
# FINAL PROJECT

# CUSTOMER CHURN PREDICTION USING APACHE SPARK FOR IDENTIFYING ROBUST DATA, POWER BI FOR VISUALIZATION AND DOCKERIZING TO AUTOMATE DEPLOYMENT

**(COCSC19)**

BY:

Sushant 2021UCS1537

Deepak Kr Meena 2021UCS1544

Antriksh Bhandari 2021UCS1541

Kaushal Kumar 2021UCS1546

**1. OBJECTIVE:**

The objective of a project involving Apache Spark and Power BI for customer churn prediction is to develop a robust data analytics solution that can accurately identify customers who are likely to churn and provide actionable insights to mitigate churn.

**2. LIBRARIES & TOOLS USED:**

pandas: Used for data manipulation and analysis, including reading and preprocessing the dataset.

numpy: Essential for numerical computations and array manipulations, often used in conjunction with pandas.

scikit-learn (sklearn): A comprehensive machine learning library in Python that provides tools for data preprocessing,

model training, evaluation, and hyperparameter tuning.

matplotlib: A plotting library for creating static, interactive, and animated visualizations in Python.

seaborn: Built on top of matplotlib, seaborn provides additional statistical plots and enhances the visual appeal of plots.

**3. APPLICATIONS**

Telecommunications: Telecom companies can use churn prediction to identify customers at risk of leaving their services. They can then offer targeted promotions or discounts to retain these customers, ultimately reducing churn and improving customer satisfaction.

Banking: Banks can use churn prediction to identify customers who are likely to close their accounts or switch to other banks. By understanding the reasons for churn, banks can improve their services and offer personalized incentives to retain customers.

E-commerce: E-commerce platforms can use churn prediction to identify customers who are likely to stop making purchases or abandon their carts. They can then send targeted marketing campaigns or offers to re-engage these customers and prevent churn.

Subscription-based Services: Companies offering subscription-based services, such as streaming platforms or software-as-a-service (SaaS) providers, can use churn prediction to identify customers who are likely to cancel their subscriptions. By proactively reaching out to these customers and addressing their concerns, companies can reduce churn and improve customer lifetime value.

## 4. IMPLEMENTATION:

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report, f1_score, precision_score, recall_score, roc_auc_score, roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from catboost import CatBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import accuracy_score,recall_score
from xgboost import XGBClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, GridSearchCV

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)



# to display all columns and rows:
```

[2]:
```
from pyspark.sql import SparkSession

# Get or create SparkSession
spark = SparkSession.builder \
    .appName("churnApp") \
    .master("spark://192.168.0.103:7077") \
    .getOrCreate()
```

[3]:
```
import pandas as pd
import numpy as np

# Create pd_temp
pd_temp = pd.DataFrame(np.random.random(10))

# Create spark_temp from pd_temp
spark_temp = spark.createDataFrame(pd_temp)

# Examine the tables in the catalog
print(spark.catalog.listTables())

# Add spark_temp to the catalog
spark_temp.createOrReplaceTempView("temp")

# Examine the tables in the catalog again
print(spark.catalog.listTables())
```

```
[]
[Table(name='temp', catalog=None, namespace=[], description=None, tableType='TEMPORARY', isTemporary=True)]
```

[4]:
```
df = pd.read_csv("C:\\Users\\D30K\\Downloads\\churnbank.csv")
```

[5]:
```
df.head()
```

[5]:

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | 101348.88 | |
| **1** | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | 112542.58 | |
| **2** | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | 113931.57 | |
| **3** | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | 93826.63 | |
| **4** | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 | 1 | 79084.10 | |

[6]:
```
df.shape
```

[6]:
```
(10000, 14)
```

[7]:
```
df.info()
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

[8]:
```
categorical_variables = [col for col in df.columns if col in "O"
                         or df[col].nunique() <=11
                         and col not in "Exited"]

categorical_variables
```

[8]:
```
['Geography',
 'Gender',
 'Tenure',
 'NumOfProducts',
 'HasCrCard',
 'IsActiveMember']
```

[9]:
```
# 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
```

[9]:
```
['RowNumber', 'CreditScore', 'Age', 'Balance', 'EstimatedSalary']
```

[10]:
```
# Frequency of classes of dependent variable
df["Exited"].value_counts()
```

[10]:
```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

[11]:
```
# Customers leaving the bank
churn = df.loc[df["Exited"]==1]
```

[12]:
```
# Customers who did not leave the bank
not_churn = df.loc[df["Exited"]==0]
```

[13]:
```
# Frequency of not_churn group according to Tenure
not_churn["Tenure"].value_counts().sort_values()
```

[13]:
```
Tenure
0     318
10    389
6     771
9     771
4     786
3     796
1     803
5     803
8     828
2     847
7     851
Name: count, dtype: int64
```

[14]:
```
# Frequency of churn group according to Tenure
churn["Tenure"].value_counts().sort_values()
```

[14]:
```
Tenure
0      95
10    101
7     177
6     196
8     197
2     201
4     203
5     209
3     213
```

[15]:
```
# Frequency of not_churn group according to NumOfProducts
not_churn["NumOfProducts"].value_counts().sort_values()
```

[15]:
```
NumOfProducts
3      46
1    3675
2    4242
Name: count, dtype: int64
```

[16]:
```
# Frequency of churn group according to NumOfProducts
churn["NumOfProducts"].value_counts().sort_values()
```

[16]:
```
NumOfProducts
4      60
3     220
2     348
1    1409
Name: count, dtype: int64
```

[17]:
```
# examining the HasCrCard of the not_churn group
not_churn["HasCrCard"].value_counts()
```

[17]:
```
HasCrCard
1    5631
0    2332
Name: count, dtype: int64
```

[18]:
```
# examining the HasCrCard of the churn group
churn["HasCrCard"].value_counts()
```

[22]:
```
Geography
Spain      413
France     810
Germany    814
Name: count, dtype: int64
```

[23]:
```
# Frequency of not_churn group according to Gender
not_churn.Gender.value_counts()
```

[23]:
```
Gender
Male      4559
Female    3404
Name: count, dtype: int64
```

[24]:
```
# Frequency of churn group according to Gender
churn.Gender.value_counts()
```

[24]:
```
Gender
Female    1139
Male       898
Name: count, dtype: int64
```

[25]:
```
# Let's examine the credit score of the not_churn group
not_churn["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[18]:
```
HasCrCard
1    1424
0     613
Name: count, dtype: int64
```

[19]:
```
# examining the IsActiveMember of the not_churn group
not_churn["IsActiveMember"].value_counts()
```

[19]:
```
IsActiveMember
1    4416
0    3547
Name: count, dtype: int64
```

[20]:
```
# examining the IsActiveMember of the churn group
churn["IsActiveMember"].value_counts()
```

[20]:
```
IsActiveMember
0    1302
1     735
Name: count, dtype: int64
```

[21]:
```
# Frequency of not_churn group according to Geography
not_churn.Geography.value_counts().sort_values()
```

[21]:
```
Geography
Germany    1695
Spain      2064
France     4204
Name: count, dtype: int64
```

[22]:
```
# Frequency of churn group according to Geography
churn.Geography.value_counts().sort_values()
```

[22]:
```
Geography
Spain      413
France     810
Germany    814
Name: count, dtype: int64
```

[23]:
```
# Frequency of not_churn group according to Gender
not_churn.Gender.value_counts()
```

[23]:
```
Gender
Male      4559
Female    3404
Name: count, dtype: int64
```

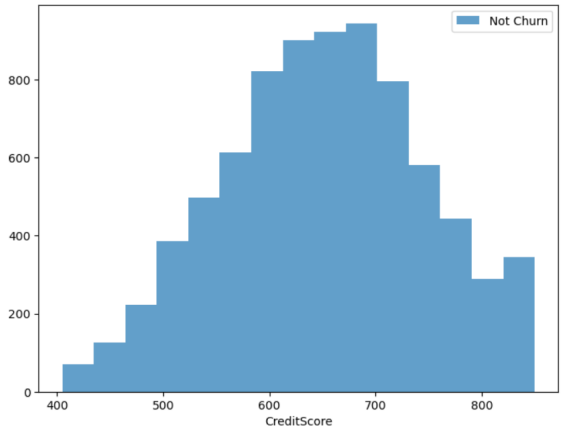
[24]:
```
# Frequency of churn group according to Gender
churn.Gender.value_counts()
```

[24]:
```
Gender
Female    1139
Male       898
Name: count, dtype: int64
```

[25]:
```
# Let's examine the credit score of the not_churn group
not_churn["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[25]:
```
count    7963.000000
mean      651.853196
std        95.653837
min       405.000000
5%        492.000000
25%       585.000000
50%       653.000000
75%       718.000000
90%       778.000000
95%       812.000000
99%       850.000000
max       850.000000
Name: CreditScore, dtype: float64
```

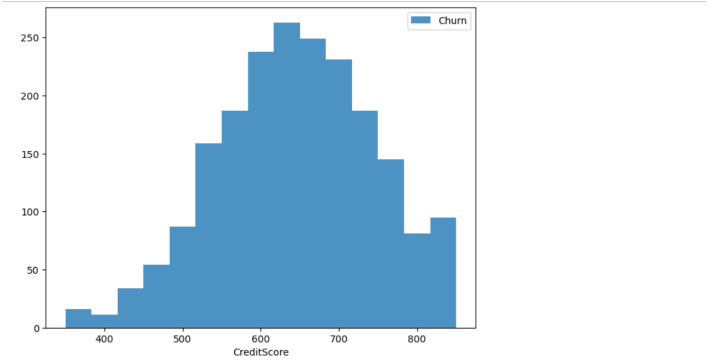
[26]:
```
# distribution of the Credit Score for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('CreditScore')
pyplot.hist(not_churn["CreditScore"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




[27]:
```
#Let's examine the credit score of the churn group
churn["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[27]:
```
count    2037.000000
mean      645.351497
std       100.321503
min       350.000000
5%        479.000000
25%       578.000000
50%       646.000000
75%       716.000000
90%       776.400000
95%       812.200000
99%       850.000000
max       850.000000
Name: CreditScore, dtype: float64
```

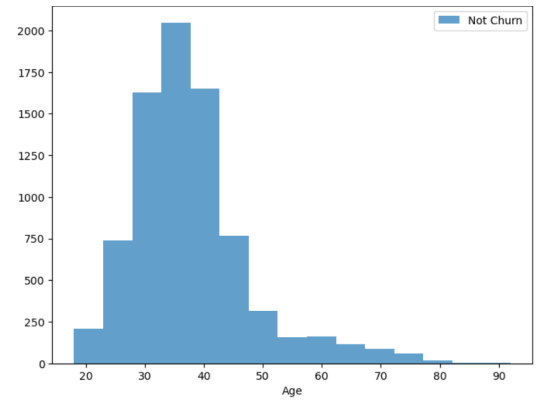
[28]:
```
# distribution of the Credit Score for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('CreditScore')
pyplot.hist(churn["CreditScore"],bins=15, alpha=0.8, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# examining the age of the not_churn group
not_churn["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[29]:
```
count    7963.000000
mean       37.408389
std        10.125363
min        18.000000
5%         24.000000
25%        31.000000
50%        36.000000
75%        41.000000
90%        49.000000
95%        59.000000
99%        73.000000
max        92.000000
Name: Age, dtype: float64
```

[30]:
```
# distribution of the Age for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age')
pyplot.hist(not_churn["Age"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```



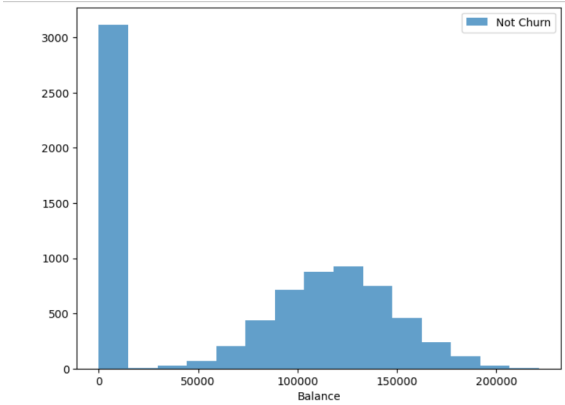

```
# examine the age of the churn group
churn["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[32]:
```
# distribution of the Age for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age')
pyplot.hist(churn["Age"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```

[33]:
```
# examining the Balance of the not_churn group
not_churn["Balance"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[33]:
```
count      7963.000000
mean      72745.296779
std       62848.040701
min           0.000000
5%            0.000000
25%           0.000000
50%       92072.680000
75%      126410.280000
90%      148730.298000
95%      161592.595000
99%      183753.906200
max      221532.800000
Name: Balance, dtype: float64
```

[34]:
```
# distribution of the Balance for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Balance')
pyplot.hist(not_churn["Balance"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




[35]:
```
# examining the Balance of the churn group
churn["Balance"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

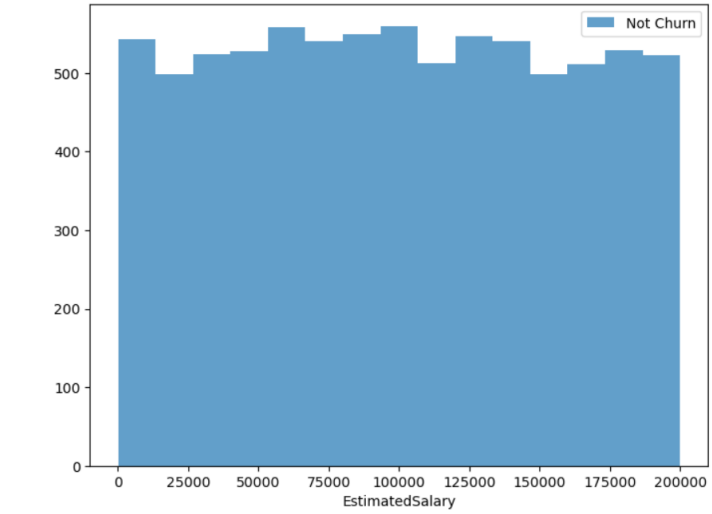
[35]:
```
count      2037.000000
mean      91108.539337
std       58360.794816
min           0.000000
5%            0.000000
25%       38340.020000
50%      109349.290000
75%      131433.330000
90%      152080.618000
95%      167698.240000
99%      197355.288400
max      250898.090000
Name: Balance, dtype: float64
```

[36]:
```
# distribution of the Balance for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Balance')
pyplot.hist(churn["Balance"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```

[37]:
```
not_churn["EstimatedSalary"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[37]:
```
count      7963.000000
mean      99738.391772
std       57405.586966
min          90.070000
5%         9773.542000
25%       50783.490000
50%       99645.040000
75%      148609.955000
90%      179453.212000
95%      190107.557000
99%      198131.465200
max      199992.480000
Name: EstimatedSalary, dtype: float64
```

[38]:
```
# distribution of the Balance for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('EstimatedSalary')
pyplot.hist(not_churn["EstimatedSalary"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```

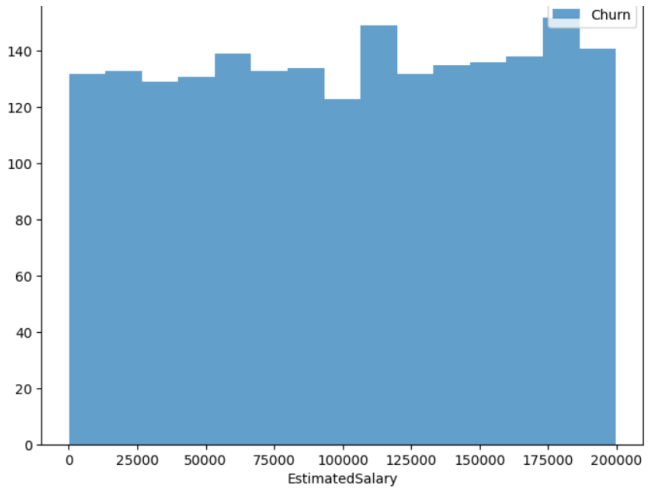



[39]:
```
churn["EstimatedSalary"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[39]:
```
churn["EstimatedSalary"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[39]:
```
count      2037.000000
mean     101465.677531
std       57912.418071
min          11.580000
5%        10030.760000
25%       51907.720000
50%      102460.840000
75%      152422.910000
90%      180169.390000
95%      190328.982000
99%      197717.297600
max      199808.100000
Name: EstimatedSalary, dtype: float64
```

[40]:
```
# distribution of the EstimatedSalary for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('EstimatedSalary')
pyplot.hist(churn["EstimatedSalary"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




[1]:
```
# # Missing Observation Analysis
df.isnull().sum()
```

[41]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

[42]:
```
# To determine the threshold value for outliers
def outlier_thresholds(dataframe, variable, low_quantile=0.05, up_quantile=0.95):
    quantile_one = dataframe[variable].quantile(low_quantile)
    quantile_three = dataframe[variable].quantile(up_quantile)
    interquantile_range = quantile_three - quantile_one
    up_limit = quantile_three + 1.5 * interquantile_range
    low_limit = quantile_one - 1.5 * interquantile_range
    return low_limit, up_limit
```

[43]:
```
# Are there any outliers in the variables
def has_outliers(dataframe, numeric_columns, plot=False):
   # variable_names = []
    for col in numeric_columns:
        low_limit, up_limit = outlier_thresholds(dataframe, col)
        if dataframe[(dataframe[col] > up_limit) | (dataframe[col] < low_limit)].any(axis=None):
            number_of_outliers = dataframe[(dataframe[col] > up_limit) | (dataframe[col] < low_limit)].shape[0]
            print(col, " : ", number_of_outliers, "outliers")
            #variable_names.append(col)
            if plot:
                sns.boxplot(x=dataframe[col])
                plt.show()
    #return variable_names
```

[44]:
```
# There is no outlier
for var in numeric_variables:
    print(var, "has " , has_outliers(df, [var]),  "Outliers")
```

```
RowNumber has  None Outliers
CreditScore has  None Outliers
Age has  None Outliers
Balance has  None Outliers
EstimatedSalary has  None Outliers
```

[45]:
```
# we standardize tenure with age
df["NewTenure"] = df["Tenure"]/df["Age"]
df["NewCreditsScore"] = pd.qcut(df['CreditScore'], 6, labels = [1, 2, 3, 4, 5, 6])
df["NewAgeScore"] = pd.qcut(df['Age'], 8, labels = [1, 2, 3, 4, 5, 6, 7, 8])
df["NewBalanceScore"] = pd.qcut(df['Balance'].rank(method="first"), 5, labels = [1, 2, 3, 4, 5])
df["NewEstSalaryScore"] = pd.qcut(df['EstimatedSalary'], 10, labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

[46]:
```
df.head()
```

[46]:

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | 101348.88 | |
| **1** | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | 112542.58 | |
| **2** | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | 113931.57 | |
| **3** | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | 93826.63 | |
| **4** | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 | 1 | 79084.10 | |

[47]:
```
# Variables to apply one hot encoding
list = ["Gender", "Geography"]
df = pd.get_dummies(df, columns =list, drop_first = True)
```

[48]:
```
df.head()
```

[48]:

| | RowNumber | CustomerId | Surname | CreditScore | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exited | NewTenure | New |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | 42 | 2 | 0.00 | 1 | 1 | 1 | 101348.88 | 1 | 0.047619 | |
| **1** | 2 | 15647311 | Hill | 608 | 41 | 1 | 83807.86 | 1 | 0 | 1 | 112542.58 | 0 | 0.024390 | |
| **2** | 3 | 15619304 | Onio | 502 | 42 | 8 | 159660.80 | 3 | 1 | 0 | 113931.57 | 1 | 0.190476 | |
| **3** | 4 | 15701354 | Boni | 699 | 39 | 1 | 0.00 | 2 | 0 | 0 | 93826.63 | 0 | 0.025641 | |
| **4** | 5 | 15737888 | Mitchell | 850 | 43 | 2 | 125510.82 | 1 | 1 | 1 | 79084.10 | 0 | 0.046512 | |

[49]:
```
# Removing variables that will not affect the dependent variable
df = df.drop(["CustomerId","Surname"], axis = 1)
```

[50]:
```
def robust_scaler(variable):
    var_median = variable.median()
    quartile1 = variable.quantile(0.25)
    quartile3 = variable.quantile(0.75)
    interquantile_range = quartile3 - quartile1
    if int(interquantile_range) == 0:
        quartile1 = variable.quantile(0.05)
        quartile3 = variable.quantile(0.95)
        interquantile_range = quartile3 - quartile1
        if int(interquantile_range) == 0:
            quartile1 = variable.quantile(0.01)
            quartile3 = variable.quantile(0.99)
            interquantile_range = quartile3 - quartile1
            z = (variable - var_median) / interquantile_range
            return round(z, 3)

        z = (variable - var_median) / interquantile_range
        return round(z, 3)
    else:
        z = (variable - var_median) / interquantile_range
        return round(z, 3)
```

[51]:
```
new_cols_ohe = ["Gender_Male","Geography_Germany","Geography_Spain"]
like_num = [col for col in df.columns if df[col].dtypes != 'O' and len(df[col].value_counts()) <= 10]
cols_need_scale = [col for col in df.columns if col not in new_cols_ohe
                   and col not in "Exited"
                   and col not in like_num]
```

[52]:
```
df.head()
```

[52]:

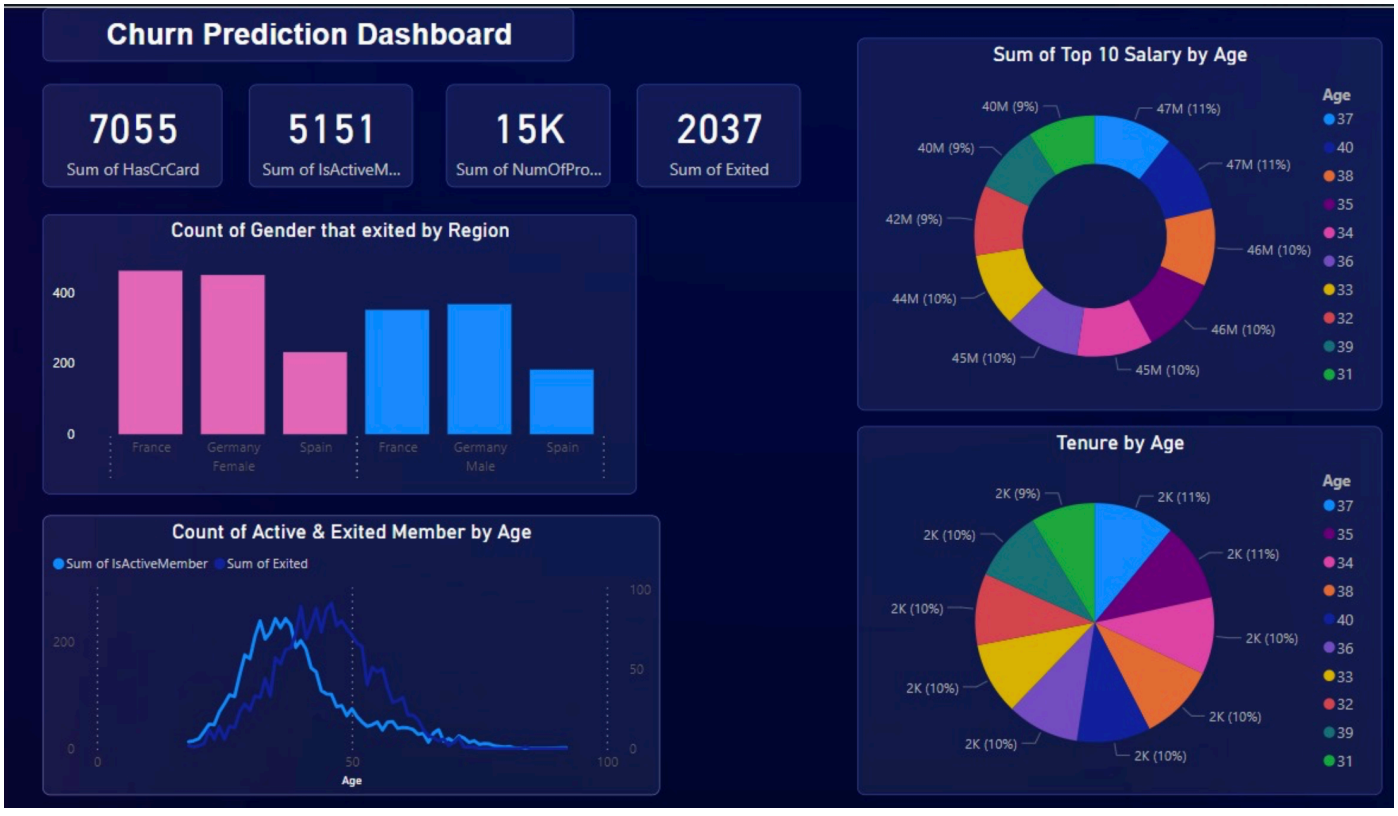
| | RowNumber | CreditScore | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exited | NewTenure | NewCreditsScore | NewAgeSc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -1.000 | -0.246 | 0.417 | -0.75 | -0.761 | 1 | 1 | 1 | 0.012 | 1 | -0.217 | 3 | |
| **1** | -1.000 | -0.328 | 0.333 | -1.00 | -0.105 | 1 | 0 | 1 | 0.126 | 0 | -0.279 | 2 | |
| **2** | -1.000 | -1.119 | 0.417 | 0.75 | 0.489 | 3 | 1 | 0 | 0.140 | 1 | 0.164 | 1 | |
| **3** | -0.999 | 0.351 | 0.167 | -1.00 | -0.761 | 2 | 0 | 0 | -0.065 | 0 | -0.276 | 5 | |
| **4** | -0.999 | 1.478 | 0.500 | -0.75 | 0.222 | 1 | 1 | 1 | -0.215 | 0 | -0.220 | 6 | |

[53]:
```
X = df.drop("Exited", axis=1)
y = df["Exited"]

# Train-Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=12345)

# Models for Classification
models = [('LR', LogisticRegression(random_state=123456)),
          ('KNN', KNeighborsClassifier()),
          ('CART', DecisionTreeClassifier(random_state=123456)),
          ('RF', RandomForestClassifier(random_state=123456)),
          ('SVR', SVC(gamma='auto', random_state=123456)),
          ('GB', GradientBoostingClassifier(random_state=12345))]

results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=123456)  # Set shuffle to True
    cv_results = cross_val_score(model, X, y, cv=kfold)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

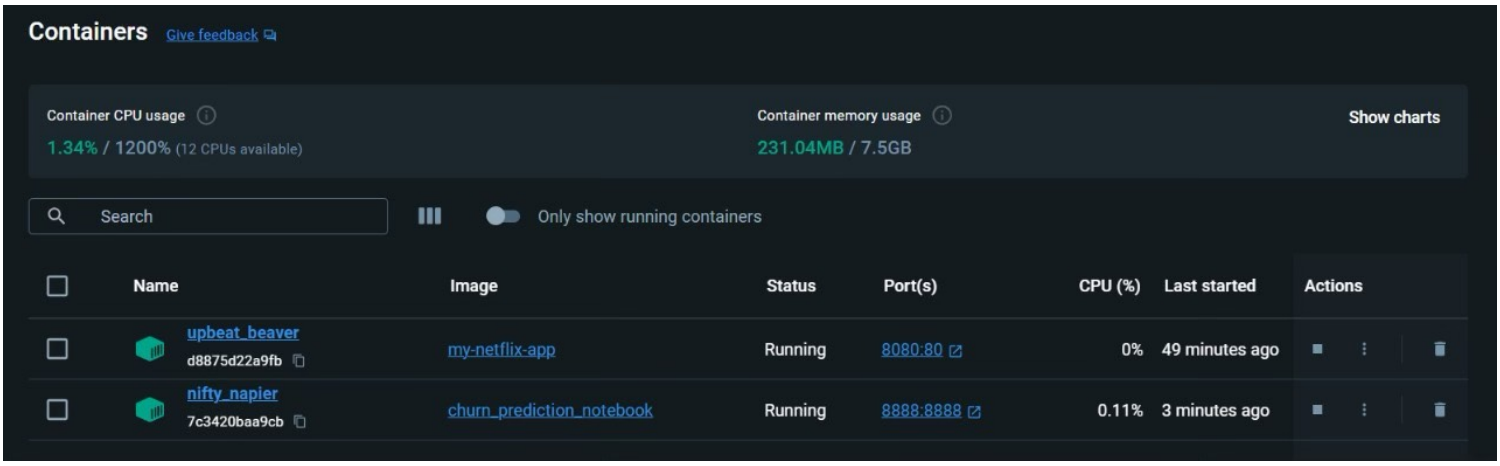
```
LR: 0.824100 (0.009439)
KNN: 0.820400 (0.010229)
CART: 0.794000 (0.012961)
RF: 0.862800 (0.006384)
SVR: 0.846000 (0.008764)
GB: 0.864300 (0.009778)
```

[54]:
```
spark.stop()
```

## POWER BI VISUALIZATION



**DOCKER**




churnprediction.ipynb | Dockerfile

Dockerfile > ...

```dockerfile
# Use an official Python runtime as a parent image
FROM jupyter/base-notebook:latest

# Set the working directory in the container
WORKDIR /home/jovyan/churn

# Copy the notebook into the container at /home/jovyan/churn
COPY churnprediction.ipynb /home/jovyan/churn


# Expose port 8888 to access Jupyter Notebook
EXPOSE 8888

# Start Jupyter Notebook
CMD ["jupyter", "notebook", "--ip='*'", "--port=8888", "--no-browser", "--allow-root"]

```

```
PS C:\Users\sushant kaddu\Desktop\churn> docker build -t churn_prediction_notebook .
>>
[+] Building 0.0s (0/0)  docker:default
[+] Building 2.7s (8/8) FINISHED                                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                                        0.0s
 => => transferring dockerfile: 609B                                                                                        0.0s
 => [internal] load metadata for docker.io/jupyter/base-notebook:latest                                                     2.4s
 => [internal] load .dockerignore                                                                                           0.0s
 => => transferring context: 2B                                                                                             0.0s
 => [1/3] FROM docker.io/jupyter/base-notebook:latest@sha256:8c903974902b0e9d45d9823c2234411de0614c5c98c4bb782b3d4f55b3e435e6  0.0s
 => [internal] load build context                                                                                           0.0s
 => => transferring context: 44B                                                                                            0.0s
 => CACHED [2/3] WORKDIR /home/jovyan/churn                                                                                 0.0s
 => CACHED [3/3] COPY churnprediction.ipynb /home/jovyan/churn                                                              0.0s
 => exporting to image                                                                                                      0.0s
 => => exporting layers                                                                                                     0.0s
 => => writing image sha256:e3a5731881946cbe113d09aff725425b45a519f83634a7127521efbd28e2111f                                0.0s
 => => naming to docker.io/library/churn_prediction_notebook                                                                0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```

```
PS C:\Users\sushant kaddu\Desktop\churn> docker run -p 8888:8888 churn_prediction_notebook
>>
[I 2024-05-03 03:04:52.525 ServerApp] Package notebook took 0.0000s to import
[I 2024-05-03 03:04:52.592 ServerApp] Package jupyter_lsp took 0.0653s to import
[W 2024-05-03 03:04:52.592 ServerApp] A `_jupyter_server_extension_points` function was not found in jupyter_lsp. Ins
er_extension_paths` function was found and will be used for now. This function name will be deprecated in future rele
.
[I 2024-05-03 03:04:52.620 ServerApp] Package jupyter_server_terminals took 0.0273s to import
[I 2024-05-03 03:04:52.621 ServerApp] Package jupyterlab took 0.0000s to import
[I 2024-05-03 03:04:54.068 ServerApp] Package nbclassic took 0.0098s to import
[W 2024-05-03 03:04:54.074 ServerApp] A `_jupyter_server_extension_points` function was not found in nbclassic. Inste
_extension_paths` function was found and will be used for now. This function name will be deprecated in future releas
[I 2024-05-03 03:04:54.075 ServerApp] Package notebook_shim took 0.0000s to import
[W 2024-05-03 03:04:54.075 ServerApp] A `_jupyter_server_extension_points` function was not found in notebook_shim. I
rver_extension_paths` function was found and will be used for now. This function name will be deprecated in future re
er.
[I 2024-05-03 03:04:54.077 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-05-03 03:04:54.088 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-05-03 03:04:54.100 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-05-03 03:04:54.110 ServerApp] nbclassic | extension was successfully linked.
[I 2024-05-03 03:04:54.123 ServerApp] notebook | extension was successfully linked.
[I 2024-05-03 03:04:54.128 ServerApp] Writing Jupyter server cookie secret to /home/jovyan/.local/share/jupyter/runti
et
[I 2024-05-03 03:04:54.921 ServerApp] notebook_shim | extension was successfully linked.
[W 2024-05-03 03:04:54.972 ServerApp] WARNING: The Jupyter server is listening on all IP addresses and not using encr
commended.
```

```
[I 2024-05-03 03:04:54.972 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-05-03 03:04:54.976 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-05-03 03:04:54.978 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-05-03 03:04:54.982 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.11/site-packages/jupyterlab
[I 2024-05-03 03:04:54.982 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 2024-05-03 03:04:54.983 LabApp] Extension Manager is 'pypi'.
[I 2024-05-03 03:04:54.987 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-05-03 03:04:54.996 ServerApp] nbclassic | extension was successfully loaded.
[I 2024-05-03 03:04:55.000 ServerApp] notebook | extension was successfully loaded.
[I 2024-05-03 03:04:55.002 ServerApp] Serving notebooks from local directory: /home/jovyan/churn
[I 2024-05-03 03:04:55.002 ServerApp] Jupyter Server 2.8.0 is running at:
[I 2024-05-03 03:04:55.002 ServerApp] http://localhost:8888/tree?token=96dcfb9f7c05d4c2c7522ed47c3ad6029badad61558cd04f
[I 2024-05-03 03:04:55.003 ServerApp]     http://127.0.0.1:8888/tree?token=96dcfb9f7c05d4c2c7522ed47c3ad6029badad61558cd04f
[I 2024-05-03 03:04:55.003 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation
[C 2024-05-03 03:04:55.008 ServerApp]

    To access the server, open this file in a browser:
        file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/tree?token=96dcfb9f7c05d4c2c7522ed47c3ad6029badad61558cd04f
        http://127.0.0.1:8888/tree?token=96dcfb9f7c05d4c2c7522ed47c3ad6029badad61558cd04f
[I 2024-05-03 03:04:56.178 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-nodej
typescript-langserver, jedi-language-server, julia-language-server, pyright, python-language-server, python-lsp-server, r-lan
ql-language-server, texlab, typescript-language-server, unified-language-server, vscode-css-languageserver-bin, vscode-html-l
bin, vscode-json-languageserver-bin, yaml-language-server
[I 2024-05-03 03:05:26.090 ServerApp] 302 GET / (@172.17.0.1) 0.50ms
```

```
[I 2024-05-03 03:05:26.090 ServerApp] 302 GET / (@172.17.0.1) 0.50ms
[I 2024-05-03 03:05:26.105 JupyterNotebookApp] 302 GET /tree? (@172.17.0.1) 0.86ms
[I 2024-05-03 03:05:29.890 ServerApp] 302 GET / (@172.17.0.1) 0.52ms
[I 2024-05-03 03:05:29.906 JupyterNotebookApp] 302 GET /tree? (@172.17.0.1) 0.63ms
[I 2024-05-03 03:05:34.844 JupyterNotebookApp] 302 GET /tree? (@172.17.0.1) 1.08ms
[I 2024-05-03 03:05:55.786 ServerApp] 302 GET / (@172.17.0.1) 0.46ms
[I 2024-05-03 03:05:55.798 JupyterNotebookApp] 302 GET /tree? (@172.17.0.1) 0.73ms
[I 2024-05-03 03:06:17.470 JupyterNotebookApp] 302 GET /tree?token=[secret] (@172.17.0.1) 0.95ms
[I 2024-05-03 03:06:25.633 JupyterNotebookApp] 302 GET /notebooks/churnprediction.ipynb (@172.17.0.1) 1.35ms
[I 2024-05-03 03:06:31.995 JupyterNotebookApp] 302 GET /tree (@172.17.0.1) 2.09ms
[I 2024-05-03 03:06:32.920 JupyterNotebookApp] 302 GET /notebooks/churnprediction.ipynb (@172.17.0.1) 1.71ms
0.00s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
[I 2024-05-03 03:07:35.168 ServerApp] Writing notebook-signing key to /home/jovyan/.local/share/jupyter/notebook_secret
[W 2024-05-03 03:07:35.182 ServerApp] Notebook churnprediction.ipynb is not trusted
[W 2024-05-03 03:07:37.428 ServerApp] Notebook churnprediction.ipynb is not trusted
[I 2024-05-03 03:07:37.745 ServerApp] Kernel started: 74c1afb4-6476-4aaf-b0ad-f8d5cdafcda4
0.00s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
[I 2024-05-03 03:07:38.857 ServerApp] Connecting to kernel 74c1afb4-6476-4aaf-b0ad-f8d5cdafcda4.
```

**ON LOCALHOST:8888 WE GET**