1. Classification

Dataset :- Our data consist of 13 columns, out of those many of them are categorical features and also there are non categorical features and 1lacs+ data entries in total in our train set and in test set there are same no of columns and about 64000 rows on which I have to predict my target variables.

Task :- Given data related to country first preference to visit , work with features to predict **probability** of **TARGET country which is being visited by any person** which I have label encoded them by using the label encoder library in scikit learn .

Model 1 :- Using Logistic Regression

Approach 1 :

Step 1 :- Data Preprocessing and Cleaning

1. First checked if data have any null values or missing values but found none.
2. Checked duplicate values but found none.

3. Then after checked the categorical and non categorical value then done the preprocessing on the basis of correlation which I calculated .

4. I have replaced the null values of age by the median age value which was one of the most important part of preprocessing .

5. Then I have done one hot encoding of those categorical columns ehich was most importantly correlated .

Step 2 :- Splitting dataset and Standarized the data

Split the dataset into 80% training set and 20% testing set, also extract Target values into Y. Then use sklearn “StandardScaler” method to standarize the data.

MODEL 1 :- **Applying Logistic Regression model and calculate accuracy score**

Used sklearn to apply the Logistic Regression model and get an accuracy score of 58% which suggests that there are less accuracy in our model means we have to preprocess more data columns which are ctaegoricland which I have dropped.

So here is muy code where I have used softmax regression with liblinear as a SOLVER:-

```
[ ]
    ###  here using ssoftmax regression classifier using solver as lbfgs

    from sklearn.linear_model import LogisticRegression



    softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=140)
    softmax_reg.fit(train_X,train_Y)
```

```
    /usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning:
      y = column_or_1d(y, warn=True)
    LogisticRegression(C=140, class_weight=None, dual=False, fit_intercept=True,
                       intercept_scaling=1, l1_ratio=None, max_iter=100,
                       multi_class='multinomial', n_jobs=None, penalty='l2',
                       random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                       warm_start=False)
```

```
[ ]  softmax_reg.predict(test_X)

    softmax_reg.score(test_X,test_Y)
```

```
    0.5827997769102063
```

MODEL 2 : Using PCA

All the steps are the same as above. The only different thing that we tried now is to apply **PCA** in our dataset. As our dataset is pretty large we tried to increase our accuracy but it somewhat improves my model accuracy bur not much.

```
[ ]  ###  NOW HERE I AM USING PCA CLASSIFIER TO TRAIN MY MODEL


     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     #Fit on training set only.


     scaler.fit(train_X)


     # Apply transform to both the training set and the test set.
     train_X = scaler.transform(train_X)
     test_X = scaler.transform(test_X)
```

```
[ ]  from sklearn.decomposition import PCA
     # Make an instance of the Model
     pca = PCA(.95)
```

```
[ ]  pca.fit(train_X)
```

```
[ ]  train_X = pca.transform(train_X)
     test_X = pca.transform(test_X)
```

Model 2 :- Using Decision Tree Classifier:-(just a curosity):-

All the preprocessing and standarization steps are similar to the above model. But this time I use Decision tree classifier using sklearn. Again I got a high accuracy score and this time I got more accuracy than all the algorithm which I have used previously .

```
from sklearn.tree import DecisionTreeClassifier
dtree_model = DecisionTreeClassifier(max_depth =9).fit(train_X, train_Y)   ###fitting opur train data
dtree_model.predict(test_X)

dtree_model.score(test_X,test_Y)   #### calculating the accuracy
```

```
0.6061349693251534
```

```
[ ]
```

```
[ ] Y_predicted_DT = dtree_model.predict(test_df)
```

```
[ ] Y_predicted_DT=Y_predicted_DT.astype('int64')
```

```
[ ] Y_predicted_DT = label_encoder.inverse_transform(Y_predicted_DT)  ### same doing inverse transform for replacing with the origional values
```

```
[ ] result = pd.DataFrame({'user_id':X_id,'target_country':Y_predicted_DT}) ### then making the result datframe
```

```
[ ] #result.to_csv('predictions_DT_again.csv',index=False,header=True)
```

Model 3 :- Using K_NEAREST_NEIGHBOURS:-

In this to improve accuracy, I did one extra step. I remove outliers as data is too big and there are outliers also so I remove them and clean the data a bit. After that when I apply KNN.

Below is the code for the model :

I took the another classifier that is K-Nearest-Neighbour and chosen 60 neighbours to calculate tyhe eucledian distance and got some enhancement in the score .

```
[ ]  ## lets take another classsifier


     from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors = 60).fit(train_X, train_Y)

     # accuracy on X_test


     accuracy = knn.score(test_X, test_Y)
     print (accuracy)
```

```
     /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DataConversionWarning: A column-vector y was passed when a 1d
       """
     0.5842275515895148
```

```
[ ]  Y_predicted_knn = knn.predict(test_df)
```

```
[ ]  Y_predicted_knn=Y_predicted_knn.astype('int64')
```

```
[ ]  Y_predicted_knn = label_encoder.inverse_transform(Y_predicted_knn)
```

In this I got accuracy 61 % which is improvement from our previous results.