```
import pandas as pd
url = 'https://raw.githubusercontent.com/rahul-1415/DataSet/main/Churn_Modelling.csv'
df = pd.read_csv(url)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8
2	3	15619304	Onio	502	France	Female	42	8	15
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12
...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	
9996	9997	15569892	Johnstone	516	France	Male	35	10	5
9997	9998	15584532	Liu	709	France	Female	36	7	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	7
9999	10000	15628319	Walker	792	France	Female	28	4	13

10000 rows × 14 columns

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	1	15634602	Hargrave	619	France	Female	42	2	0
1	2	15647311	Hill	608	Spain	Female	41	1	83807
2	3	15619304	Onio	502	France	Female	42	8	159660
3	4	15701354	Boni	699	France	Female	39	1	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510

```
df.tail()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	
9996	9997	15569892	Johnstone	516	France	Male	35	10	5
9997	9998	15584532	Liu	709	France	Female	36	7	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	7
9999	10000	15628319	Walker	792	France	Female	28	4	13

```
df.shape
```

```
(10000, 14)
```

```
df.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

```
df.dtypes
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
df.select_dtypes(include=['int64','float64','Int64']).dtypes
```

```
RowNumber            int64
CustomerId           int64
CreditScore          int64
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

```
# custom function for easy and efficient analysis of numerical univariate
import matplotlib.pyplot as plt
```

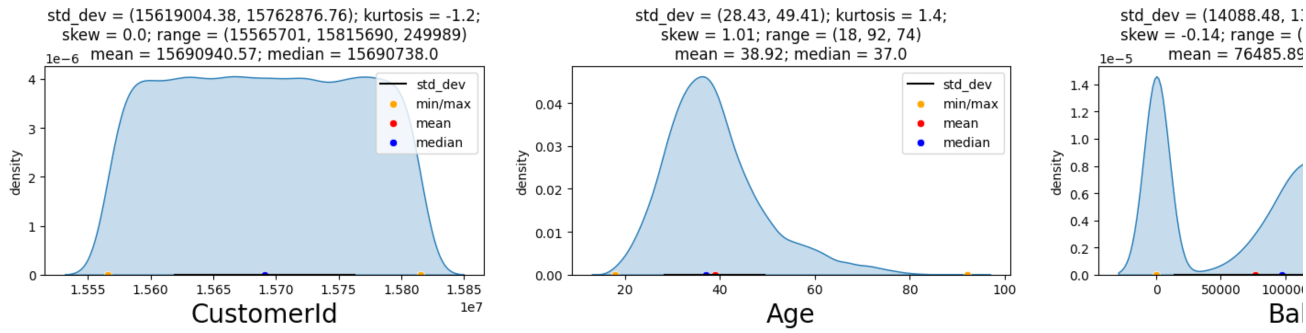
```
def UVA_numeric(data, var_group):
  '''
  Univariate_Analysis_numeric
  takes a group of variables (INTEGER and FLOAT) and plot/print all the descriptives and pro

  Runs a loop: calculate all the descriptives of i(th) variable and plot/print it
  '''

  size = len(var_group)
  plt.figure(figsize = (7*size,3), dpi = 100)

  #looping for each variable
  for j,i in enumerate(var_group):

    # calculating descriptives of variable
    mini = data[i].min()
    maxi = data[i].max()
    ran = data[i].max()-data[i].min()
    mean = data[i].mean()
    median = data[i].median()
    st_dev = data[i].std()
    skew = data[i].skew()
    kurt = data[i].kurtosis()

    # calculating points of standard deviation
    points = mean-st_dev, mean+st_dev

    #Plotting the variable with every information
    plt.subplot(1,size,j+1)
    sns.kdeplot(data[i], shade=True)
    sns.lineplot(points, [0,0], color = 'black', label = "std_dev")
    sns.scatterplot([mini,maxi], [0,0], color = 'orange', label = "min/max")
    sns.scatterplot([mean], [0], color = 'red', label = "mean")
    sns.scatterplot([median], [0], color = 'blue', label = "median")
    plt.xlabel('{}'.format(i), fontsize = 20)
    plt.ylabel('density')
```



```
}; median = {}'.
```

```
import seaborn as sns
customer_details = ['CustomerId','Age','Balance']
UVA_numeric(df,customer_details)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
```




```
#Univariate analysis outliers
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

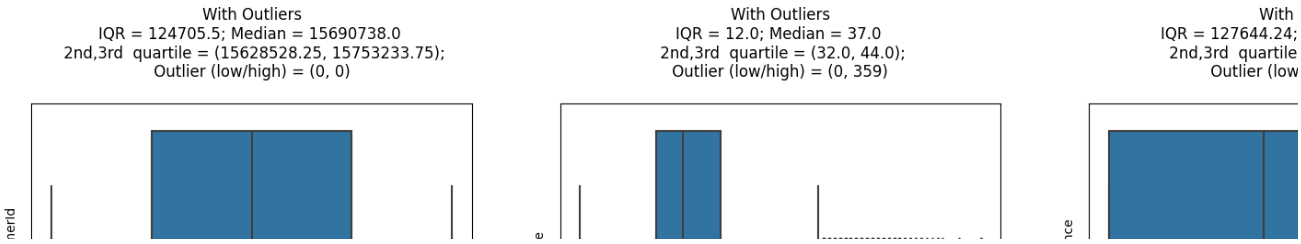
```
  takes a group of variables (INTEGER and FLOAT) and plot/print boplot and descriptives\n
  Runs a loop: calculate all the descriptives of i(th) variable and plot/print it \n\n

  data : dataframe from which to plot from\n
  var_group : {list} type Group of Continuous variables\n
  include_outlier : {bool} whether to include outliers or not, default = True\n
  '''

  size = len(var_group)
  plt.figure(figsize = (7*size,4), dpi = 100)

  #looping for each variable
  for j,i in enumerate(var_group):

    # calculating descriptives of variable
    quant25 = data[i].quantile(0.25)
    quant75 = data[i].quantile(0.75)
```

```
        IQR = quant75 - quant25
        med = data[i].median()
        whis_low = quant25-(1.5*IQR)
        whis_high = quant75+(1.5*IQR)

        # Calculating Number of Outliers
        outlier_high = len(data[i][data[i]>whis_high])
        outlier_low = len(data[i][data[i]<whis_low])

        if include_outlier == True:
          #Plotting the variable with every information
          plt.subplot(1,size,j+1)
          sns.boxplot(data[i], orient="v")
          plt.ylabel('{}'.format(i))
          plt.title('With Outliers\nIQR = {}; Median = {} \n 2nd,3rd  quartile = {};\n Outlier (









        else:
          # replacing outliers with max/min whisker
          data2 = data[var_group][:]
          data2[i][data2[i]>whis_high] = whis_high+1
          data2[i][data2[i]<whis_low] = whis_low-1

          # plotting without outliers
          plt.subplot(1,size,j+1)
          sns.boxplot(data2[i], orient="v")
          plt.ylabel('{}'.format(i))
          plt.title('Without Outliers\nIQR = {}; Median = {} \n 2nd,3rd  quartile = {};\n Outlie
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
UVA_outlier(df,customer_details,)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_core.py:1326: UserWarning: Vertical or:
  warnings.warn(single_var_warning.format("Vertical", "x"))
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_core.py:1326: UserWarning: Vertical or:
  warnings.warn(single_var_warning.format("Vertical", "x"))
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_core.py:1326: UserWarning: Vertical or:
  warnings.warn(single_var_warning.format("Vertical", "x"))
```




```
#find missing values
```

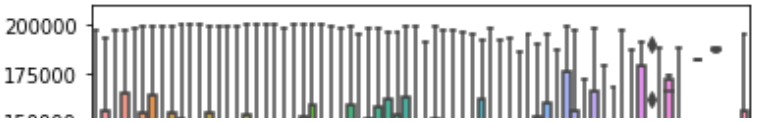
```
df.isnull().sum()
```

```
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
```

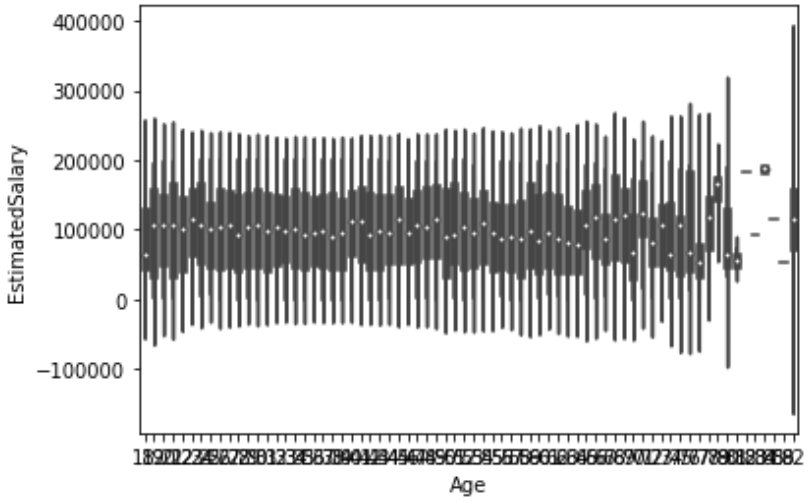


```
#bivariate analysis
```

```
sns.boxplot(x='Age',y='EstimatedSalary',data=df)
plt.show()
```

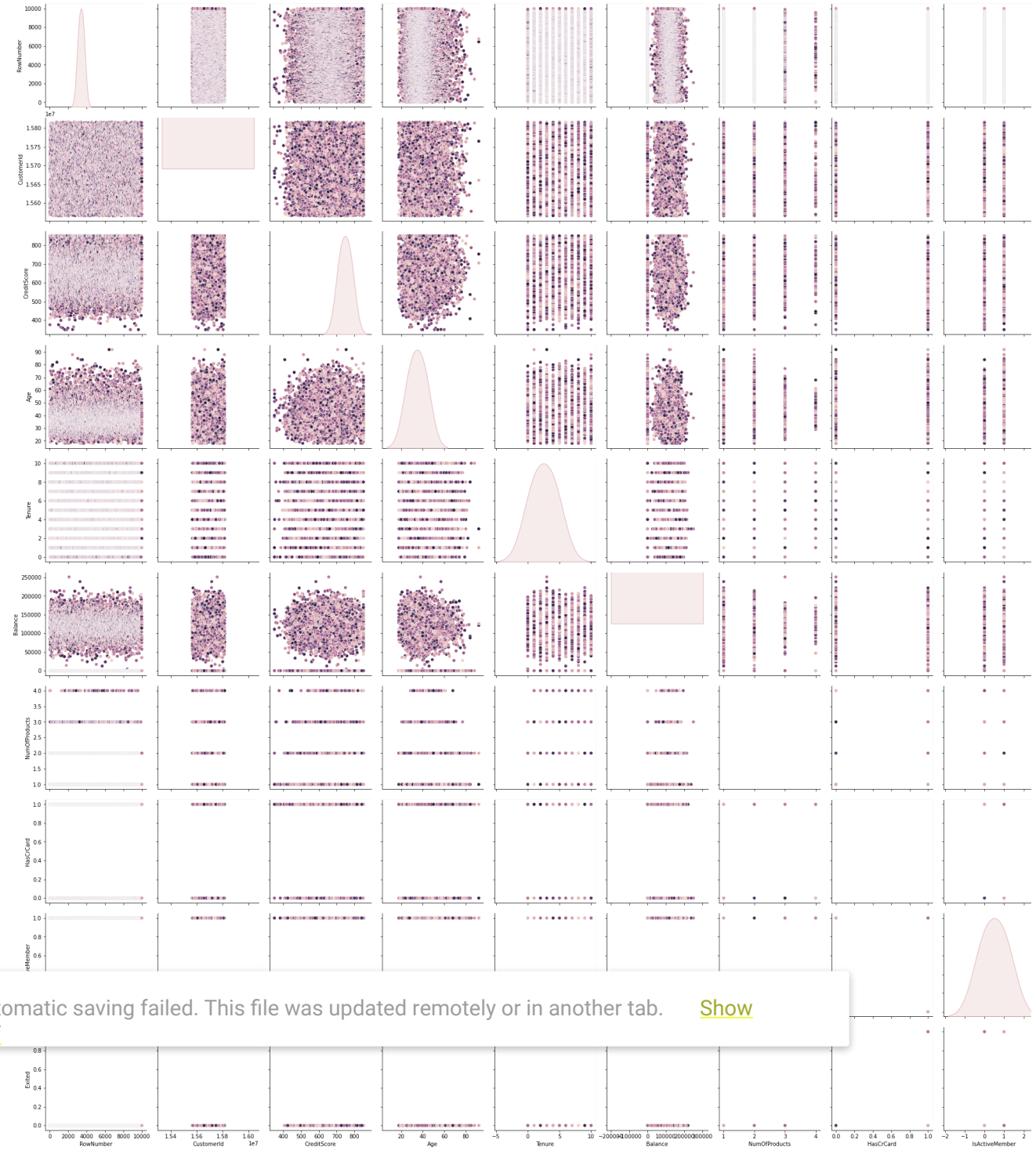



```
sns.violinplot(x='Age',y='EstimatedSalary',data=df,size = 8)
plt.show()
```




```
#multivariate analysis


sns.pairplot(hue='EstimatedSalary',data=df,size = 3)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:2076: UserWarning: The `size
  warnings.warn(msg, UserWarning)
```



Automatic saving failed. This file was updated remotely or in another tab. Show diff

detecting outliers

```
def detect_outlier(df):
 outlier = []
 threshold = 3
 mean = np.mean(df)
 std = np.std(df)
 for i in df:
```

```
    z_score = (i - mean)/std
    if np.abs(z_score)>threshold:
        outlier.append(i)
    return outlier
    CreditScore_list = df['CreditScore'].tolist()
    Balance_list = df['Balance'].tolist()
    EstimatedSalary_list = df['EstimatedSalary'].tolist()
    CreditScore_outlier = detect_outlier(CreditScore_list)
    CreditScore_outlier
    Output-[359, 350, 350, 358, 351, 350, 350, 350]
    Balance_outlier = detect_outlier(Balance_list)
    Balance_outlier
    EstimatedSalary_outlier = detect_outlier(EstimatedSalary_list)
    EstimatedSalary_outlier
```

```
# Shape of Data before removing the outliers
```

```
print("Shape of Data before removing outliers: {}".format(df.shape))
```

```
    Shape of Data before removing outliers: (10000, 14)
```

```
# Removing the outlier
```

```
df.drop(df[df['CreditScore'] <= 359].index, inplace = True)
```

```
#Shape of Data after removing the outliers
```

```
 print("Shape of Data after removing outliers: {}".format(df.shape))
```

```
    Shape of Data after removing outliers: (9992, 14)
```



```
def dataoveriew(df, message):
    print(f'{message}:\n')
    print('Number of rows: ', df.shape[0])
    print("\nNumber of features:", df.shape[1])
    print("\nData Features:")
    print(df.columns.tolist())
    print("\nMissing values:", df.isnull().sum().values.sum())
    print("\nUnique values:")
    print(df.nunique())

dataoveriew(df, 'Overview of the dataset')
```

```
    Overview of the dataset:

    Number of rows:  9992
```

```
Number of features: 14

Data Features:
['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Te

Missing values: 0

Unique values:
RowNumber          9992
CustomerId         9992
Surname            2931
CreditScore         456
Geography             3
Gender                2
Age                  70
Tenure               11
Balance            6376
NumOfProducts         4
HasCrCard             2
IsActiveMember        2
EstimatedSalary    9991
Exited                2
dtype: int64
```

```
import plotly.express as px
target_instance = df["Exited"].value_counts().to_frame()
target_instance = target_instance.reset_index()
target_instance = target_instance.rename(columns={'index': 'Category'})
fig = px.pie(target_instance, values='Exited', names='Category', color_discrete_sequence=["g
             title='Distribution of Churn')
fig.show()
```

Distribution of Churn

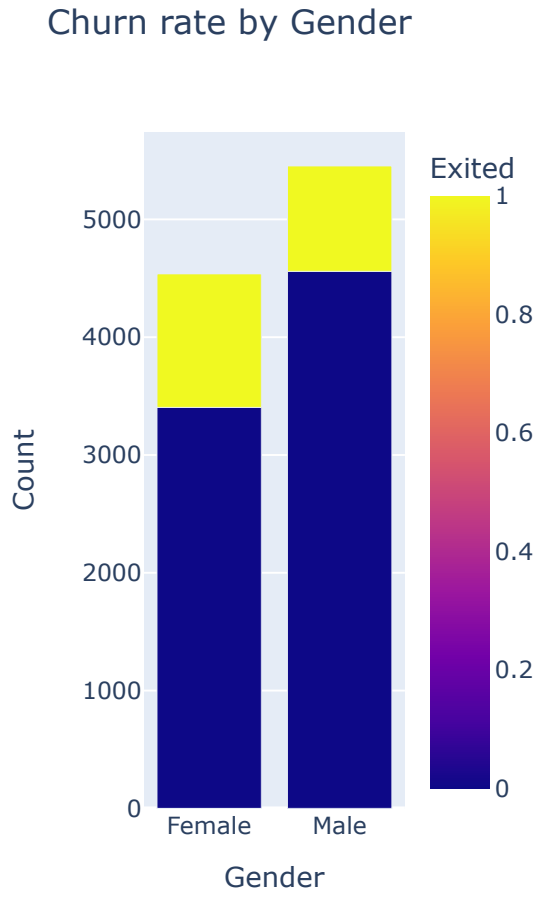
```
def bar(feature, df=df ):
    #Groupby the categorical feature
    temp_df = df.groupby([feature, 'Exited']).size().reset_index()
    temp_df = temp_df.rename(columns={0:'Count'})
    #Calculate the value counts of each distribution and it's corresponding Percentages
    value_counts_df = df[feature].value_counts().to_frame().reset_index()
    categories = [cat[1][0] for cat in value_counts_df.iterrows()]
    #Calculate the value counts of each distribution and it's corresponding Percentages
    num_list = [num[1][1] for num in value_counts_df.iterrows()]
    div_list = [element / sum(num_list) for element in num_list]
    percentage = [round(element * 100,1) for element in div_list]
    #Defining string formatting for graph annotation
    #Numeric section
    def num_format(list_instance):
        formatted_str = ''
        for index,num in enumerate(list_instance):
            if index < len(list_instance)-2:
                formatted_str=formatted_str+f'{num}%, ' #append to empty string(formatted_st
            elif index == len(list_instance)-2:
                formatted_str=formatted_str+f'{num}% & '
            else:
                formatted_str=formatted_str+f'{num}%'
        return formatted_str
    #Categorical section
    def str_format(list_instance):
        formatted_str = ''
        for index, cat in enumerate(list_instance):
            if index < len(list_instance)-2:
                formatted_str=formatted_str+f'{cat}, '
            elif index == len(list_instance)-2:
```



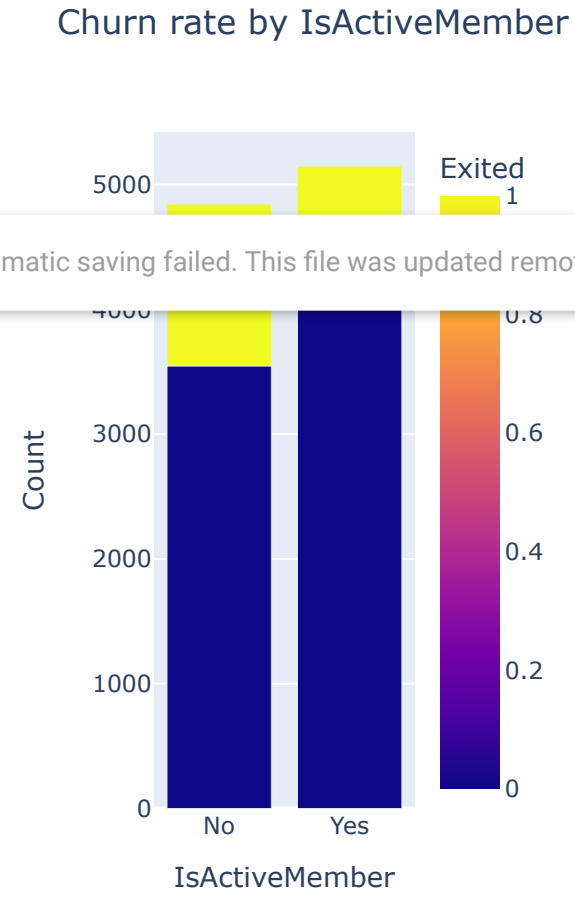
```
                formatted_str=formatted_str+f'{cat}'
        return formatted_str


    #Running the formatting functions
    num_str = num_format(percentage)
    cat_str = str_format(categories)

    #Setting graph framework
    fig = px.bar(temp_df, x=feature, y='Count', color='Exited', title=f'Churn rate by {featu
    fig.add_annotation(
                text=f'Value count of distribution of {cat_str} are<br>{num_str} percentage
                align='left',
                showarrow=False,
                xref='paper',
                yref='paper',
                x=1.4,
                y=1.3,
```

```
                bordercolor='black',
                borderwidth=1)
    fig.update_layout(
        # margin space for the annotations on the right
        margin=dict(r=400),
    )

    return fig.show()

#Gender feature plot
bar('Gender')
#IsActiveMember feature plot
df.loc[df.IsActiveMember==0,'IsActiveMember'] = "No"   #convert 0 to No in all data instance
df.loc[df.IsActiveMember==1,'IsActiveMember'] = "Yes"  #convert 1 to Yes in all data instanc
bar('IsActiveMember')
#Hascreditcard feature plot
bar('HasCrCard')
#Geography feature plot
bar('Geography')
```

e count of distribution of Male & Female are
,% & 45.4% percentage respectively.

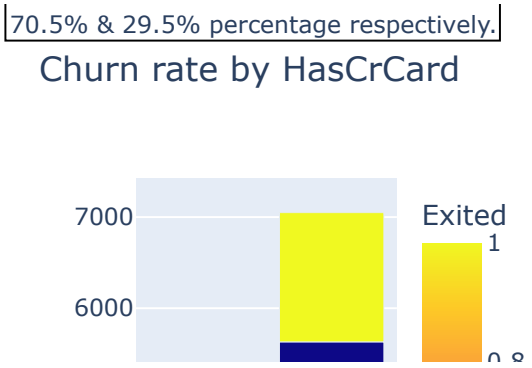


Value count of distribution of Yes & No are
51.5% & 48.5% percentage respectively.



Value count of distribution of 1 & 0 are

70.5% & 29.5% percentage respectively.




```
#Split the data into dependent and independent variables.


# Split the Dataset
X= df.drop(['Exited'], axis = 1)
y = df['Exited']



# Creating dummy variables
Dummies = pd.get_dummies(X[['Geography', 'Gender']],drop_first=True)
X = X.drop(['Geography', 'Gender'], axis = 1)
X = pd.concat([X, Dummies], axis = 1)



#Split the data into training and testing
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42


#Scale the independent variables


from sklearn.preprocessing import StandardScaler


# Feature Scaling using standard scaler
X_train = StandardScaler().fit_transform(X_train)
X_test = StandardScaler().transform(X_test)
```