Customer Data Analytics and Churn Predictions for ABC Multinational Bank

UCD x FIL Data Analytics Certificate Program

Eugenia Fang - November 3, 2022

GitHub URL: https://github.com/EugenlilBox/UCDPA_EugeniaFang

Dataset

This is dataset contains account holders at ABC Multinational Bank, available on Kaggle (https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset)

Abstract

(Short overview of the entire project and features)

Introduction

For every business customer retention plays the most important role in business sustainability. It is a fact that to acquire new customers cost more than retaining existing customers. According to studies from HubSpot, generally, increasing the customer retention rate by 5% will easily resulted at least 25% increase in profit. Because returning customers will likely spends 67% more. Additionally, speaking from my own experience with a retail banking digital acquisition background, the time and money on convincing new customers is very expensive.

The dataset I selected from Kaggle contains a set of account holders information at ABC Multinational Bank, and can be used to predict the customer churn.

Implementation Process

1. Importing data
   - Import a CSV file into a Pandas DataFrame (10) (Illastration: py_cd_1)
     - Import packages for data analyzing
     - Import CSV file into a Panda DataFrame
     - Examine the shape of the imported data frame

```
In [1]: #import packages for analyzing data
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np

In [2]: #import the csv data file, and understand the shape of the dataset
        data = pd.read_csv('Bank Customer Churn Prediction.csv')
        data.shape

Out[2]: (10000, 12)
```

Illastration: py_cd_1

2. Analyzing data
   - Understand the data structure, data types of each column, and exam whether there are any missing values for data cleansing. (Illastration: py_cd_2 - 3)
     - data.head()
     - data.columns
     - data.info()
     - data.isnull().sum()

- data.nunique()

```
In [5]: #understand the data types
        data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customer_id       10000 non-null  int64  
 1   credit_score      10000 non-null  int64  
 2   country           10000 non-null  object 
 3   gender            10000 non-null  object 
 4   age               10000 non-null  int64  
 5   tenure            10000 non-null  int64  
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64  
 8   credit_card       10000 non-null  int64  
 9   active_member     10000 non-null  int64  
 10  estimated_salary  10000 non-null  float64
 11  churn             10000 non-null  int64  
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

Illastration: py_cd_2

```
In [6]: #look for any null or missing value in the dataset
        data.isnull().sum()
```

```
Out[6]: customer_id         0
        credit_score        0
        country             0
        gender              0
        age                 0
        tenure              0
        balance             0
        products_number     0
        credit_card         0
        active_member       0
        estimated_salary    0
        churn               0
        dtype: int64
```

```
In [7]: #drop missing value in the data frame that either rows or columns has at least one element is missing
        data.dropna()
        data.dropna(axis='columns')
        #understand unique values in the dataset
        data.nunique()
```

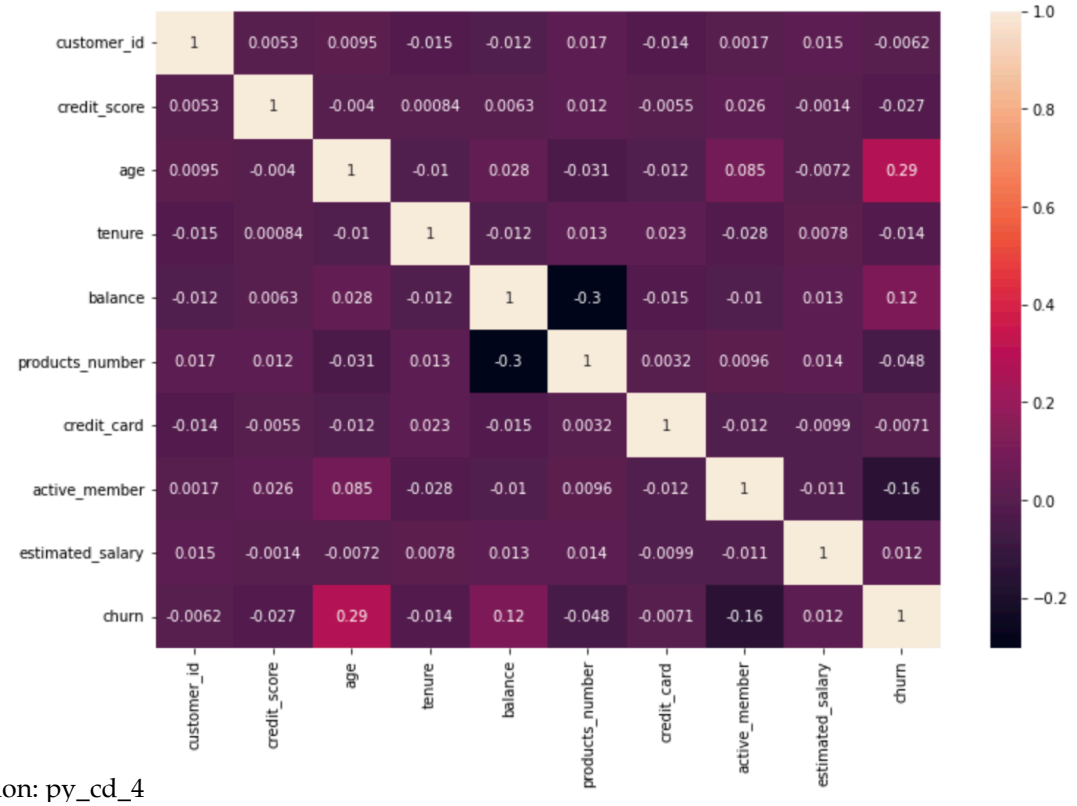
```
Out[7]: customer_id         10000
        credit_score          460
        country                 3
        gender                  2
        age                    70
        tenure                 11
        balance              6382
        products_number         4
        credit_card             2
        active_member           2
        estimated_salary     9999
        churn                   2
        dtype: int64
```

Illastration: py_cd_3

After the first glance of the data frame, we can easily observe that there are total 12 columns and 1,000 row of records in the dataset, with no missing value in the dataset. However, I still performed the dropna() function for double insure the data we are going to exam further is clean and with no missing values. [Demonstrated: Replace missing values or drop duplicates (10)] Also, it is easy to discover that there are 8 columns has the integer data type, 2 columns assigned float data type, and 2 columns came with object data type. For column: "country", "gender", "tenure", "products_number", "credit_card", "active_member", and "Churn" can be potentially turn into categorical data type for further data exploratory analysis. In addition, it is easy to discover that there are 70 unique values in the age column, which potentially can be grouped into age ranges, need to run distribution analysis to determine the size of the range

- Exploratory Data Analysis with Visualizing charts - [Demonstrated - two charts with Seaborn or Matplotlib (20)]
  - Run an Seaborn in heat map to uncover any correlation between 2 variables in the data frame. If considering 0.8 as heavy correlation [Demonstrated - insight 1], our dataset does not has an clear correlation between any two items in the dataset, will need to perform further exploring to uncover insights (Illastration: py_cd_4)

```
In [8]: # run initiating corelation heatmap to check any corelation exist between variables in the data frame
        plt.figure(figsize=(12,8))
        sns.heatmap(data.corr(), annot=True)
        plt.show()
```



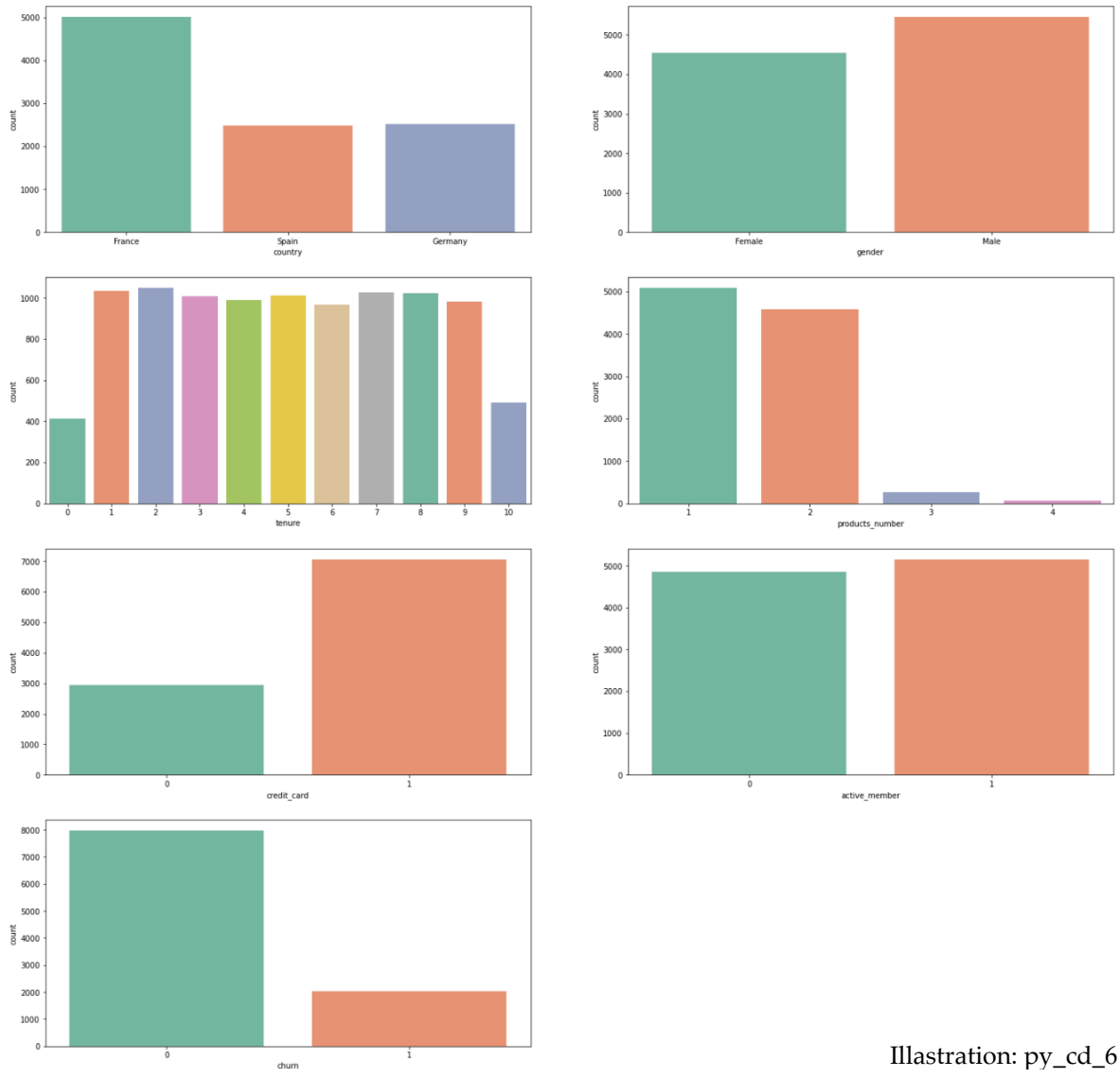
Illastration: py_cd_4

- visualizing the distributions based on the 7 potential categorical data type we have define in the previous step. (Illastration: py_cd_5 - 6)

```
In [10]: # visualizing the distributions based in the 7 potential categorical data type
         plt.figure(figsize = (25,25))

         plt.subplot(4,2,1)
         sns.countplot(x = 'country', palette='Set2', data = data, label = 'Country')

         plt.subplot(4,2,2)
         sns.countplot(x = 'gender', palette='Set2', data = data)

         plt.subplot(4,2,3)
         sns.countplot(x = 'tenure', palette='Set2', data = data)

         plt.subplot(4,2,4)
         sns.countplot(x = 'products_number', palette='Set2', data = data)

         plt.subplot(4,2,5)
         sns.countplot(x = 'credit_card', palette='Set2', data = data)

         plt.subplot(4,2,6)
         sns.countplot(x = 'active_member', palette='Set2', data = data)

         plt.subplot(4,2,7)
         sns.countplot(x = 'churn', palette='Set2', data = data)
```

Illastration: py_cd_5

```
Out[10]: <AxesSubplot:xlabel='churn', ylabel='count'>
```

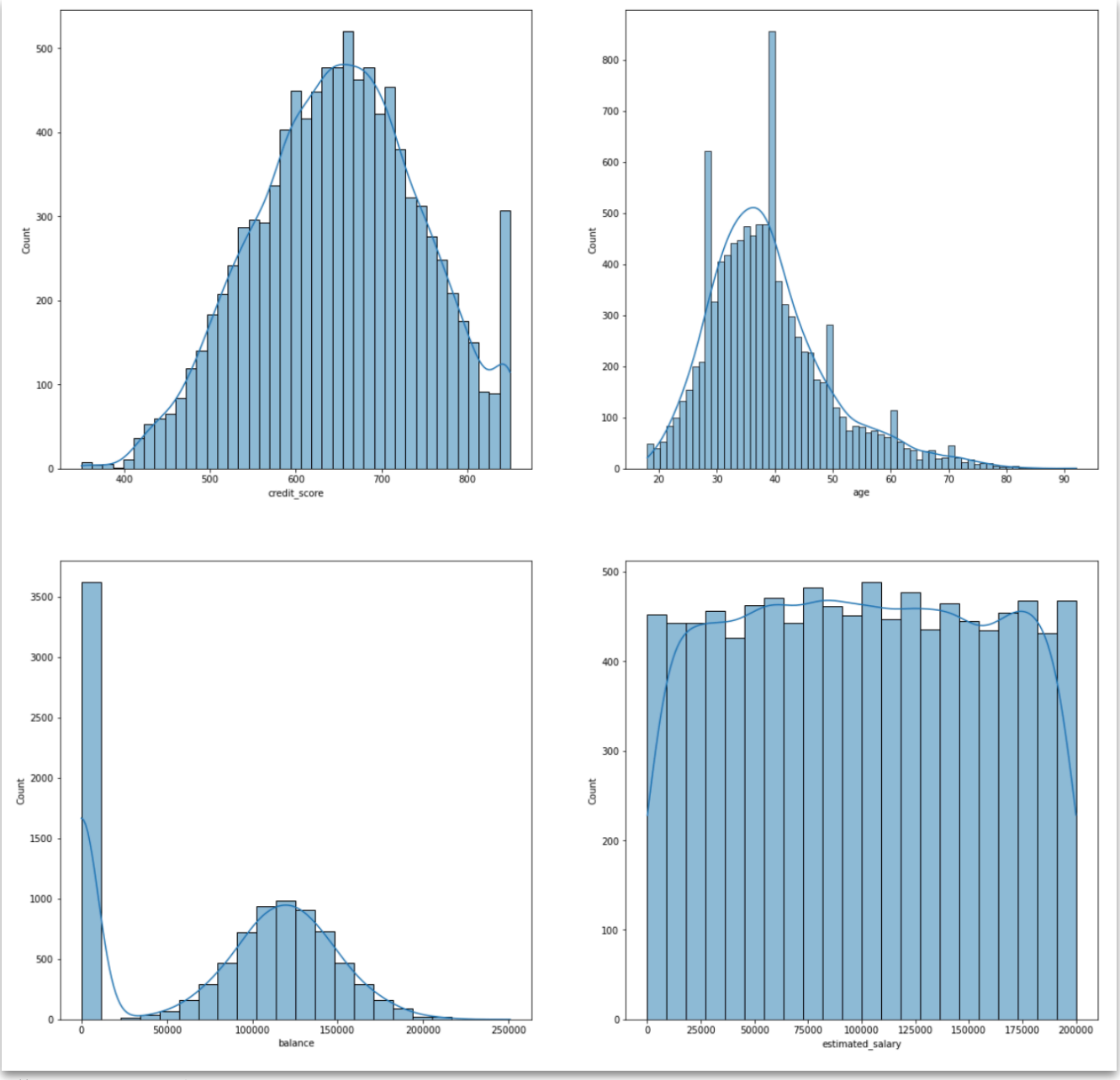


Illastration: py_cd_6

- Visualizing the distributions in the 4 numerical data type columns.(Illastration: py_cd_7 - 8) [Define a custom function to create reusable code (10)]

```
In [12]: # visualizing the distributions in the 4 numerical data type columns
         sub_num_cols = ['credit_score','age','balance','estimated_salary']
         rows = 2
         cols = 2
         idx = 0
         fig, axs = plt.subplots(rows, cols, figsize=(20,20))
         for i in range(rows):
             for j in range(cols):
                 sns.histplot(ax=axs[i, j], data=data, x=sub_num_cols[idx], kde=True)
                 idx += 1
```

Illastration: py_cd_7



Illastration: py_cd_8

After running below commands (Illastration: py_cd_9), it is easy to further understand for the 3 numerical data columns. There are 233 customers scores the max of 850 in credit scores; and 3617 customers has a zero-balance on their account.

```
In [13]: #analyzing customer credit score range
         data['credit_score'].describe()

Out[13]: count    10000.000000
         mean       650.528800
         std         96.653299
         min        350.000000
         25%        584.000000
         50%        652.000000
         75%        718.000000
         max        850.000000
         Name: credit_score, dtype: float64

In [14]: #large number of customer has credit score at max
         credit_score_850 = data[data['credit_score']==850]
         credit_score_850.shape

Out[14]: (233, 12)

In [15]: # analyzing the customer age range and customer distribution based on Age
         data['age'].describe()

Out[15]: count    10000.000000
         mean        38.921800
         std         10.487806
         min         18.000000
         25%         32.000000
         50%         37.000000
         75%         44.000000
         max         92.000000
         Name: age, dtype: float64

In [16]: #large number of customer has balance = 0
         balance_0 = data[data['balance']==0]
         balance_0.shape

Out[16]: (3617, 12)
```

Illastration: py_cd_9

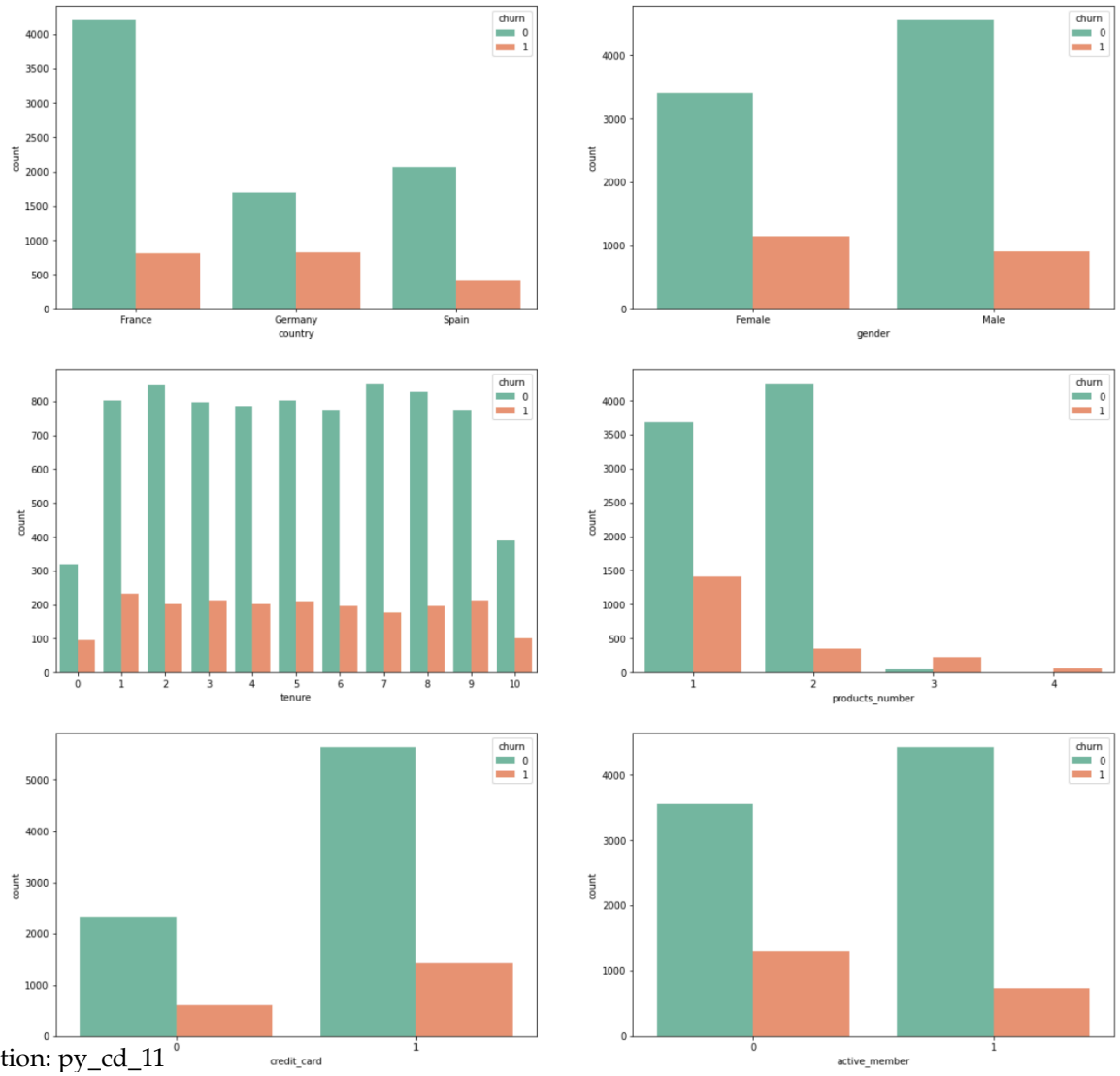
The main objective of this project is to use the dataset to analyze and predict about the customer churn issues, the next step I will explore the relationship between churn and each variables. I will perform 2 separate sets of Seaborn charts, one for categorical data and one for numerical data, to visualize the relationship.

- Visualizing the Churn relationship with each categorical data. (Illastration: py_cd_10 - 11)

```
In [17]: #understanding the churn relationship with each categorical data
         plt.figure(figsize = (20,20))

         plt.subplot(3,2,1)
         sns.countplot(x = 'country', hue='churn', palette='Set2', data = data)

         plt.subplot(3,2,2)
         sns.countplot(x = 'gender', hue= 'churn', palette='Set2', data = data)

         plt.subplot(3,2,3)
         sns.countplot(x = 'tenure', hue= 'churn', palette='Set2', data = data)

         plt.subplot(3,2,4)
         sns.countplot(x = 'products_number', hue= 'churn', palette='Set2', data = data)

         plt.subplot(3,2,5)
         sns.countplot(x = 'credit_card', hue= 'churn', palette='Set2', data = data)

         plt.subplot(3,2,6)
         sns.countplot(x = 'active_member', hue= 'churn', palette='Set2', data = data)
```

Illastration: py_cd_10

```
Out[17]: <AxesSubplot:xlabel='active_member', ylabel='count'>
```



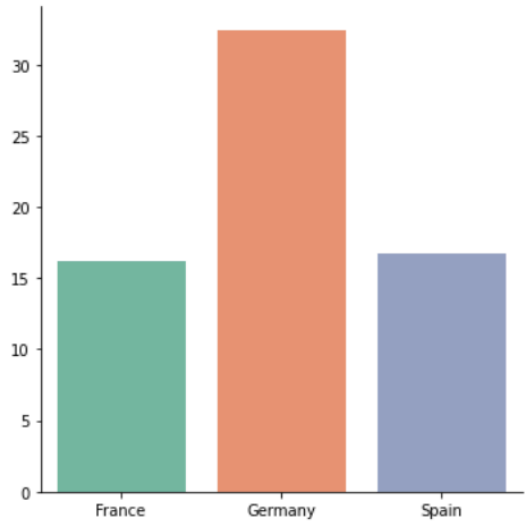
Illastration: py_cd_11

From the countplots display against Churn (Illastration: py_cd_11), the 'country' and 'product_numbers' might causing the churn issue. The next step, I will perform additional analysis on churn rates based on these 2 columns to identify the churn rates categorized.

- Customer churn rate based on 'country' (Illastration: py_cd_12) [Demonstrated: Dictionary or Lists (10), NumPy (10), Merge DataFrames (10)]

In [19]:
```
# churn rate based on country

country = np.array(pd.Categorical(data['country']).categories)
#check array order
#print(country)

cust_FR = data[data['country'] == 'France']
cust_GR = data[data['country'] == 'Germany']
cust_SP = data[data['country'] == 'Spain']
churn_FR = data[np.logical_and(data['country']== 'France', data['churn']== 1)]
churn_GR = data[np.logical_and(data['country']== 'Germany', data['churn']== 1)]
churn_SP = data[np.logical_and(data['country']== 'Spain', data['churn']== 1)]
total_cust_by_country = [len(cust_FR), len(cust_GR), len(cust_SP)]
churn_by_country = [len(churn_FR), len(churn_GR), len(churn_SP)]

churn_rate_country = []
for churn_1, total_cust in zip (churn_by_country,total_cust_by_country):
    churn_rate_country.append((churn_1/total_cust)*100)
#check array order
#print(churn_rate_country)

churn_rate_country_df = pd.DataFrame(np.column_stack(churn_rate_country), columns= country)
#check new dataframe structure
print(churn_rate_country_df)

sns.catplot(data=churn_rate_country_df, kind='bar',palette='Set2')
plt.show()
```

```
      France     Germany       Spain
0  16.154767  32.443204  16.673395
```



Illastration: py_cd_12

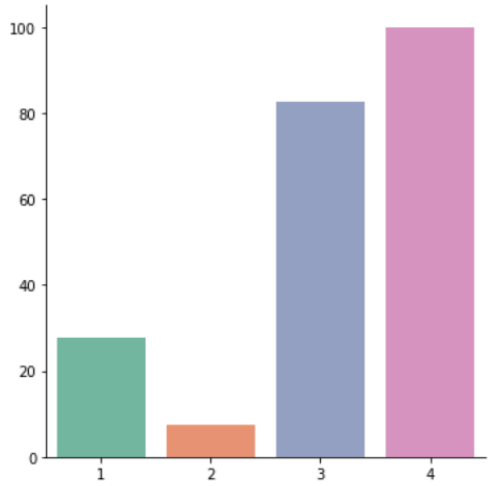
Based on the catplot (Illastration: py_cd_12), German customers has the highest customer churn rate, while customers from France and Spain enjoys the similar customer churn rate. [Demonstrated: Insight 2]

- Customer churn rate based on 'number of products holds' (Illastration: py_cd_13)

[Demonstrated: Dictionary or Lists (10), NumPy (10), Merge DataFrames (10)]

In [20]:
```
#churn rate in product numbers
#convert number of product to categorical data
data['products_number'] = data.products_number.astype('category')

num_of_prod = np.array(pd.Categorical(data['products_number']).categories)
#check array order
#print(num_of_prod)

cust_prod_1 = data[data['products_number'] == 1]
cust_prod_2 = data[data['products_number'] == 2]
cust_prod_3 = data[data['products_number'] == 3]
cust_prod_4 = data[data['products_number'] == 4]
churn_prod_1 = data[np.logical_and(data['products_number']== 1, data['churn']== 1)]
churn_prod_2 = data[np.logical_and(data['products_number']== 2, data['churn']== 1)]
churn_prod_3 = data[np.logical_and(data['products_number']== 3, data['churn']== 1)]
churn_prod_4 = data[np.logical_and(data['products_number']== 4, data['churn']== 1)]
total_cust_by_prod = [len(cust_prod_1), len(cust_prod_2), len(cust_prod_3),len(cust_prod_4)]
churn_by_prod = [len(churn_prod_1), len(churn_prod_2), len(churn_prod_3),len(churn_prod_4)]

churn_rate_prod = []
for churn_1, total_cust in zip (churn_by_prod,total_cust_by_prod):
    churn_rate_prod.append((churn_1/total_cust)*100)
#check array order
#print(churn_rate_prod)

churn_rate_prod_df = pd.DataFrame(np.column_stack(churn_rate_prod), columns= num_of_prod)
#check new dataframe structure
print(churn_rate_prod_df)

sns.catplot(data=churn_rate_prod_df, kind='bar',palette='Set2')
plt.show()
```

```
           1          2          3      4
0  27.714398   7.581699  82.706767  100.0
```



Illastration: py_cd_13

Based on the catplot above(Illastration: py_cd_10 - 13), customers who holds 3 or more products in the same account would more than likely to churn, and customers who holds 2 products with the bank are easily to retain. [Demonstrated: Insight 3]

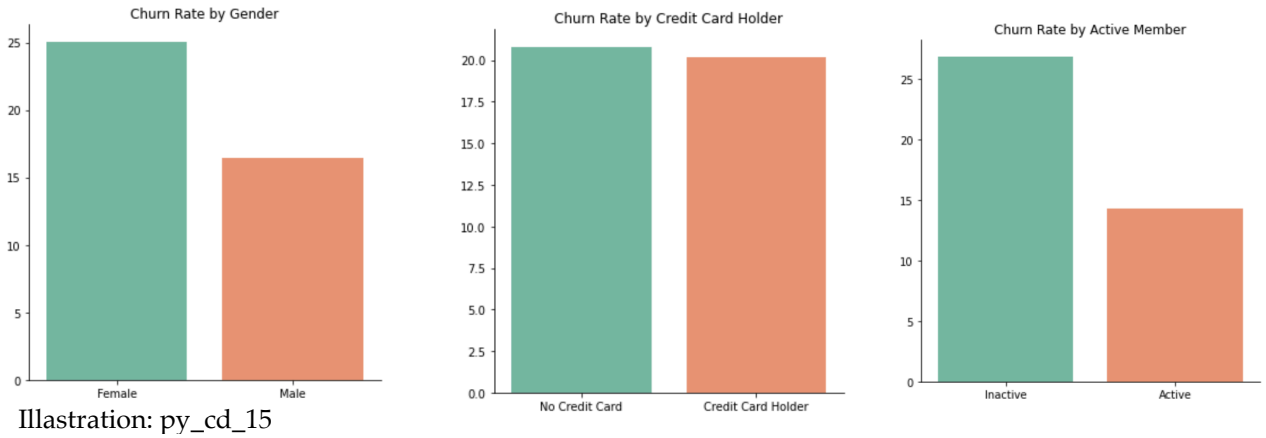
- Customer churn rate based on other categorical variables (Illastration: py_cd_14)

In [21]:
```
#perform similar churn rate analysis based on gender, number of credit cards hold, active members

#covnert dtype to categorical
data['gender'] = data.gender.astype('category')
data['credit_card'] = data.credit_card.astype('category')
data['active_member'] = data.active_member.astype('category')


####### gender ####################################
gender = np.array(pd.Categorical(data['gender']).categories)
#check array order
#print(gender)

cust_F = data[data['gender'] == 'Female']
cust_M = data[data['gender'] == 'Male']
churn_F = data[np.logical_and(data['gender']== 'Female', data['churn']== 1)]
churn_M = data[np.logical_and(data['gender']== 'Male', data['churn']== 1)]
total_cust_by_gender = [len(cust_F), len(cust_M)]
churn_by_gender = [len(churn_F), len(churn_M)]

churn_rate_gender = []
for churn_1, total_cust in zip (churn_by_gender,total_cust_by_gender):
    churn_rate_gender.append((churn_1/total_cust)*100)
#check array order
#print(churn_rate_gender)

churn_rate_gender_df = pd.DataFrame(np.column_stack(churn_rate_gender), columns= gender)

###### credit_card_num ##############################
credit_card_num = np.array(pd.Categorical(data['credit_card']).categories)
#check array order
#print(credit_card_num)

cust_CC_0 = data[data['credit_card'] == 0]
cust_CC_1 = data[data['credit_card'] == 1]
churn_CC_0 = data[np.logical_and(data['credit_card']== 0, data['churn']== 1)]
churn_CC_1 = data[np.logical_and(data['credit_card']== 1, data['churn']== 1)]
total_cust_by_CC = [len(cust_CC_0), len(cust_CC_1)]
churn_by_CC = [len(churn_CC_0), len(churn_CC_1)]

churn_rate_CC = []
for churn_1, total_cust in zip (churn_by_CC,total_cust_by_CC):
    churn_rate_CC.append((churn_1/total_cust)*100)
#check array order
#print(churn_rate_CC)

churn_rate_CC_df = pd.DataFrame(np.column_stack(churn_rate_CC), columns= credit_card_num)

###### active_member ##############################
active_member = np.array(pd.Categorical(data['active_member']).categories)
#check array order
#print(active_member)

cust_active_0 = data[data['active_member'] == 0]
cust_active_1 = data[data['active_member'] == 1]
churn_active_0 = data[np.logical_and(data['active_member']== 0, data['churn']== 1)]
churn_active_1 = data[np.logical_and(data['active_member']== 1, data['churn']== 1)]
total_cust_by_active = [len(cust_active_0), len(cust_active_1)]
churn_by_active = [len(churn_active_0), len(churn_active_1)]

churn_rate_active_member = []
for churn_1, total_cust in zip (churn_by_active,total_cust_by_active):
    churn_rate_active_member.append((churn_1/total_cust)*100)
#check array order
#print(churn_rate_active_member)

churn_rate_active_member_df = pd.DataFrame(np.column_stack(churn_rate_active_member), columns= active_member)

print(churn_rate_gender_df)
sns.catplot(data=churn_rate_gender_df, kind='bar',
            palette='Set2').set(title='Churn Rate by Gender')
print(churn_rate_CC_df)
sns.catplot(data=churn_rate_CC_df, kind='bar',
            palette='Set2').set(title='Churn Rate by Credit Card Holder').set_xticklabels(["No Credit Card",
                                                                                           "Credit Card Holder"])
print(churn_rate_active_member_df)
sns.catplot(data=churn_rate_active_member_df, kind='bar',
            palette='Set2').set(title='Churn Rate by Active Member').set_xticklabels(["Inactive", "Active"])
```

Illastration: py_cd_14

I also performed similar churn rate analysis based on 'gender', 'number of credit cards hold', and 'members status', to reconfirmed the assumption that these category has minimum impact on churn issue.

The command code showed in above illustration (Illastration: py_cd_15), and the catplots resulted from those commands as follow (Illastration: py_cd_15)



Illastration: py_cd_15

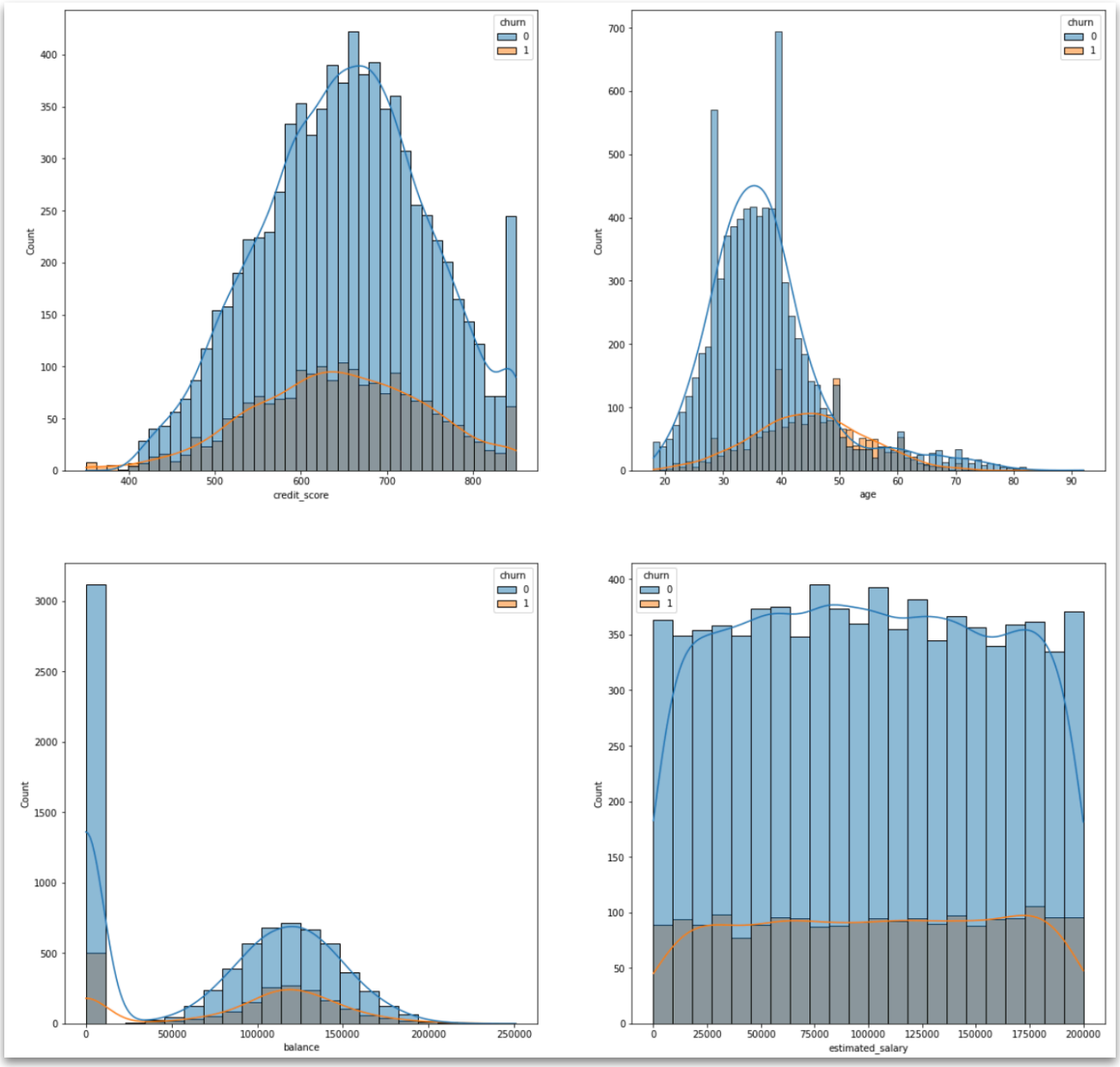
- Understanding the churn distribution with each numerical columns by running histplots with kde turns on and hue = 'churn', to exam the distribution shape (Illastration: py_cd_16)

```
In [22]: #understanding the churn distribution with each numerical columns
         sub_num_cols = ['credit_score','age','balance','estimated_salary']
         rows = 2
         cols = 2
         idx = 0
         fig, axs = plt.subplots(rows, cols, figsize=(20,20))
         for num_count in range(rows):
             for num_val in range(cols):
                 sns.histplot(ax=axs[num_count, num_val], data=data, x=sub_num_cols[idx], hue='churn', kde=True)
                 idx += 1
```

Illastration: py_cd_16

with the result displayed in below histplots (Illastration: py_cd_17), it is easy to see the difference between the distribution shape of when Churn = 0 and when churn = 1. The histplots demonstrated a clear skew on the 'age' based distributions, while the 'credit_score', 'balance', and 'estimated_salary' distribution shows not much of a difference in distributions shapes. [Demonstrated: Insight 4]

In fact, for further analyzation and/or prediction, I may potentially drop the column of 'estimated_salary'



Illastration: py_cd_17

- As mentioned in above analysis and insights, to better understand the dataset, it is necessary to group "age" into certain age groups and use it as an additional categorical variable for further in sights.
    - Add additional categorical column for age groups

I have created a function (AgeGroup(Age)) that identifies the value in the "age" column and returns a designated age group values, then assign to the correspond row in the newly added column. (Illastration: py_cd_18) [Demonstrated: Make use of iterators (10), Define a custom function to create reusable code (10)]

```
In [23]: #adding an addtional categorical column for age groups
Age = 0
def AgeGroup(Age):
    if Age < 20:
        return 'blow 20'
    if Age >= 20 and Age < 25:
        return '20-25'
    if Age >= 25 and Age <30:
        return  '25-30'
    if Age >= 30 and Age <35:
        return '30-35'
    if Age >= 35 and Age <40:
        return  '35-40'
    if Age >= 40 and Age <45:
        return  '40-45'
    if Age >= 45 and Age <50:
        return  '45-50'
    if Age >= 50 and Age <55:
        return  '50-55'
    if Age >= 55 and Age <60:
        return  '55-60'
    if Age >= 60 and Age <65:
        return  '60-65'
    if Age >= 65 and Age <70:
        return  '65-70'
    if Age >= 70 and Age <75:
        return  '70-75'
    else:
        return  '75 above'

data['age_group']= data['age'].apply(AgeGroup)
data.head()
```

Illastration: py_cd_18

The new “age_group” columns was added correctly, and assigned with correct values (Illastration: py_cd_19)

Out[23]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn	age_group
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1	40-45
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0	40-45
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1	40-45
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0	35-40
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0	40-45

Illastration: py_cd_19

After converted the data type to categorial data for the “age_group” column (Illastration: py_cd_20), I ran a similar process to visualize the churn rate by “age_group” (Illastration: py_cd_21)

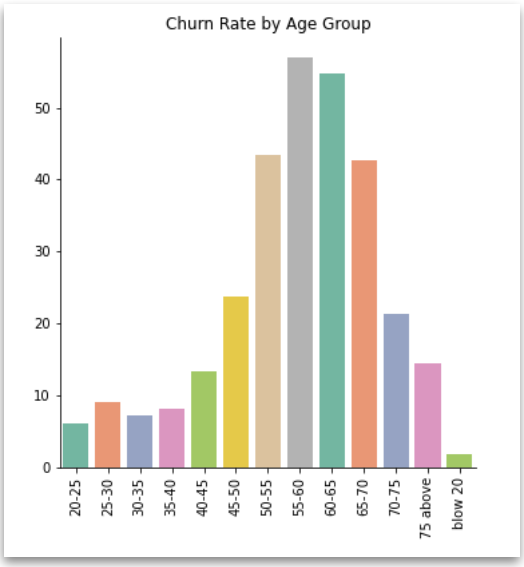
```
In [24]: data['age_group'] = data.age_group.astype('category')
         data['age_group'].info()

         <class 'pandas.core.series.Series'>
         RangeIndex: 10000 entries, 0 to 9999
         Series name: age_group
         Non-Null Count  Dtype
         --------------  -----
         10000 non-null  category
         dtypes: category(1)
         memory usage: 10.5 KB
```

Illastration: py_cd_20

In [26]:
```
#churn rate at each age_group
age_range = np.array(pd.Categorical(data['age_group']).categories)
#check array order
#print(num_of_prod)

cust_age_20 = data[data['age_group'] =='blow 20']
cust_age_20_25 = data[data['age_group'] =='20-25']
cust_age_25_30 = data[data['age_group'] =='25-30']
cust_age_30_35 = data[data['age_group'] =='30-35']
cust_age_35_40 = data[data['age_group'] =='35-40']
cust_age_40_45 = data[data['age_group'] =='40-45']
cust_age_45_50 = data[data['age_group'] =='45-50']
cust_age_50_55 = data[data['age_group'] =='50-55']
cust_age_55_60 = data[data['age_group'] =='55-60']
cust_age_60_65 = data[data['age_group'] =='60-65']
cust_age_65_70 = data[data['age_group'] =='65-70']
cust_age_70_75 = data[data['age_group'] =='70-75']
cust_age_75 = data[data['age_group'] =='75 above']
churn_age_20 = data[np.logical_and(data['age_group'] =='blow 20', data['churn']== 1)]
churn_age_20_25 = data[np.logical_and(data['age_group'] =='20-25', data['churn']== 1)]
churn_age_25_30 = data[np.logical_and(data['age_group'] =='25-30', data['churn']== 1)]
churn_age_30_35 = data[np.logical_and(data['age_group'] =='30-35', data['churn']== 1)]
churn_age_35_40 = data[np.logical_and(data['age_group'] =='35-40', data['churn']== 1)]
churn_age_40_45 = data[np.logical_and(data['age_group'] =='40-45', data['churn']== 1)]
churn_age_45_50 = data[np.logical_and(data['age_group'] =='45-50', data['churn']== 1)]
churn_age_50_55 = data[np.logical_and(data['age_group'] =='50-55', data['churn']== 1)]
churn_age_55_60 = data[np.logical_and(data['age_group'] =='55-60', data['churn']== 1)]
churn_age_60_65 = data[np.logical_and(data['age_group'] =='60-65', data['churn']== 1)]
churn_age_65_70 = data[np.logical_and(data['age_group'] =='65-70', data['churn']== 1)]
churn_age_70_75 = data[np.logical_and(data['age_group'] =='70-75', data['churn']== 1)]
churn_age_75 = data[np.logical_and(data['age_group'] =='75 above', data['churn']== 1)]

total_cust_age_group = [len(cust_age_20),
                        len(cust_age_20_25),
                        len(cust_age_25_30),
                        len(cust_age_30_35),
                        len(cust_age_35_40),
                        len(cust_age_40_45),
                        len(cust_age_45_50),
                        len(cust_age_50_55),
                        len(cust_age_55_60),
                        len(cust_age_60_65),
                        len(cust_age_65_70),
                        len(cust_age_70_75),
                        len(cust_age_75)]
churn_by_age_group = [len(churn_age_20),
                      len(churn_age_20_25),
                      len(churn_age_25_30),
                      len(churn_age_30_35),
                      len(churn_age_35_40),
                      len(churn_age_40_45),
                      len(churn_age_45_50),
                      len(churn_age_50_55),
                      len(churn_age_55_60),
                      len(churn_age_60_65),
                      len(churn_age_65_70),
                      len(churn_age_70_75),
                      len(churn_age_75)]

churn_rate_age_group = []
for churn_1, total_cust in zip (churn_by_age_group,total_cust_age_group):
    churn_rate_age_group.append((churn_1/total_cust)*100)
#check array order
#print(churn_rate_active_member)

churn_rate_age_group_df = pd.DataFrame(np.column_stack(churn_rate_age_group), columns= age_range)

print(churn_rate_age_group_df)
sns.catplot(data=churn_rate_age_group_df, kind='bar',palette='Set2').set(title='Churn Rate by Age Group')
plt.xticks(rotation=90)
plt.show()
```

Illastration: py_cd_21

below catplot (Illastration: py_cd_22) demonstrates that in the customer churn rate by age group, customers in their 50 -70 is more than likely to leave the bank. [Demonstrated: Insight 5]



Illastration: py_cd_22

✓ Results from exploratory analysis

Based on the exploratory data analysis, customers in age 50-70, holds 3 or 4 products, lives in Germany, have higher potential of churn. As business, if we can identify a list of customer that have churn potential and create a specific treatment to for this group of customers, it would be ecumenically beneficial. As mentioned at the beginning of this report, by lowering 5% of customer churn rate would increase at least 25% in profit.

I have ran below python commends to identify a group of customers who has high potential to churn. (Illastration: py_cd_23)

In [27]:
```
#Identify high churn rate customers by customer IDs

# Customers from Germany - use the sub datasets constructed above
cust_GR.shape

#construct other sub datasets

#Customers holds 3 or 4 products
cust_prod_hold = data[data['products_number'].isin([3,4])]
cust_prod_hold.shape
#customer in age group 50 - 70
cust_high_churn_age_range = data[data['age_group'].isin(['50-55','55-60','60-65','65-70'])]
cust_high_churn_age_range.head()

# merge by customer ID to get a group of hight churn posibility
#customer age 50 - 70 lives in Germany
high_churn_loc_n_age_range = pd.merge(cust_GR,cust_high_churn_age_range, on='customer_id', how='inner')
high_churn_loc_n_age_range.head()
#customer holds 3 or 4 product lives in Germany
high_churn_loc_n_prod_hold = pd.merge(cust_GR,cust_prod_hold, on='customer_id', how='inner')
high_churn_loc_n_prod_hold.head()
#customers in age 50-70, holds 3 or 4 prodcuts, lives in Germany
high_churn = pd.merge(high_churn_loc_n_age_range, high_churn_loc_n_prod_hold, on = 'customer_id', how = 'inner')
high_churn.head()
```

Illastration: py_cd_23

3. Machine learning

During exploratory data analysis, I have found that the 'credit_card', 'tenure', and 'estimated_salary' has no impact to churn rate; and the distribution shape between churn and not churn customers on the 'balance' columns shows no skew in the churn analysis. Therefore, the final data will be used for machine learning models will drop those columns. (Illastration: py_ml_1)

```
In [28]: # As 'credit_card','tenure','estimated_salary','balance', no show of skewed shape in the churn analysis
         #data.head()
         final_data = data.drop(columns = ['credit_card','estimated_salary','tenure','balance'])
         final_data.head()
```

Out[28]:

	customer_id	credit_score	country	gender	age	products_number	active_member	churn	age_group
0	15634602	619	France	Female	42	1	1	1	40-45
1	15647311	608	Spain	Female	41	1	1	0	40-45
2	15619304	502	France	Female	42	3	0	1	40-45
3	15701354	699	France	Female	39	2	0	0	35-40
4	15737888	850	Spain	Female	43	1	1	0	40-45

Illastration: py_ml_1

Import necessary packages (Illastration: py_ml_2)

```
In [29]: #import ML packs
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.neighbors import NearestNeighbors
         from sklearn.model_selection import train_test_split
```

Illastration: py_ml_2

Convert 'country' and 'gender' column values to numeric values, for training machine learning model (Illastration: py_ml_3)

```
In [30]: #country and gender to numeric values
         to_numeric = {
             'country':{'France': 1, 'Germany': 2, 'Spain':3},
             'gender': {'Male': 0, 'Female': 1}
         }
         final_data = final_data.replace(to_numeric)
         final_data.head()
```

Out[30]:

	customer_id	credit_score	country	gender	age	products_number	active_member	churn	age_group
0	15634602	619	1	1	42	1	1	1	40-45
1	15647311	608	3	1	41	1	1	0	40-45
2	15619304	502	1	1	42	3	0	1	40-45
3	15701354	699	1	1	39	2	0	0	35-40
4	15737888	850	3	1	43	1	1	0	40-45

Illastration: py_ml_3

Start building machine learning models by following below steps. (Illastration: py_ml_4)
Step 1 : Define supervised learning NumPy arrays
Step 2: Split the dataset available into training model dataset, and test dataset
Step 3: Calculating the accuracy rate of the training model

```
In [38]: X = final_data.drop(["churn","age_group"], axis = 1).values
         y = final_data["churn"].values

         # Split into training and test sets
         X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.3,random_state=42)

         #print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

         # Fit the classifier to the training data
         knn = KNeighborsClassifier(n_neighbors=2)
         knn.fit(X_train, y_train)

         print(knn.score(X_test, y_test))

         0.7846666666666666
```

Illastration: py_ml_4

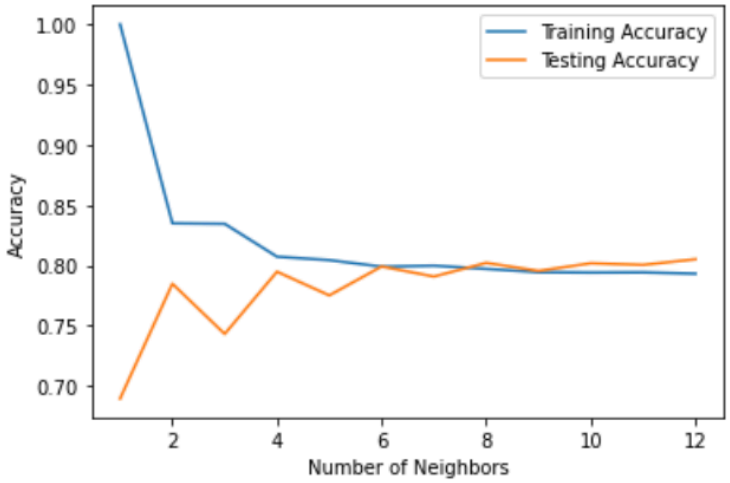
Based on the output, the training model only has 78% accuracy rate, when the 'neighbors' set at 2. The next step is to understand how the model's accuracy rate changes when fitting with different number of neighbors.
Use a for loop to calculate the correspond accuracy rate (Illastration: py_ml_5)

```
In [39]: # Create neighbors
         neighbors = np.arange(1, 13)
         train_accuracies = {}
         test_accuracies = {}

         for neighbor in neighbors:

             # Set up a KNN Classifier
             knn = KNeighborsClassifier(n_neighbors=neighbor)

             # Fit the model
             knn.fit(X_train, y_train)

             # Compute accuracy
             train_accuracies[neighbor] = knn.score(X_train, y_train)
             test_accuracies[neighbor] = knn.score(X_test, y_test)
```

Illastration: py_ml_5

Visualize the trend (Illastration: py_ml_6)

```
In [40]: # Plot training accuracies
         plt.plot(neighbors, train_accuracies.values(), label="Training Accuracy")

         # Plot test accuracies
         plt.plot(neighbors, test_accuracies.values(), label="Testing Accuracy")
         plt.legend()
         plt.xlabel("Number of Neighbors")
         plt.ylabel("Accuracy")
         plt.show()
```



Illastration: py_ml_6

Based on the result plot (Illastration: py_ml_6), when neighbor set to 8, the training model predict the most accurate result. After we fit the model again, have the optimized model using KNeighborClassifier (Illastration: py_ml_7)

```
In [41]: knn = KNeighborsClassifier(n_neighbors=8)
         knn.fit(X_train, y_train)

         print(knn.score(X_test, y_test))
```

```
0.802
```

Illastration: py_ml_7

Result and Insights

During the data exploratory process, we uncovered:

- No clear strong correlation between any 2 of the variables.

During advance data exploratory process, we uncovered:

- German customers enjoys a high customer churn rate
- Customers who holds 3 or 4 products has a high customer churn rate
- Customers in age 50 - 70 is more than likely to churn
- Customer holds how many credit cards, the tenure, how much is in their balance, and their estimated salary, has low or no impact on the churn issue.
- Identified a list of customers who is in between age 50 - 70, holds 3 or 4 products, and lives in Germany. This is a great starting point for customer relation departments to design a/several specific treatments to retained the customer in this group.

Model training using supervised classification, we uncovered:

- Set the n_neighbors to 8, with 70/30 data split will have the optimal KNeighborClassification model.