Final Project: One-Week Project on Customer Dataset

Importing libraries:

```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```

Loading file:

```
In [ ]: cust_churn_df = pd.read_csv(r'C:\Users\User\PycharmProjects\PythonProject2\Exerc
        cust_churn_df
```

Content Dataset checks:

```
In [ ]: #Number of rows and columns
        total_rows = cust_churn_df.shape[0]
        print(f'Total number of rows in the dataset are: {total_rows}')
        print(' ')
        print('--------------------')
        total_columns = cust_churn_df.shape[1]
        print(f'Total number of columns in the dataset are: {total_columns}')
        print(' ')
        print('--------------------')
        #Column Names of DataFrame
        my_columns = cust_churn_df.columns
        print('\nThe columns of the DataFrame are:\n','\n',my_columns)
        print(' ')
        print('--------------------')
        #First 5 rows of DataFrame
        first_5_rows = cust_churn_df.head()
        print(first_5_rows)
        print(' ')
        print('--------------------')
        #Last 5 rows of DataFrame
        last_five_rows = cust_churn_df.tail()
        print(last_five_rows)
        print(' ')
        print('--------------------')
        #Key details of DataFrame
        df_info = cust_churn_df.info(show_counts=True)
        print(df_info)
        print(' ')
        print('--------------------')
        #Description of DataFrame
        df_describe = cust_churn_df.describe(include='all').round()
        print(df_describe)
        print(' ')
        print('--------------------')
```

Understanding the values inside our columns:

```
In [ ]: #Understanding values in Column Age
        unique_age_count = cust_churn_df['Age'].nunique()
        print('The unique values in Column Age are:',unique_age_count)
        print(' ')

        print('The split of values in column Age is formed:')
        print(cust_churn_df['Age'].value_counts())
        print(' ')

        sns.countplot(data=cust_churn_df,x='Age',color='lightblue')
        plt.title('Distribution of Age')
        plt.show()
```

In [ ]:
```
#Understanding values in Column Gender
unique_gender_count = cust_churn_df['Gender'].nunique()
print('The unique values in Column Gender are:',unique_gender_count)
print(' ')

print('The split of values in column Gender is formed:')
print(' ')
print(cust_churn_df['Gender'].value_counts())

#Fixing values for men to be all Male, and for women to be all Female
cust_churn_df['Gender'] = cust_churn_df['Gender'].str.replace('male','Male')
cust_churn_df['Gender'] = cust_churn_df['Gender'].str.replace('FeMale','Female')
cust_churn_df['Gender'] = cust_churn_df['Gender'].str.replace('FEMALE','Female')
cust_churn_df['Gender'] = cust_churn_df['Gender'].str.replace('feMale','Female')

print('The updated split of values in column Gender is formed:')
print(' ')
print(cust_churn_df['Gender'].value_counts())

sns.countplot(data=cust_churn_df,x='Gender',color='lightblue')
plt.title('Distribution of Gender')
plt.show()
```

In [ ]:
```
#Understanding values in Column MaritalStatus
unique_married_count = cust_churn_df['MaritalStatus'].nunique()
print('The unique values in Column MaritalStatus are:',unique_married_count)
print(' ')

print('The split of values in column MaritalStatus is formed:')
print(' ')
print(cust_churn_df['MaritalStatus'].value_counts())

sns.countplot(data=cust_churn_df,x='MaritalStatus',color='lightblue')
plt.title('Married Status')
plt.show()
```

In [ ]:
```
#Understanding values in Column Income
unique_income_count = cust_churn_df['Income'].nunique()
print('The unique values in Column Income are:',unique_income_count)
print(' ')

print('The split of values in column Income is formed:')
print(' ')
print(cust_churn_df['Income'].value_counts())

sns.countplot(data=cust_churn_df,x='Income',color='lightblue')
plt.title('Distribution of Income')
plt.show()
```

In [ ]:
```
#Understanding values in Column Savings
unique_savings_count = cust_churn_df['Savings'].nunique()
print('The unique values in Column Savings are:',unique_savings_count)
print(' ')

print('The split of values in column Savings is formed:')
print(' ')
print(cust_churn_df['Savings'].value_counts())

sns.countplot(data=cust_churn_df,x='Savings',color='lightblue')
plt.title('Distribution of Savings')
plt.show()
```

In [ ]:
```
plt.hist(cust_churn_df['Savings'],bins=30,color='lightblue')
plt.title('Distribution of Savings')
plt.xlabel('Savings')
plt.ylabel('Frequency')
plt.show()
```

In [ ]:
```
#Understanding values in Column CreditScore
unique_credit_score_count = cust_churn_df['CreditScore'].nunique()
print('The unique values in Column CreditScore are:',unique_credit_score_count)
print(' ')

print('The split of values in column CreditScore is formed:')
print(' ')
print(cust_churn_df['CreditScore'].value_counts())

sns.countplot(data=cust_churn_df,x='CreditScore',color='lightblue')
plt.title('Distribution of CreditScore')
plt.show()
```

In [ ]:
```
#Understanding values in Column LoanAmount
unique_loan_amount_count = cust_churn_df['LoanAmount'].nunique()
print('The unique values in Column LoanAmount are:',unique_loan_amount_count)
print(' ')

print('The split of values in column LoanAmount is formed:')
print(' ')
print(cust_churn_df['LoanAmount'].value_counts())

sns.countplot(data=cust_churn_df,x='LoanAmount',color='lightblue')
plt.title('Distribution of Loan Amount')
plt.show()
```

In [ ]:
```
#Understanding values in Column LoanStatus
unique_loan_status_count = cust_churn_df['LoanStatus'].nunique()
print('The unique values in Column LoanStatus are:',unique_loan_status_count)
print(' ')

print('The split of values in column LoanStatus is formed:')
print(' ')
print(cust_churn_df['LoanStatus'].value_counts())

sns.countplot(data=cust_churn_df,x='LoanStatus',color='lightblue')
plt.title('Distribution of Loan Status')
plt.show()
```

In [ ]:
```
#Understanding values in Column AccountType
unique_account_type_count = cust_churn_df['AccountType'].nunique()
print('The unique values in Column AccountType are:',unique_account_type_count)
print(' ')

print('The split of values in column AccountType is formed:')
print(' ')
print(cust_churn_df['AccountType'].value_counts())

sns.countplot(data=cust_churn_df,x='AccountType',color='lightblue')
plt.title('Distribution of Account Types')
plt.show()
```

In [ ]:
```
#Understanding values in Column YearsWithBank
unique_years_count = cust_churn_df['YearsWithBank'].nunique()
print('The unique values in Column YearsWithBank are:',unique_years_count)
print(' ')

print('The split of values in column YearsWithBank is formed:')
print(' ')
print(cust_churn_df['YearsWithBank'].value_counts())

sns.countplot(data=cust_churn_df,x='YearsWithBank',color='lightblue')
plt.title('Distribution of Years With the Bank')
plt.show()

#YearsWithBank with value 100 seems to be an outlier. Will check later.
```

In [ ]:
```
#Understanding values in Column Churn
unique_churn_count = cust_churn_df['Churn'].nunique()
print('The unique values in Column Churn are:',unique_churn_count)
print(' ')

print('The split of values in column Churn is formed:')
print(' ')
print(cust_churn_df['Churn'].value_counts())

sns.countplot(data=cust_churn_df,x='Churn',color='lightblue')
plt.title('Distribution of Churn')
plt.show()
```

Duplicate rows Check:

In [ ]:
```
#Duplicate rows check in our DataFrame
print(cust_churn_df.duplicated())
duplicated_rows_count = cust_churn_df.duplicated().sum()
print(f'Total number of duplicated rows in the dataset are: {duplicated_rows_cou
print(' ')

#Finding from below output is that we have no duplicate rows in our DataFrame
```

NaN values Check:

In [ ]:
```
df_NaN = cust_churn_df.isna().sum()
print(' ')
print('The analysis for NaN values is:')
print(df_NaN)
```

Handling NaN values for below columns :

- Age : 162 & 5% from total values of col Age
- Income : 135 & 5% from total values of col Income
- Savings : 146 & 5% from total values of col Savings
- CreditScore : 143 & 5% from total values of col CreditScore
- LoanAmount : 133 & 4% from total values of col LoanAmount
- YearsWithBank : 287 & 10% from total values of col YearsWithBank

In [ ]:
```
#Finding the index of rows where column Age is NaN.
NaN_rows_Age = cust_churn_df[cust_churn_df['Age'].isna()].index
print(NaN_rows_Age)
print(' ')

cust_churn_df[cust_churn_df['Age'].isna()]
```

In [ ]:
```
#Finding the index of rows where column Income is NaN.
NaN_rows_Income = cust_churn_df[cust_churn_df['Income'].isna()].index
print(NaN_rows_Income)
print(' ')
cust_churn_df[cust_churn_df['Income'].isna()]
```

In [ ]:
```
#Finding the index of rows where column Savings is NaN.
NaN_rows_Savings = cust_churn_df[cust_churn_df['Savings'].isna()].index
print(NaN_rows_Savings)
print(' ')
cust_churn_df[cust_churn_df['Savings'].isna()]
```

In [ ]:
```
#Finding the index of rows where column CreditScore is NaN.
NaN_rows_CreditScore = cust_churn_df[cust_churn_df['CreditScore'].isna()].index
print(NaN_rows_CreditScore)
print(' ')
cust_churn_df[cust_churn_df['CreditScore'].isna()]
```

In [ ]:
```
#Finding the index of rows where column LoanAmount is NaN.
NaN_rows_LoanAmount = cust_churn_df[cust_churn_df['LoanAmount'].isna()].index
print(NaN_rows_LoanAmount)
print(' ')
cust_churn_df[cust_churn_df['LoanAmount'].isna()]
```

In [ ]:
```
#Finding the index of rows where column YearsWithBank is NaN.
NaN_rows_years = cust_churn_df[cust_churn_df['YearsWithBank'].isna()].index
print(NaN_rows_years)
print(' ')
cust_churn_df[cust_churn_df['YearsWithBank'].isna()]
```

In [ ]:
```
#creating a subset of columns to fill with mean
columns_to_fill=['Age','Income','Savings','CreditScore','LoanAmount','YearsWithB

#Calculating Average of columns & filling with median
for column in columns_to_fill:
    column_median=int(cust_churn_df[column].median())
    print(f"Median for {column}: {column_median}")
    cust_churn_df[column]=cust_churn_df[column].fillna(column_median)
print(' ')
```

```
#Checking that we no longer have NaN values in any of our columns that needed to
print('Current columns with NaN are:')
print(cust_churn_df[columns_to_fill].isna().sum())
print(' ')

#Checking sample indexes -found previously- for column Age to be NaN, to now be
print('Previously NaN index values of column Age are now set as mean check. Mean
print(cust_churn_df.loc[[35,50,55,85,91,113,119],'Age'])
print(' ')

#Checking sample indexes -found previously- for column Income to be NaN, to now
print('Previously NaN index values of column Income are now set as mean check. M
print(cust_churn_df.loc[[26,53,69,82,106,135,174],'Income'])
print(' ')


#Checking sample indexes -found previously- for column Savings to be NaN, to now
print('Previously NaN index values of column Savings are now set as mean check.
print(cust_churn_df.loc[[84,121,144,169,177,216],'Savings'])
print(' ')

#Checking sample indexes -found previously- for column CreditScore to be NaN, to
print('Previously NaN index values of column CreditScore are now set as mean che
print(cust_churn_df.loc[[9,16,39,98,99,106,135],'CreditScore'])
print(' ')

#Checking sample indexes -found previously- for column LoanAmount to be NaN, to
print('Previously NaN index values of column LoanAmount are now set as mean chec
print(cust_churn_df.loc[[45,48,166,177,185,188],'LoanAmount'])
print(' ')

#Checking sample indexes -found previously- for column YearsWithBank to be NaN,
print('Previously NaN index values of column YearsWithBank are now set as mean c
print(cust_churn_df.loc[[0,3,13,14,17,33],'YearsWithBank'])
print(' ')
```

Cleaning DataFrame

In [ ]:
```
#Dropping reduntant column CustomerID
cust_churn_df = cust_churn_df.drop('CustomerID', axis=1)

#Main columns for Churn as Target Column
main_columns_for_churn = ['LoanStatus','YearsWithBank','AccountType','Income','S
                          'LoanAmount','Age','Gender','MaritalStatus']

#Checking that CustomerID is Dropped
cust_churn_df.head()
```

Checking for Insights and for Outliers

In [ ]:
```
#Target Column is column Churn
#Checking current statistics of the updated DataFrame numerical columns
cust_churn_df.describe().T
```

Insights found for numerical columns from .describe( ):

1. Age of customers to have min 25 and max 55 seems normal. (We don't see any outlier for example Age=3).
2. YearsWithBank min=1 is considered normal if we understand that this refers to new customers.
3. For YearsWithBank column we see that the percentile of 75% is 10 years.
4. YearsWithBank max=100 seems an outlier value.
5. We see that CreditScore column mean and median are almost identical, meaning that customers are distributed close to score 700.

In [ ]:
```
#Target Column is column Churn
#Main categorical columns that may affect the Target column are: Age,Income,Savi
categorical_cols=['Gender','MaritalStatus','LoanStatus','AccountType','Churn']
```

In [ ]:
```
#Checking current statistics of the updated DataFrame categorical columns
cust_churn_df[categorical_cols].describe().T
```

Insights found for categorical columns from describe( ) and with previous value_counts checks:

1. Gender customers distribution is almost equal. Female customers are slightly more.
2. Most customers are single & married. Single are slightly more than married customers. A few customers are widowed and divorced. Widowed are slightly more than divorced customers.
3. Most loan status are approved (60%). Approximately half dynamic of the approved, have the rejected ones(31%) and follow at last a few defaulted ones(9%).
4. Majority of Account Types are savings accounts (41%), checking accounts follow(29%), investment(20%) & joint ones(10%).
5. Majority of customers haven't churned (70%).

In [ ]:
```
#Checking if we have Outliers of column Age that we havn't understand from above
plt.figure(figsize=(4,5))
sns.boxplot(y=cust_churn_df['Age'], medianprops={'color': 'black', 'linewidth':
plt.title('Distribution of Customer Age')
plt.ylabel('Age')
plt.show()

#We see no outliers in column Age
```

In [ ]:
```
#Checking if we have Outliers of column YearsWithBank
plt.figure(figsize=(4,5))
sns.boxplot(y=cust_churn_df['YearsWithBank'], medianprops={'color': 'black', 'li
plt.title('Distribution of Customer Years within the Bank')
plt.ylabel('YearsWithBank')
plt.show()
```

In [ ]:
```
#Creating a list for all values of column YearsWithBank
years_within_bank = cust_churn_df['YearsWithBank'].to_list()

#Splitting YearsWithBank values in percentiles
Q1 = np.percentile(years_within_bank,25,interpolation='midpoint')
Q2 = np.percentile(years_within_bank,50,interpolation='midpoint')
Q3 = np.percentile(years_within_bank,75,interpolation='midpoint')
```

```
#Calculating the Interquartile Range
IQR=Q3-Q1

print('Old YearsWithBank Outliers are:')
print('Minimum:', Q1 - 1.5 * IQR)  #lower outlier fence
print('Maximum:', Q3 + 1.5 * IQR)  #maximum outlier fence
```

In [ ]:
```
#Filling outliers list with values that are considered above max YearsWithBank v
#expected to see all values thus >22 age and <-10 years
outliers_years_within_bank = []
for i in years_within_bank:
    if (i>Q3 + 1.5 * IQR) or (i< Q1 - 1.5 * IQR):
        outliers_years_within_bank.append(i)
outliers_years_within_bank
```

In [ ]:
```
#Finding rows with outliers values in column YearsWithBank
cust_churn_df[cust_churn_df['YearsWithBank'].isin(outliers_years_within_bank)]
```

In [ ]:
```
#Dropping row with outlier value in column YearsWithBank
cust_churn_df=cust_churn_df[~ cust_churn_df['YearsWithBank'].isin(outliers_years
```

In [ ]:
```
#Checking updated shape of DataFrame (we had (3000,11) should be (2999,22) as we
cust_churn_df.shape
```

In [ ]:
```
#Checking if we have Outliers of column CreditScore
plt.figure(figsize=(4,5))
sns.boxplot(y=cust_churn_df['CreditScore'], medianprops={'color': 'black', 'line
plt.title('Distribution of Customer Credit Score')
plt.ylabel('CreditScore')
plt.show()

#We see no outliers in column CreditScore
```

In [ ]:
```
#Checking if we have Outliers of column Income
plt.figure(figsize=(4,5))
sns.boxplot(y=cust_churn_df['Income'], medianprops={'color': 'black', 'linewidth
plt.title('Distribution of Customer Income')
plt.ylabel('Income')
plt.show()

#We see no outliers in column Income
```

In [ ]:
```
#Checking if we have Outliers of column Savings
plt.figure(figsize=(4,5))
sns.boxplot(y=cust_churn_df['Savings'], medianprops={'color': 'black', 'linewidt
plt.title('Distribution of Customer Savings')
plt.ylabel('Savings')
plt.show()
```

In [ ]:
```
#Creating a list for all values of column Savings
savings = cust_churn_df['Savings'].to_list()

#Splitting Savings values in percentiles
Q1 = np.percentile(savings,25,interpolation='midpoint')
Q2 = np.percentile(savings,50,interpolation='midpoint')
Q3 = np.percentile(savings,75,interpolation='midpoint')

#Calculating the Interquartile Range
```

```
IQR=Q3-Q1

print('Old Savings Outliers are:')
print('Minimum:', Q1 - 1.5 * IQR) #lower outlier fence
print('Maximum:', Q3 + 1.5 * IQR) #maximum outlier fence
```

In [ ]:
```
#Filling outliers list with values that are considered above max Savings values
#expected to see all values thus >35000 age and <-5000 years
outliers_savings = []
for i in savings:
    if (i>Q3 + 1.5 * IQR) or (i< Q1 - 1.5 * IQR):
        outliers_savings.append(i)
```

In [ ]:
```
#Finding rows with outliers values in column Savings
cust_churn_df[cust_churn_df['Savings'].isin(outliers_savings)]

#Outliers of Savings appear to be approx a 10% of all dataset, choosing to leave
```

In [ ]:
```
#Reseting index of DataFrame
cust_churn_df.reset_index(drop=True,inplace=True)
cust_churn_df
```

In [ ]:
```
#Remembering what type of data are in the DataFrame Columns
print(cust_churn_df.dtypes)
print(' ')
```

Comparing the important columns with the target column Churn

In [ ]:
```
#Age vs Churn

sns.boxplot(x='Churn',y='Age',color='lightblue',data=cust_churn_df)
plt.title('Age vs Churn')
plt.show()

#Insights:
#Age has no clear impact on whether a customer churns as two boxplots are nearly
```

In [ ]:
```
#Churn Rate by Gender

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='Gender',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Gender')
plt.show()

#Insights:
#Churn rate both for 'churn=yes' and 'churn=no' are almost identical for both Fe
```

In [ ]:
```
#Churn Rate by MaritalStatus

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='MaritalStatus',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Marital Status')
plt.show()

#Insights
#The churn ratio of the 'churn=yes' and 'churn=no'—is very similar proportionall
#For Single: The 'churn=yes' bar is about ~40-45%) of the 'churn=no' bar.
#For Married: The churn ratio is almost the same as the "single" group.
```

In [ ]:

```
#Churn Rate by LoanStatus

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='LoanStatus',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Loan Status')
plt.show()

#Insights
#The churn rate of the "churn=yes" bar to the "churn=no" bar is high and similar
#A large fraction (about 40-45%) of Approved loan status customers churned.
#The Defaulted loan status customers have a very high churn rate (looks like ~50
#The Rejected loan status customers a very high churn rate (about 40-45%).
```

In [ ]:

```
#Churn Rate by AccountType

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='AccountType',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Account Type')
plt.show()

#Insights
#The churn rate of the "churn=yes" bar to the "churn=no" bar is very similar acr
#Joint: A high percentage of these customers churned (the "churn=yes" bar is a b
```

In [ ]:

```
#Churn Rate by Years with the Bank
```

```
color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='YearsWithBank',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Years with the Bank')
plt.show()

#Insights:
#1. Customers with the longest years in bank (20 years) have the highest churn r
#2. For groups from 1 to 10 years, the black "churn=yes" bar is significantly sm
#3. For the 20 years group, the "churn=yes" bar is more than half the "churn=no"
```

In [ ]:

```
#Churn Rate by Income

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='Income',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Income')
plt.show()

#Insights:
#The ratio of "churn=yes" bar to the "churn=no" bar is very similar across all f
#regardless of their income bracket, are churning at a similarly high rate (arou
```

In [ ]:

```
#Churn Rate by Savings

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='Savings',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Savings')
plt.show()

#Insights
#All customers, regardless of their savings bracket, appear to churn at a simila
```

In [ ]:

```
#Churn Rate by LoanAmount

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='LoanAmount',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Loan Amount')
plt.show()

#Insights
#Customers regardless of their loan amount, are churning at a similarly high rat
```

In [ ]:
```
#Churn Rate by CreditScore

color_palette = {
    'no': 'lightblue',
    'yes': 'black' }

sns.countplot(
    data=cust_churn_df,
    x='CreditScore',
    hue='Churn',
    palette=color_palette
)
plt.title('Churn Rate by Credit Score')
plt.show()

#Insights
#The proportion of customers who churn appear to be consistent, whether they hav
```

In [ ]:
```
#Calculating the exact percentage of churn for each numerical non Target columns
print((cust_churn_df.groupby('Age')['Churn'].value_counts(normalize=True) * 100)
print(' ')
print('--------------------')
print((cust_churn_df.groupby('Income')['Churn'].value_counts(normalize=True) * 1
print(' ')
print('--------------------')
print((cust_churn_df.groupby('Savings')['Churn'].value_counts(normalize=True) *
print(' ')
print('--------------------')
print((cust_churn_df.groupby('CreditScore')['Churn'].value_counts(normalize=True
print(' ')
print('--------------------')
print((cust_churn_df.groupby('LoanAmount')['Churn'].value_counts(normalize=True)
print(' ')
print('--------------------')
print((cust_churn_df.groupby('YearsWithBank')['Churn'].value_counts(normalize=Tr
print(' ')
print('--------------------')
```

In [ ]:
```
#Calculating the exact percentage of churn for each categorical columns.
print((cust_churn_df.groupby('Gender')['Churn'].value_counts(normalize=True) * 1
print(' ')
print('--------------------')
print((cust_churn_df.groupby('MaritalStatus')['Churn'].value_counts(normalize=Tr
print(' ')
print('--------------------')
print((cust_churn_df.groupby('LoanStatus')['Churn'].value_counts(normalize=True)
print(' ')
print('--------------------')
print((cust_churn_df.groupby('AccountType')['Churn'].value_counts(normalize=True
print(' ')
```

In [ ]:
```
#Making a copy of our DataFrame to a new one to adjust in new one values from Ch
cust_churn_df_corr = cust_churn_df.copy()
cust_churn_df_corr['Churn'] = cust_churn_df_corr['Churn'].map({'no': 0, 'yes': 1
```

In [ ]:
```
numerical_cols = ['Age', 'Income', 'Savings', 'CreditScore', 'LoanAmount', 'Year
corr_matrix = cust_churn_df_corr[numerical_cols].corr()

# Plotting a heatmap to check if our previous found insights are correct.
plt.figure(figsize=(6, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

Dropping more Columns per newest above findings

In [ ]:
```
#1. Dropping Column Gender

#As for both Female and Male genders 'churn rate=no' is 70 & 'churn rate=yes' is
#in our Target column if it a Male or a Female, thus decision is dropping the co
cust_churn_df = cust_churn_df.drop('Gender', axis=1)

#2. Dropping Column AccountType
#Between all types of accounts for 'churn rate=yes' values there seems to be a d
#'churn rate=yes':
#checking->29% ,investment->30%, joint->28%, savings->32%
cust_churn_df = cust_churn_df.drop('AccountType', axis=1)

#4. Dropping Column LoanAmount & Column Income
#By comparing -money based- groups against each other we understand:
#Group 1 (0-40k) -> Poor Group : 30% of them churned per loan amount and 30% of
#Group 2 (40.01k-100k) ->Rich Group : 31% of them churned per loan amount and 32
# =>The Rich Group behaved exactly the same way as the Poor Group. Thus decision
cust_churn_df = cust_churn_df.drop('LoanAmount', axis=1)
cust_churn_df = cust_churn_df.drop('Income', axis=1)

#5. Dropping Column CreditScore
#Between all types of accounts for 'churn rate=yes' values there seems to be a d
#'churn rate=yes':
#600->28% ,650->31%, 700->29%, 750->32% , 800->32%
cust_churn_df = cust_churn_df.drop('CreditScore', axis=1)

#6. Dropping Column Savings examination
#Due to Outliers found earlier -that were not dropped from Dataset- it is decide
```

In [ ]:
```
#Making another copy of our DataFrame to a newest one to adjust in new one value
cust_churn_df_corr_02 = cust_churn_df.copy()
cust_churn_df_corr_02['Churn'] = cust_churn_df_corr_02['Churn'].map({'no': 0, 'y
```

In [ ]:
```
numerical_cols_final = ['Age', 'Savings', 'YearsWithBank', 'Churn']
corr_matrix = cust_churn_df_corr_02[numerical_cols_final].corr()

# Plotting a heatmap to check if our previous found insights are correct.
```

```
plt.figure(figsize=(6, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

Conclusions

1. Column CustomerID is reduntant in regards to our target column Churn, and thus removed from dataset.

2. Age column doesn't have any clear impact in our target column Churn but has a relevance with columns YearsWithBank & Savings.

   - Age of customers to have min 25 and max 55 seems normal. We don't see for example any outlier Age=3 or Age=130.

3. Gender column doesn't have any impact in target column Churn.

   - Gender customers distribution is almost equal. Female customers are slightly more (1521 & 50,7%) than Male customers (1479 & 49,3%) by 1,4%.
   - Churn rate ratio for both 'churn=yes' and 'churn=no' for both Female & Male are nearly the same. So, column is
   - not needed for Target Column Churn, and thus removed.

4. Marital Status column appears to have a small impact on whether a customer churns.

   - Most customers are single (40,5%) & married (40,4%). Single are slightly more than married customers by 0,1%.
   - A few customers are widowed (10,03%) and divorced (9%).
   - Widowed are slightly more than divorced customers by 1,03%.
   - Singles churn ratio is about 40-45%. Married churn ratio appears very close to singles churn ratio from Countplot.
   - Churn rate ratio for both 'churn=yes' and 'churn=no'for all 4 groups appears in general similar. It appers to be a 5% gap between Divorced/Single in the churn rate ratio for "churn=yes".

5. Loan Status column doesn't appear to have a clear impact in our target column Churn.

   - Most loan status are approved (60%). Approximately half dynamic of the approved, have the rejected ones(31%) and follow at last a few defaulted ones(9%).
   - Churn rate ratio for both 'churn=yes' and 'churn=no' for all 4 groups in Countplot are very similar across all three groups.
   - Approved: A large part of these customers churned (around 40-45%).
   - Rejected: This group also has a high churn rate (around 40-45%).
   - Defaulted: This group also has a high churn rate (around 50%).
   - Customers of Approved loan status are leaving at nearly the same high rate as Defaulted and Rejected.

6. Account Type appears do not have a significant impact on whether a customer churns - as between all types of accounts for 'churn rate=yes' values there seems to

be a deviation +-2% -, and this is the reason why this column is dropped from dataset.

- The majority of Account Types are savings accounts (41%). Next follow checking accounts (29%), investment (20%) & joint (10%).

7. YearsWithBank appears to be the best predictor for target column Churn.

   - YearsWithBank min=1 is considered normal as this refers to new customers.
   - Percentile of 75% is 10 years, meaning 75% of the customers have been with the bank for 10 years or less.
   - There was a value equal to 100, which was an outlier and such row was dropped from the dataset.
   - Customers with the longest years within the bank (20 years), have the highest churn rate. This means that they may feel neglected or that their long-term needs are not being met by the bank (Countplot insight).
   - For the 20 years group, the customers that churned are 37% and the customers that didn't churn 63%. Meaning 58,7% from customers of the 20 years group churned.

8. Income column doesn't have any visible impact in our target column Churn and this is the reason why this column is dropped from dataset.

   - The ratio of the 'churn=yes' bar to the 'churn=no' bar— appears in Countplot similar across all four income groups.
   - Customers, regardless of their income group -Poor or Rich- are churning at a similar rate (Poor 30% and Rich 32%).

9. Savings column doesn't appear to have a visible clear impact in our target column Churn but due to the outliers found 304 rows that werte not dropped, this column is kept in the dataset.

   - More specifically, savings column has 304 rows appearing outliers, representing about 10% of Dataset.
   - Customers, regardless of their savings group, per Counterplot are churning at a similarly high rate.

10. Loan Amount column doesn't have any impact in our target column Churn, thus it is decided to be removed from Dataset.

    - Customers regardless of their loan amount, apper to churn at a similarly high rate.

11. Credit Score column doesn't have any visible impact in our target column Churn, thus it is decided to be removed from Dataset.

    - CreditScore column mean and median are almost identical, meaning that customers are distributed close to score 700.
    - The proportion of customers who churn appears to be consistent, whether they have a 600 or 800 credit score.

12. Majority of customers that haven't churned represent the 70%, while customers that churned represent the 30%.

In [ ]:
```
#Saving the updated Dataset after all corrections done in our original DataFrame
cust_churn_df.to_csv('cleaned_bank_data_cust_churn.csv', index=False)
```

In [ ]:
```
#Reading DataFrame saved before adjusting values of Churn column for heat map to
cust_churn_df
```