MACHINE LEARNING DEPLOYMENT USING IBM WATSON STUDIO

TEAM LEADER:

1.NAME:NIRUPAMA K

REG NO:211521243110

TEAM MEMBER:

2.NAME:NEHA SHRUTHI.U

REG NO:211521243109

3. NAME:ASWATHI SWARNA SREE

REG NO:211521243025

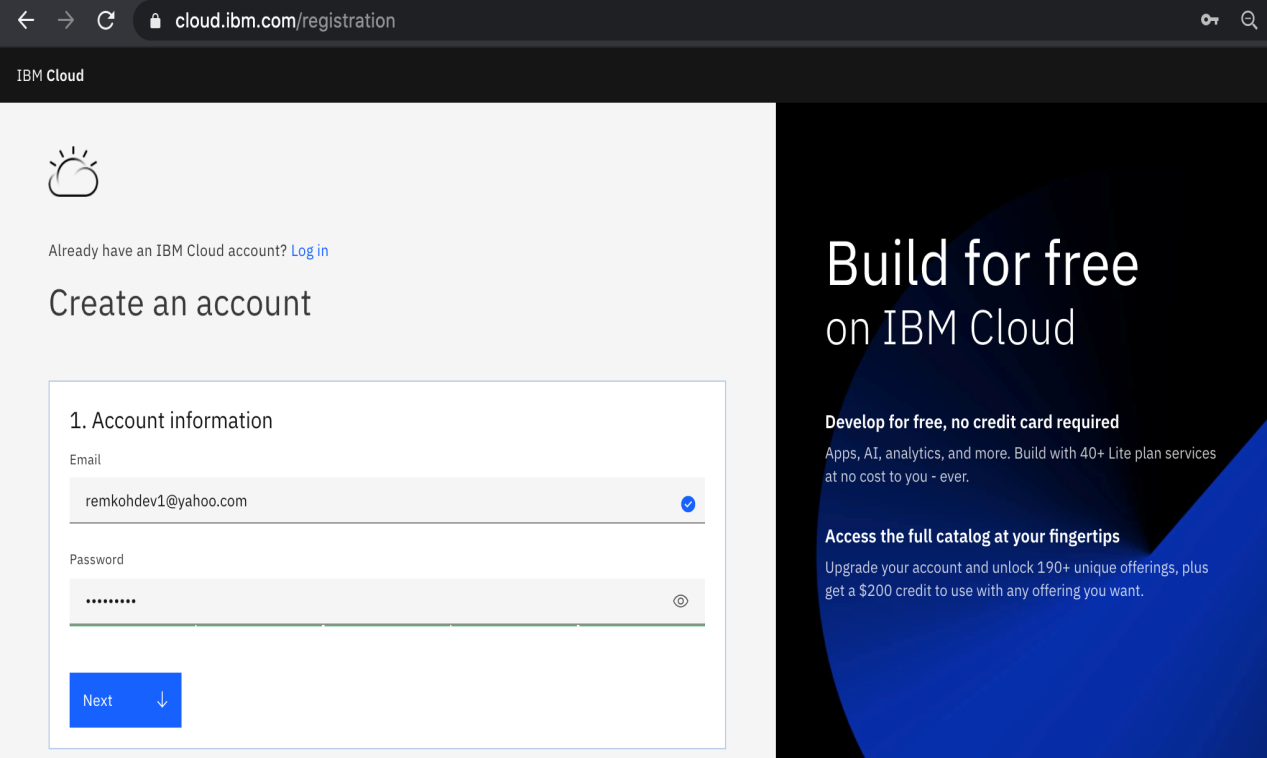
4. NAME:PRIYADHARSHINI N

REG NO: 211521243121

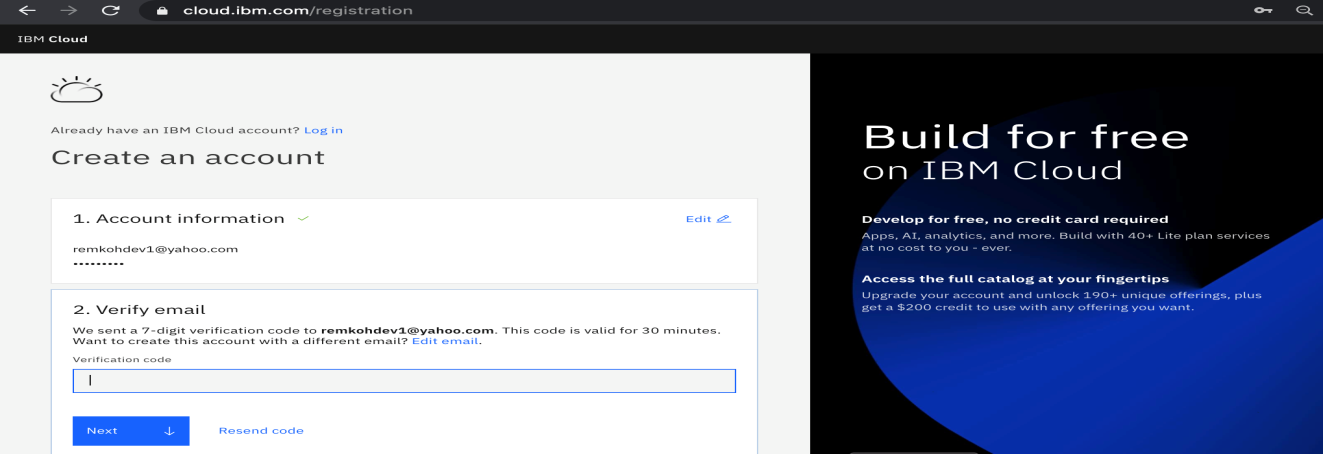
MACHINE LEARNING MODEL DEPLOYMENT USING IBM WATSON STUDIO

To create a new account, follow the steps below,

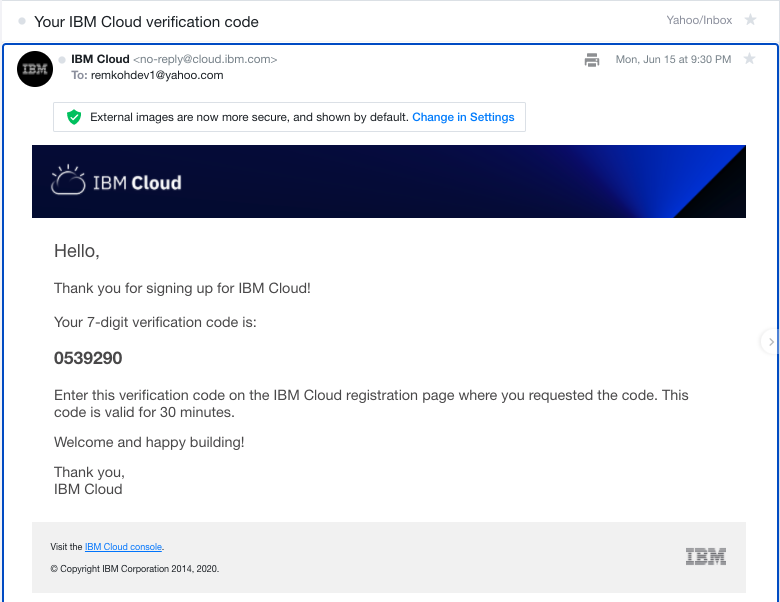
1. You need an IBM Cloud account to access your cluster,
2. If you do not have an IBM Cloud account yet, go to <https://cloud.ibm.com/registration> to register,
3. In the Create an account window, enter your email and password,



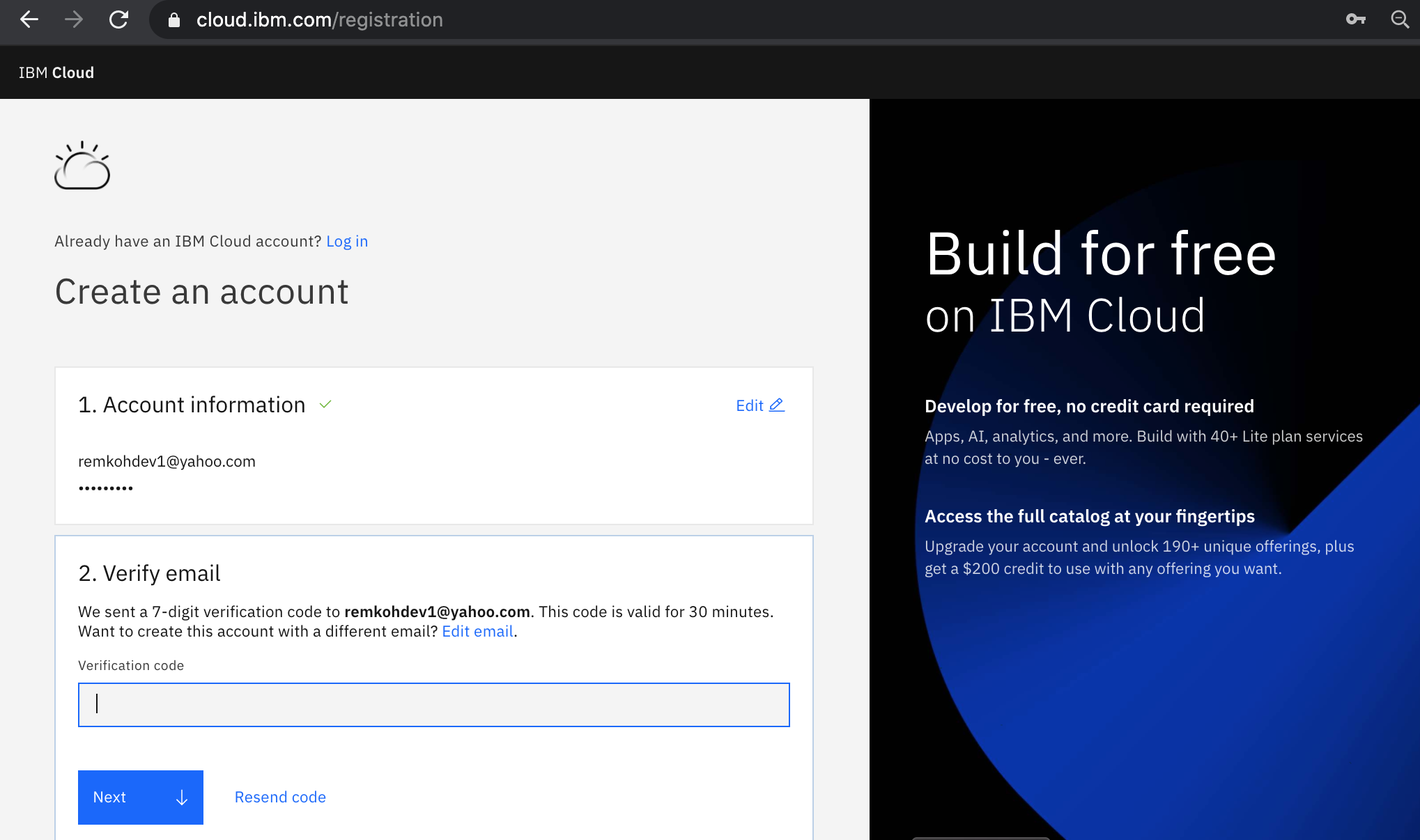
1. The Verify email section will inform you that a verification code was sent to your email,



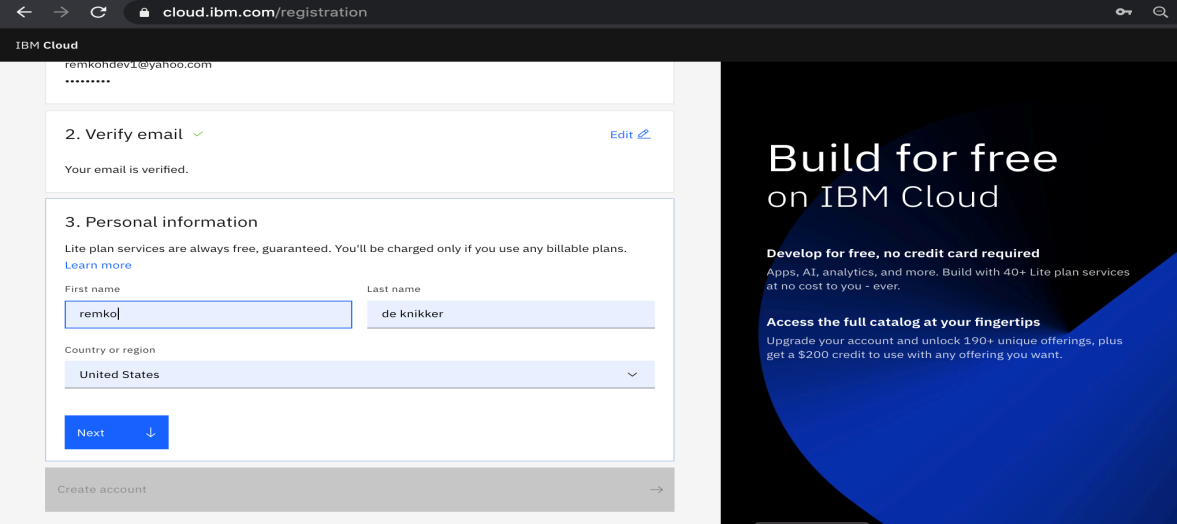
1. Switch to your email provider to retrieve the verification code,



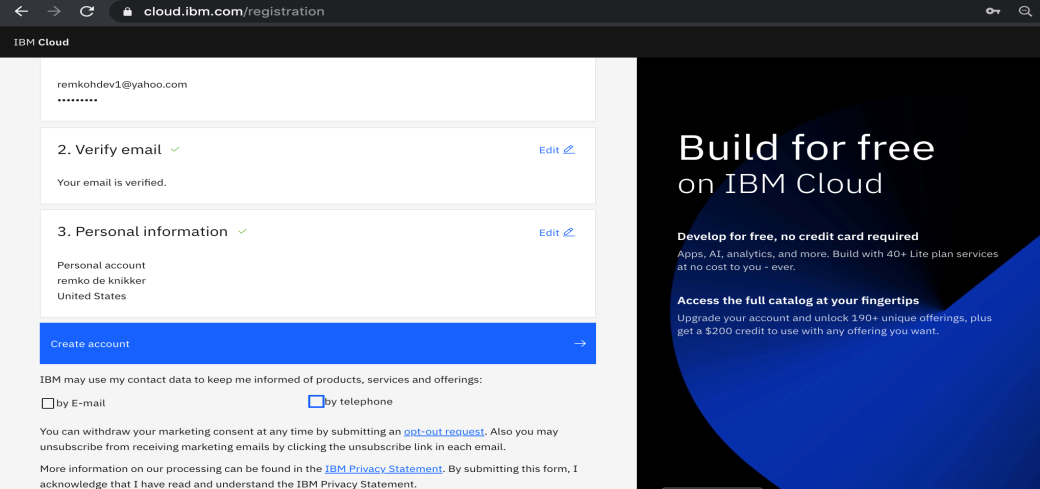
1. Enter the verification code in the Verify email section, and click Next,



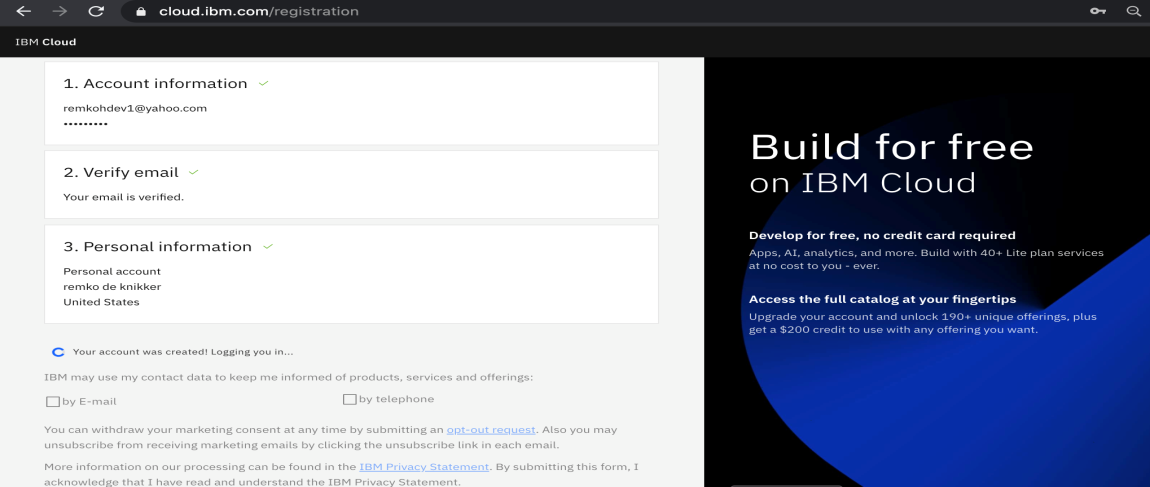
1. Enter your first name, last name and country in the Personal information section and click Next,



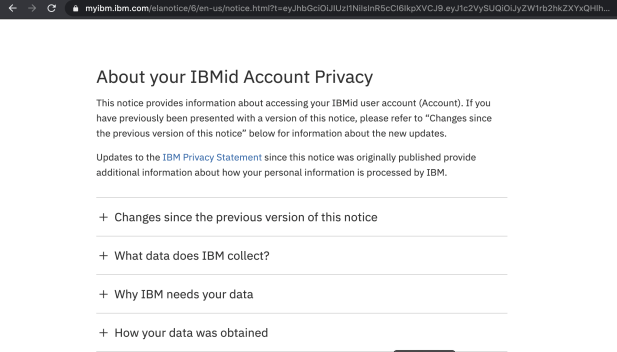
1. Click Create account,



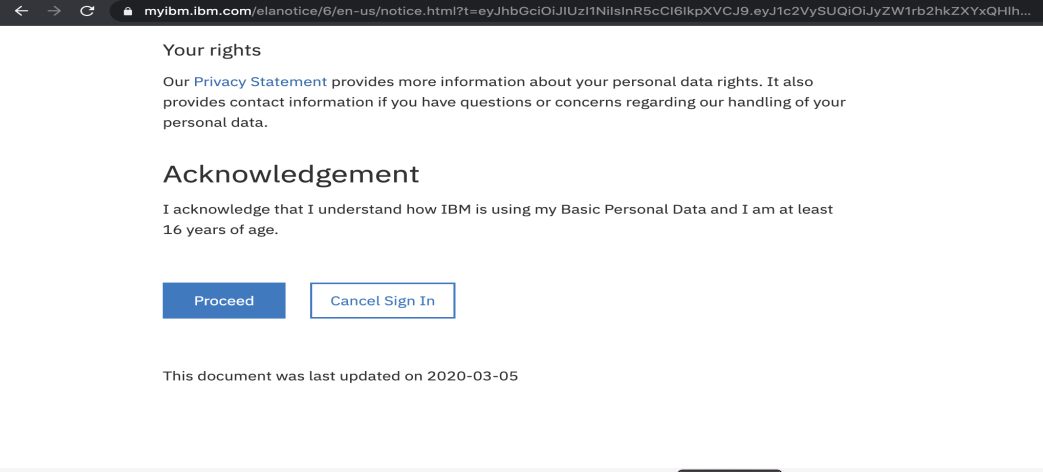
1. Your account is being created,



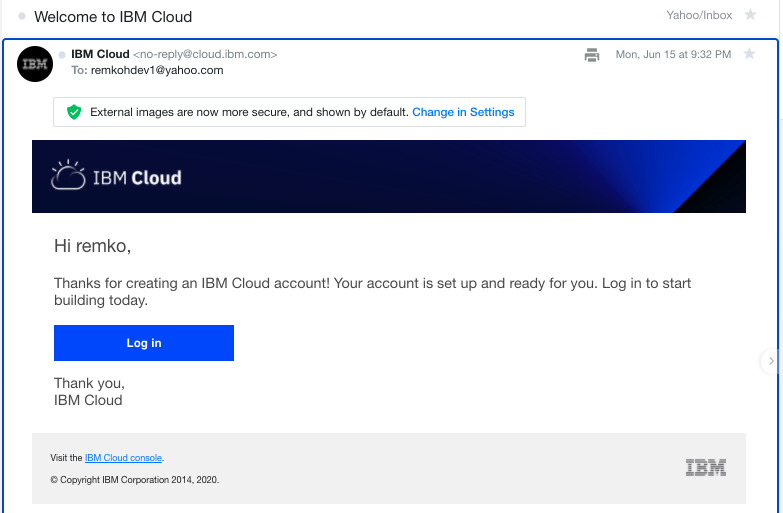
1. Review the IBM Privacy Statement,



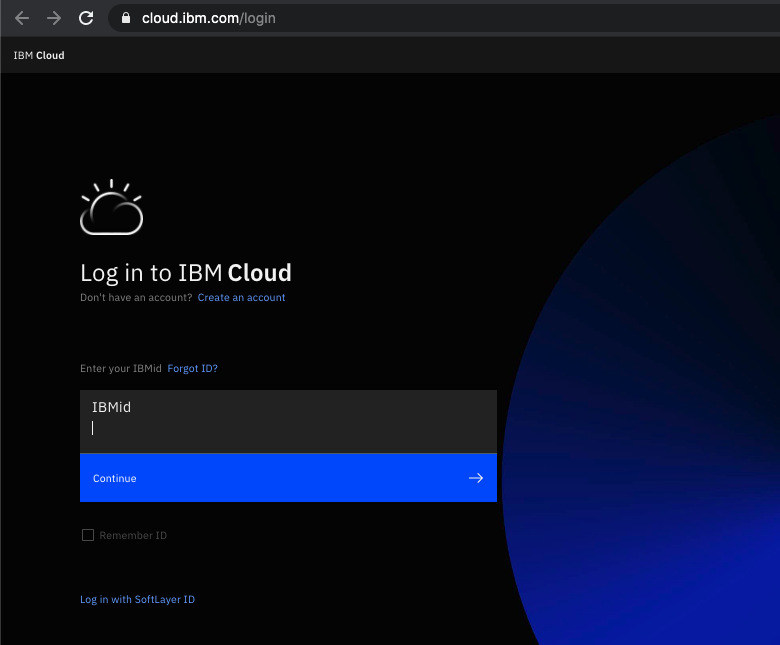
1. Click Proceed to acknowledge the privacy statement,



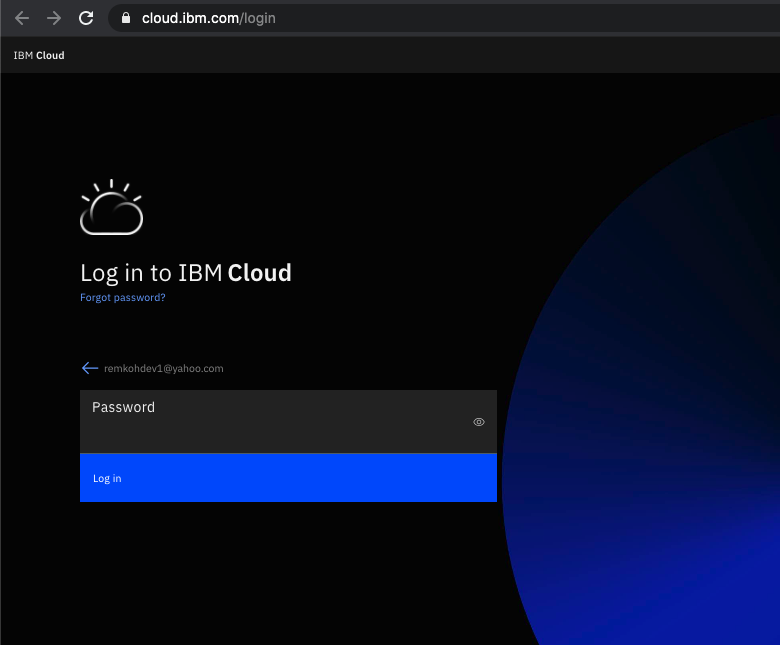
1. Switch to your email provider to review the Welcome to IBM Cloud email, and click the Login link,



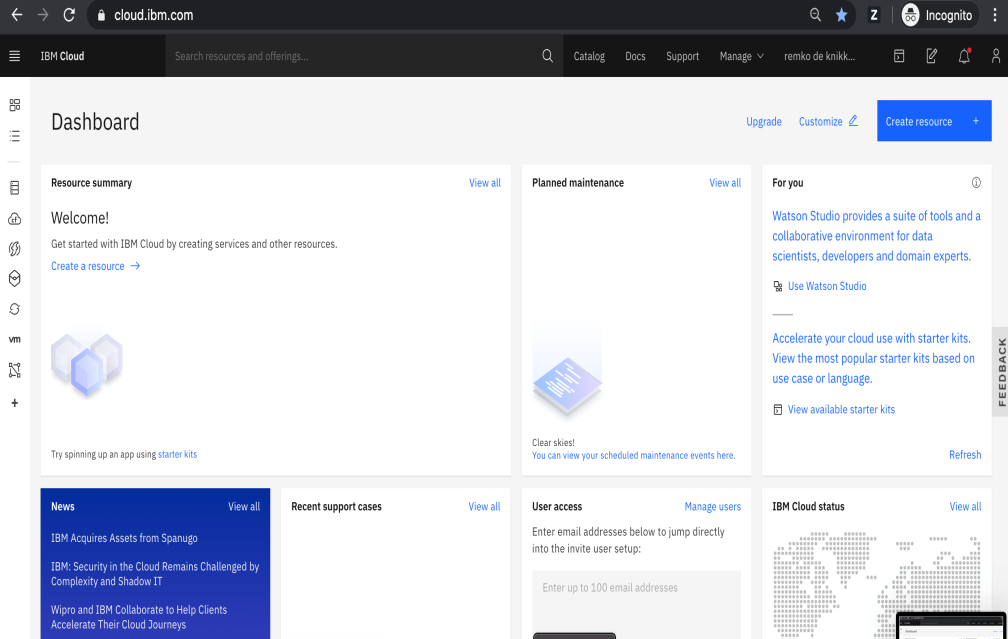
1. Enter your IBM Id to login,



1. Enter your password to login,



1. The IBM Cloud dashboard page should load,



1. You have successfully registered a new IBM Cloud account.

## Churn Prediction

### A Machine Learning Model That Can Predict Customers Who Will Leave The CompanyThe aim is to predict whether a bank's customers leave the bank or not. If the Client has closed his/her bank account, he/she has left.

## Dataset:

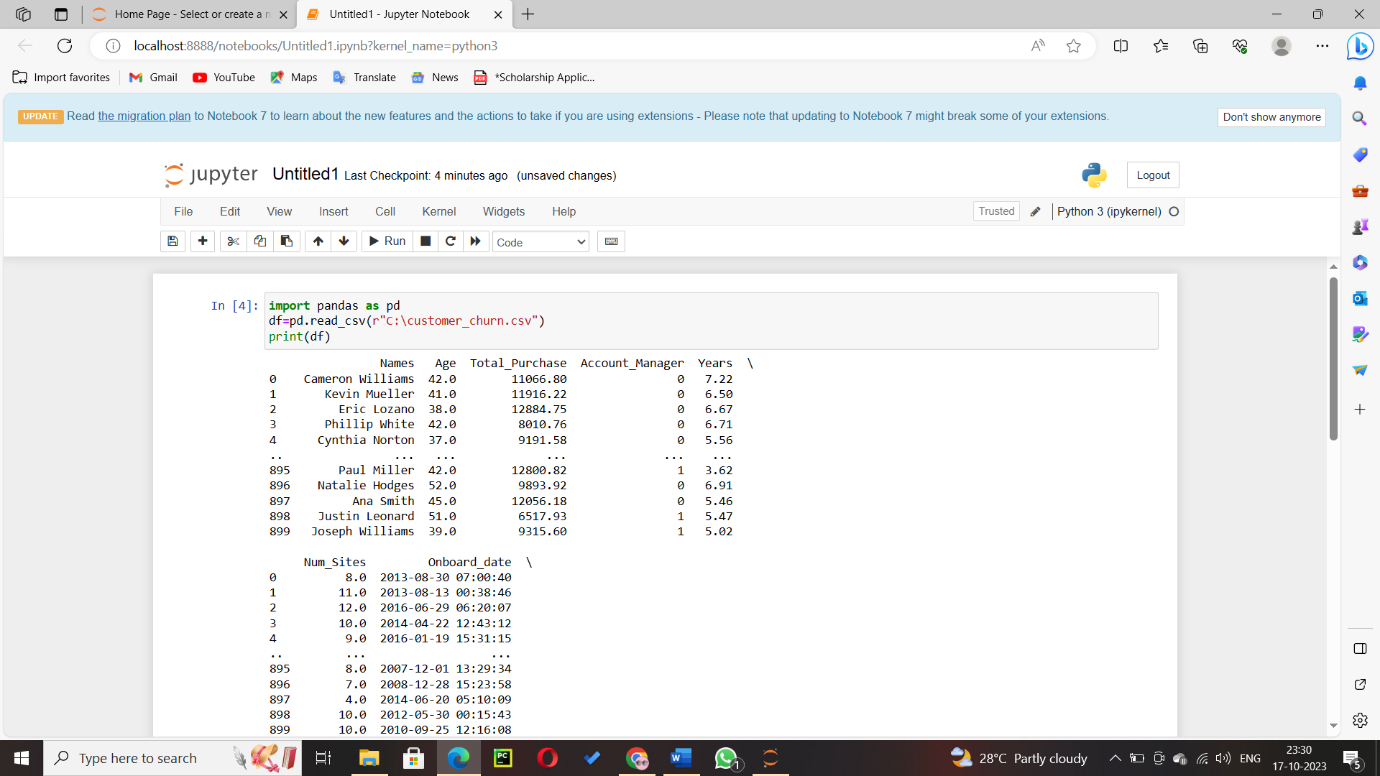
* **RowNumber:** corresponds to the record (row) number and has no effect on the output.
* **CustomerId:** contains random values and has no effect on customer leaving the bank.
* **Surname:** the surname of a customer has no impact on their decision to leave the bank.
* **CreditScore:** can have an 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.
* **Gender:** it’s interesting to explore whether gender plays a role in a customer leaving the bank.
* **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:** also a very good indicator of customer churn, 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.
* **IsActiveMember:** active customers are less likely to leave the bank.
* **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. (0=No,1=Yes)

Code:

import pandas as pd

df=pd.read\_csv(r”c:\customer\_churn.csv”)

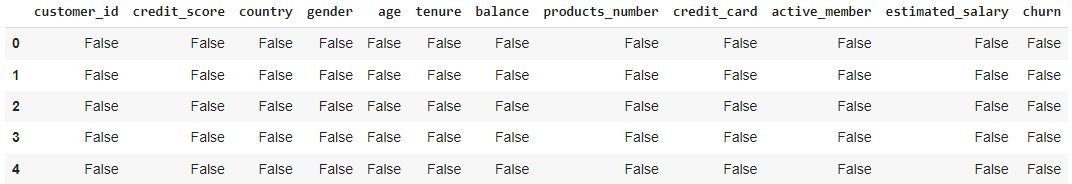
print(df)



Exploratory Data Analysis:

The first thing we have to do in Exploratory Data Analysis is checked if there are null values in the dataset.

df.isnull().head()



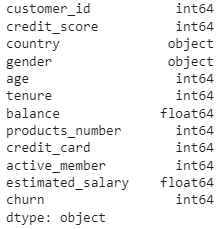
#This is used to check any null value.

df.isnull().sum()



**#Checking Data types**

df.dtypes



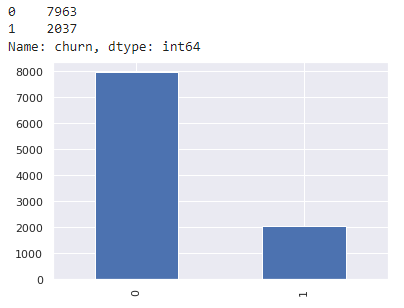
**#Counting 1 and 0 Value in Churn column**

color\_wheel = {1: "#0392cf", 2: "#7bc043"}

colors = df["churn"].map(lambda x: color\_wheel.get(x + 1))

print(df.churn.value\_counts())

p=df.churn.value\_counts().plot(kind="bar")



**#Change value in country column**

df['country'] = df['country'].replace(['Germany'],'0')

df['country'] = df['country'].replace(['France'],'1')

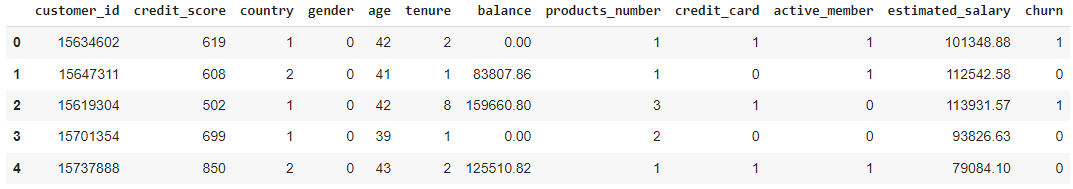
df['country'] = df['country'].replace(['Spain'],'2')

#Change value in gender column

df['gender'] = df['gender'].replace(['Female'],'0')

df['gender'] = df['gender'].replace(['Male'],'1')

df.head()

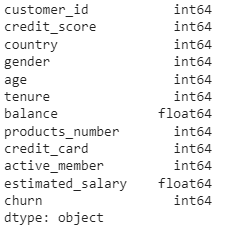


**#convert object data types column to integer**

df['country'] = pd.to\_numeric(df['country'])

df['gender'] = pd.to\_numeric(df['gender'])

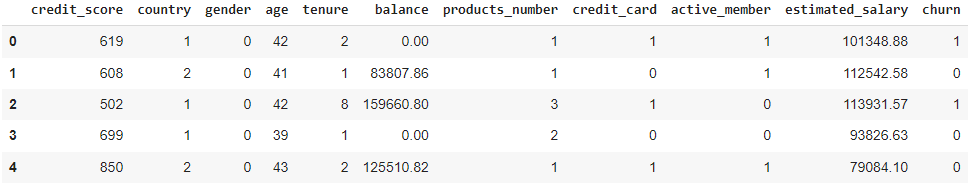
df.dtypes



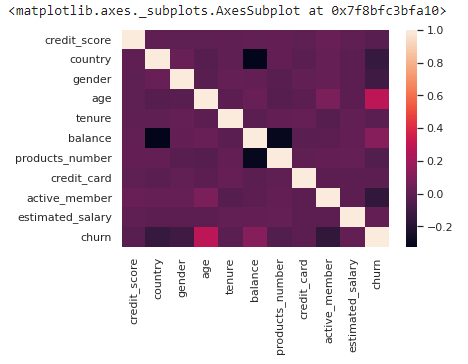
**#Remove customer\_id column**

df2 = df.drop('customer\_id', axis=1)

df2.head()



sns.heatmap(df2.corr(), fmt='.2g')



Build Machine Learning Model:

X = df2.drop('churn', axis=1)

y = df2['churn']

#test size 20% and train size 80%

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, cross\_val\_predict

from sklearn.metrics import accuracy\_score

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2,random\_state=7)

Decision Tree:

from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()

dtree.fit(X\_train, y\_train)

y\_pred = dtree.predict(X\_test)

print("Accuracy Score :", accuracy\_score(y\_test, y\_pred)\*100, "%")



Random Forest:

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

y\_pred = rfc.predict(X\_test)

print("Accuracy Score :", accuracy\_score(y\_test, y\_pred)\*100, "%")



Support Vector Machine:

from sklearn import svm

svm = svm.SVC()

svm.fit(X\_train, y\_train)

y\_pred = svm.predict(X\_test)

print("Accuracy Score :", accuracy\_score(y\_test, y\_pred)\*100, "%")



**XGBoost:**

from xgboost import XGBClassifier

xgb\_model = XGBClassifier()

xgb\_model.fit(X\_train, y\_train)

y\_pred = xgb\_model.predict(X\_test)

print("Accuracy Score :", accuracy\_score(y\_test, y\_pred)\*100, "%")



Visualize Random Forest and XGBoost Algorithm because Random Forest and XGBoost Algorithm have the Best Accuracy:

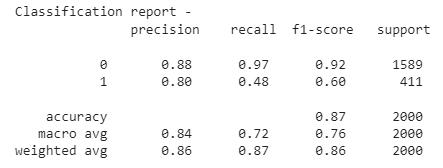
#importing classification report and confusion matrix from sklearn

from sklearn.metrics import classification\_report, confusion\_matrix

**Random Forest:**

y\_pred = rfc.predict(X\_test)

print("Classification report - n", classification\_report(y\_test,y\_pred))



cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(5,5))

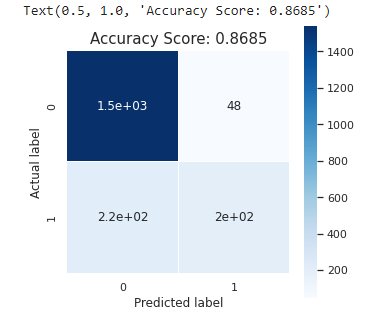
sns.heatmap(data=cm,linewidths=.5, annot=True,square = True, cmap = 'Blues')

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

all\_sample\_title = 'Accuracy Score: {0}'.format(rfc.score(X\_test, y\_test))

plt.title(all\_sample\_title, size = 15)



from sklearn.metrics import roc\_curve, roc\_auc\_score

y\_pred\_proba = rfc.predict\_proba(X\_test)[:][:,1]

df\_actual\_predicted = pd.concat([pd.DataFrame(np.array(y\_test), columns=['y\_actual']), pd.DataFrame(y\_pred\_proba, columns=['y\_pred\_proba'])], axis=1)

df\_actual\_predicted.index = y\_test.index

fpr, tpr, tr = roc\_curve(df\_actual\_predicted['y\_actual'], df\_actual\_predicted['y\_pred\_proba'])

auc = roc\_auc\_score(df\_actual\_predicted['y\_actual'], df\_actual\_predicted['y\_pred\_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)

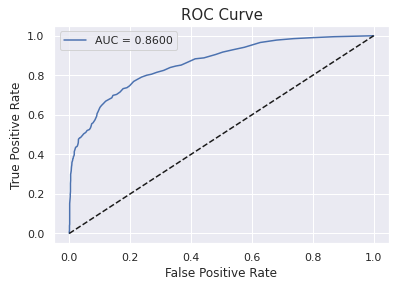
plt.plot(fpr, fpr, linestyle = '--', color='k')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve', size = 15)

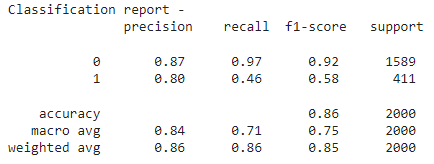
plt.legend()



**XGBoost:**

y\_pred = xgb\_model.predict(X\_test)

print("Classification report - n", classification\_report(y\_test,y\_pred))



cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(5,5))

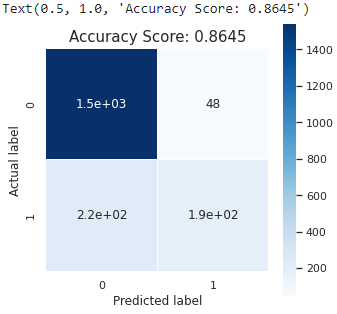
sns.heatmap(data=cm,linewidths=.5, annot=True,square = True, cmap = 'Blues')

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

all\_sample\_title = 'Accuracy Score: {0}'.format(xgb\_model.score(X\_test, y\_test))

plt.title(all\_sample\_title, size = 15)



from sklearn.metrics import roc\_curve, roc\_auc\_score

y\_pred\_proba = xgb\_model.predict\_proba(X\_test)[:][:,1]

df\_actual\_predicted = pd.concat([pd.DataFrame(np.array(y\_test), columns=['y\_actual']), pd.DataFrame(y\_pred\_proba, columns=['y\_pred\_proba'])], axis=1)

df\_actual\_predicted.index = y\_test.index

fpr, tpr, tr = roc\_curve(df\_actual\_predicted['y\_actual'], df\_actual\_predicted['y\_pred\_proba'])

auc = roc\_auc\_score(df\_actual\_predicted['y\_actual'], df\_actual\_predicted['y\_pred\_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)

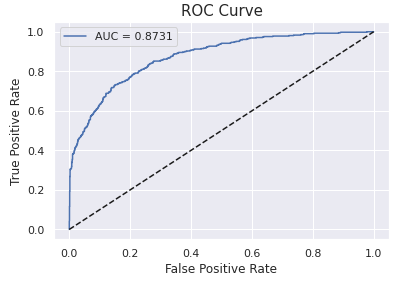
plt.plot(fpr, fpr, linestyle = '--', color='k')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve', size = 15)

plt.legend()



*# Descriptive statistics of the data set*

df.describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])

Out[6]:

|  | CustomerId | CreditScore | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exited |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 1.000000e+04 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.00000 | 10000.000000 | 10000.000000 | 10000.000000 |
| mean | 1.569094e+07 | 650.528800 | 38.921800 | 5.012800 | 76485.889288 | 1.530200 | 0.70550 | 0.515100 | 100090.239881 | 0.203700 |
| std | 7.193619e+04 | 96.653299 | 10.487806 | 2.892174 | 62397.405202 | 0.581654 | 0.45584 | 0.499797 | 57510.492818 | 0.402769 |
| min | 1.556570e+07 | 350.000000 | 18.000000 | 0.000000 | 0.000000 | 1.000000 | 0.00000 | 0.000000 | 11.580000 | 0.000000 |
| 5% | 1.557882e+07 | 489.000000 | 25.000000 | 1.000000 | 0.000000 | 1.000000 | 0.00000 | 0.000000 | 9851.818500 | 0.000000 |
| 25% | 1.562853e+07 | 584.000000 | 32.000000 | 3.000000 | 0.000000 | 1.000000 | 0.00000 | 0.000000 | 51002.110000 | 0.000000 |
| 50% | 1.569074e+07 | 652.000000 | 37.000000 | 5.000000 | 97198.540000 | 1.000000 | 1.00000 | 1.000000 | 100193.915000 | 0.000000 |
| 75% | 1.575323e+07 | 718.000000 | 44.000000 | 7.000000 | 127644.240000 | 2.000000 | 1.00000 | 1.000000 | 149388.247500 | 0.000000 |
| 90% | 1.579083e+07 | 778.000000 | 53.000000 | 9.000000 | 149244.792000 | 2.000000 | 1.00000 | 1.000000 | 179674.704000 | 1.000000 |
| 95% | 1.580303e+07 | 812.000000 | 60.000000 | 9.000000 | 162711.669000 | 2.000000 | 1.00000 | 1.000000 | 190155.375500 | 1.000000 |
| 99% | 1.581311e+07 | 850.000000 | 72.000000 | 10.000000 | 185967.985400 | 3.000000 | 1.00000 | 1.000000 | 198069.734500 | 1.000000 |
| max | 1.581569e+07 | 850.000000 | 92.000000 | 10.000000 | 250898.090000 | 4.000000 |  |  |  |  |

*# categorical Variables*

categorical\_variables = [col for col **in** df.columns if col **in** "O"

**or** df[col].nunique() <=11

**and** col **not** **in** "Exited"]

categorical\_variables

Out[7]:

['Geography',

'Gender',

'Tenure',

'NumOfProducts',

'HasCrCard',

'IsActiveMember']

In [8]:

*# Numeric Variables*

numeric\_variables = [col for col **in** df.columns if df[col].dtype != "object"

**and** df[col].nunique() >11

**and** col **not** **in** "CustomerId"]

numeric\_variables

Out[8]:

['CreditScore', 'Age', 'Balance', 'EstimatedSalary']

## Exited (Dependent Variable)

In [9]:

*# Frequency of classes of dependent variable*

df["Exited"].value\_counts()

Out[9]:

0 7963

1 2037

Name: Exited, dtype: int64

In [10]:

*# Customers leaving the bank*

churn = df.loc[df["Exited"]==1]

In [11]:

*# Customers who did not leave the bank*

not\_churn = df.loc[df["Exited"]==0]

# Categorical Variables

## Tenure

In [12]:

*# Frequency of not\_churn group according to Tenure*

not\_churn["Tenure"].value\_counts().sort\_values()

Out[12]:

0 318

10 389

6 771

9 771

4 786

3 796

5 803

1 803

8 828

2 847

7 851

Name: Tenure, dtype: int64

In [13]:

*# Frequency of churn group according to Tenure*

churn["Tenure"].value\_counts().sort\_values()

Out[13]:

0 95

10 101

7 177

6 196

8 197

2 201

4 203

5 209

9 213

3 213

1 232

Name: Tenure, dtype: int64

## NumOfProducts

In [14]:

*# Frequency of not\_churn group according to NumOfProducts*

not\_churn["NumOfProducts"].value\_counts().sort\_values()

Out[14]:

3 46

1 3675

2 4242

Name: NumOfProducts, dtype: int64

In [15]:

*# Frequency of churn group according to NumOfProducts*

churn["NumOfProducts"].value\_counts().sort\_values()

Out[15]:

4 60

3 220

2 348

1 1409

Name: NumOfProducts, dtype: int64

## HasCrCard

In [16]:

*# examining the HasCrCard of the not\_churn group*

not\_churn["HasCrCard"].value\_counts()

Out[16]:

1 5631

0 2332

Name: HasCrCard, dtype: int64

In [17]:

*# examining the HasCrCard of the churn group*

churn["HasCrCard"].value\_counts()

Out[17]:

1 1424

0 613

Name: HasCrCard, dtype: int64

## IsActiveMember

In [18]:

*# examining the IsActiveMember of the not\_churn group*

not\_churn["IsActiveMember"].value\_counts()

Out[18]:

1 4416

0 3547

Name: IsActiveMember, dtype: int64

In [19]:

*# examining the IsActiveMember of the churn group*

churn["IsActiveMember"].value\_counts()

Out[19]:

0 1302

1 735

Name: IsActiveMember, dtype: int64

## Geography

In [20]:

*# Frequency of not\_churn group according to Geography*

not\_churn.Geography.value\_counts().sort\_values()

Out[20]:

Germany 1695

Spain 2064

France 4204

Name: Geography, dtype: int64

In [21]:

*# Frequency of churn group according to Geography*

churn.Geography.value\_counts().sort\_values()

#Reference dataset: https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset

**Conclusion:**

Developing a Machine Learning Model is a complex process, but it is essential for building and deploying successful machine-learning applications. By following the steps outlined in this blog, you can increase your chances of success.