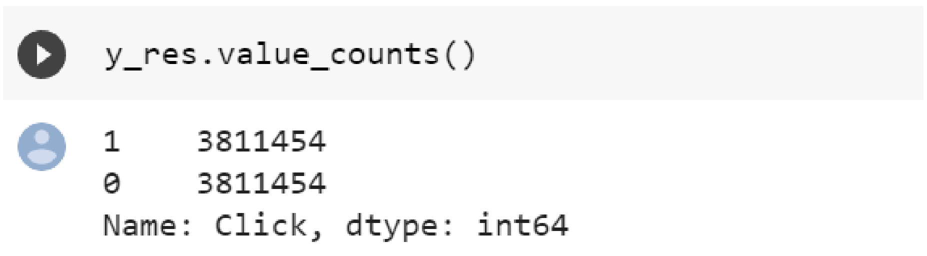


```
X = data1.drop('Click', axis=1)
y = data1['Click']
```

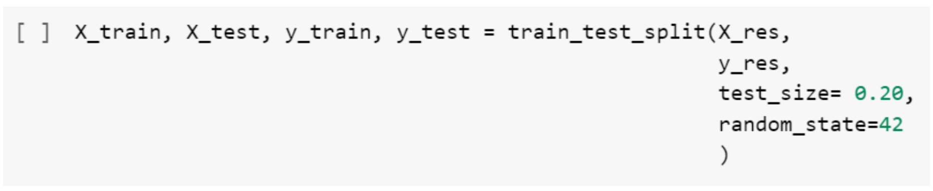
After splitting and exploring the data, we tried to deal with the imbalance of data we encountered. And to solve this problem, we imported SMOTE from imblearn.over_sampling, and managed to balance the data and split our data to train and test our data.



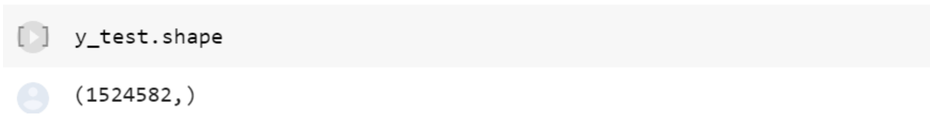
```
[ ] from imblearn.over_sampling import SMOTE
    X_res, y_res = SMOTE().fit_resample(X,y)
```



```
y_res.value_counts()

1    3811454
0    3811454
Name: Click, dtype: int64
```

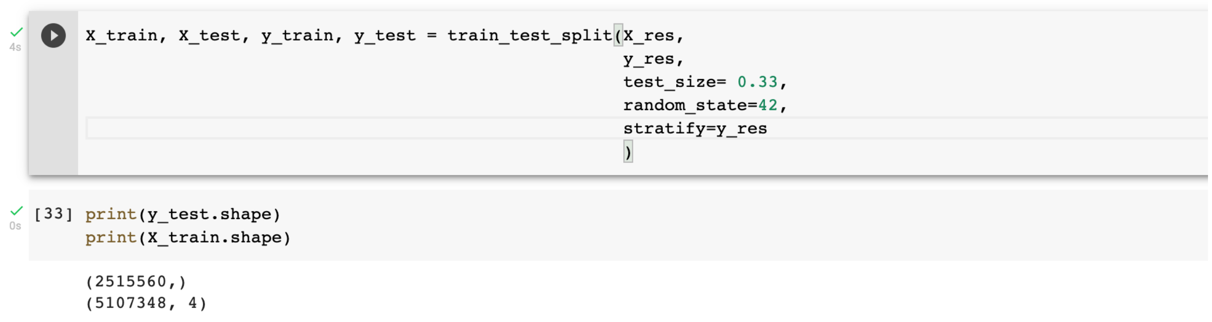


```
[ ] X_train, X_test, y_train, y_test = train_test_split(X_res,
                                                        y_res,
                                                        test_size= 0.20,
                                                        random_state=42
                                                        )
```



```
[ ] y_test.shape

(1524582,)
```

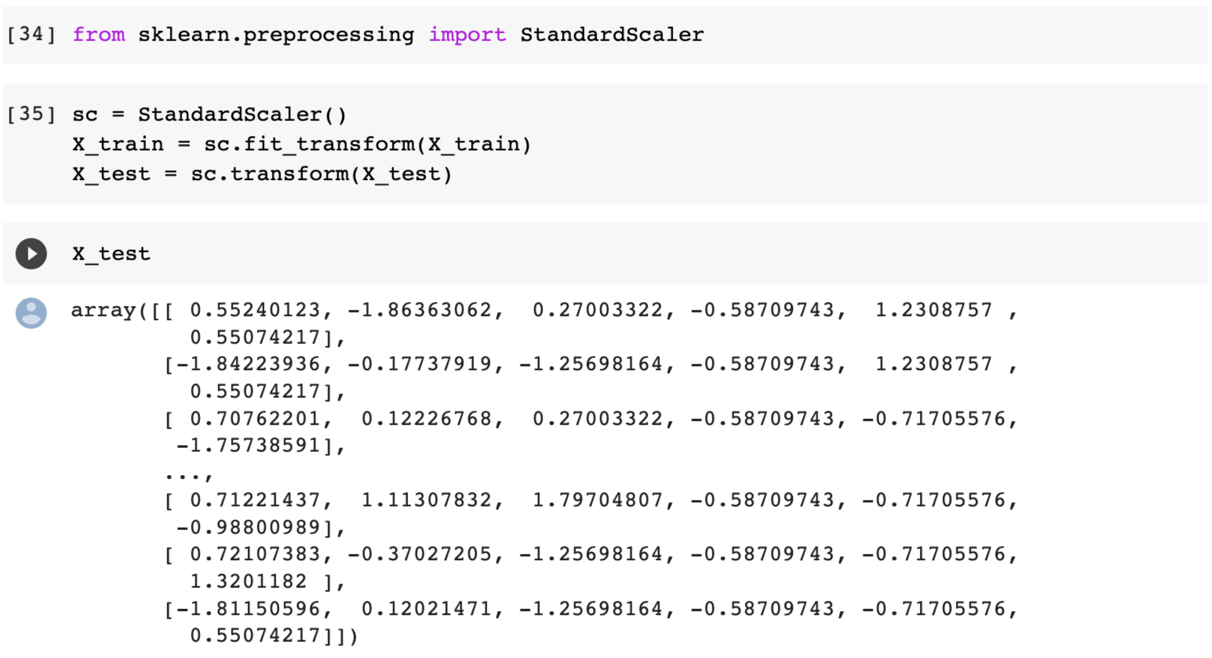
Smote works by selecting samples that are close to the feature, drawing a line between the sample in the feature space and a new sample at a point along that line. As described in the project we developed SMOTE by applying it to an imbalance classification dataset. With the split test size of 0.33 and random state of 42. (2515560,) (51073484, 4) for Y test shape and X train shape respectively.



```
X_train, X_test, y_train, y_test = train_test_split(X_res,
                                                    y_res,
                                                    test_size= 0.33,
                                                    random_state=42,
                                                    stratify=y_res
                                                    )
```

```
[33] print(y_test.shape)
     print(X_train.shape)

     (2515560,)
     (5107348, 4)
```

Feature scaling normalizes the range of features in the dataset, the features vary in terms of magnitude, range, and units. Scaling the features makes the flow of gradient descent smooth and helps algorithms quickly reach the minima of the cost function. It gave us a set of arrays in the standardscaler (X-test).

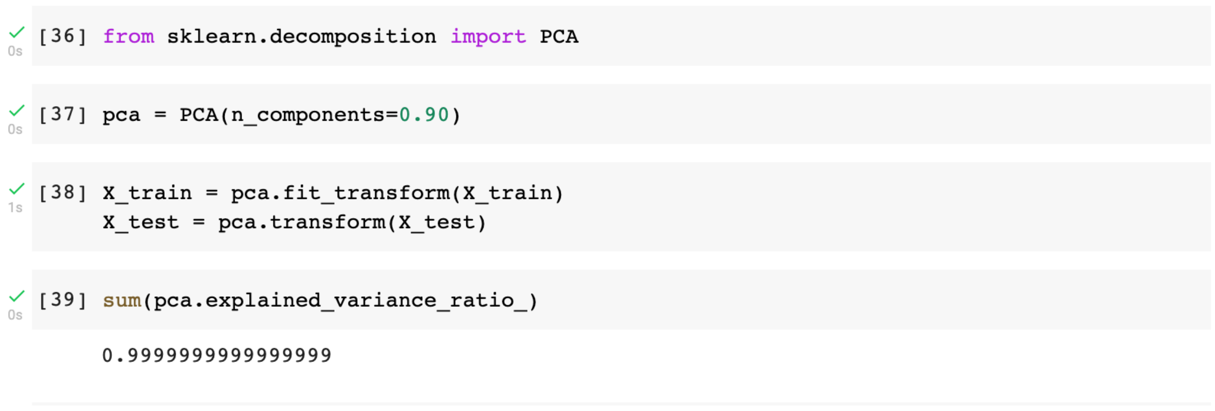


```
[34] from sklearn.preprocessing import StandardScaler
```

```
[35] sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

```
X_test

array([[ 0.55240123, -1.86363062,  0.27003322, -0.58709743,  1.2308757 ,
         0.55074217],
       [-1.84223936, -0.17737919, -1.25698164, -0.58709743,  1.2308757 ,
         0.55074217],
       [ 0.70762201,  0.12226768,  0.27003322, -0.58709743, -0.71705576,
        -1.75738591],
       ...,
       [ 0.71221437,  1.11307832,  1.79704807, -0.58709743, -0.71705576,
        -0.98800989],
       [ 0.72107383, -0.37027205, -1.25698164, -0.58709743, -0.71705576,
         1.3201182 ],
       [-1.81150596,  0.12021471, -1.25698164, -0.58709743, -0.71705576,
         0.55074217]])
```

The key aim of PCA is to reduce the number of variables of a data set while preserving as much information as possible. From sklearn decomposition we imported PCA.



```
[36] from sklearn.decomposition import PCA
```

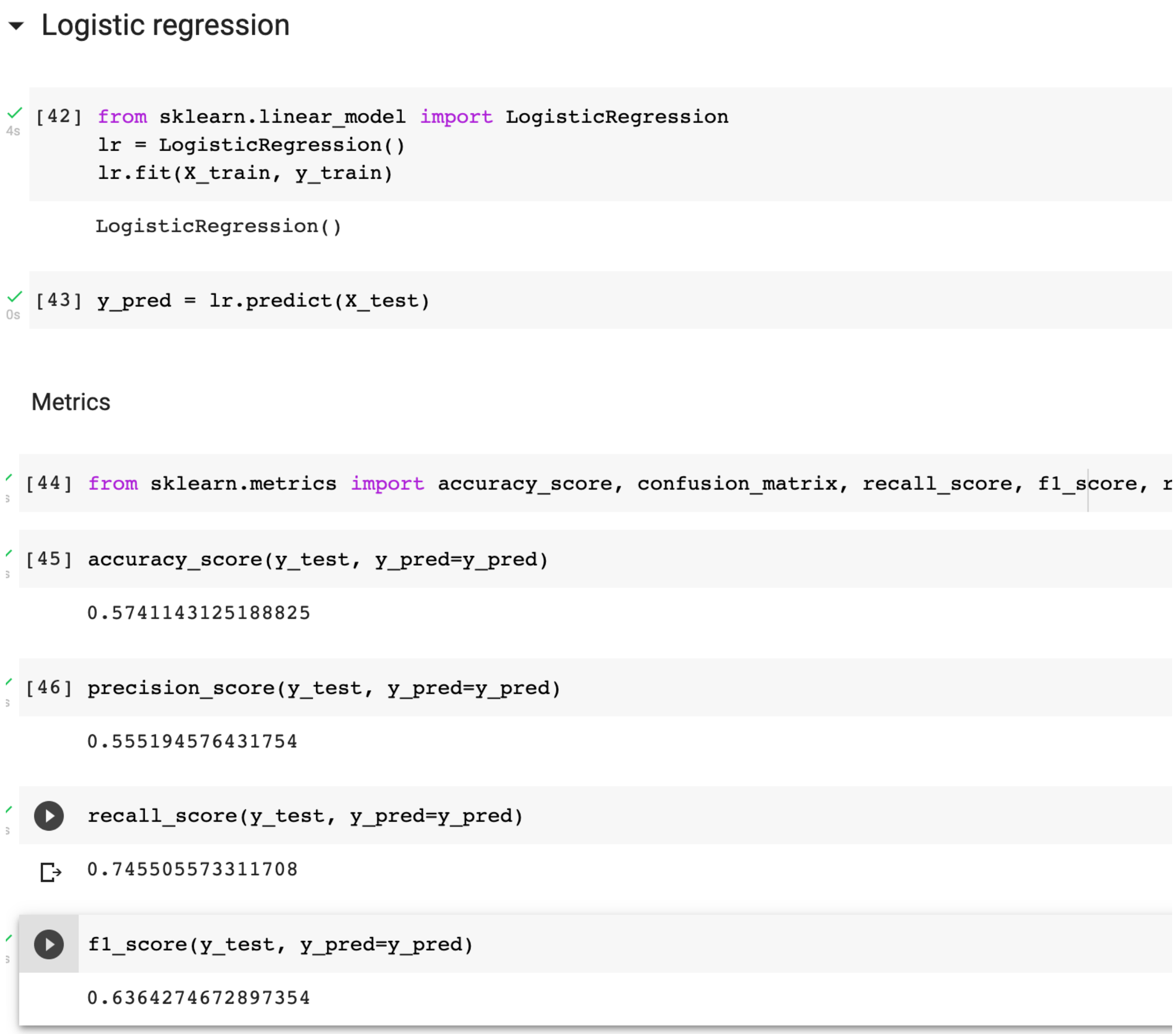
```
[37] pca = PCA(n_components=0.90)
```

```
[38] X_train = pca.fit_transform(X_train)
     X_test = pca.transform(X_test)
```

```
[39] sum(pca.explained_variance_ratio_)

     0.9999999999999999
```

.

The Metric of Regression predicted an accuracy score, precision score, recall score, and F1 score of 0.5741143125188825,0.555194576431754,0.745505573311708, and 0.6364274672897354 respectively.



Logistic regression

```
[42] from sklearn.linear_model import LogisticRegression
     lr = LogisticRegression()
     lr.fit(X_train, y_train)

     LogisticRegression()
```

```
[43] y_pred = lr.predict(X_test)
```

Metrics

```
[44] from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, f1_score, r
```

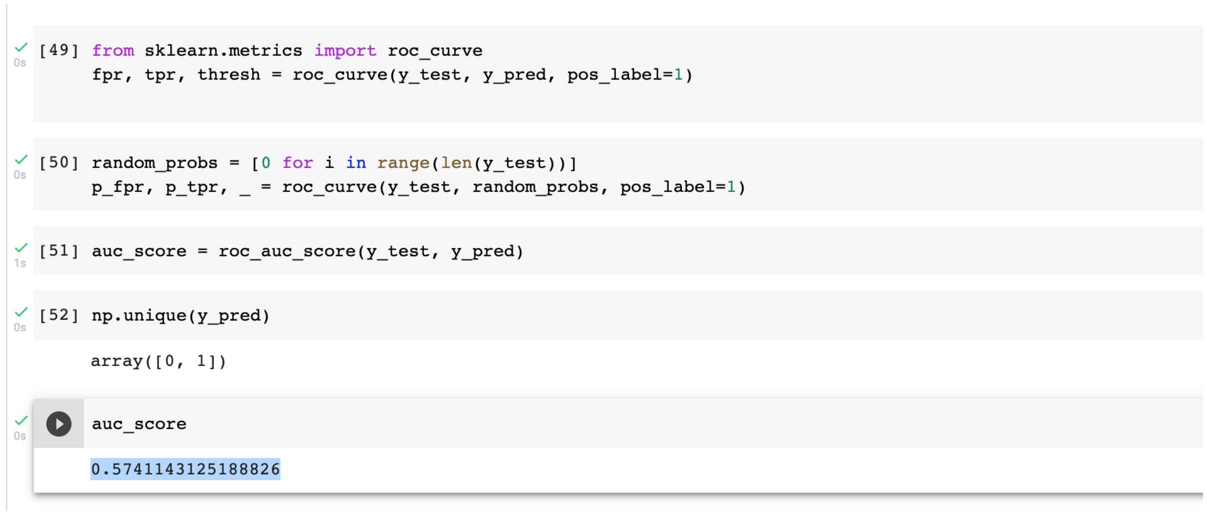
```
[45] accuracy_score(y_test, y_pred=y_pred)

     0.5741143125188825
```

```
[46] precision_score(y_test, y_pred=y_pred)

     0.555194576431754
```

```
     recall_score(y_test, y_pred=y_pred)

     0.745505573311708
```

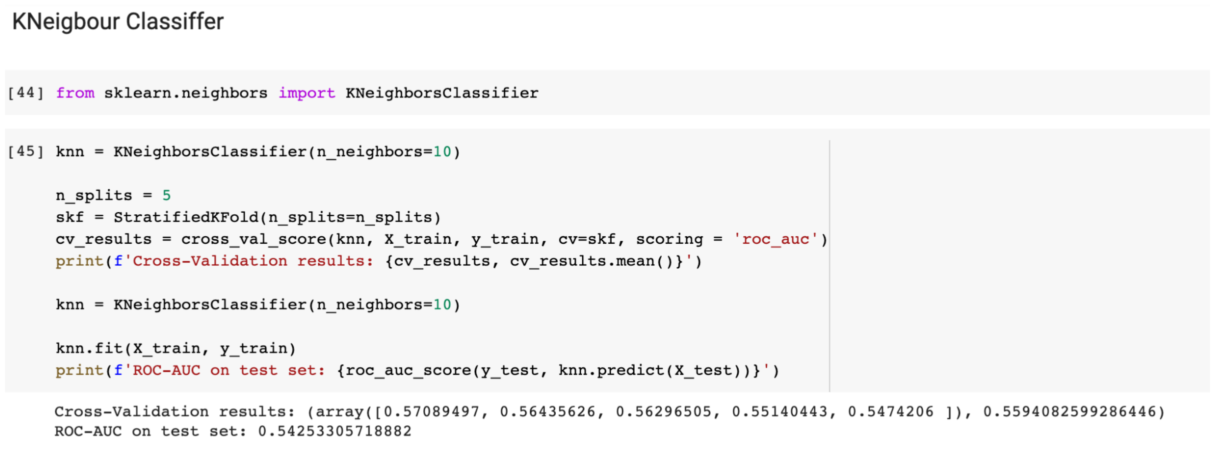
```
     f1_score(y_test, y_pred=y_pred)

     0.6364274672897354
```

The AUC score was computed using the roc_auc_score() method of sklearn. The unique(Y Prediction) gave us an array of (0,1) and AUC score of 0.5741143125188826.



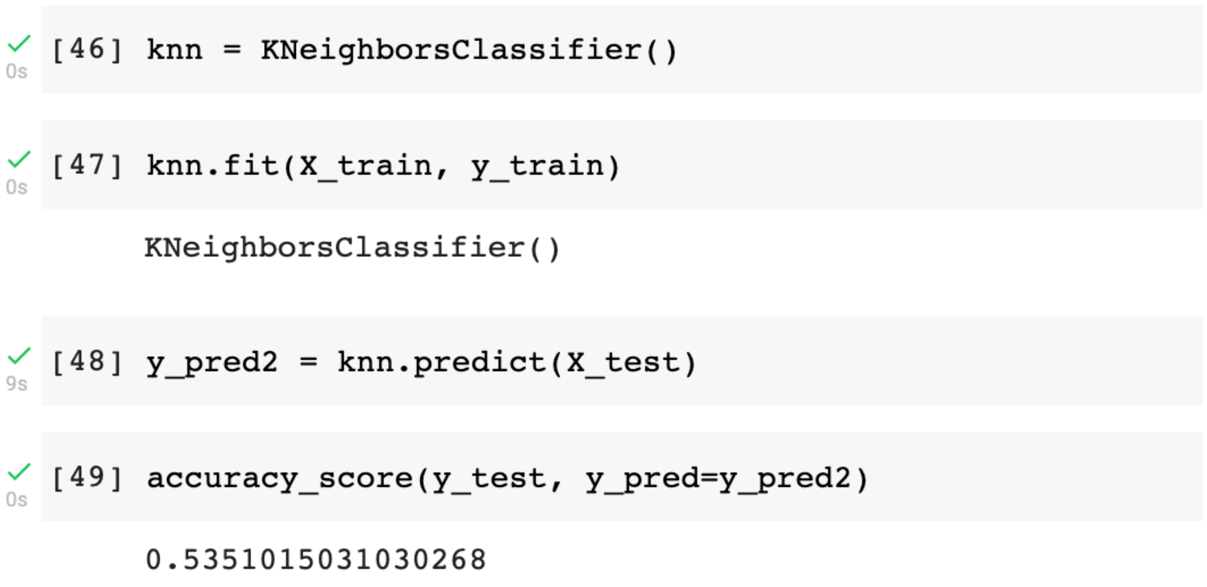
```
[49] from sklearn.metrics import roc_curve
     fpr, tpr, thresh = roc_curve(y_test, y_pred, pos_label=1)

[50] random_probs = [0 for i in range(len(y_test))]
     p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

[51] auc_score = roc_auc_score(y_test, y_pred)

[52] np.unique(y_pred)

     array([0, 1])

     auc_score

     0.5741143125188826
```

The KNN algorithm makes highly accurate predictions, Predicting an accuracy score for y to be 0.5351015031030268.



KNeigbour Classiffer

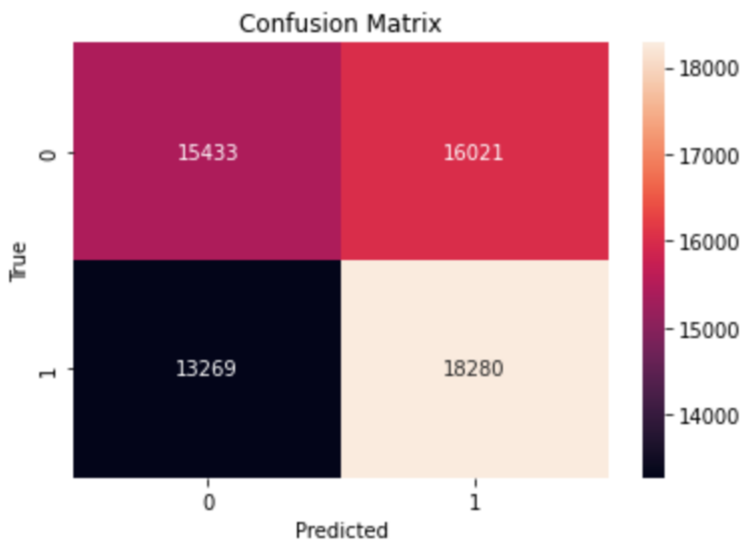
```
[44] from sklearn.neighbors import KNeighborsClassifier

[45] knn = KNeighborsClassifier(n_neighbors=10)

     n_splits = 5
     skf = StratifiedKFold(n_splits=n_splits)
     cv_results = cross_val_score(knn, X_train, y_train, cv=skf, scoring = 'roc_auc')
     print(f'Cross-Validation results: {cv_results, cv_results.mean()}')

     knn = KNeighborsClassifier(n_neighbors=10)

     knn.fit(X_train, y_train)
     print(f'ROC-AUC on test set: {roc_auc_score(y_test, knn.predict(X_test))}')

     Cross-Validation results: (array([0.57089497, 0.56435626, 0.56296505, 0.55140443, 0.5474206 ]), 0.5594082599286446)
     ROC-AUC on test set: 0.54253305718882
```



```
[46] knn = KNeighborsClassifier()

[47] knn.fit(X_train, y_train)

     KNeighborsClassifier()

[48] y_pred2 = knn.predict(X_test)

[49] accuracy_score(y_test, y_pred=y_pred2)

     0.5351015031030268
```

The matrix table defines the performance of a classification algorithm. Calculating the accuracy, precision, recall, F-1, and support score for binary classifiers. Confusion matrix and classification report for Knn.

```
              precision    recall  f1-score   support

           0       0.54      0.49      0.51     31454
           1       0.53      0.58      0.56     31549

    accuracy                           0.54     63003
   macro avg       0.54      0.54      0.53     63003
weighted avg       0.54      0.54      0.53     63003
```



accuracy_score(y_test, y_pred=y_pred5)

gradient boost classifer

```
[71] from sklearn.ensemble import GradientBoostingClassifier
```

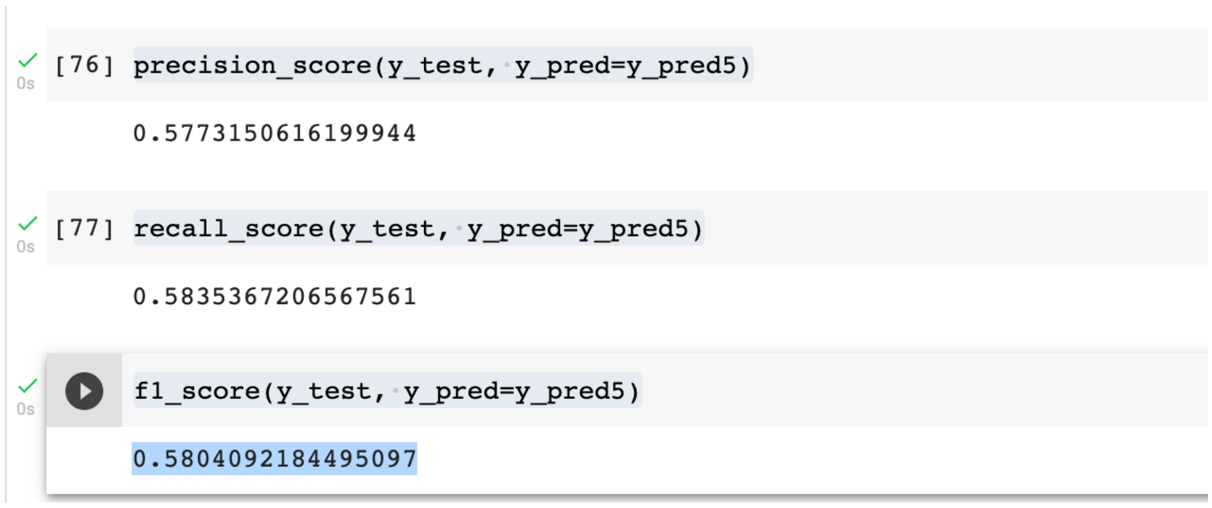
```
[72] gbc= GradientBoostingClassifier()
```

```
[73] gbc.fit(X_train, y_train)

     GradientBoostingClassifier()
```

```
[74] y_pred5 = gbc.predict(X_test)
```

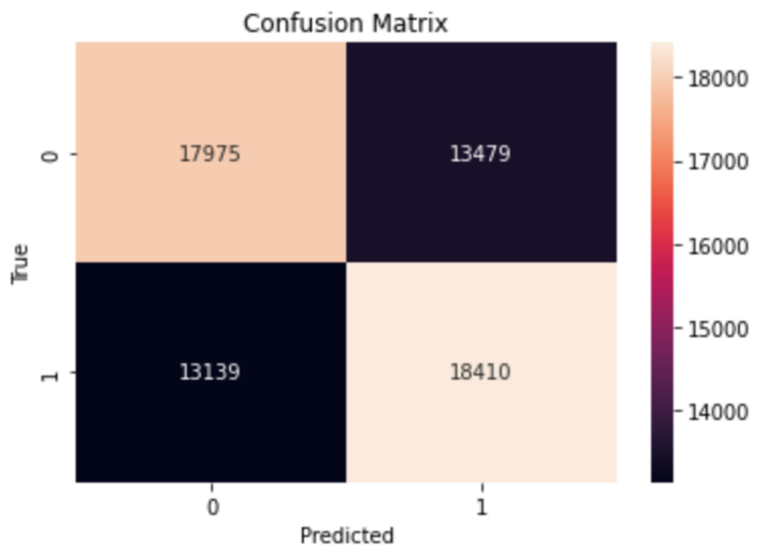
```
accuracy_score(y_test, y_pred=y_pred5)

0.5775121819595892
```

precision_score(y_test, y_pred=y_pred5) predicted 0.5773150616199944, recall_score(y_test, y_pred=y_pred5)predicted 0.5835367206567561 and f1_score(y_test, y_pred=y_pred5)predicted 0.5804092184495097



```
[76] precision_score(y_test, y_pred=y_pred5)

     0.5773150616199944

[77] recall_score(y_test, y_pred=y_pred5)

     0.5835367206567561

     f1_score(y_test, y_pred=y_pred5)

     0.5804092184495097
```

```
              precision    recall  f1-score   support

           0       0.58      0.57      0.57     31454
           1       0.58      0.58      0.58     31549

    accuracy                           0.58     63003
   macro avg       0.58      0.58      0.58     63003
weighted avg       0.58      0.58      0.58     63003
```


Confusion Matrix
True
Predicted