Predicting Churn for Bank Customers

Project Report

Domain of Project	Customer Behavior
Proposed Title	Customer Churn
Proposed Date	Dec’2020-Feb’2021
Proposed by	**NIKHIL VAIBHAV**

BUSINESS OBJECTIVE :

I have chosen this project bearing the subsequent objectives in mind:

• Visualize which factors contribute to customer churn in the banking sector across the world.

• Build and deploy a prediction model that will –

➢ given the details of the customer predict whether he/she is going to churn or not.

➢ by proposing a probabilistic approach of model of the given customer we can ascertain which all customers bank's customer services to target for retention campaigns and launch such loyalty program or offers for incentivising those customer who are at borderline of churn rate to maximize its profit.

Literature Survey:

1. HBR Post Link: the-value-of-keeping-the-rightcustomers
2. Kaggle Bank Churn Link: kaggle dataset
3. Jill Avery is a senior lecturer at Harvard Business School & an author of HBR's: Go To Market Tools
4. Customer Churn Prediction – A case Study in retail published by Sas
5. Harvard Business School case on HubSpot's Development of a Customer Happiness Index

DATASET AND DOMAIN

Dataset Information:

It consists of 10000 observations and 12 variables. Independent variables contain information about customers. Dependent variable refers to customer abandonment status.

- **RowNumber** — Corresponds to the record (row) number and has no effect on the output. This column will be removed.
- **CustomerId** — Contains random values and has no effect on customer leaving the bank. This column will be removed.
- **Surname** — The surname of a customer has no impact on their decision to leave the bank. This column will be removed.
- **CreditScore** — Can have an effect on customer churn, since a customer with a higher credit score is less likely to leave the bank.
- **Geography** — A customer's location can affect their decision to leave the bank. We'll keep this column.
- **Gender** — It's interesting to explore whether gender plays a role in a customer leaving the bank. We'll include this column, too.
- **Age** — This is certainly relevant, since older customers are less likely to leave their bank than younger ones.
- **Tenure** — Refers to the number of years that the customer has been a client of the bank. Normally, older clients are more loyal and less likely to leave a bank.
- **Balance** —As people with a higher balance in their accounts are less likely to leave the bank compared to those with lower balances.
- **NumOfProducts** — Refers to the number of products that a customer has purchased through the bank.
- **HasCrCard** — Denotes whether or not a customer has a credit card. This column is also relevant, since people with a credit card are less likely to leave the bank. (0=No,1=Yes)

- **IsActiveMember** — Active customers are less likely to leave the bank, so we'll keep this. (0=No,1=Yes)
- **EstimatedSalary** — As with balance, people with lower salaries are more likely to leave the bank compared to those with higher salaries.
- **Exited** — Whether or not the customer left the bank. This is what we have to predict. (0=No,1=Yes)

Variable categorization:

Categorical variables	Numerical variables
Geography	CreditScore
Gender	Age
HasCrCard	Tenure
IsActiveMember	Balance
Exited	NumOfProducts
	EstimatedSalary

Pre-Processing Data Analysis:

There are no missing values in the dataset. We have removed 3 columns from our dataset ('RowNumber','CustomerId','Surname') since they have no significance.

Project Justification:

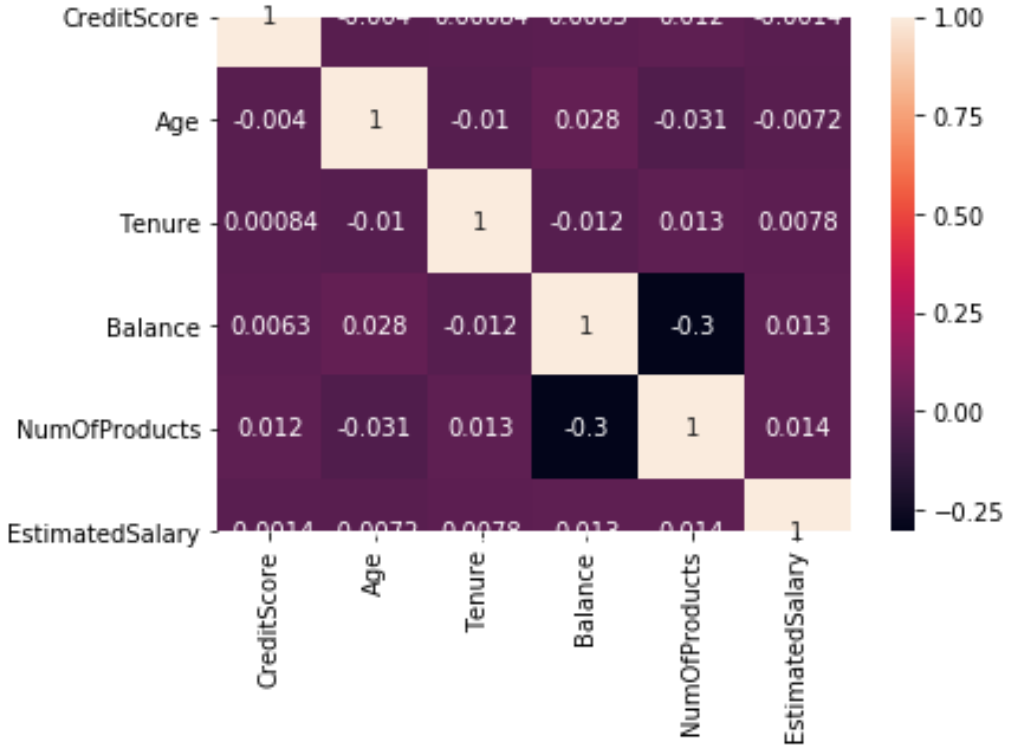
As aforementioned, decreasing churn rates is a crucial agenda for all firms mainly because of two reasons

- Acquiring new customers is anywhere between 5 to 25 times more expensive.
- Increasing retention rates by 5% results in increase in profits anywhere between 25% to 95%.
    - While most of the Projects on Kaggle on customer churn is dependent on predictive modeling for retention strategies, we propose a predictive model

reinforced with probabilities so that costumer services could focus on customers whose probability to churn is greater than 0.5 but in the vicinity. This might lead to better results as the customer services efforts and resources will be concentrated towards retaining customers who are plausible to get convinced.

Data Exploration (EDA):

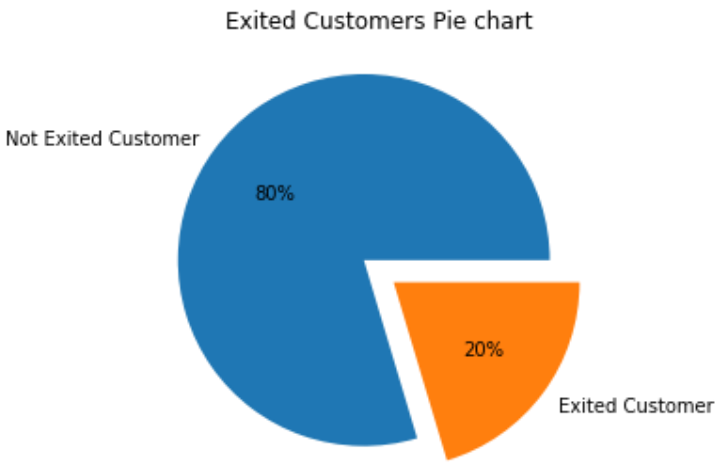
Check for Multicollinearity:

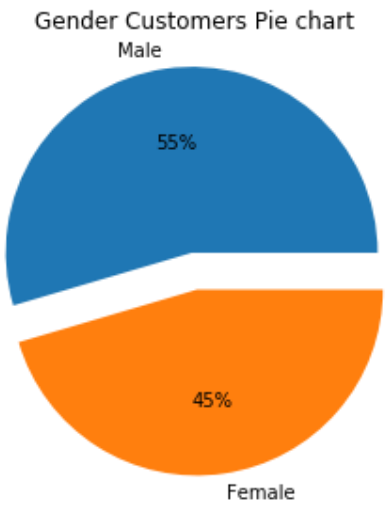


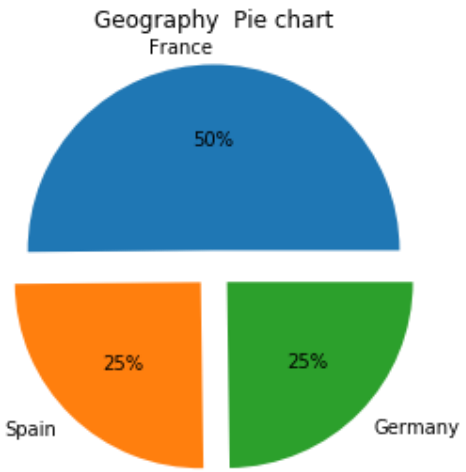
There is no strong correlation among variables so we can say that there is no multicollinearity.

	VIF
const	80.195448
CreditScore	1.002306
Age	1.013138
Tenure	1.001632
Balance	1.193985
NumOfProducts	1.194761
HasCrCard	1.002196
IsActiveMember	1.013914
EstimatedSalary	1.002233
Gender_Male	1.002219

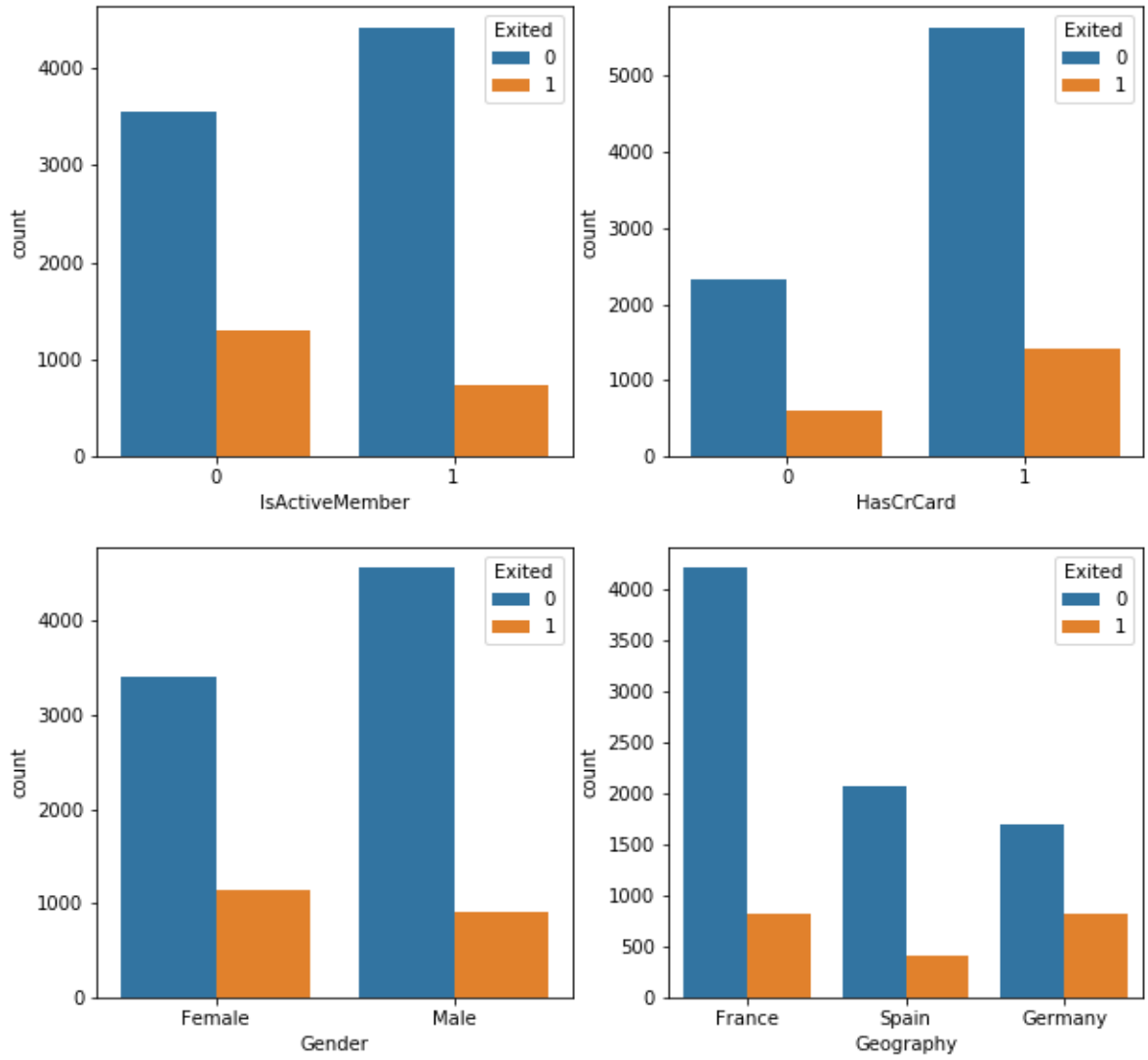
Distribution of Variables:





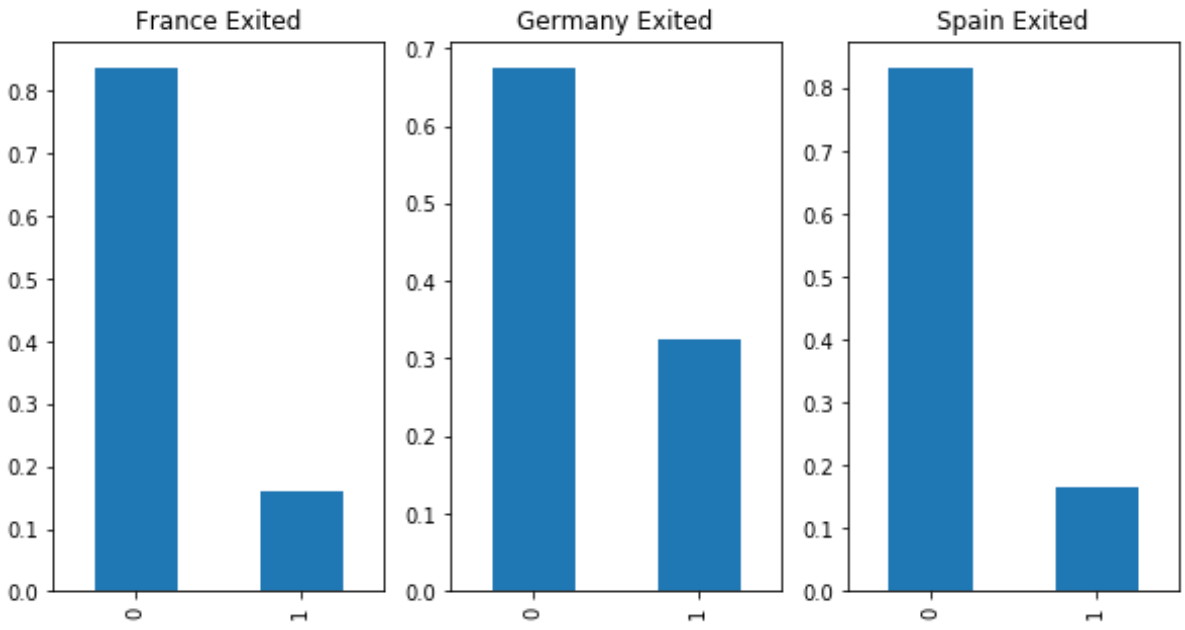


- The customer that has exited is about 20% of the overall population.
- The male and female customer proportion in bank is around 55% and 45% respectively.
- The France region contribute to around 50% of the customers to the bank and Spain and Germany contribute to around equally of the half proportion i.e. 25-25%



- Majority of the data is from persons from France. However, the proportion of churned customers is greater in Spain although having small proportion of the customers. The proportion of female customers churning is also greater than that of male customers by looking at the proportions individually.

- Majority of the customers that are churned have credit cards.
- Unsurprisingly the inactive members have a greater churn. Worryingly is that the overall proportion of inactive members is quite high suggesting that the bank may need a program implemented to turn this group to active customers as this will definitely have a positive impact on the customer churn.



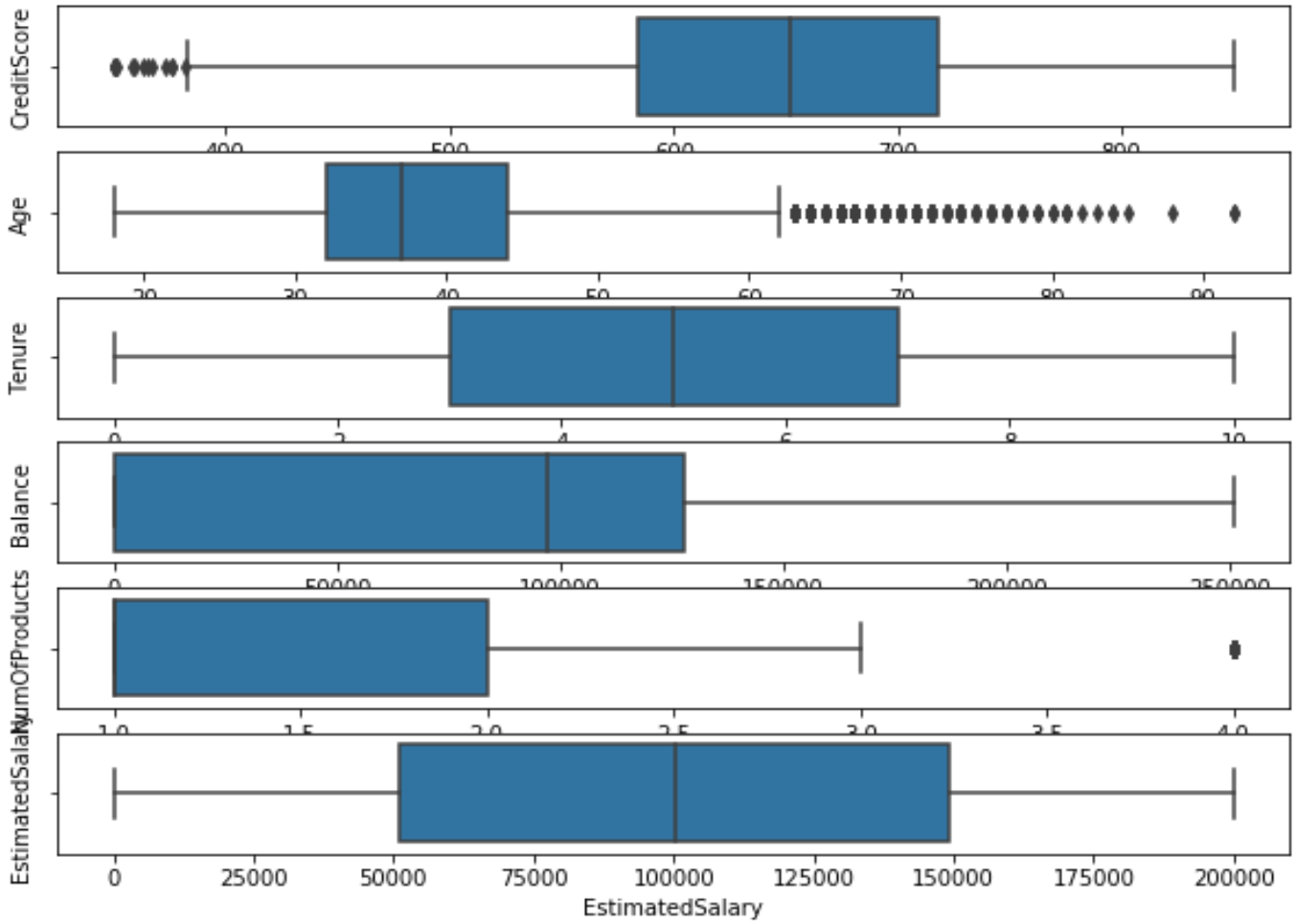

```
France Exited Proportion
 0    0.838452
1    0.161548
Name: Exited, dtype: float64

 Germany Exited Proportion
 0    0.675568
1    0.324432
Name: Exited, dtype: float64

 Spain Exited Proportion
 0    0.833266
1    0.166734
Name: Exited, dtype: float64
```

- We can infer by looking at the value count percentage that customers exiting from Germany branch has higher percentage than that of France and Spain.

Presence of Outliers:



- We can see presence of outliers in features like "CreditScore" and "Age". We will be treating them by using Power Transformer though customized transformer.

Statistical Significance of Variables:

```
df_churn=df[df['Exited']==1]
df_not_churn=df[df['Exited']==0]
```

```
for i in num_df.columns:
    _,pval=ttest_ind(df_churn[i],df_not_churn[i])
    print(i,pval)
```

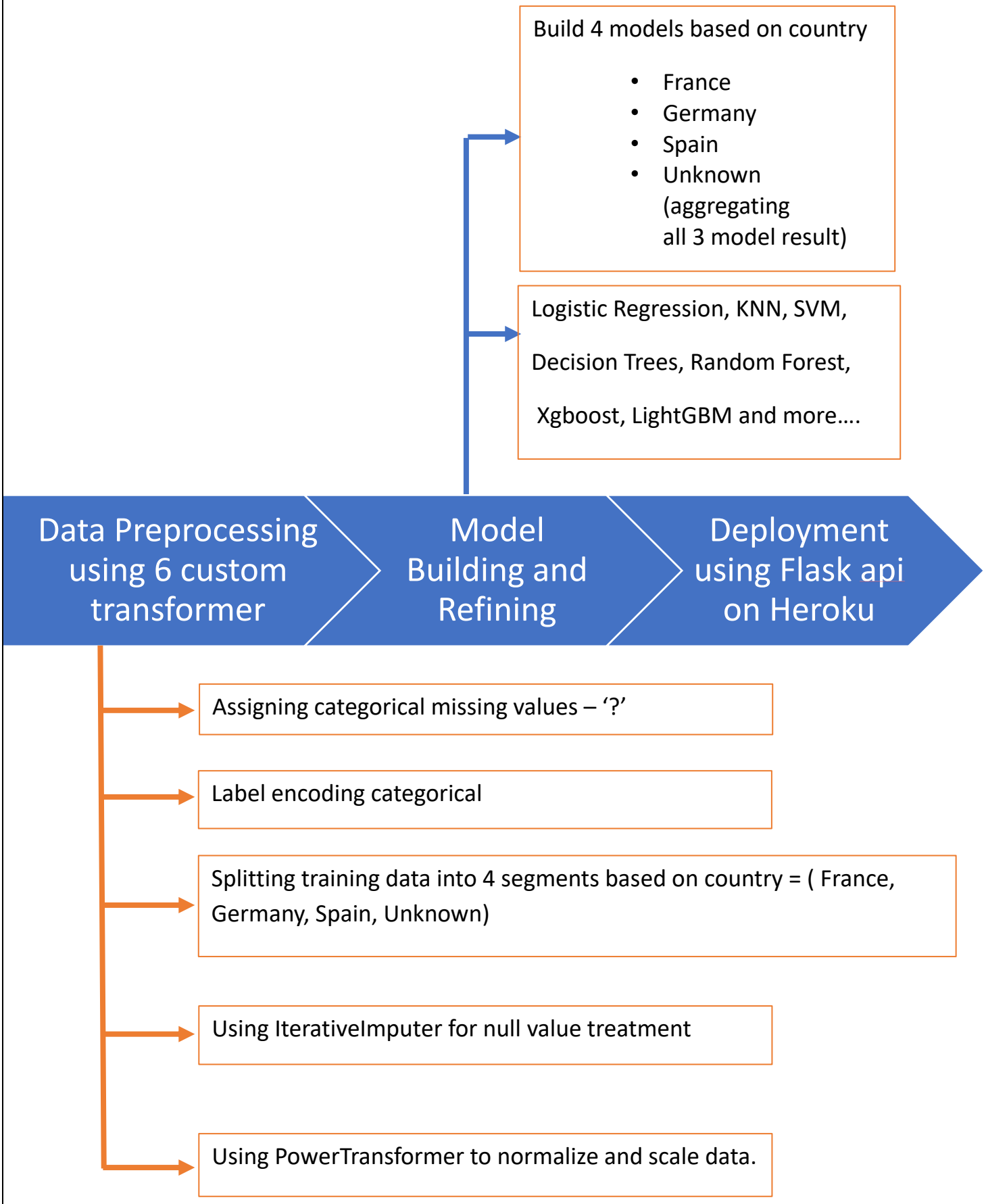
```
CreditScore 0.006738213892192373
Age 1.2399313093427738e-186
Tenure 0.16152684949473256
Balance 1.2755633191525477e-32
NumOfProducts 1.717333004804293e-06
EstimatedSalary 0.22644042802223346
```

Null Hypothesis (Ho): Variables are not significant.

Alternate Hypothesis (Ha): Variables are significant.

- Since majority of columns are having p-value <0.05 we can reject h0 and we can say that the variables are significant and we can go on and build the model on these variables.

Overview of the Final Process:



Step-by-Step walk through of the Solution :

As depicted in flowchart the previous page, the process after acquiring valuable insights from the Explorative Data Analysis, the final solution is divided into predominantly 3 parts.

1) Data Preprocessing using 6 Custom Transformers

data_prep_1 :

```
class data_prep_1(BaseEstimator, TransformerMixin):
    def __init__(self,fill_value='?'):
        self.fill_value = fill_value
    def fit(self,df):
        return self
    def transform(self,df):
        df_gender_null_counter = df['Gender'].isnull().sum()
        if df_gender_null_counter>0:
            df_isgender_null = True
            df['Gender'].fillna(value = self.fill_value)
        return (df,df_gender_null_counter)
```

The objective of building this Custom Transformer is to impute any null values present in categorical data that is not encoded (i.e., in 'object' type; in our case only the Gender column) to '?'. The reason for this being, we are using Iterative Imputer in Custom Transformer 4. The input to an Iterative Imputer needs to be encoded data. Hence, to convert our categorical data into encoded format, we must get rid of null values. We are taking '?' as value to impute null values because:

Ascii Value ('?') < Ascii Value ('Female') < Ascii Value ('Male')

Hence, if there are null value present, the encoding will take place as follows:

'?'	0
'Female'	1
'Male'	2

data_prep_2:

```
class data_prep_2(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
    def fit(self,df,df_gender_null_counter):
        return self
    def transform(self,df,df_gender_null_counter=0):
        le = LabelEncoder()
        df.loc[:,'Gender'] = le.fit_transform(df['Gender'])
        if df_gender_null_counter>0:
            df.loc[:,'Gender'] = df['Gender'].replace({0:np.nan})
        return df
```

Now, that we have the Gender as Encoded Data, it is time to replace 0s with np.nan.

Note: We cannot always change 0s to np.nan everytime! We will do this only when there were null values present. This the purpose of keeping a counter df_gender_null_counter.

data_prep_3:

```
class data_prep_3(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def fit(self,X=None):
        return self

    def transform(self,train,X=None):

        trainF = train[train['Geography']=='France']


        trainG = train[train['Geography']=='Germany']


        trainS = train[train['Geography']=='Spain']

        trainNK = train.copy()

        trainF.drop(columns=['Geography'], inplace=True)
        trainG.drop(columns=['Geography'], inplace=True)
        trainS.drop(columns=['Geography'], inplace=True)
        trainNK.drop(columns=['Geography'], inplace=True)
        return (trainF,trainG,trainS,trainNK)
```

This Custom Transformer is built solely with the purpose of segregating the training data into 4 DataFrames : trainF (data with country==’France’), trainG (data with country==’Germany’), trainS (data with country==’Spain), and trainNK with the entire dataset.

While all other Custom Transformers return the only 1 DataFrame, this Custom Transformer returns the 4 DataFrames.

data_prep_4:

```
class data_prep_4(BaseEstimator, TransformerMixin):

    def __init__(self,num_bool):
        self.num_bool=num_bool


    def fit(self,X,y=None):
        if self.num_bool==True:
            self.est = RandomForestRegressor()
            self.itimp = IterativeImputer(self.est)
            self.cols = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
                         'EstimatedSalary']

        else:
            self.est = RandomForestClassifier()
            self.itimp = IterativeImputer(self.est)
            self.cols =  ['Gender','HasCrCard','IsActiveMember']

        self.itimp.fit(X[self.cols])


        return self

    def transform(self,X,y=None):
        if X.shape[0] > 0:
            X.loc[:, self.cols] = self.itimp.transform(X[self.cols])

            return X
        else:
            return X
```

The objective of the 4th Custom Transformer is to impute the null values in the dataset.

HyperParameter : num_bool (True→passed dataset is Numeric in nature

False→ passed dataset is Categorical in Nature)

If num_bool is True:

Iterative Imputer employs the RandomForestRegressor

Elif num_bool is False:

Iterative Imputer employs the RandomForestClassifier

Note: There are total 8 instances of this Custom Transformer called in the 6th Custom Transformer (Combined_Data_Prep) which we will see in the next page.

1 Numerical and 1 Categorical instance for each country == [‘France’,’Germany’,’Spain’,’NK’]

data_prep_5:

```
class data_prep_5(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def fit(self,df):
        self.cols = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
                     'EstimatedSalary']
        self.pt = PowerTransformer()
        self.pt.fit(df[self.cols])
        return self

    def transform(self,df):
        if df.shape[0] > 0:
            df.loc[:,self.cols] = self.pt.transform(df[self.cols])
            return df
```

Finally, the 5th Custom Transformer will employ Power Transformer to treat the

Outliers and Scale the data using the method = ‘yeo-johnson’.

Combined_Data_Prep:

```
class Combined_Data_Prep(BaseEstimator, TransformerMixin):

    def __init__(self):

        self.test_bool = False
        self.country = str

        self.d1 = data_prep_1()
        self.d2 = data_prep_2()
        self.d3 = data_prep_3()


        self.d41F = data_prep_4(num_bool=True)
        self.d42F = data_prep_4(num_bool=False)

        self.d41G = data_prep_4(num_bool=True)
        self.d42G = data_prep_4(num_bool=False)

        self.d41S = data_prep_4(num_bool=True)
        self.d42S = data_prep_4(num_bool=False)

        self.d41NK = data_prep_4(num_bool=True)
        self.d42NK = data_prep_4(num_bool=False)

        self.d5F = data_prep_5()
        self.d5G = data_prep_5()
        self.d5S = data_prep_5()
        self.d5NK = data_prep_5()

    def fit(self,X,y=None):
        return self
```

```
def transform(self,X,y=None):

    if self.test_bool==False:

        train,b = self.d1.fit_transform(X)

        train = self.d2.fit_transform(train,df_gender_null_counter=b)

        trainF,trainG,trainS,trainNK= self.d3.transform(train)

        trainF = self.d41F.fit_transform(trainF)
        trainF = self.d42F.fit_transform(trainF)

        trainG = self.d41G.fit_transform(trainG)
        trainG = self.d42G.fit_transform(trainG)

        trainS = self.d41S.fit_transform(trainS)
        trainS = self.d42S.fit_transform(trainS)

        trainNK = self.d41NK.fit_transform(trainNK)
        trainNK = self.d42NK.fit_transform(trainNK)

        trainF = self.d5F.fit_transform(trainF)
        trainG = self.d5G.fit_transform(trainG)
        trainS = self.d5S.fit_transform(trainS)
        trainNK = self.d5NK.fit_transform(trainNK)

        return (trainF,trainG,trainS,trainNK)

    else:

        dptest,c = self.d1.fit_transform(X)

        dptest = self.d2.fit_transform(dptest,df_gender_null_counter=c)

        dptest.drop(columns=['Geography'],inplace=True)
        if self.country=='France':
            dptest = self.d41F.transform(dptest)
            dptest = self.d42F.transform(dptest)
            dptest = self.d5F.transform(dptest)
            return dptest

        elif self.country=='Germany':
            dptest = self.d41G.transform(dptest)
            dptest = self.d42G.transform(dptest)
            dptest = self.d5G.transform(dptest)
            return dptest

        elif self.country=='Spain':
            dptest = self.d41S.transform(dptest)
            dptest = self.d42S.transform(dptest)
            dptest = self.d5S.transform(dptest)
            return dptest

        elif self.country=='NK':
            dptest = self.d41NK.transform(dptest)
            dptest = self.d42NK.transform(dptest)
            dptest = self.d5NK.transform(dptest)
            return dptest
```

This is the 6th Custom Transformer that integrates all the 5 Custom Transforms previously built.

Initialization:

- During the initialization of an instance, two attributes (self.test_bool & self.country get initialized as False and an empty string respectively

- Next, we create an instance of each Custom Transformer data_prep_1, data_prep_2, data_prep_3

- As discussed in the note after explaining data_prep_4, we will create 2 instances of data_prep_4 for each country (3 countries + 1 Unknown = 4), hence we have in total 8 instances of data_prep_4.

  We will similarly create 4 instances of data_prep_5 ; one for each country.

Fit:

The method will do nothing but return self.

Transform:

Now, the attribute self.test_bool comes into the picture.

If self.test_bool == False : # (This is training data)

Here every instance calls fit_transform as this is the training data and they have to learn the necessary attributes of the data given.

Note: only if the data passed is training data, d3 (instance of data_prep_3) will be called

If self.text_bool == True: # (This is testing data)

Here every instance calls transform method to impute the testing data keeping in mind all that has been learnt during the training phase.

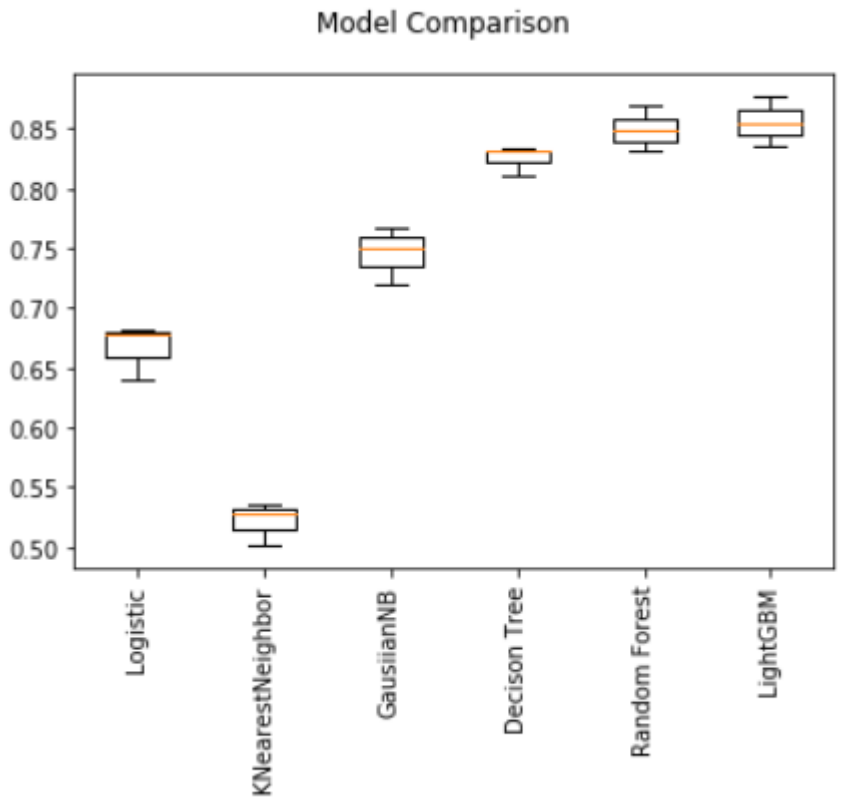
Note: Here d3 is not called as finally in the Flask api, only 1 record is tested and this will correspond to only 1 country; hence no point in using d3.

2) Model Building:

Now that we have the training data processed and have 4 dataframes –

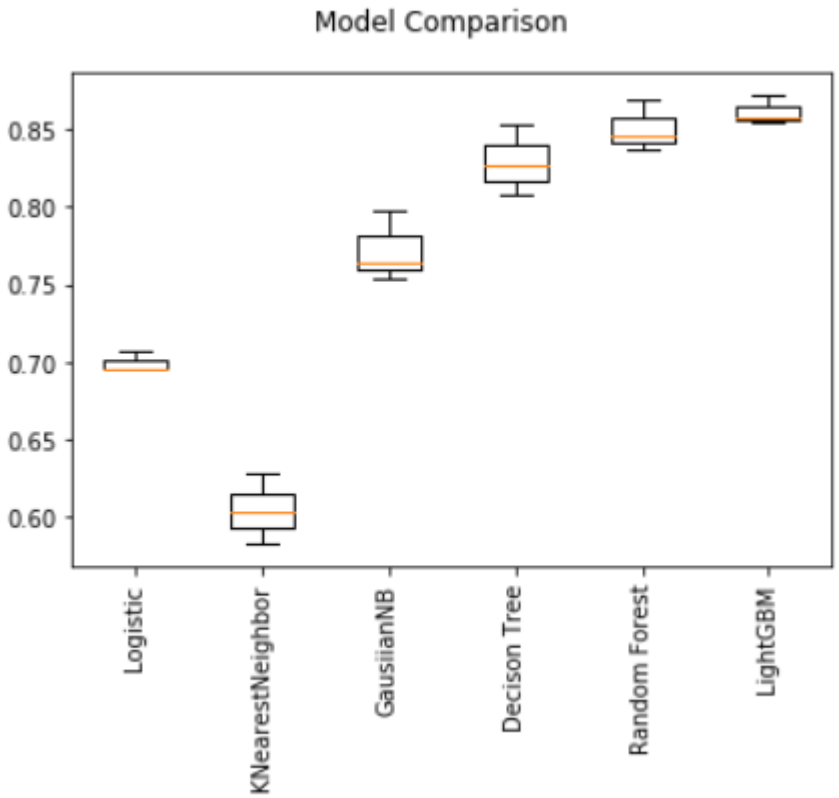
trainF, trainG, trainS, trainNK , we built several models on all these DataFrames and compared the results. For all the countries, we found LightGBM (LGBClassifier) to be working the best. The following are snapshots of the model performance :

- **France**



- As **Logistic Regression** is a basic model, and has no hyperparameters to tune so it's the static score which once got can't be improved further that is why we are getting (**roc_auc score** =**0.66**) from this model which is not best.
- The **KNN** algorithm uses 'feature similarity' to predict the class of any new data points. This means that the new point is assigned the class based on how closely it resembles the points that falls inside the no of neighbors we fixed as 'k' through hyperparameter tuning. We got the (**roc_auc score** =**0.52**) which is lowest in comparison to all other models.
- **Gaussian Naive Bayes** supports continuous valued features and models each as conforming to a Gaussian (normal) distribution. In this the prior probability of given class label is calculated, after which likelihood probability with each attribute for each class is found. Then both are put into Bayes formula in order to calculate posterior probability to get the predicted class. In this (**roc_auc score** =**0.74**) which is greater than Logistic and Knn but not all other models.
- **Decision tree** builds classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed in order to attain the maximum purity at each node for individual class label. We got (**roc_auc score** =**0.82**) which is better in comparison to all above models.
- **Random forest** like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest predict the class label and the class with the most votes becomes our model's prediction. In this we got (**roc_auc score** =**0.84**) which is better in comparison to all above models.
- **LightGBM** is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient as compared to other boosting algorithms. It is based on decision tree algorithms, it splits the tree leaf wise. Other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. In this we got (**roc_auc score** =**0.85**) which is better in comparison to all above models.

- **Germany**

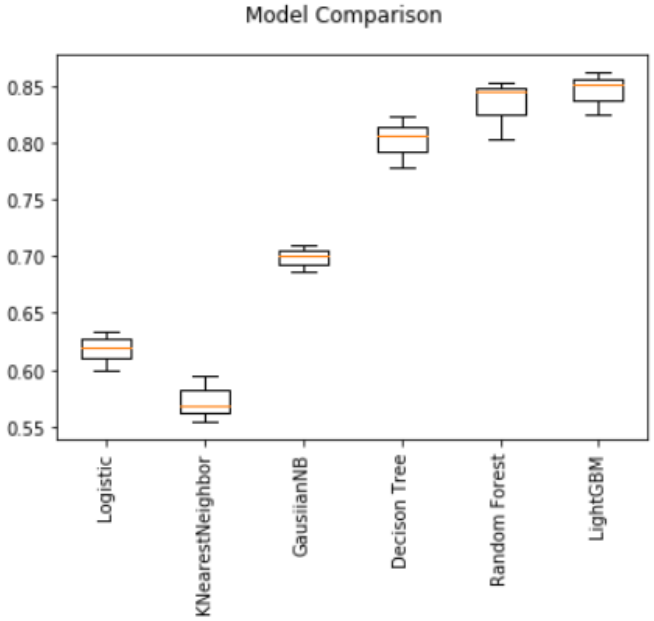


- As **Logistic Regression** is a basic model, and has no hyperparameters to tune so it's the static score which once got can't be improved further that is why we are getting (**roc_auc score** =**0.69**) from this model which is not best.
- The **KNN** algorithm uses 'feature similarity' to predict the class of any new data points. This means that the new point is assigned the class based on how closely it resembles the points that falls inside the no of neighbors we fixed as 'k' through hyperparameter tuning. We got the (**roc_auc score** =**0.60**) which is lowest in comparison to all other models.
- **Gaussian Naive Bayes** supports continuous valued features and models each as conforming to a Gaussian (normal) distribution. In this the prior probability of given class label is calculated, after which likelihood probability with each attribute for each class is found. Then both are put into Bayes formula in order to calculate posterior probability to get the predicted class. In this (**roc_auc score** =**0.77**) which is greater than Logistic and Knn but not all other models.
- **Decision tree** builds classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed in order to attain the maximum purity at each node for individual class label. We got (**roc_auc score** =**0.82**) which is better in comparison to all above models.
- **Random forest** like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest predict the class label and the class with the most votes

becomes our model's prediction. In this we got (**roc_auc score** =**0.85**) which is better in comparison to all above models.

- **LightGBM** is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient as compared to other boosting algorithms. It is based on decision tree algorithms, it splits the tree leaf wise. Other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. In this we got (**roc_auc score** =**0.86**) which is better in comparison to all above models.

- **Spain**



- As **Logistic Regression** is a basic model, and has no hyperparameters to tune so it's the static score which once got can't be improved further that is why we are getting (**roc_auc score** =**0.61**) from this model which is not best.
- The **KNN** algorithm uses 'feature similarity' to predict the class of any new data points. This means that the new point is assigned the class based on how closely it resembles the points that falls inside the no of neighbors we fixed as 'k' through hyperparameter tuning. We got the (**roc_auc score** =**0.57**) which is lowest in comparison to all other models.
- **Gaussian Naive Bayes** supports continuous valued features and models each as conforming to a Gaussian (normal) distribution. In this the prior probability of given class label is calculated, after which likelihood probability with each

attribute for each class is found. Then both are put into Bayes formula in order to calculate posterior probability to get the predicted class. In this (**roc_auc score** =**0.69**) which is greater than Logistic and Knn but not all other models.

- **Decision tree** builds classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed in order to attain the maximum purity at each node for individual class label. We got (**roc_auc score** =**0.80**) which is better in comparison to all above models.
- **Random forest** like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest predict the class label and the class with the most votes becomes our model's prediction. In this we got (**roc_auc score** =**0.83**) which is better in comparison to all above models.
- **LightGBM** is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient as compared to other boosting algorithms. It is based on decision tree algorithms, it splits the tree leaf wise. Other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. In this we got (**roc_auc score** =**0.84**) which is better in comparison to all above models.

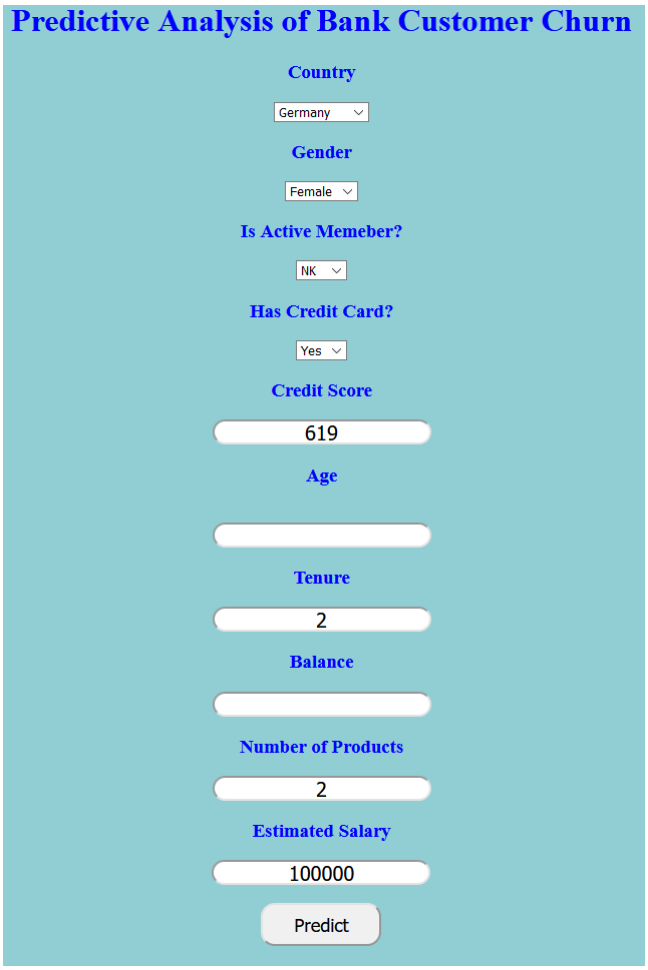
3) Building Flask api

Flask is a web framework for python. It provides functionality for building web applications. Out of the numerous functionalities that Flask has to offer, we have used predominantly 3 features:

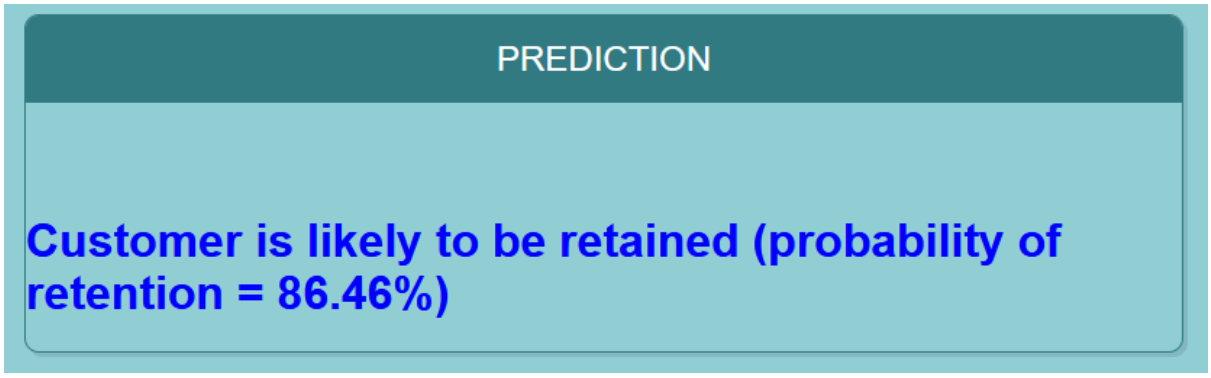
- Flask forms: To create a survey form to get the user input for different features.
- HTTP requests: Used to take the inputs given and store them in a variable and converted to a DataFrame
- Rendering Templates: Used two templates index.html which hosts the the main form. As soon as the user hits predict, the results.html file gets rendered.

Below are a 2 snaphots of the Flask api running on local machine:

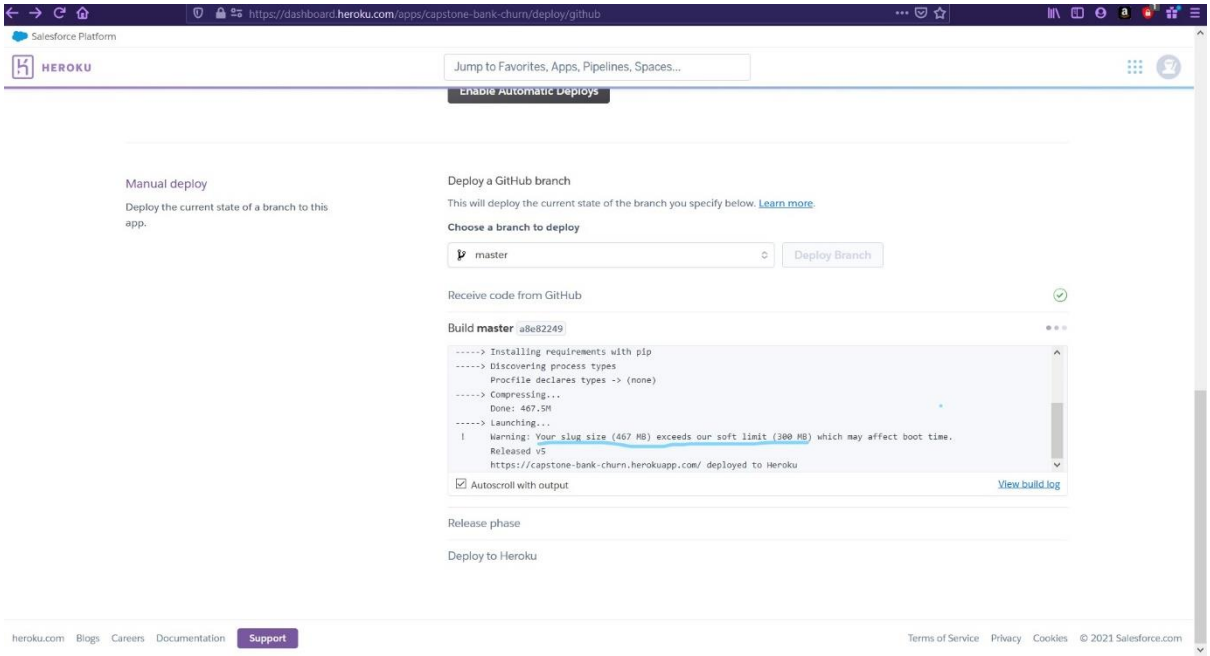
Index.html:



result.html



We also tried to deploy the project on Heroku. However, the flask app size is coming out to be 476 MBs, exceeding the Herko slug size limit of 300 MBs.



BUSINESS PROBLEM SOLUTION

Amy Gallo, an executive coach and author of Harvard Business Review, claims in a post1 she wrote on HBR that acquiring a new customer is anywhere between 5 to 25 times more expensive than retaining an existing one! To reinforce her claim, she refers to a research done by Frederick Reichheld of Bain & Company that shows increasing customer retention rates by 5% increases profits by 25% to 95%.

Bearing this in mind, our group had embarked on this project to simulate a real-world business case. We propose a Predictive Model that is reinforced with Probabilistic Model. Targeting those customers that are in the vicinity of the threshold probability of churn would help tremendously as they are the easiest to pull back.
Customer services can concentrate on the factors that we propose that influence the churn rate the most to organize retention campaigns.

LIMITATIONS

The one problem with our model is that we have assumed the threshold probability as 0.5

If we had the required domain knowledge, we could have used that to calculate the cost function keeping in mind the False Positive (loyal customers who have been predicted as customer who will churn) and False Negative (customers who are going to churn but are predicted as loyal).