Assignment 2:
IBM-Project-117-1658211897

Importing

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [4]:

```
df=pd.read_csv("E:\IBM projects Assignment Sona
College\Churn_Modelling.csv")
```

In [5]:

```
df.info()
```

```
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
```

In [6]:

```
df.describe()
```

Out[6]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.00000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

1. UNIVARIATE ANALYSIS

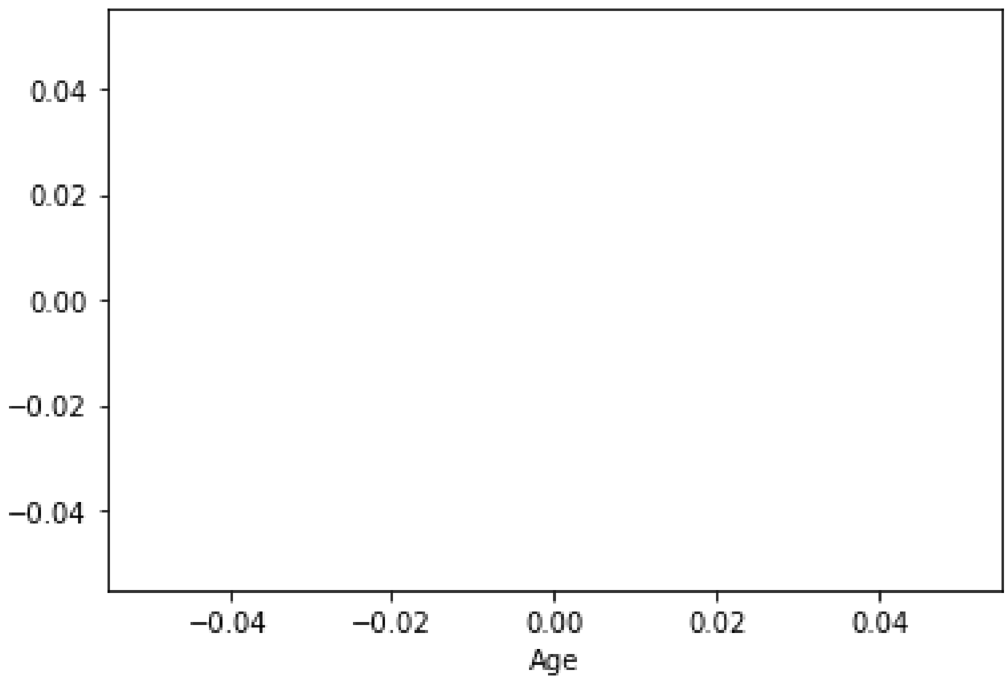
The term univariate analysis refers to the analysis of one variable. You can remember this because the prefix “uni” means “one.” There are three common ways to perform univariate analysis on one variable: 1. Summary statistics – Measures the center and spread of values.

In [78]:

```
df_france=df.loc[df['Geography']=='France']
df_spain=df.loc[df['Geography']=='Spain']
df_germany=df.loc[df['Geography']=='Germany']
```

In [79]:

```
plt.plot(df_france['Balance'],np.zeros_like(df_france['Balance']),'o')
plt.plot(df_spain['Balance'],np.zeros_like(df_spain['Balance']),'o')
plt.plot(df_germany['Balance'],np.zeros_like(df_germany['Balance']),'o')
plt.xlabel('Age')
plt.show()
```

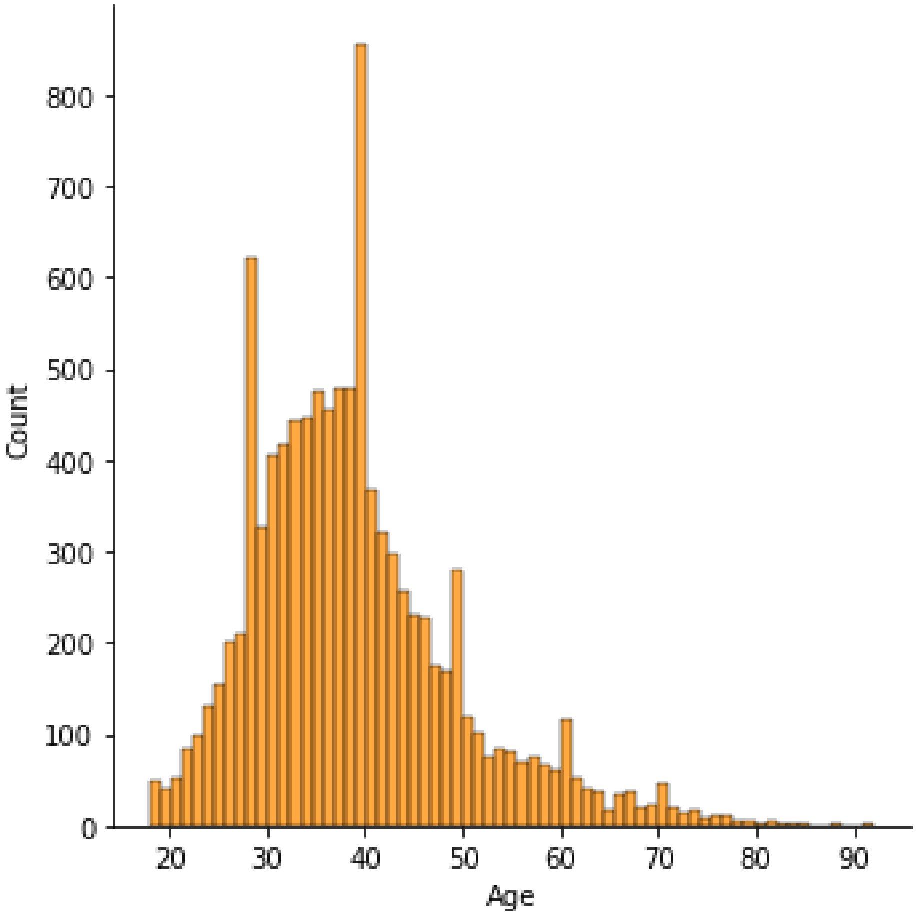


Histogram

In [7]:

```
sns.displot(df["Age"], color='darkorange')
```

Out[7]:

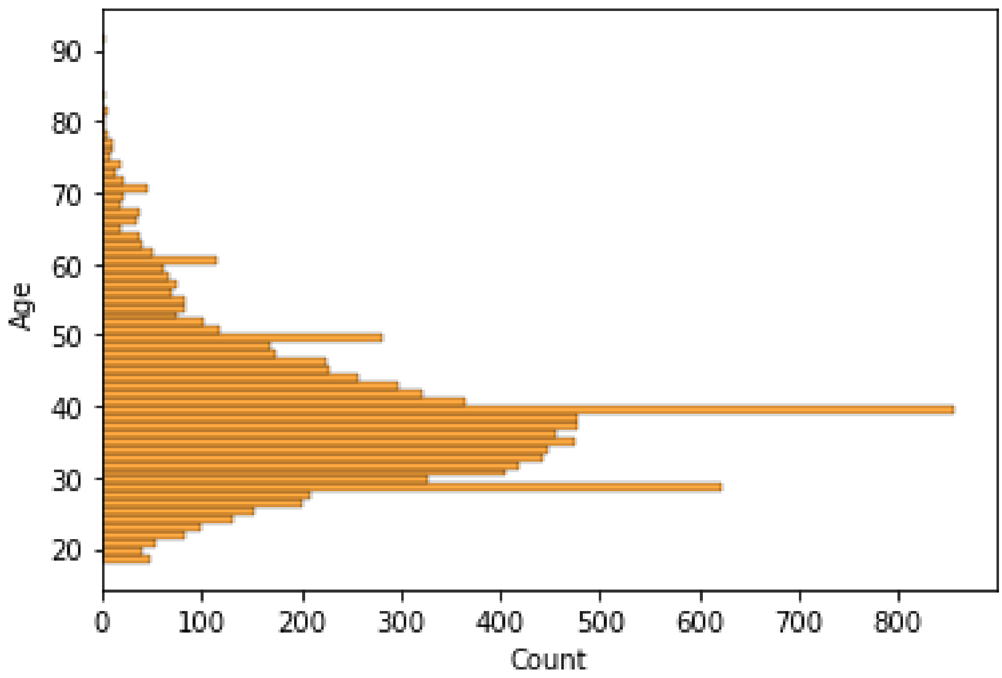


In Histogram, we can do it vertically too, by just changing the axis

In [8]:

```
sns.histplot(y="Age",data=df,color='darkorange')
```

Out[8]:

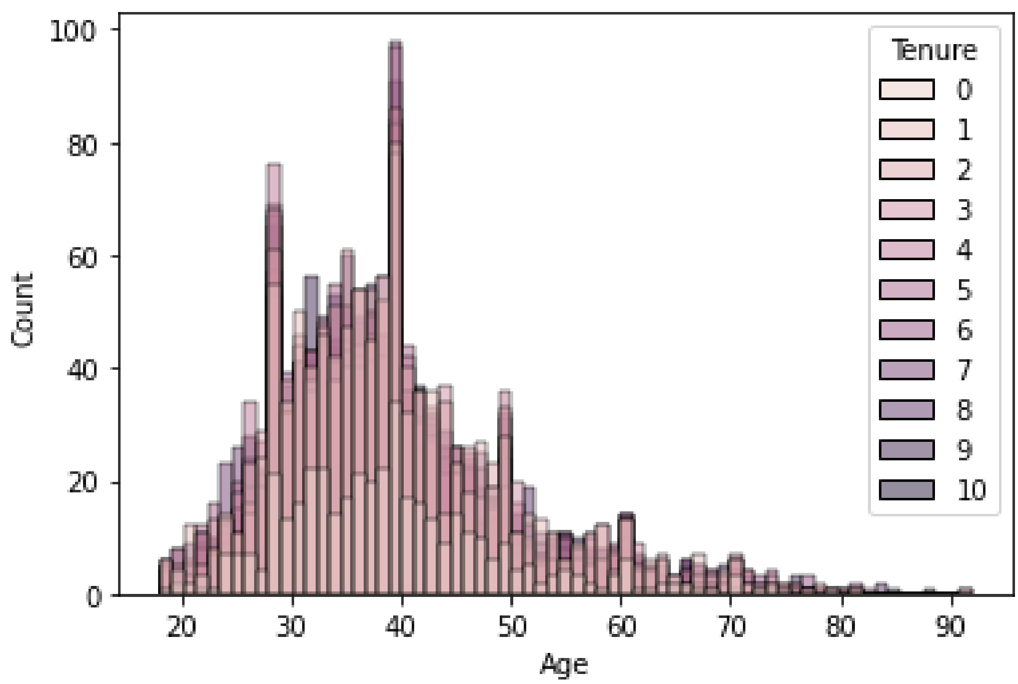


Now, we can also use Histogram for categorical variables

In [9]:

```
sns.histplot(x='Age',data=df,hue=df['Tenure'])
```

Out[9]:

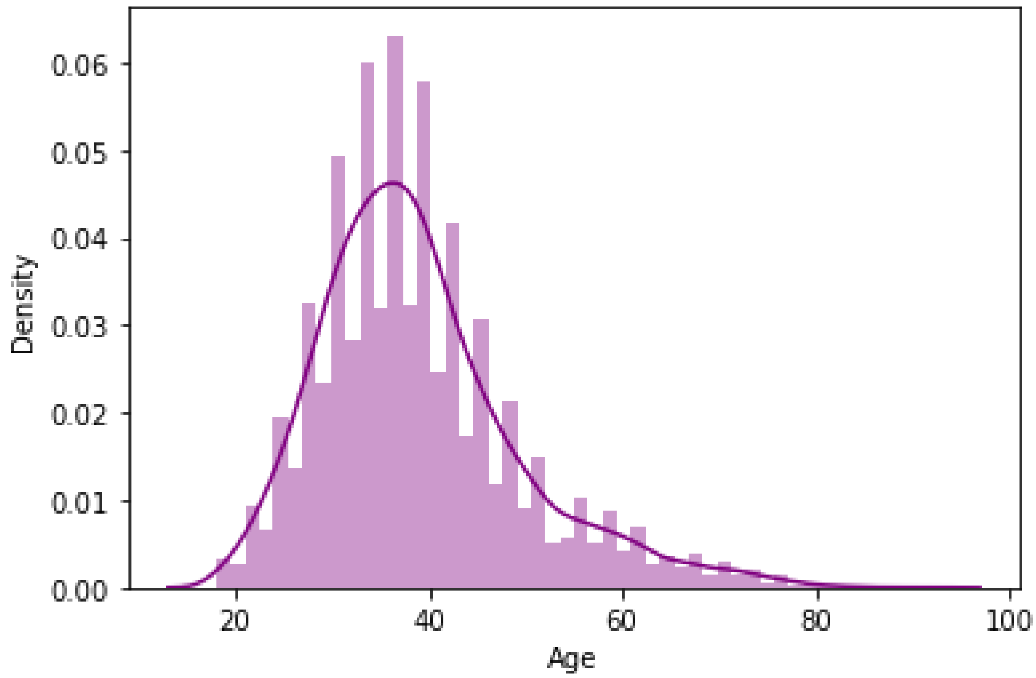


Distplot

In [10]:

```
sns.distplot(df["Age"],color='purple')
```

Out[10]:

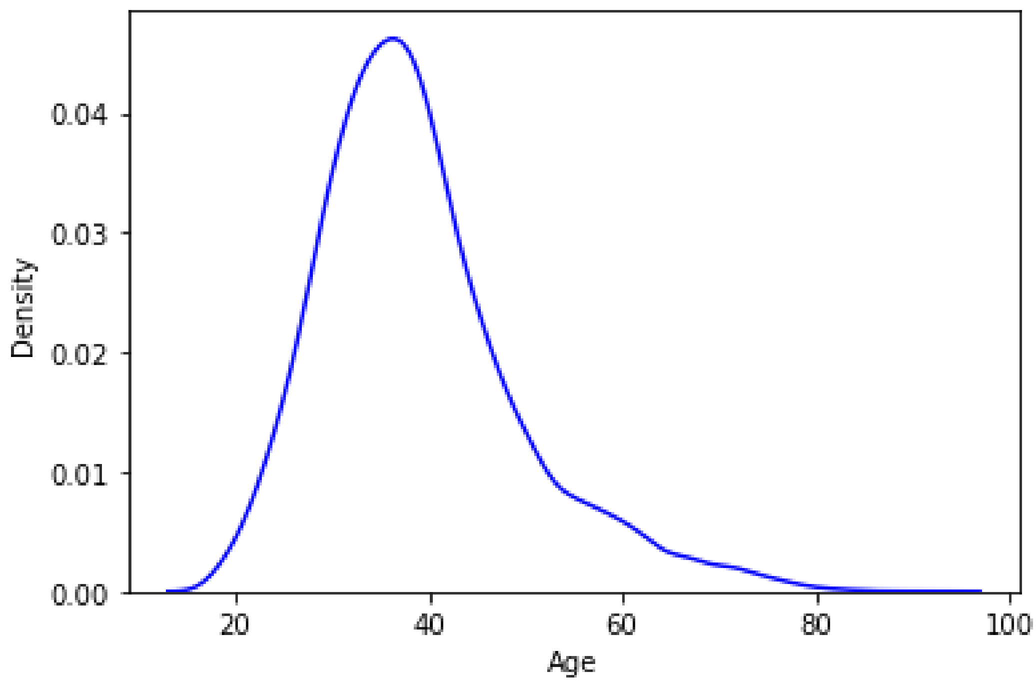


This is visualising displot alone

In [11]:

```
sns.distplot(df["Age"],hist=False,color='blue')
```

Out[11]:

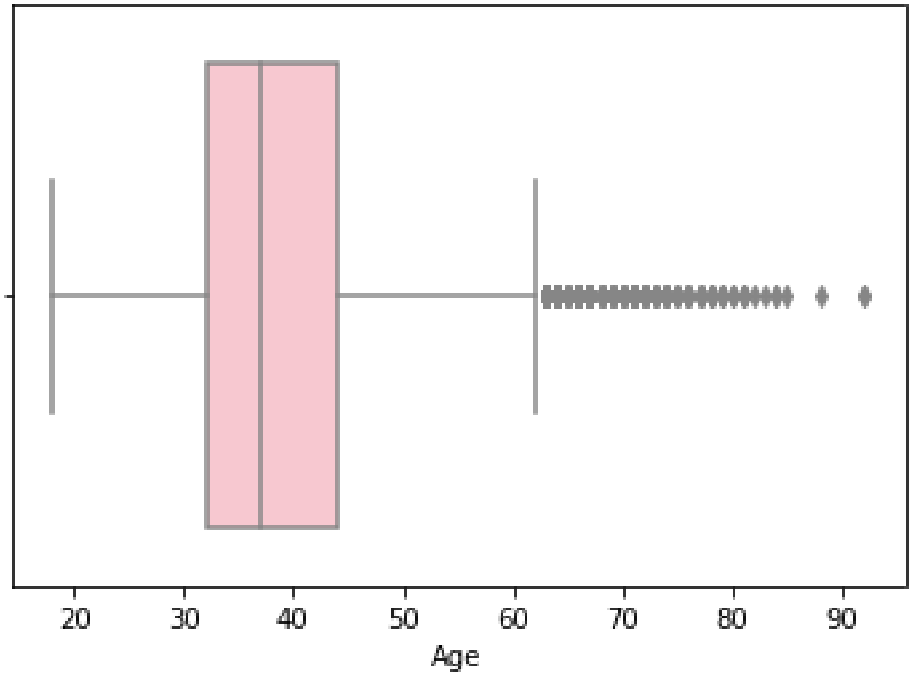


Boxplot

In [12]:

```
sns.boxplot(df["Age"],color='pink')
```

Out[12]:

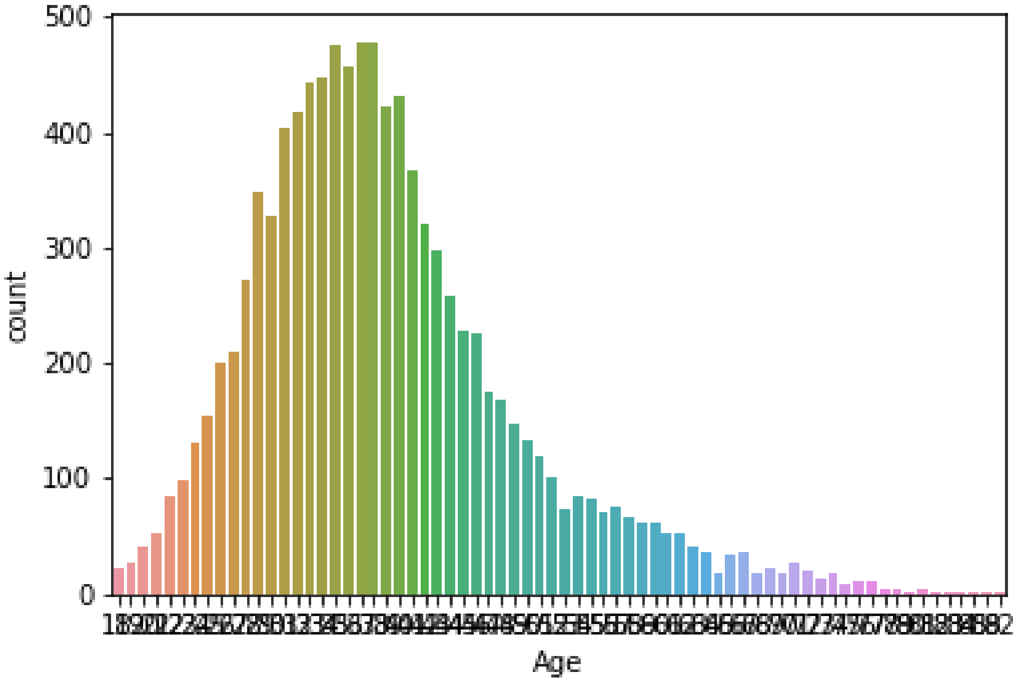


Countplot

In [14]:

```
sns.countplot(df['Age'])
```

Out[14]:

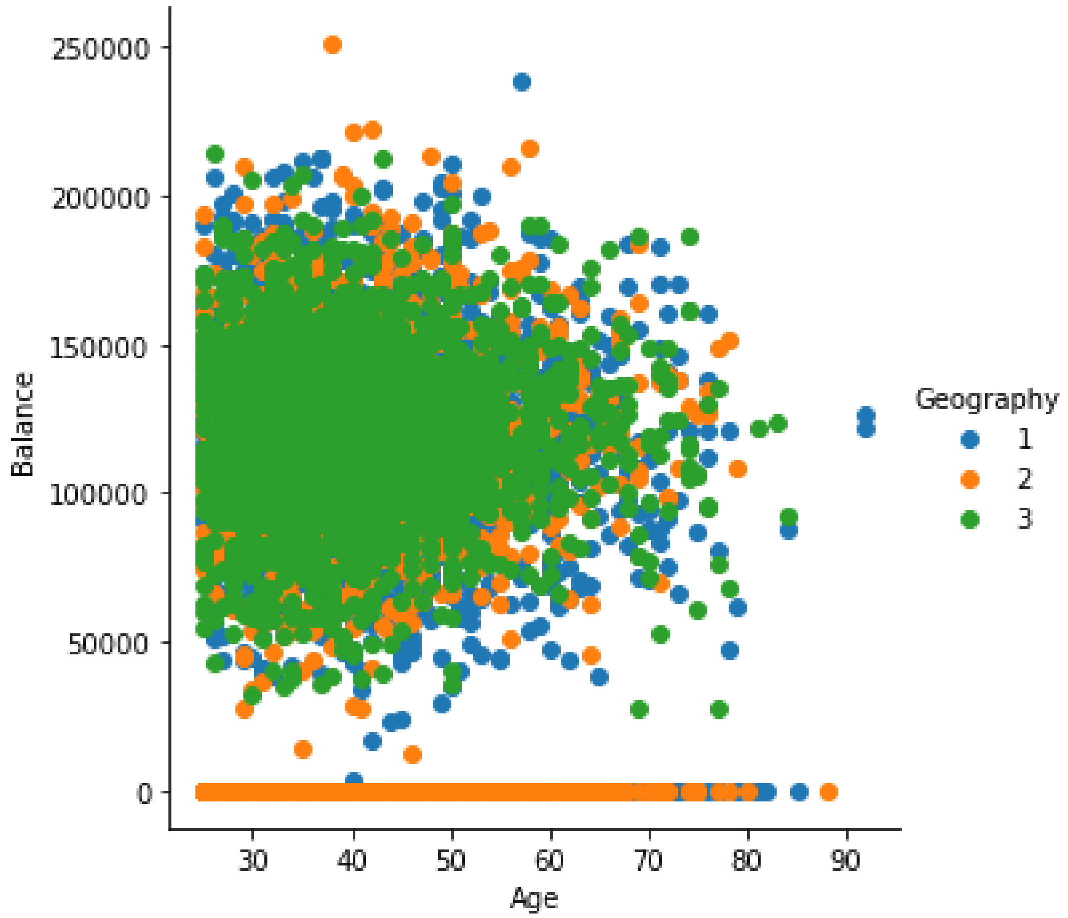


2. BIVARIATE ANALYSIS

Image result for bivariate analysis in python It is a methodical statistical technique applied to a pair of variables (features/ attributes) of data to determine the empirical relationship between them. In order words, it is meant to determine any concurrent relations (usually over and above a simple correlation analysis).

In [80]:

```
sns.FacetGrid(df,hue="Geography",size=5).map(plt.scatter,"Age","Balance").a
dd_legend();
plt.show()
```

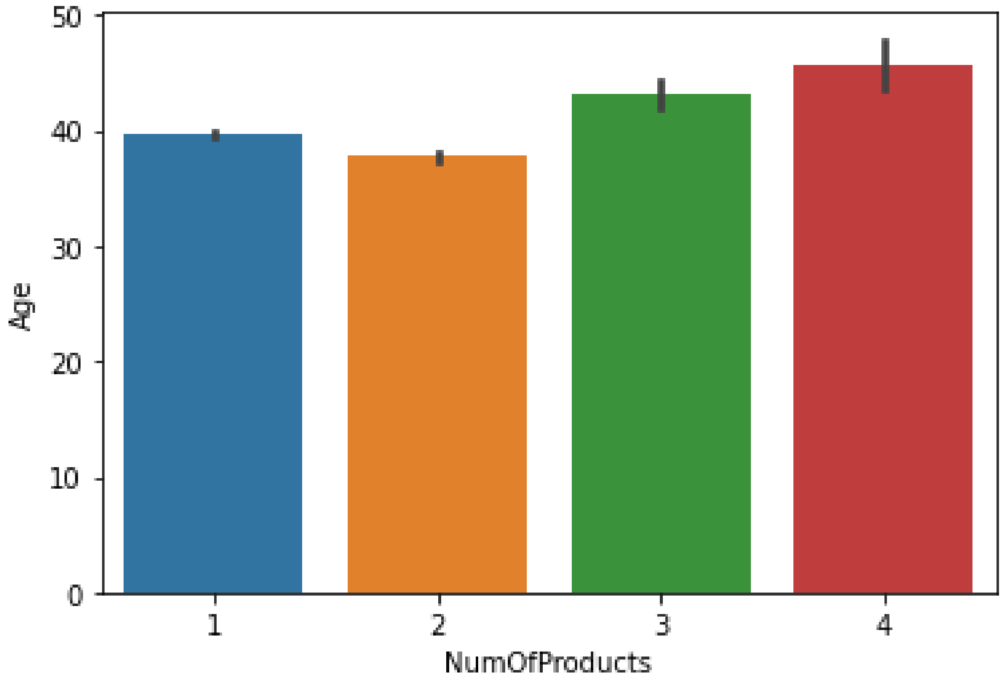


Barplot

In [15]:

```
sns.barplot(df["NumOfProducts"],df["Age"])
```

Out[15]:

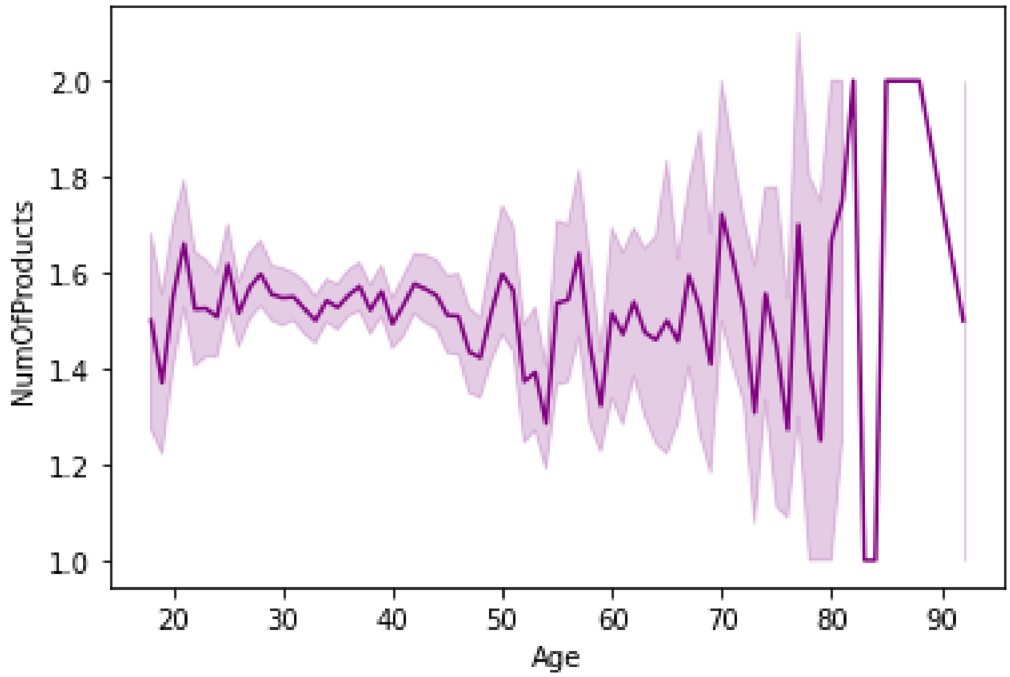


Linearplot

In [16]:

```
sns.lineplot(df["Age"],df["NumOfProducts"], color='purple')
```

Out[16]:

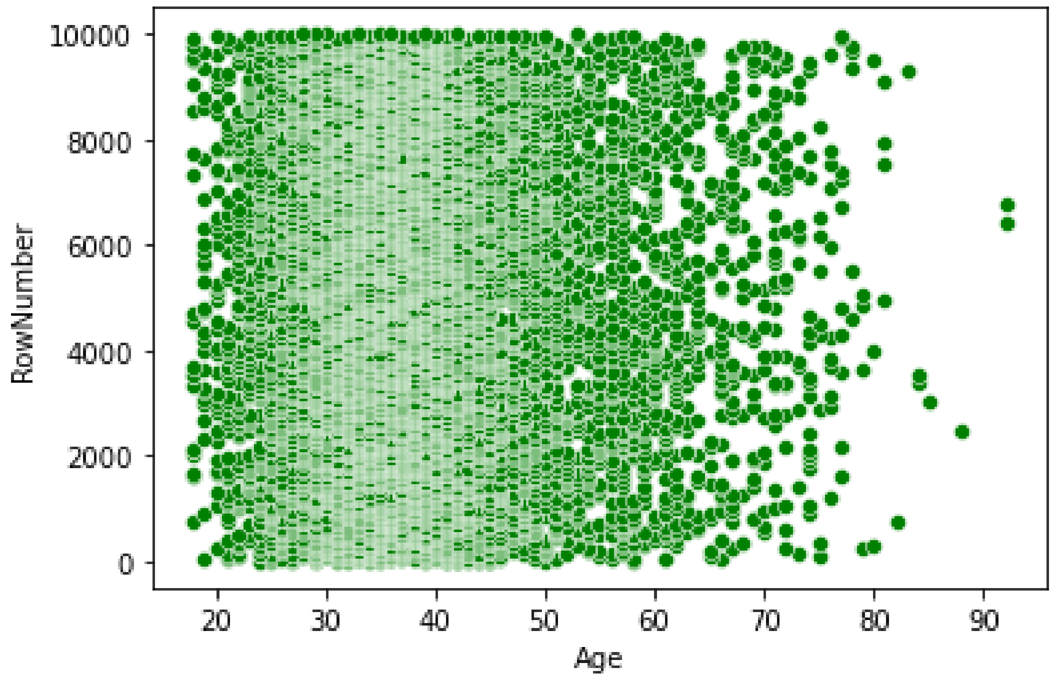


Scatterplot

In [17]:

```
sns.scatterplot(x=df.Age,y=df.RowNumber,color='green')
```

Out[17]:

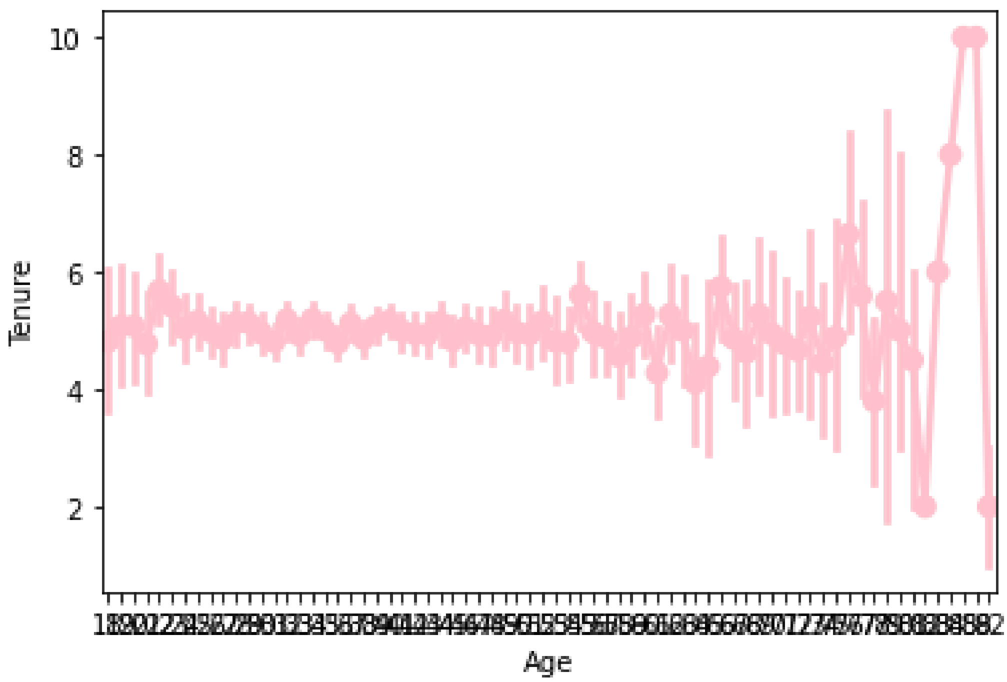


Pointplot

In [18]:

```
sns.pointplot(x='Age',y='Tenure',data=df,color='pink')
```

Out[18]:

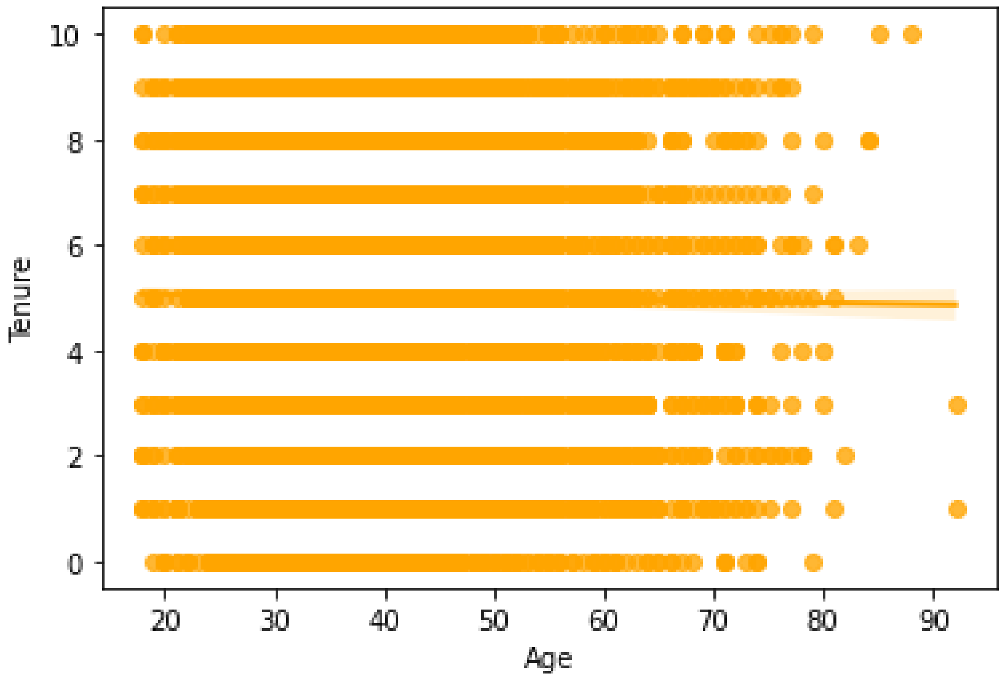


Regplot

In [19]:

```
sns.regplot(df['Age'],df['Tenure'],color='orange')
```

Out[19]:



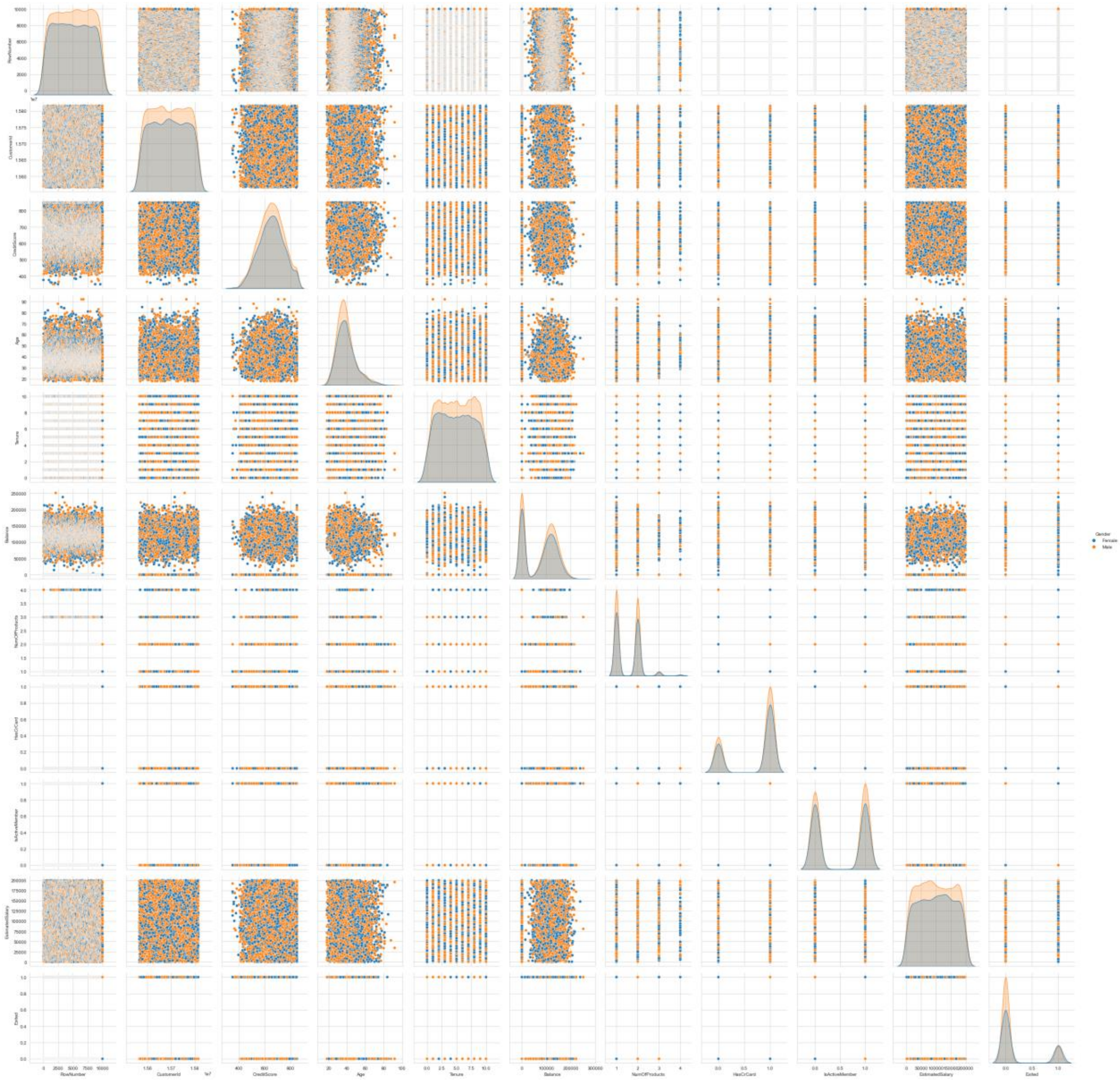
3. MULTI - VARIATE ANALYSIS

Multivariate analysis is based in observation and analysis of more than one statistical outcome variable at a time. In design and analysis, the technique is used to perform trade studies across multiple dimensions while taking into account the effects of all variables on the responses of interest.

In [187]:

```
sns.pairplot(df,hue="Gender",size=3)
```

Out[187]:

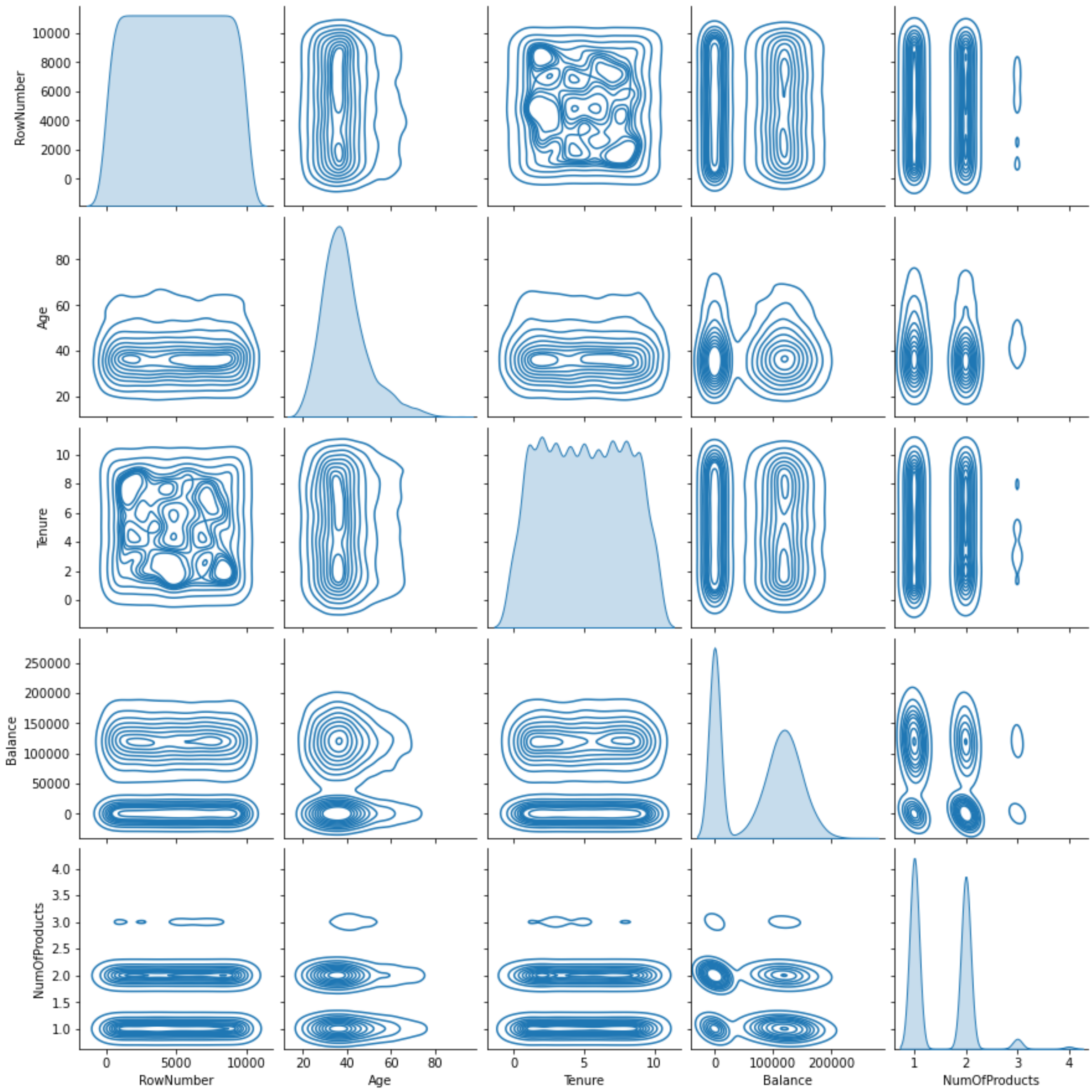


Pairplot

In [21]:

```
sns.pairplot(data=df[["RowNumber","Age","Tenure","Balance","NumOfProducts"]
],kind="kde")
```

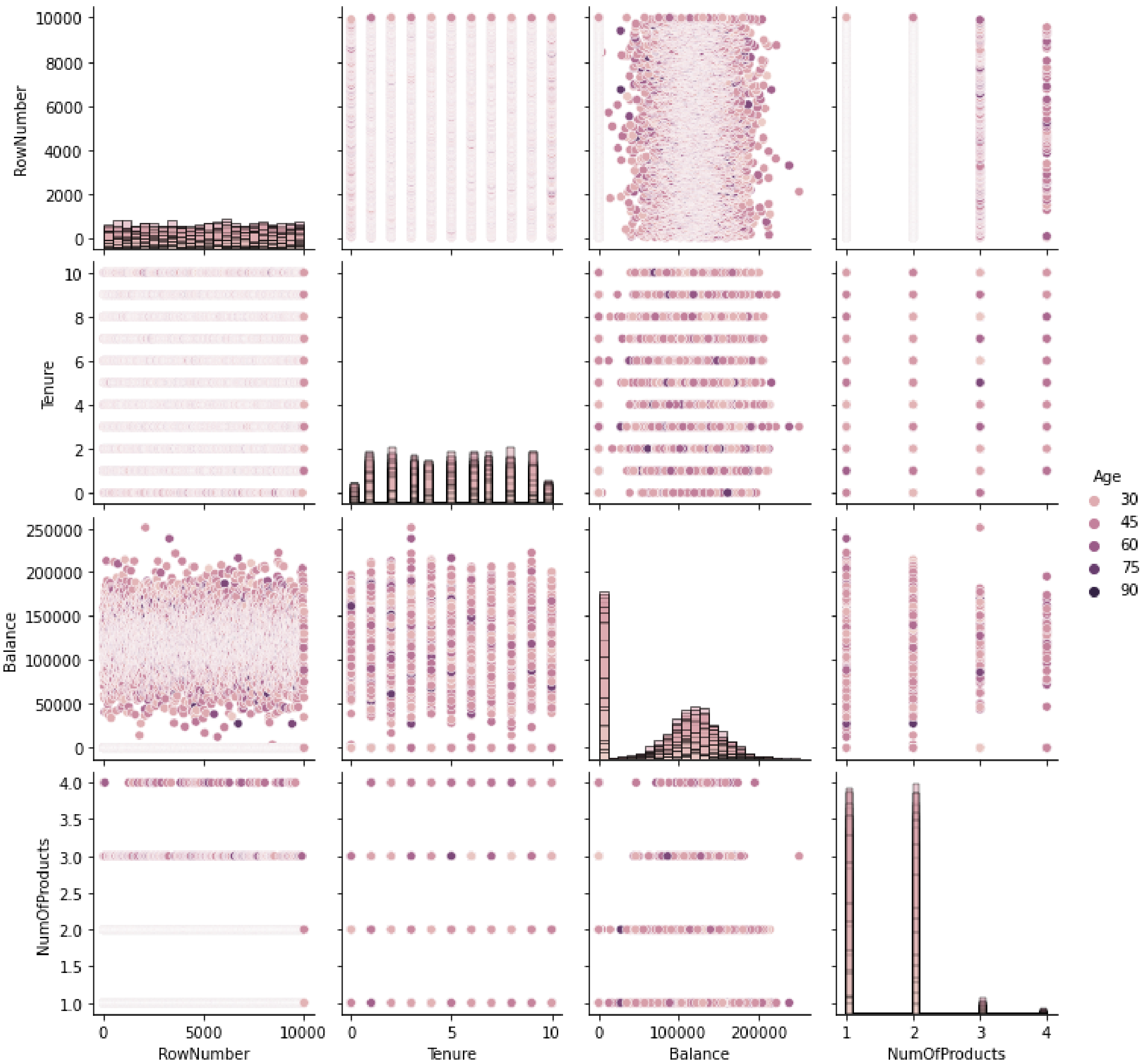
Out[21]:



In [22]:

```
sns.pairplot(data=df[["RowNumber","Age","Tenure","Balance","NumOfProducts"]
], hue="Age", diag_kind="hist")
```

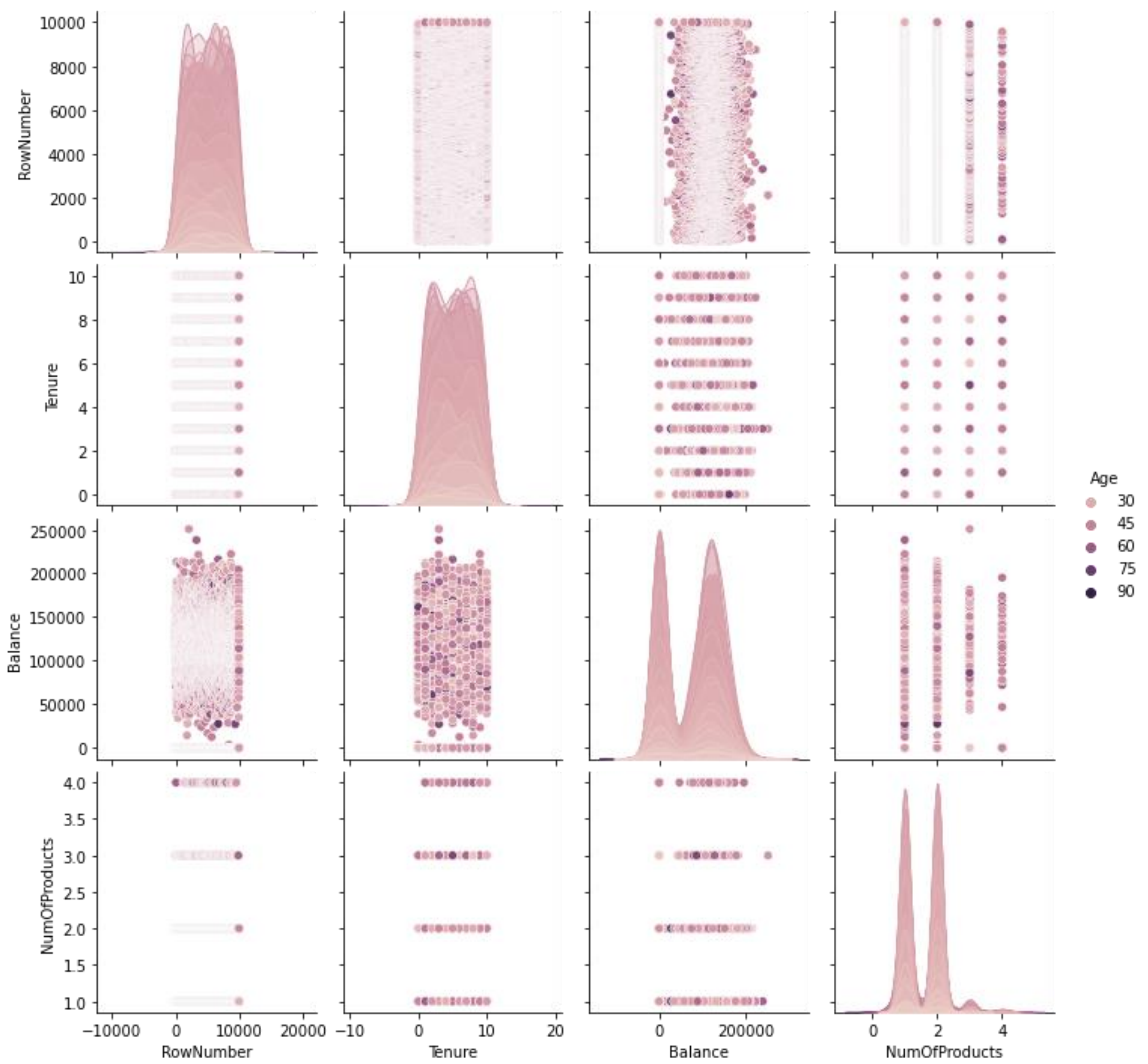
Out[22]:



In [23]:

```
sns.pairplot(data=df[["RowNumber","Age","Tenure","Balance","NumOfProducts"]], hue="Age")
```

Out[23]:



4. Perform descriptive statistics on the dataset

Image result for descriptive statistics in python Python Descriptive Statistics process describes the basic features of data in a study. It delivers summaries on the sample and the measures and does not use the data to learn about the population it represents. Under descriptive statistics, fall two sets of properties- central tendency and dispersion.

In [24]:

```
df.describe()
```

Out[24]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

In [81]:

```
df.head()
```

Out[81]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	1	0	42	2	0.00	1	1	1	101348.88	1

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
1	2	15647311	Hill	608	2	0	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	1	0	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	1	0	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	2	0	43	2	125510.82	1	1	1	79084.10	0

In [82]:

```
df.mean()  # Get the mean of each column
```

Out[82]:

```
RowNumber          5.000500e+03
CustomerId         1.569094e+07
CreditScore        6.505288e+02
Geography          1.749500e+00
Gender             5.457000e-01
Age                4.019530e+01
Tenure             5.012800e+00
Balance            7.648589e+04
NumOfProducts      1.530200e+00
HasCrCard          7.055000e-01
IsActiveMember     5.151000e-01
EstimatedSalary    1.000902e+05
Exited             2.037000e-01
dtype: float64
```

In [83]:

```
df.mean(axis=1)           # Get the mean of each row
```

Out[83]:

```
0       1.210509e+06
1       1.218794e+06
2       1.222574e+06
3       1.215071e+06
4       1.226414e+06
            ...
9995    1.208717e+06
9996    1.210733e+06
9997    1.202875e+06
9998    1.220088e+06
9999    1.215960e+06
Length: 10000, dtype: float64
```

In [84]:

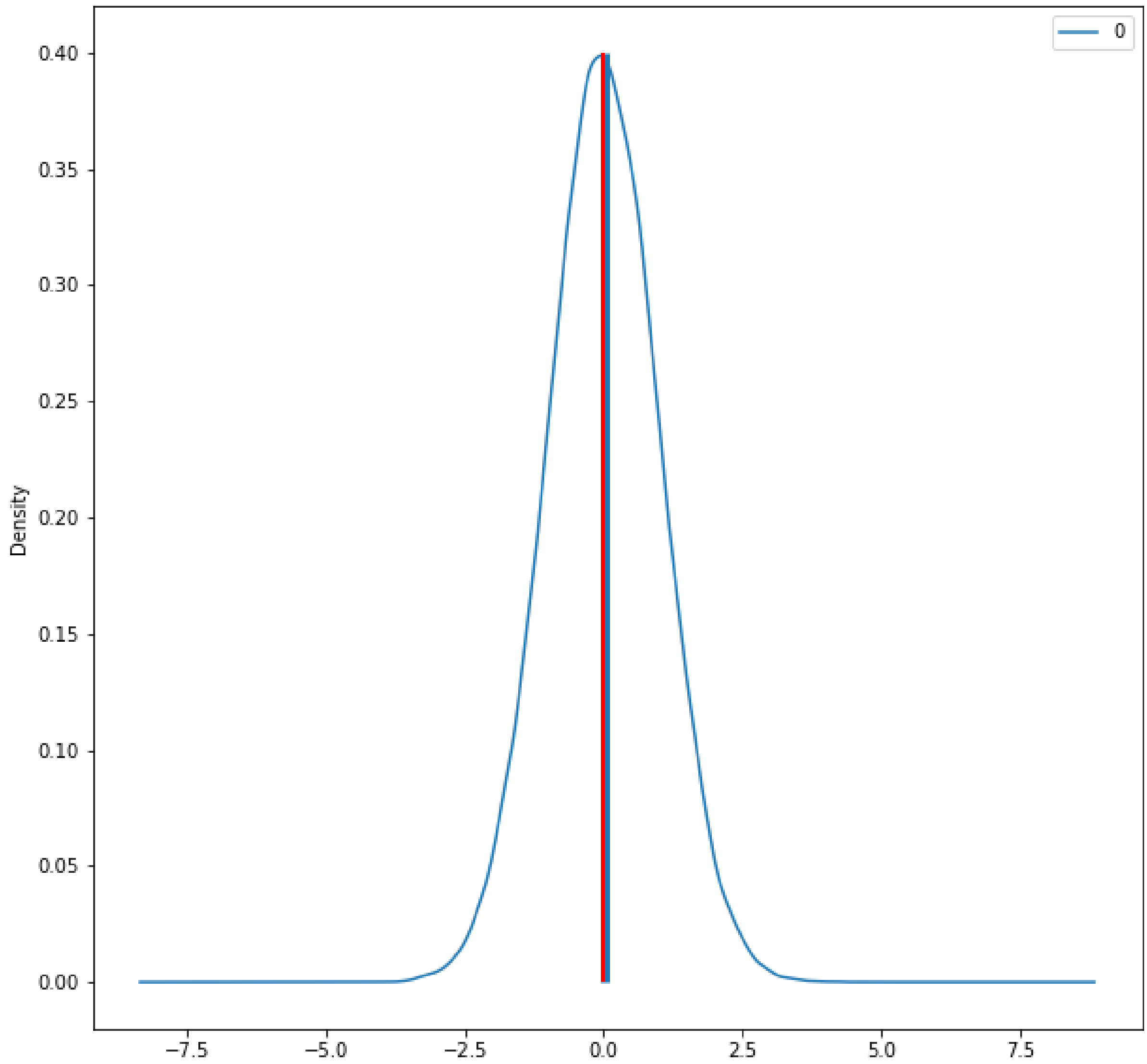
```
df.median()                  # Get the median of each column
```

Out[84]:

```
RowNumber          5.000500e+03
CustomerId         1.569074e+07
CreditScore        6.520000e+02
Geography          1.000000e+00
Gender             1.000000e+00
Age                3.800000e+01
Tenure             5.000000e+00
Balance            9.719854e+04
NumOfProducts      1.000000e+00
HasCrCard          1.000000e+00
IsActiveMember     1.000000e+00
EstimatedSalary    1.001939e+05
Exited             0.000000e+00
dtype: float64
```

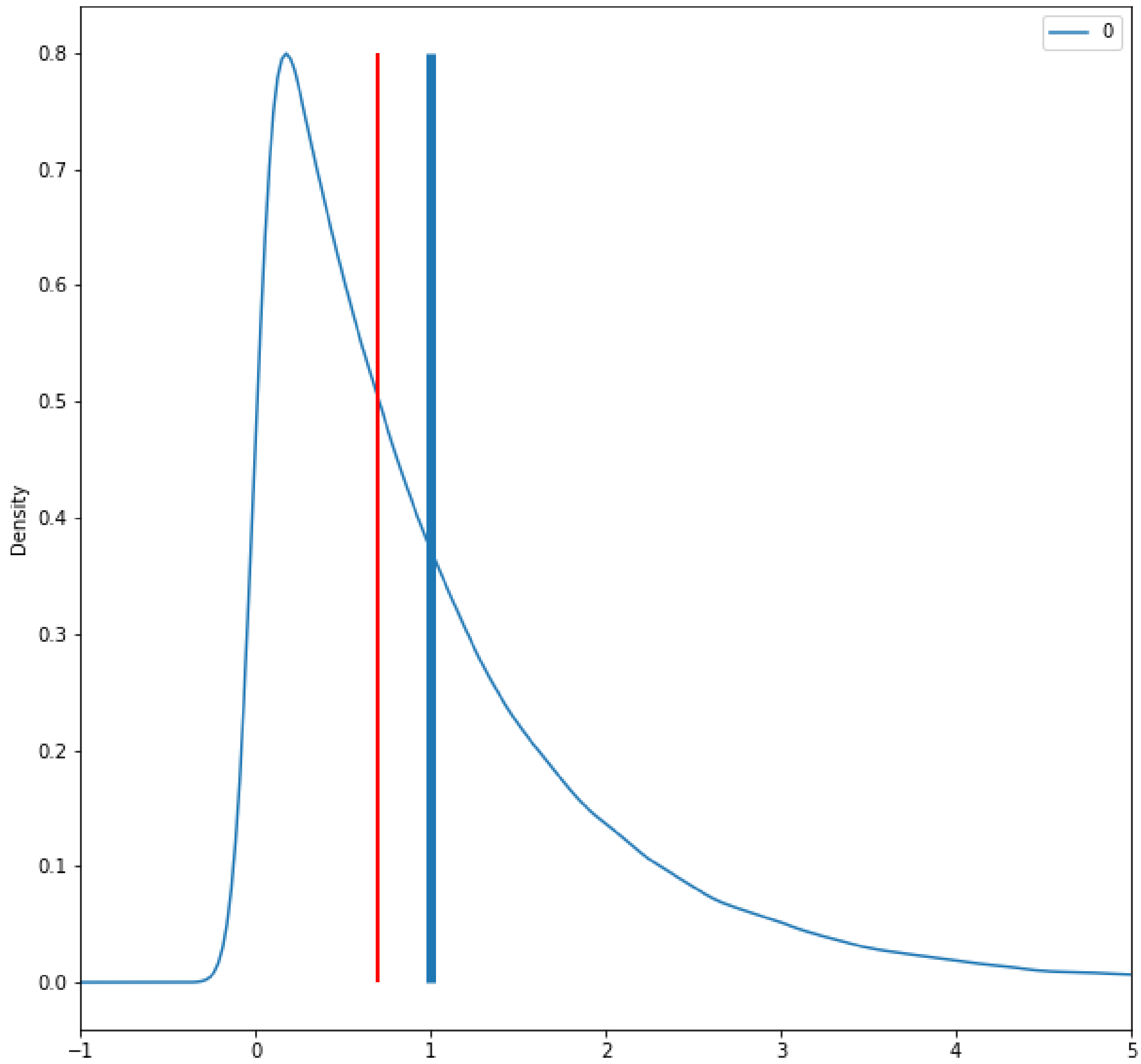
In [85]:

```
norm_data = pd.DataFrame(np.random.normal(size=100000))

norm_data.plot(kind="density",
              figsize=(10,10));
plt.vlines(norm_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.4,
           linewidth=5.0);

plt.vlines(norm_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.4,
           linewidth=2.0,
           color="red");
```



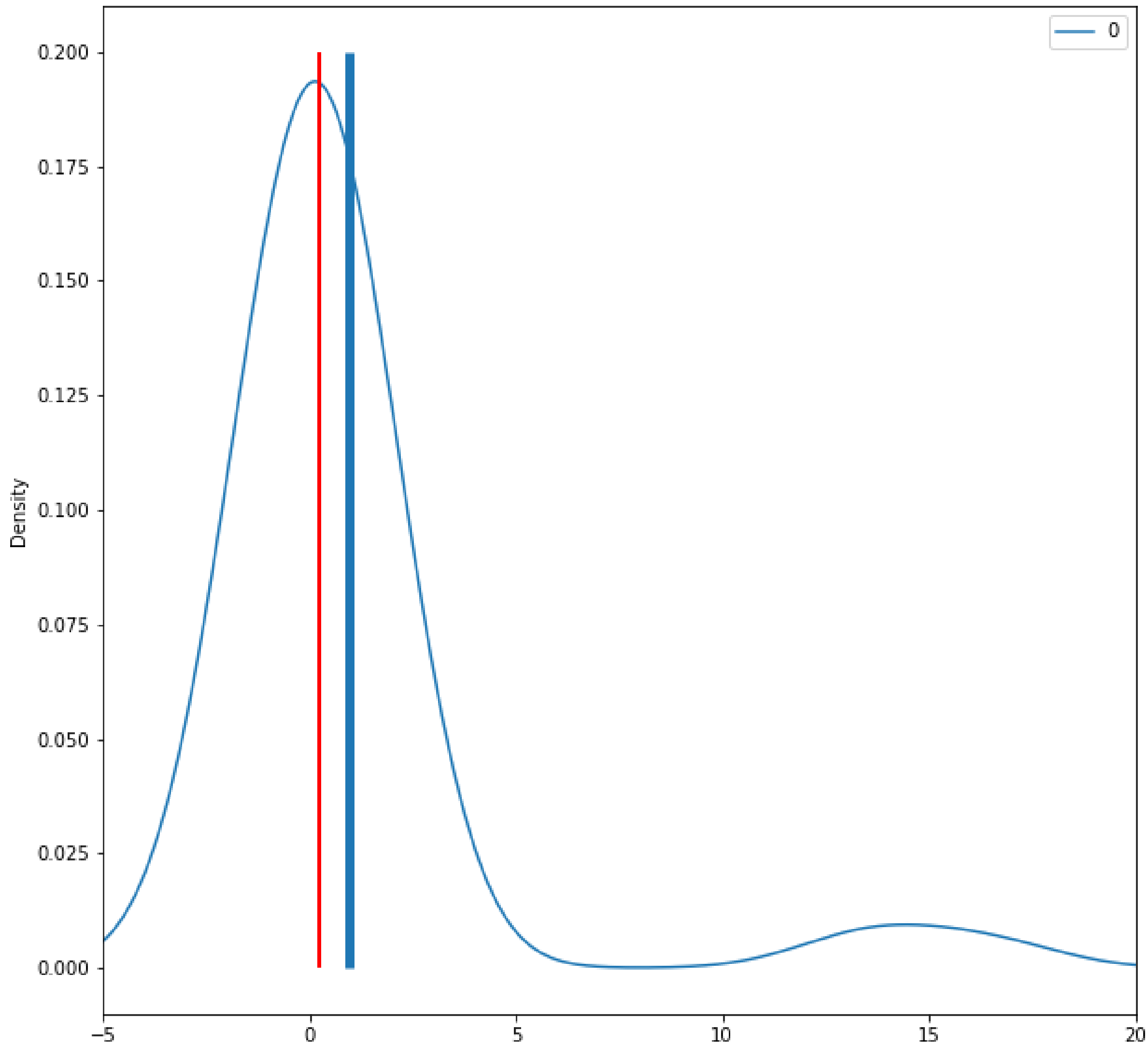
In [86]:

```
skewed_data = pd.DataFrame(np.random.exponential(size=100000))

skewed_data.plot(kind="density",
              figsize=(10,10),
              xlim=(-1,5));


plt.vlines(skewed_data.mean(),      # Plot black line at mean
           ymin=0,
           ymax=0.8,
           linewidth=5.0);

plt.vlines(skewed_data.median(),    # Plot red line at median
           ymin=0,
           ymax=0.8,
           linewidth=2.0,
           color="red");
```



In [87]:

```
norm_data = np.random.normal(size=50)
outliers = np.random.normal(15, size=3)
combined_data = pd.DataFrame(np.concatenate((norm_data, outliers), axis=0))

combined_data.plot(kind="density",
              figsize=(10,10),
              xlim=(-5,20));


plt.vlines(combined_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.2,
           linewidth=5.0);

plt.vlines(combined_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.2,
           linewidth=2.0,
           color="red");
```



In [88]:

```
df.mode()
```

Out[88]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15565701	Smith	850.0	1.0	1.0	50.0	2.0	0.0	1.0	1.0	1.0	24924.92	0.0
1	2	15565706	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	3	15565714	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
3	4	15565779	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	5	15565796	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15815628	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9996	9997	15815645	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9997	9998	15815656	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9998	9999	15815660	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9999	10000	15815690	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

10000 rows × 14 columns

Measures of Spread

In [89]:

```
max(df["Age"]) - min(df["Age"])
```

Out[89]:

```
67
```

In [90]:

```
five_num = [df["Age"].quantile(0),
            df["Age"].quantile(0.25),
            df["Age"].quantile(0.50),
```

```
            df["Age"].quantile(0.75),
            df["Age"].quantile(1)]

five_num
```

Out[90]:

```
[25.0, 33.0, 38.0, 46.0, 92.0]
```

In [91]:

```
df["Age"].describe()
```

Out[91]:

```
count    10000.000000
mean        40.195300
std         10.047729
min         25.000000
25%         33.000000
50%         38.000000
75%         46.000000
max         92.000000
Name: Age, dtype: float64
```

In [92]:

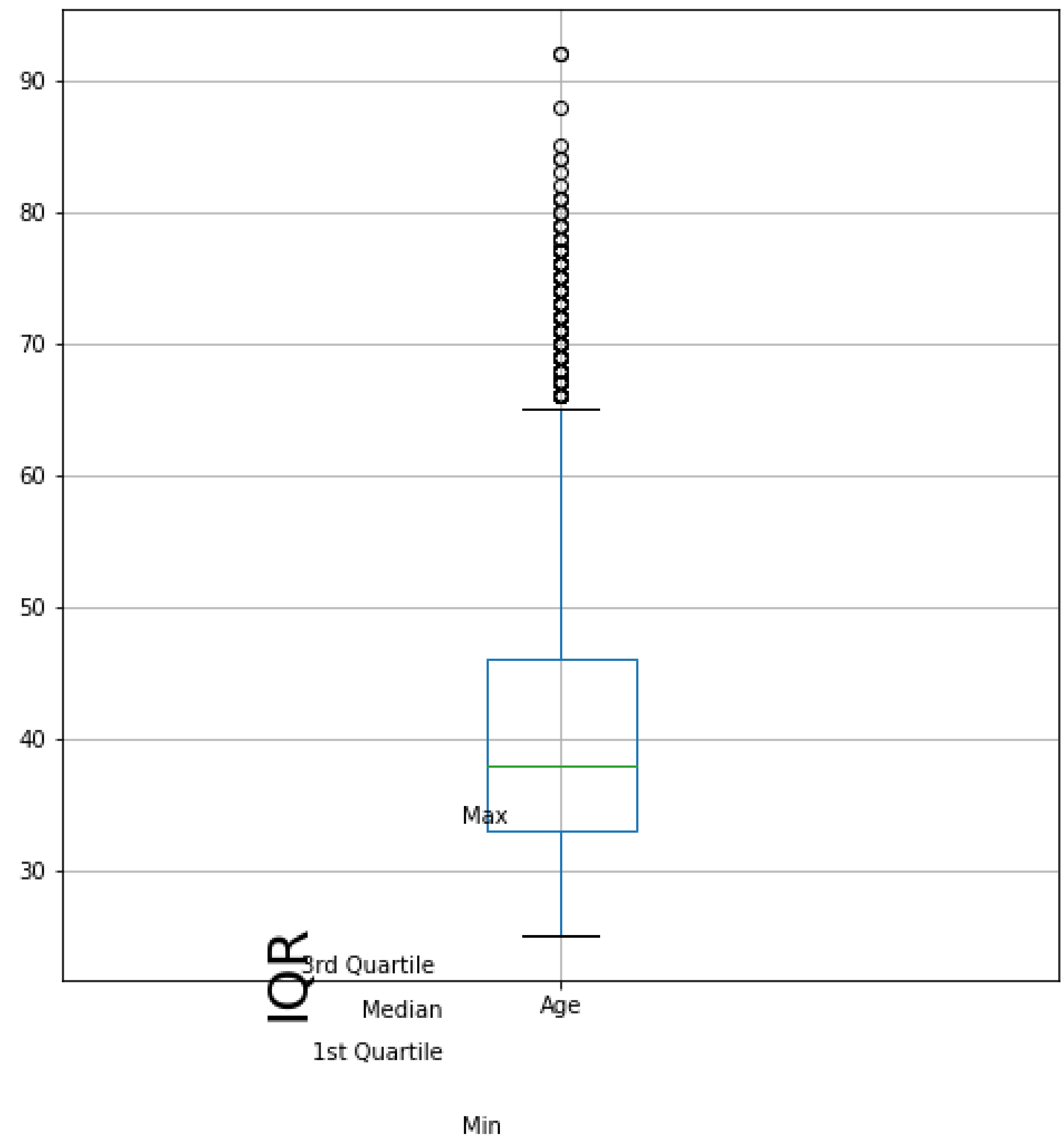
```
df["Age"].quantile(0.75) - df["Age"].quantile(0.25)
```

Out[92]:

```
13.0
```

In [93]:

```
df.boxplot(column="Age",
               return_type='axes',
               figsize=(8,8))

plt.text(x=0.74, y=22.25, s="3rd Quartile")
plt.text(x=0.8, y=18.75, s="Median")
plt.text(x=0.75, y=15.5, s="1st Quartile")
plt.text(x=0.9, y=10, s="Min")
plt.text(x=0.9, y=33.5, s="Max")
plt.text(x=0.7, y=19.5, s="IQR", rotation=90, size=25);
```



In [94]:

```
df["Age"].var()
```

Out[94]:

```
100.95685359535904
```

In [95]:

```
df["Age"].std()
```

Out[95]:

```
10.047728777955694
```

In [96]:

```
abs_median_devs = abs(df["Age"] - df["Age"].median())

abs_median_devs.median() * 1.4826
```

Out[96]:

```
8.8956
```

Skewness and Kurtosis

In [97]:

```
df["Age"].skew()  # Check skewness
```

Out[97]:

```
1.0495460120728233
```

In [98]:

```
df["Age"].kurt()  # Check kurtosis
```

Out[98]:

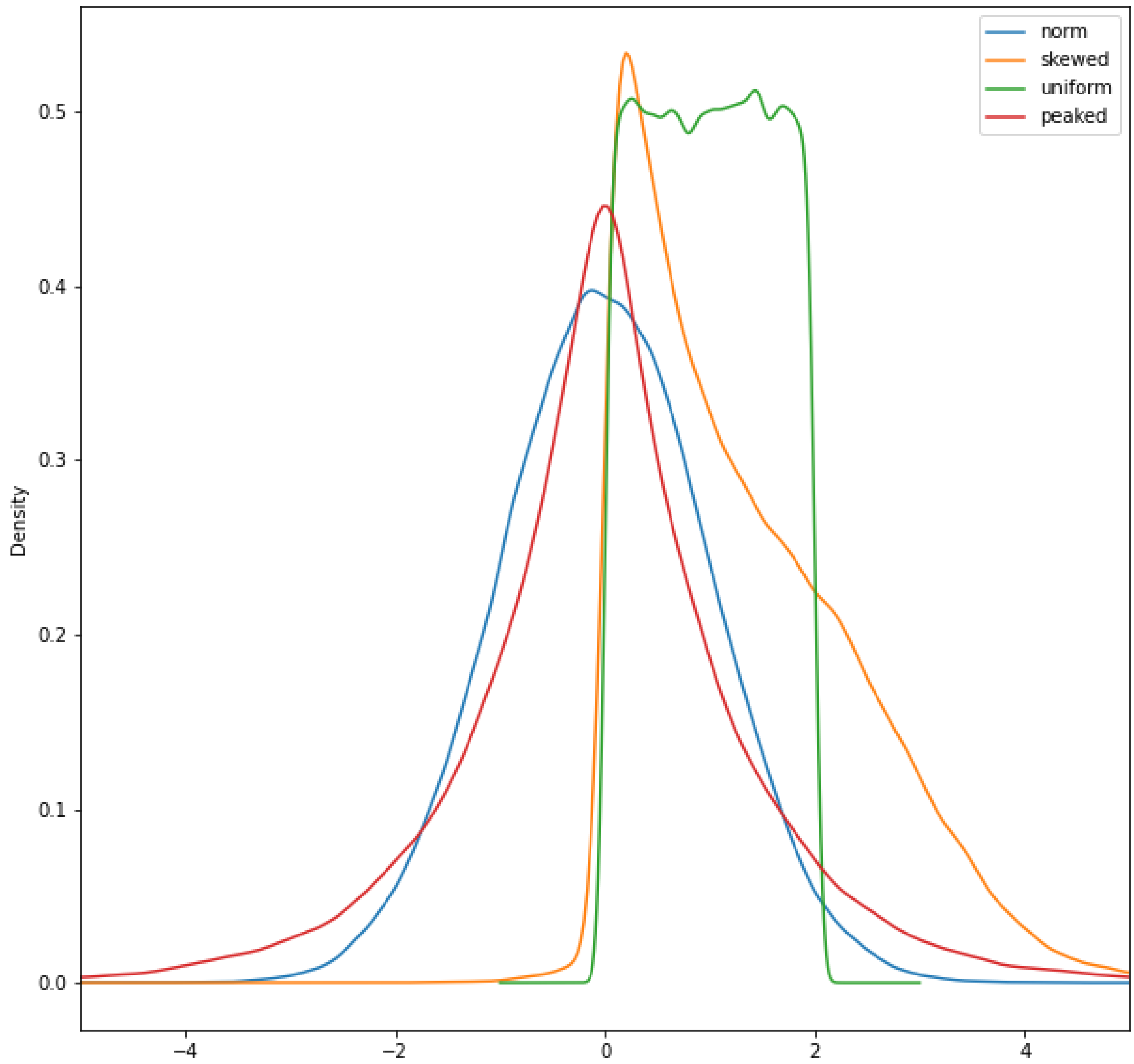
```
1.2747003702904487
```

In [99]:

```
norm_data = np.random.normal(size=100000)
skewed_data = np.concatenate((np.random.normal(size=35000)+2,
                             np.random.exponential(size=65000)),
                             axis=0)
uniform_data = np.random.uniform(0,2, size=100000)
peaked_data = np.concatenate((np.random.exponential(size=50000),
                             np.random.exponential(size=50000)*(-1)),
                             axis=0)

data_df = pd.DataFrame({"norm":norm_data,
                       "skewed":skewed_data,
                       "uniform":uniform_data,
                       "peaked":peaked_data})
```

In [100]:

```
data_df.plot(kind="density",
            figsize=(10,10),
            xlim=(-5,5));
```



In [101]:

```
data_df.skew()
```

Out[101]:

```
norm       0.009355
skewed     1.019859
uniform   -0.003685
peaked    -0.010567
dtype: float64
```

In [102]:

```
data_df.kurt()
```

Out[102]:

```
norm       0.019495
skewed     1.526498
uniform   -1.201582
peaked     3.080928
dtype: float64
```

5. Handle the Missing values.

In [103]:

```
df.head()
```

Out[103]:

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

In [104]:

```
df.isnull()
```

Out[104]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False

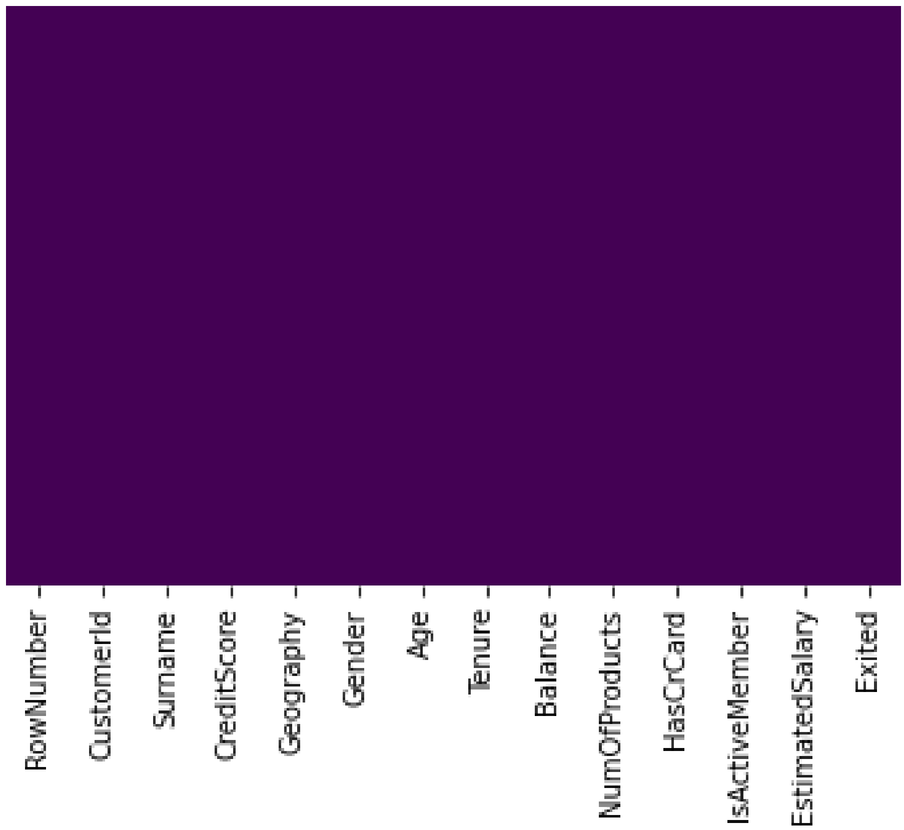
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9996	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9997	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9998	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9999	False	False	False	False	False	False	False	False	False	False	False	False	False	False

10000 rows × 14 columns

In [105]:

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

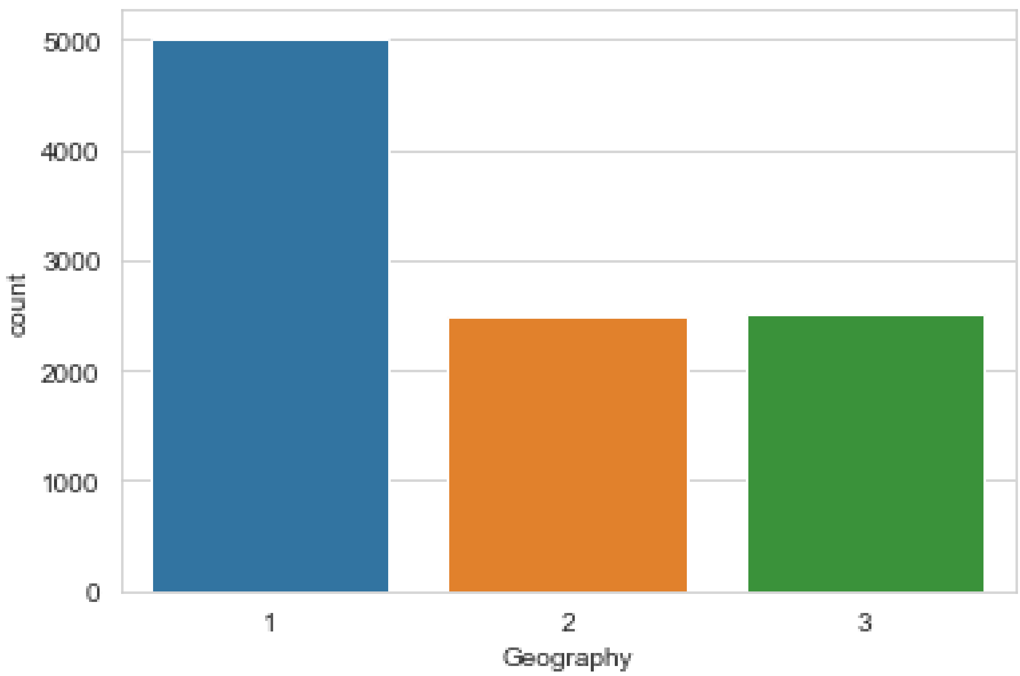
Out[105]:



In [106]:

```
sns.set_style('whitegrid')
sns.countplot(x='Geography',data=df)
```

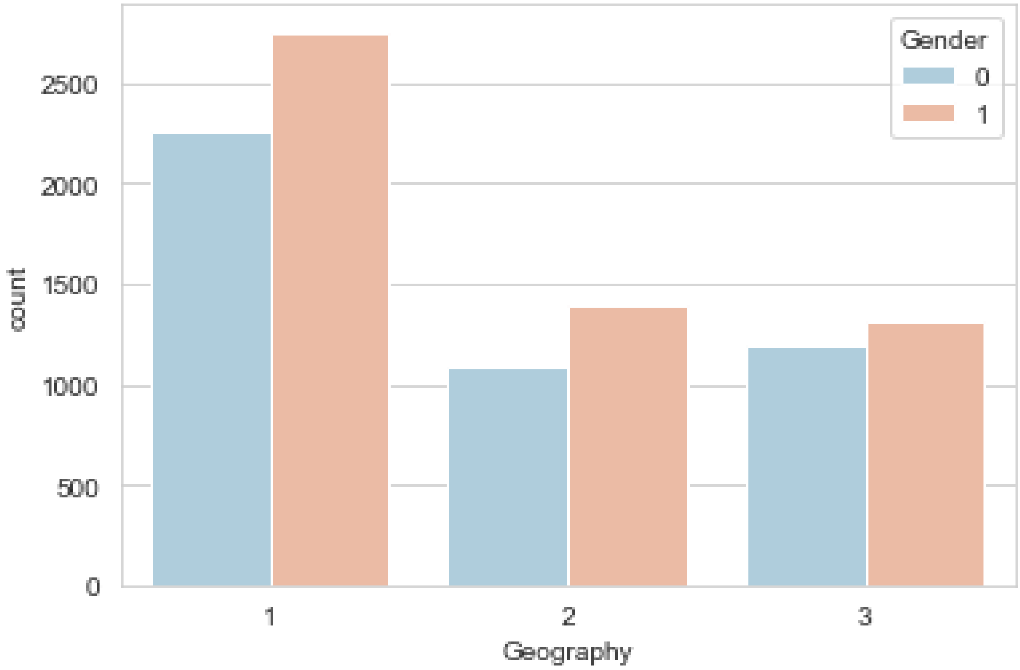
Out[106]:



In [107]:

```
sns.set_style('whitegrid')
sns.countplot(x='Geography',hue='Gender',data=df,palette='RdBu_r')
```

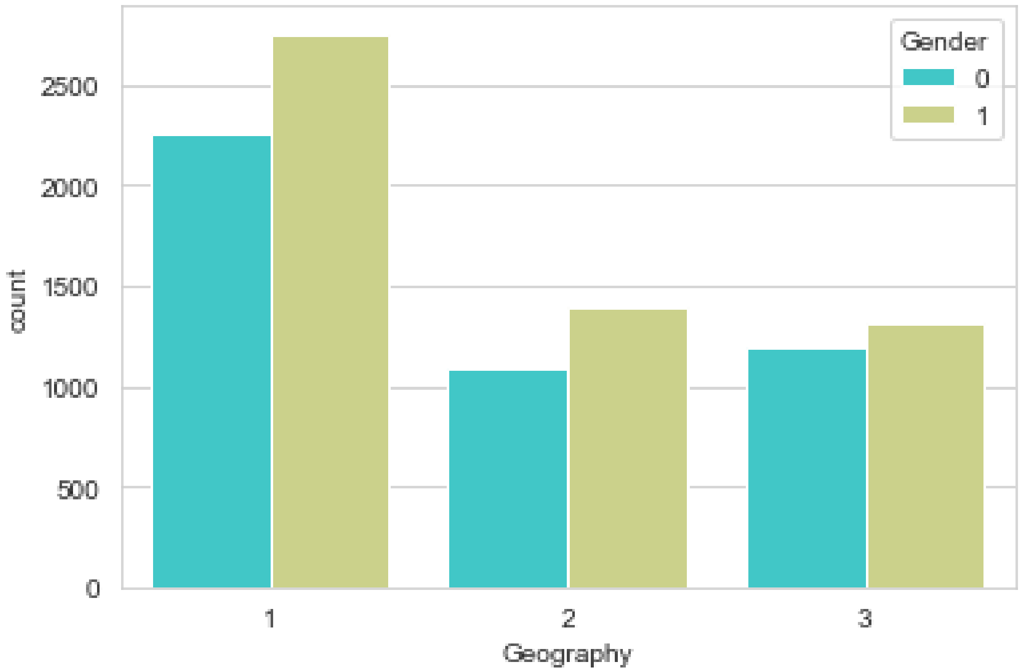
Out[107]:



In [108]:

```
sns.set_style('whitegrid')
sns.countplot(x='Geography',hue='Gender',data=df,palette='rainbow')
```

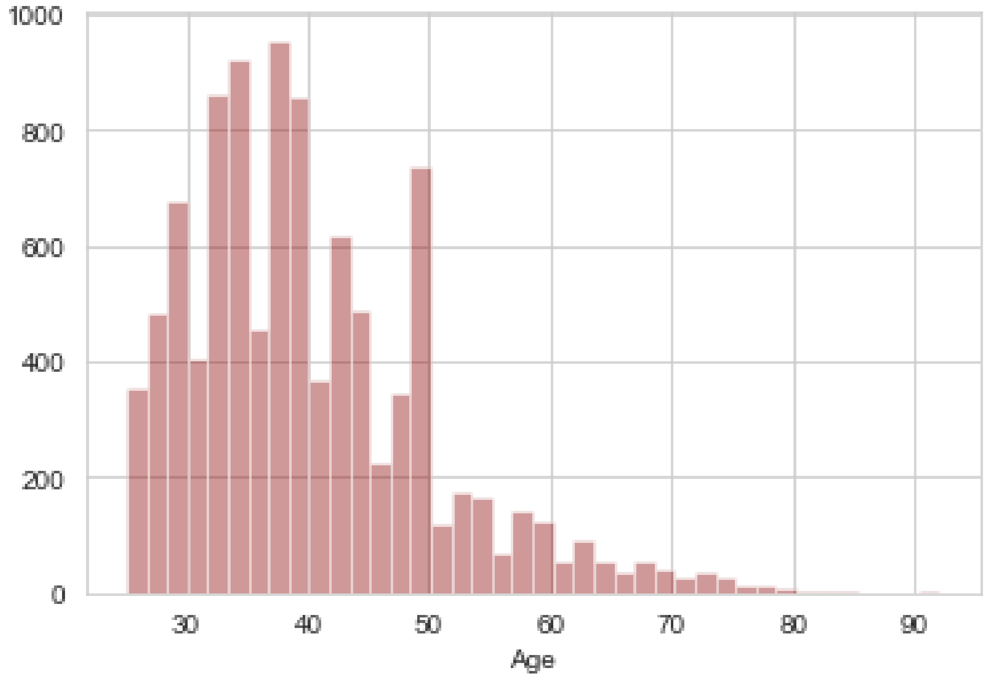
Out[108]:



In [109]:

```
sns.distplot(df['Age'].dropna(),kde=False,color='darkred',bins=40)
```

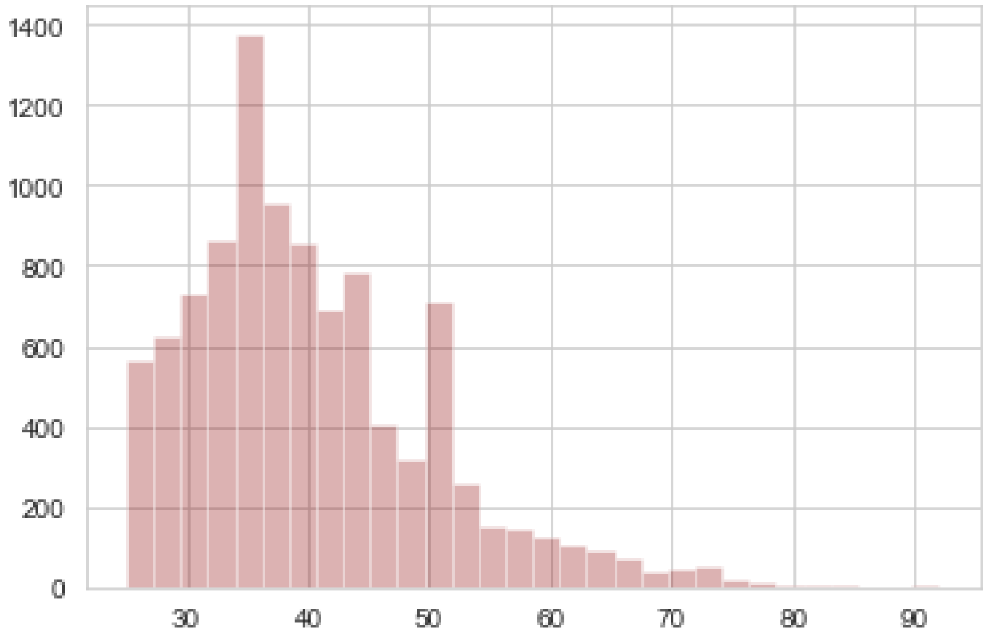
Out[109]:



In [110]:

```
df['Age'].hist(bins=30,color='darkred',alpha=0.3)
```

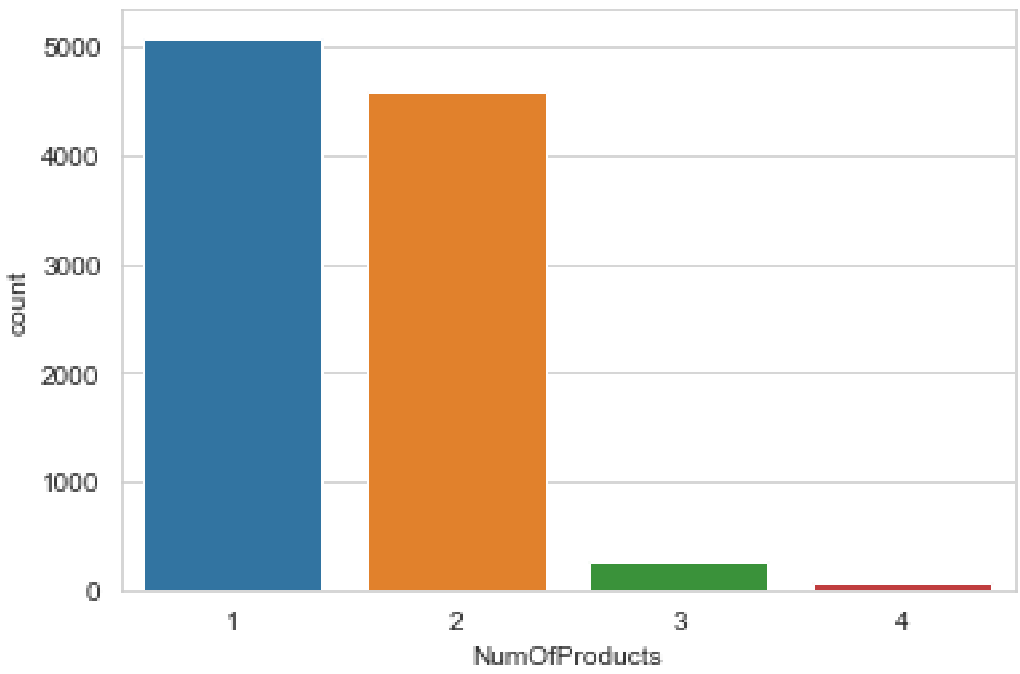
Out[110]:



In [111]:

```
sns.countplot(x='NumOfProducts',data=df)
```

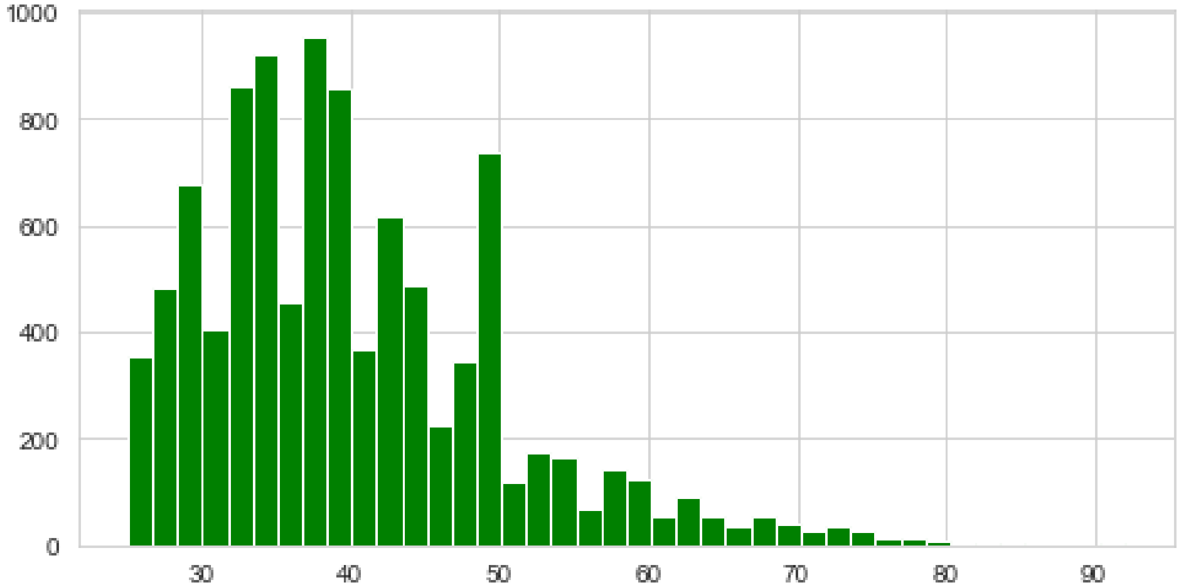
Out[111]:



In [112]:

```
df['Age'].hist(color='green',bins=40,figsize=(8,4))
```

Out[112]:



Cufflinks for plots

In [113]:

```
import cufflinks as cf
cf.go_offline()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Input In [113], in ()
----> 1 import cufflinks as cf
      2 cf.go_offline()
```

```
ModuleNotFoundError: No module named 'cufflinks'
```

In [114]:

```
df['Age'].iplot(kind='hist',bins=30,color='green')
```

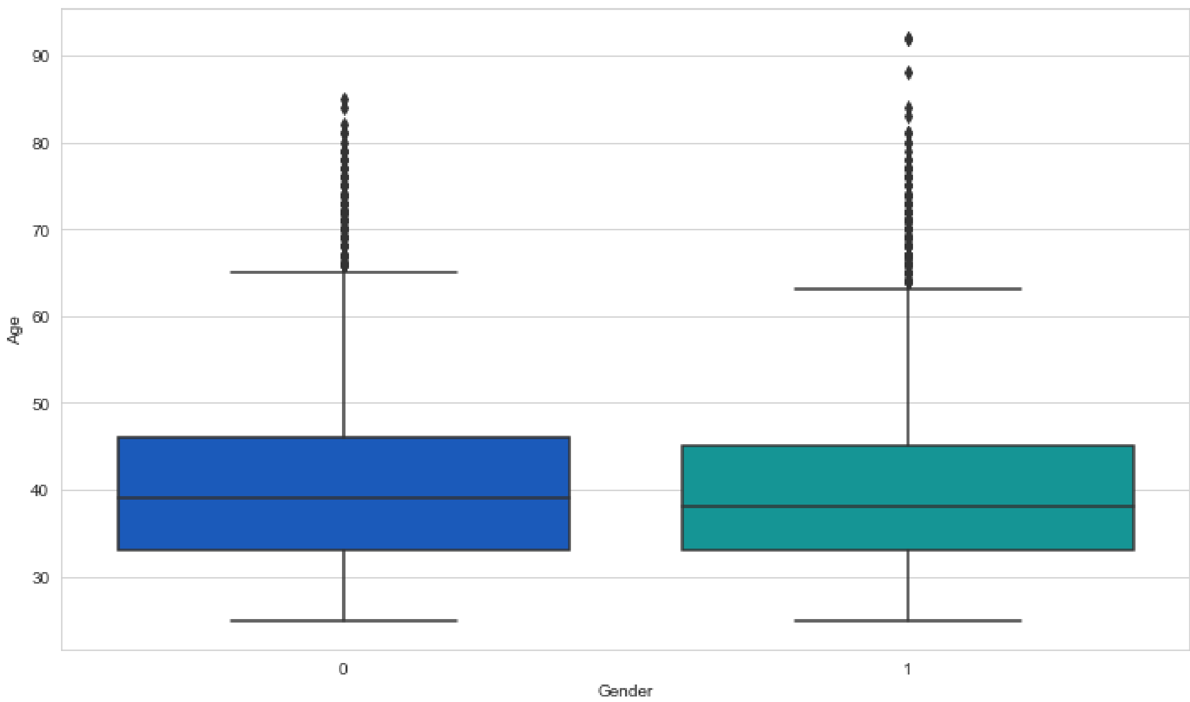
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [114], in ()
----> 1 df['Age'].iplot(kind='hist',bins=30,color='green')

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:5575, in NDFrame.
__getattr__(self, name)
   5568 if (
   5569     name not in self._internal_names_set
   5570     and name not in self._metadata
   5571     and name not in self._accessors
   5572     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   5573 ):
   5574     return self[name]
-> 5575 return object.__getattribute__(self, name)

AttributeError: 'Series' object has no attribute 'iplot'
```

Data Cleaning

In [115]:

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Gender',y='Age',data=df,palette='winter')
```

Out[115]:



In [116]:

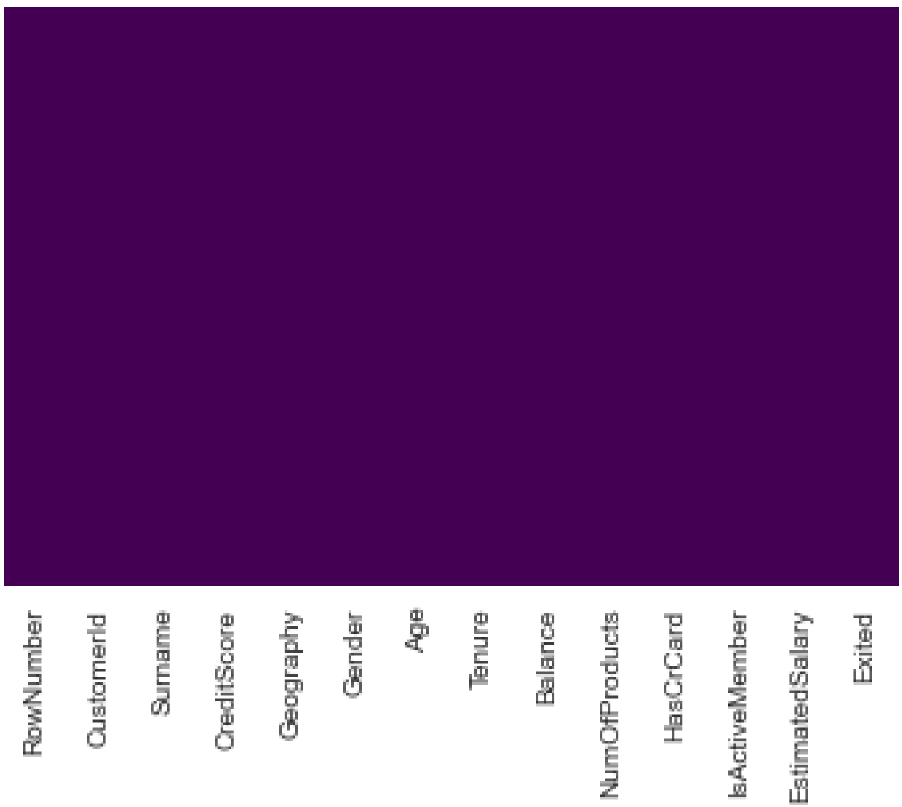
```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):
```

```
        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

In [117]:

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[117]:



In [118]:

```
df.drop('Gender',axis=1,inplace=True)
```

In [119]:

```
df.head()
```

Out[119]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	1	42	2	0.00	1	1	1	101348.88	1

	RowNumber	CustomerId	Surname	CreditScore	Geography	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
1	2	15647311	Hill	608	2	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	1	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	1	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	2	43	2	125510.82	1	1	1	79084.10	0

Converting Categorical Features

In [120]:

```
df.info()
```

```
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 13 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  int64  
 5   Age              10000 non-null  int64  
 6   Tenure           10000 non-null  int64  
 7   Balance          10000 non-null  float64
 8   NumOfProducts    10000 non-null  int64  
 9   HasCrCard        10000 non-null  int64  
 10  IsActiveMember   10000 non-null  int64  
 11  EstimatedSalary  10000 non-null  float64
 12  Exited           10000 non-null  int64  
dtypes: float64(2), int64(10), object(1)
memory usage: 1015.8+ KB
```

In [121]:

```
pd.get_dummies(df['Geography'],drop_first=True).head()
```

Out[121]:

	2	3
0	0	0

	2	3
1	1	0
2	0	0
3	0	0
4	1	0

In [122]:

```
df.info
```

Out[122]:

In [123]:

```
sex = pd.get_dummies(df['Age'],drop_first=True)
embark = pd.get_dummies(df['Balance'],drop_first=True)
```

In [124]:

```
df.drop(['Age','HasCrCard','Surname','CustomerId'],axis=1,inplace=True)
```

In [125]:

```
df.head()
```

Out[125]:

	RowNumber	CreditScore	Geography	Tenure	Balance	NumOfProducts	IsActiveMember	EstimatedSalary	Exited
0	1	619	1	2	0.00	1	1	101348.88	1
1	2	608	2	1	83807.86	1	1	112542.58	0
2	3	502	1	8	159660.80	3	0	113931.57	1
3	4	699	1	1	0.00	2	0	93826.63	0
4	5	850	2	2	125510.82	1	1	79084.10	0

In [126]:

```
train = pd.concat([df,sex,embark],axis=1)
```

In [127]:

```
train.head()
```

Out[127]:

	RowNumber	CreditScore	Geography	Tenure	Balance	NumOfProducts	IsActiveMember	EstimatedSalary	Exited	26	...	212692.97	212696.32	212778.2	213146.2	214346.96	216109.88	221532.8	222267.63	238387.56	250898.09
0	1	619	1	2	0.00	1	1	101348.88	1	0	...	0	0	0	0	0	0	0	0	0	0
1	2	608	2	1	83807.86	1	1	112542.58	0	0	...	0	0	0	0	0	0	0	0	0	0
2	3	502	1	8	159660.80	3	0	113931.57	1	0	...	0	0	0	0	0	0	0	0	0	0
3	4	699	1	1	0.00	2	0	93826.63	0	0	...	0	0	0	0	0	0	0	0	0	0
4	5	850	2	2	125510.82	1	1	79084.10	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 6452 columns

In [128]:

```
data=pd.DataFrame({"a":[1,2,np.nan],"b":[1,np.nan,np.nan],"c":[1,2,4]})
data
```

Out[128]:

	a	b	c
0	1.0	1.0	1

	a	b	c
1	2.0	NaN	2
2	NaN	NaN	4

In [129]:

```
data.isnull().any()
```

Out[129]:

```
a     True
b     True
c    False
dtype: bool
```

In [130]:

```
data.isnull().sum()
```

Out[130]:

```
a    1
b    2
c    0
dtype: int64
```

In [131]:

```
data.fillna(value = "S")
```

Out[131]:

	a	b	c
0	1.0	1.0	1
1	2.0	S	2
2	S	S	4

In [132]:

```
data["a"].mean()
```

Out[132]:

```
1.5
```

In [133]:

```
data["a"].median()
```

Out[133]:

```
1.5
```

6. Find the outliers and replace the outliers

For data that follows a normal distribution, the values that fall more than three standard deviations from the mean are typically considered outliers. Outliers can find their way into a dataset naturally through variability, or they can be the result of issues like human error, faulty equipment, or poor sampling.

In [68]:

```
dataset= [11,10,12,14,12,15,14,13,15,102,12,14,17,19,107,
10,13,12,14,12,108,12,11,14,13,15,10,15,12,10,14,13,15,10]
```

Detecting outlier using Z score

Using Z score

In [69]:

```
outliers=[]
def detect_outliers(data):

    threshold=3
    mean = np.mean(data)
    std =np.std(data)


    for i in data:
        z_score= (i - mean)/std
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

In [70]:

```
outlier_pt=detect_outliers(dataset)
```

In [71]:

```
outlier_pt
```

Out[71]:

```
[0        Hargrave
 1            Hill
 2            Onio
 3            Boni
 4        Mitchell
           ...
 9995     Obijiaku
 9996    Johnstone
 9997          Liu
 9998    Sabbatini
 9999       Walker
 Name: Surname, Length: 10000, dtype: object,
 0        Hargrave
 1            Hill
 2            Onio
 3            Boni
 4        Mitchell
           ...
 9995     Obijiaku
 9996    Johnstone
 9997          Liu
 9998    Sabbatini
```

```
 9999       Walker
 Name: Surname, Length: 10000, dtype: object,
 0        Hargrave
 1            Hill
 2            Onio
 3            Boni
 4        Mitchell
           ...
 9995     Obijiaku
 9996    Johnstone
 9997          Liu
 9998    Sabbatini
 9999       Walker
 Name: Surname, Length: 10000, dtype: object]
```

In [72]:

```
## Perform all the steps of IQR
sorted(dataset)
```

Out[72]:

```
[10,
 10,
 10,
 10,
 10,
 11,
 11,
 12,
 12,
 12,
 12,
 12,
 12,
 12,
 13,
 13,
 13,
 13,
 14,
 14,
 14,
 14,
 14,
 14,
 15,
 15,
 15,
 15,
 15,
 17,
 19,
 102,
 107,
 108]
```

In [73]:

```
quantile1, quantile3= np.percentile(dataset,[25,75])
```

In [74]:

```
print(quantile1,quantile3)
```

```
12.0 15.0
```

In [75]:

```
## Find the IQR

iqr_value=quantile3-quantile1
print(iqr_value)
```

```
3.0
```

In [76]:

```
## Find the lower bound value and the higher bound value

lower_bound_val = quantile1 -(1.5 * iqr_value)
upper_bound_val = quantile3 +(1.5 * iqr_value)
```

In [77]:

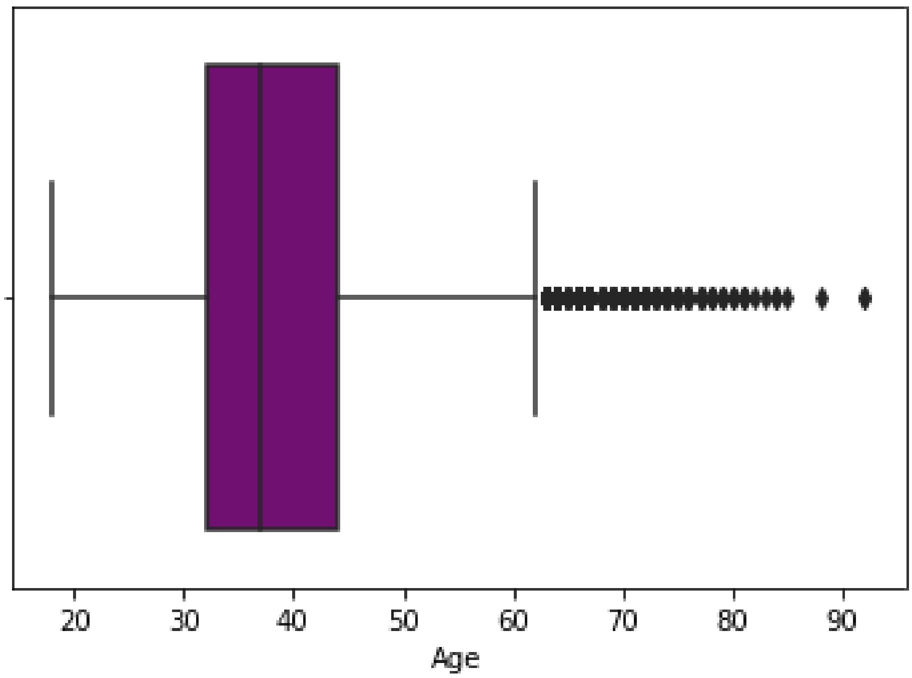
```
print(lower_bound_val,upper_bound_val)
```

```
7.5 19.5
```

In [46]:

```
sns.boxplot(df["Age"],color='purple')
```

Out[46]:



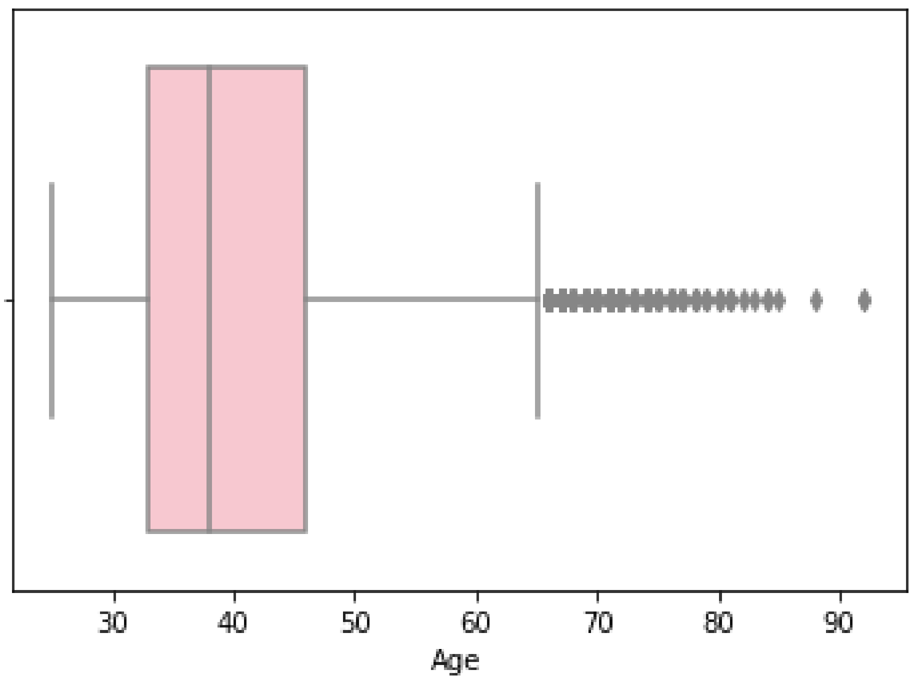
In [47]:

```
df["Age"]=np.where(df["Age"]<25,50,df["Age"])
```

In [48]:

```
sns.boxplot(df["Age"],color='pink')
```

Out[48]:



7. Check for Categorical columns and perform encoding.

Categorical Columns : Categorical are a Pandas data type. A string variable consisting of only a few different values.

Encoding : For efficient storage of these strings, the sequence of code points is converted into a set of bytes. The process is known as encoding.

In [139]:

```
df=pd.read_csv("E:\IBM projects Assignment Sona College\Churn_Modelling.csv")
```

In [140]:

```
df.head()
```

Out[140]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

In [141]:

```
df_numeric = df[['RowNumber', 'CustomerId', 'CreditScore', 'Age', 'Tenure', 'Balance',
'NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Exited']]
df_categorical = df[['Surname', 'Geography', 'Gender']]
```

In [142]:

```
df_numeric.head()
```

Out[142]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	619	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	608	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	502	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	699	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	850	43	2	125510.82	1	1	1	79084.10	0

In [143]:

```
df_categorical.head()
```

Out[143]:

	Surname	Geography	Gender
0	Hargrave	France	Female
1	Hill	Spain	Female
2	Onio	France	Female
3	Boni	France	Female
4	Mitchell	Spain	Female

In [144]:

```
print(df['Surname'].unique())
print(df['Geography'].unique())
print(df['Gender'].unique())
```

```
['Hargrave' 'Hill' 'Onio' ... 'Kashiwagi' 'Aldridge' 'Burbidge']
['France' 'Spain' 'Germany']
['Female' 'Male']
```

In [145]:

```
from sklearn.preprocessing import LabelEncoder

marry_encoder = LabelEncoder()
```

In [146]:

```
marry_encoder.fit(df_categorical['Gender'])
```

Out[146]:

```
LabelEncoder()
```

In [147]:

```
marry_values = marry_encoder.transform(df_categorical['Gender'])
```

In [148]:

```
print("Before Encoding:", list(df_categorical['Gender'][-10:]))
print("After Encoding:", marry_values[-10:])
print("The inverse from the encoding result:",
marry_encoder.inverse_transform(marry_values[-10:]))
```

```
Before Encoding: ['Male', 'Female', 'Male', 'Male', 'Female', 'Male', 'Male
', 'Female', 'Male', 'Female']
After Encoding: [1 0 1 1 0 1 1 0 1 0]
The inverse from the encoding result: ['Male' 'Female' 'Male' 'Male' 'Femal
e' 'Male' 'Male' 'Female' 'Male'
 'Female']
```

In [149]:

```
residence_encoder = LabelEncoder()
residence_values =
residence_encoder.fit_transform(df_categorical['Geography'])

print("Before Encoding:", list(df_categorical['Geography'][:5]))
print("After Encoding:", residence_values[:5])
print("The inverse from the encoding result:",
residence_encoder.inverse_transform(residence_values[:5]))
```

```
Before Encoding: ['France', 'Spain', 'France', 'France', 'Spain']
After Encoding: [0 2 0 0 2]
The inverse from the encoding result: ['France' 'Spain' 'France' 'France' '
Spain']
```

In [150]:

```python
from sklearn.preprocessing import OneHotEncoder

gender_encoder = OneHotEncoder()
```

In [151]:

```python
from sklearn.preprocessing import OneHotEncoder
import numpy as np

gender_encoder = OneHotEncoder()
gender_reshaped = np.array(df_categorical['Gender']).reshape(-1, 1)
gender_values = gender_encoder.fit_transform(gender_reshaped)

print(df_categorical['Gender'][:5])
print()
print(gender_values.toarray()[:5])
print()
print(gender_encoder.inverse_transform(gender_values)[:5])
```

```
0    Female
1    Female
2    Female
3    Female
4    Female
Name: Gender, dtype: object

[[1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]

[['Female']
 ['Female']
 ['Female']
 ['Female']
 ['Female']]
```

In [152]:

```python
smoke_encoder = OneHotEncoder()
smoke_reshaped = np.array(df_categorical['Surname']).reshape(-1, 1)
smoke_values = smoke_encoder.fit_transform(smoke_reshaped)

print(df_categorical['Surname'][:5])
print()
print(smoke_values.toarray()[:5])
print()
print(smoke_encoder.inverse_transform(smoke_values)[:5])
```

```
0    Hargrave
1        Hill
2        Onio
3        Boni
4    Mitchell
Name: Surname, dtype: object
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

[['Hargrave']
 ['Hill']
 ['Onio']
 ['Boni']
 ['Mitchell']]
```

In [153]:

```
work_encoder = OneHotEncoder()
work_reshaped = np.array(df_categorical['Geography']).reshape(-1, 1)
work_values = work_encoder.fit_transform(work_reshaped)

print(df_categorical['Geography'][:5])
print()
print(work_values.toarray()[:5])
print()
print(work_encoder.inverse_transform(work_values)[:5])
```

```
0    France
1     Spain
2    France
3    France
4     Spain
Name: Geography, dtype: object

[[1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]

[['France']
 ['Spain']
 ['France']
 ['France']
 ['Spain']]
```

In [154]:

```
df_categorical_encoded = pd.get_dummies(df_categorical, drop_first=True)
df_categorical_encoded.head()
```

Out[154]:

	Surname_Abbie	Surname_Abbott	Surname_Abdullah	Surname_Abdulov	Surname_Abel	Surname_Abernathy	Surname_Abramov	Surname_Abramova	Surname_Abramovich	Surname_Abramowitz	...	Surname_Zotova	Surname_Zox	Surname_Zubarev	Surname_Zubareva	Surname_Zuev	Surname_Zuyev	Surname_Zuyeva	Geography_Germany	Geography_Spain	Gender_Male
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0

5 rows × 2934 columns

In [155]:

```
df_new = pd.concat([df_numeric, df_categorical_encoded], axis=1)
df_new.head()
```

Out[155]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	...	Surname_Zotova	Surname_Zox	Surname_Zubarev	Surname_Zubareva	Surname_Zuev	Surname_Zuyev	Surname_Zuyeva	Geography_Germany	Geography_Spain	Gender_Male
0	1	1563	619	42	2	0.	1	1	1	101348	...	0	0	0	0	0	0	0	0	0	0

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	...	Surname_Zotova	Surname_Zox	Surname_Zubarev	Surname_Zubareva	Surname_Zuev	Surname_Zuyev	Surname_Zuyeva	Geography_Germany	Geography_Spain	Gender_Male
		4602				00				.88											
1	2	15647311	608	41	1	83807.86	1	0	1	112542.58	...	0	0	0	0	0	0	0	0	1	0
2	3	15619304	502	42	8	159660.80	3	1	0	113931.57	...	0	0	0	0	0	0	0	0	0	0
3	4	15701354	699	39	1	0.00	2	0	0	93826.63	...	0	0	0	0	0	0	0	0	0	0
4	5	15737888	850	43	2	125510.82	1	1	1	79084.10	...	0	0	0	0	0	0	0	0	1	0

5 rows × 2945 columns

8. Split the data into dependent and independent variables.

Dependent Variable : A dependent variable is a variable whose value depends on another variable.

Independent Variable : An Independent variable is a variable whose value never depends on another variable but the researcher.

In [156]:

```
df=pd.read_csv("E:\IBM projects Assignment Sona College\Churn_Modelling.csv")
```

In [157]:

```
print(df["Balance"].min())
print(df["Balance"].max())
print(df["Balance"].mean())
```

```
0.0
250898.09
76485.88928799961
```

In [158]:

```
print(df.count(0))
```

```
RowNumber          10000
CustomerId         10000
Surname            10000
CreditScore        10000
Geography          10000
Gender             10000
Age                10000
Tenure             10000
Balance            10000
NumOfProducts      10000
HasCrCard          10000
IsActiveMember     10000
EstimatedSalary    10000
Exited             10000
dtype: int64
```

In [159]:

```
print(df.shape)
```

```
(10000, 14)
```

In [160]:

```
print(df.size)
```

```
140000
```

In [161]:

```
X = df.iloc[:, :-1].values
print(X)
```

```
[[1 15634602 'Hargrave' ... 1 1 101348.88]
 [2 15647311 'Hill' ... 0 1 112542.58]
 [3 15619304 'Onio' ... 1 0 113931.57]
 ...
 [9998 15584532 'Liu' ... 0 1 42085.58]
 [9999 15682355 'Sabbatini' ... 1 0 92888.52]
 [10000 15628319 'Walker' ... 1 0 38190.78]]
```

In [162]:

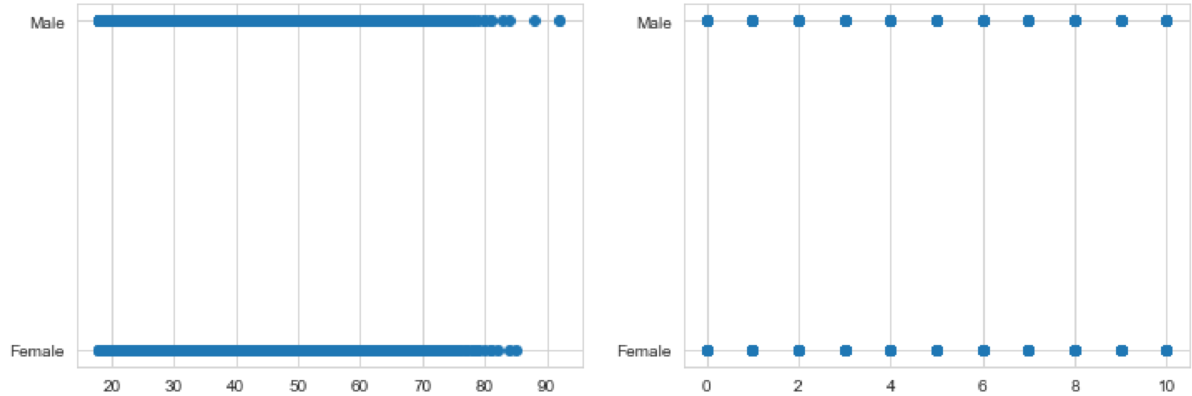
```
Y = df.iloc[:, -1].values
print(Y)
```

```
[1 0 1 ... 1 1 0]
```

9. Scale the independent variables

In [164]:

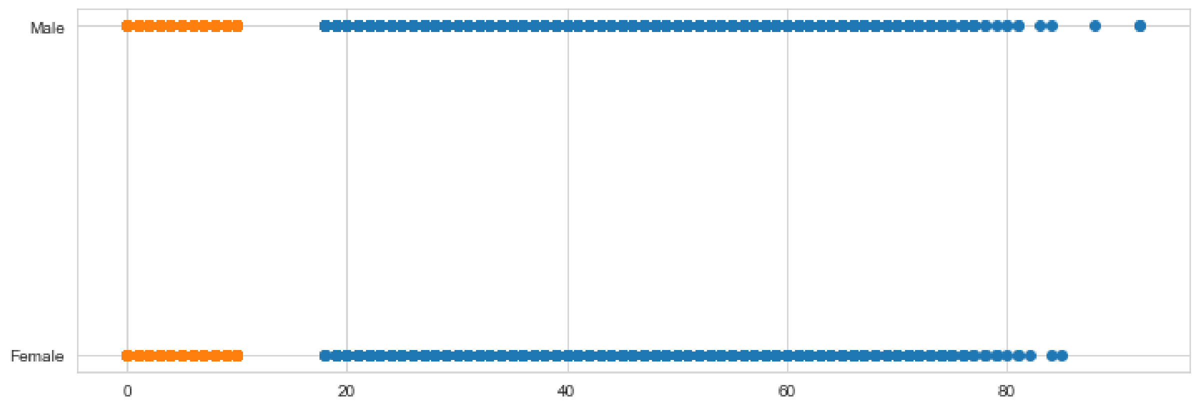
```
df = pd.read_csv("E:\IBM projects Assignment Sona
College\Churn_Modelling.csv")

x = df[['Age', 'Tenure']].values
y = df['Gender'].values

fig, ax = plt.subplots(ncols=2, figsize=(12, 4))

ax[0].scatter(x[:,0], y)
ax[1].scatter(x[:,1], y)

plt.show()
```



In [165]:

```
fig, ax = plt.subplots(figsize=(12, 4))

ax.scatter(x[:,0], y)
ax.scatter(x[:,1], y)
```

Out[165]:



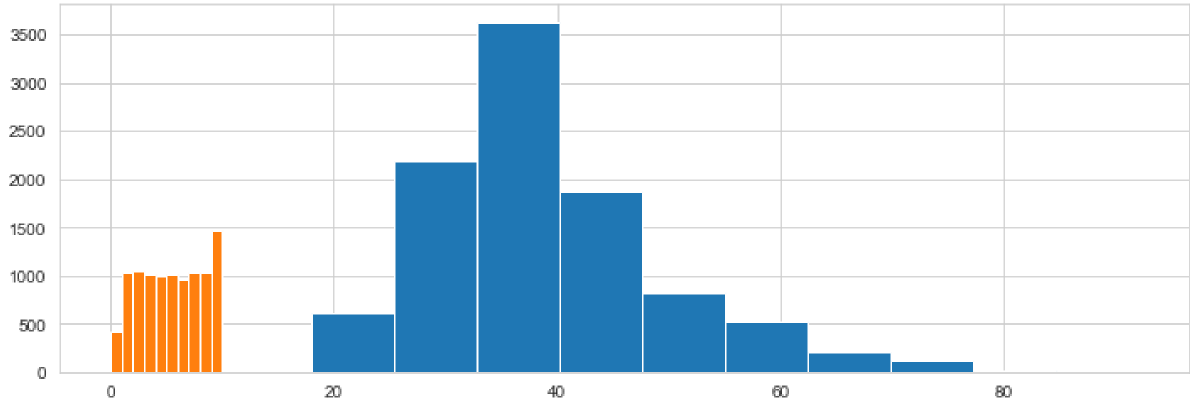
In [166]:

```
fig, ax = plt.subplots(figsize=(12, 4))
```

```
ax.hist(x[:,0])
ax.hist(x[:,1])
```

Out[166]:

```
(array([ 413., 1035., 1048., 1009.,  989., 1012.,  967., 1028., 1025.,
        1474.]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]),
 )
```

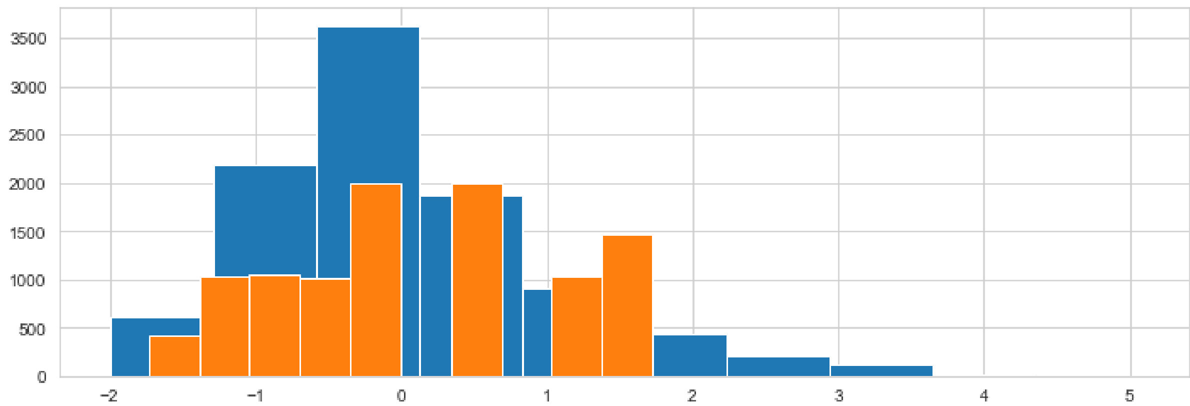


In [167]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
fig, ax = plt.subplots(figsize=(12, 4))

scaler = StandardScaler()
x_std = scaler.fit_transform(x)

ax.hist(x_std[:,0])
ax.hist(x_std[:,1])
```

Out[167]:

```
(array([ 413., 1035., 1048., 1009., 2001.,    0., 1995.,    0., 1025.,
        1474.]),
 array([-1.73331549, -1.38753759, -1.04175968, -0.69598177, -0.35020386,
        -0.00442596,  0.34135195,  0.68712986,  1.03290776,  1.37868567,
         1.72446358]),
 )
```

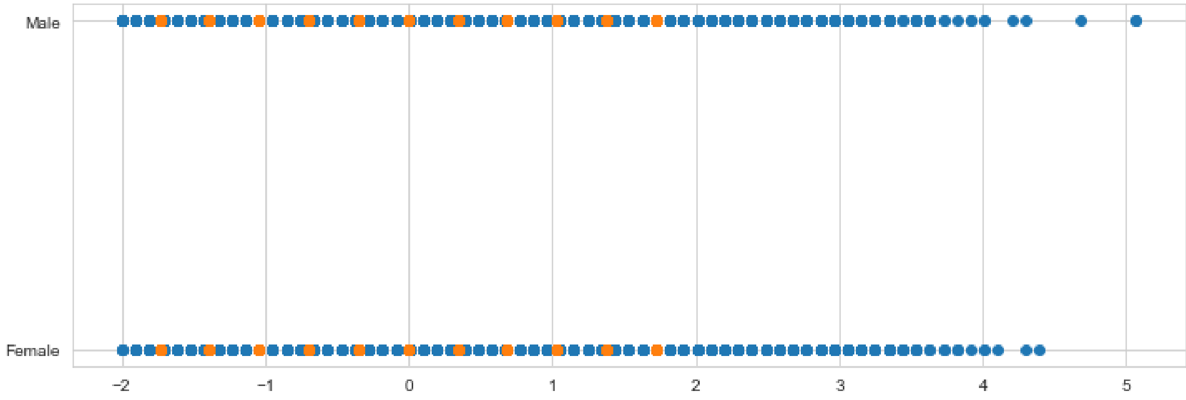


In [168]:

```
fig, ax = plt.subplots(figsize=(12, 4))

scaler = StandardScaler()
x_std = scaler.fit_transform(x)
```

```
ax.scatter(x_std[:,0], y)
ax.scatter(x_std[:,1], y)
```

Out[168]:

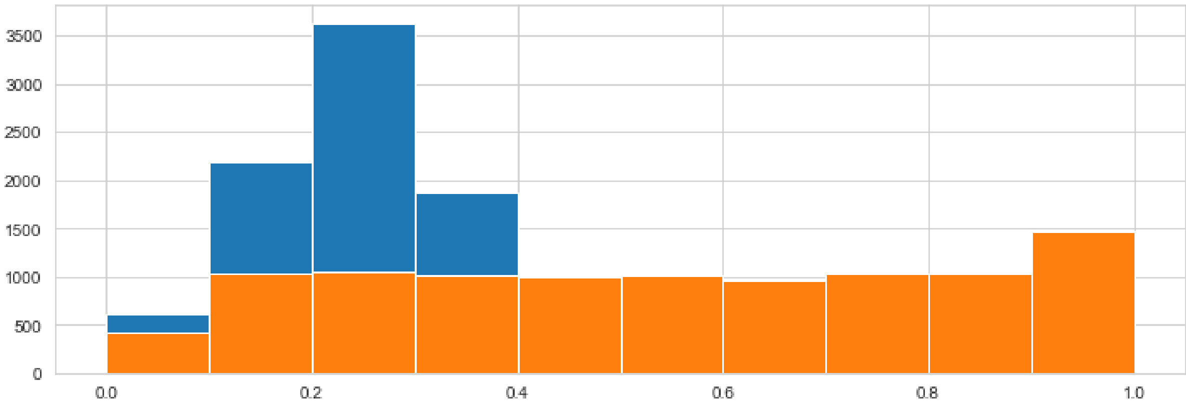


In [169]:

```
fig, ax = plt.subplots(figsize=(12, 4))

scaler = MinMaxScaler()
x_minmax = scaler.fit_transform(x)

ax.hist(x_minmax [:,0])
ax.hist(x_minmax [:,1])
```

Out[169]:

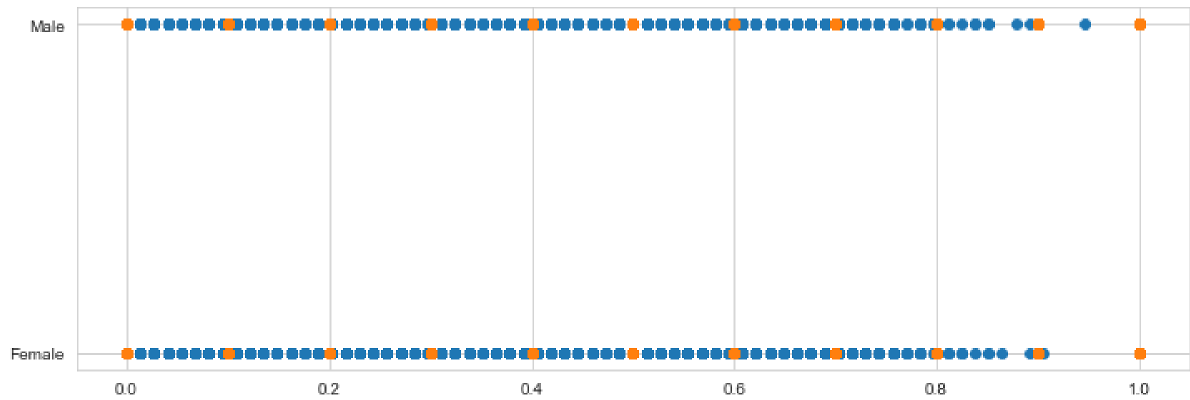
```
(array([ 413., 1035., 1048., 1009.,  989., 1012.,  967., 1028., 1025.,
        1474.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 )
```



In [170]:

```
fig, ax = plt.subplots(figsize=(12, 4))

scaler = MinMaxScaler()
x_minmax = scaler.fit_transform(x)

ax.scatter(x_minmax [:,0], y)
ax.scatter(x_minmax [:,1], y)
```

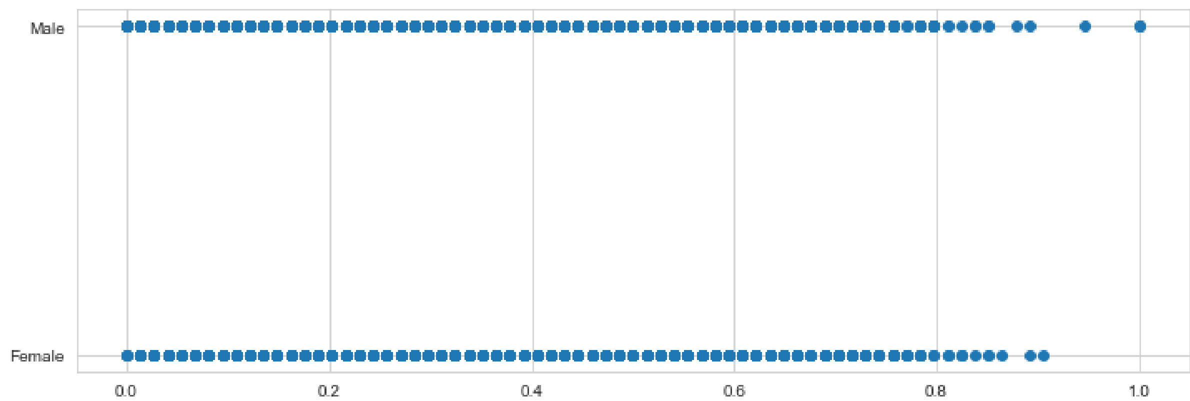
Out[170]:



In [171]:

```
fig, ax = plt.subplots(figsize=(12, 4))

scaler = MinMaxScaler()
x_minmax = scaler.fit_transform(x)

ax.scatter(x_minmax [:,0], y)
```

Out[171]:



fig, ax = plt.subplots(figsize=(12, 4))

scaler = MinMaxScaler() x_minmax = scaler.fit_transform(x)

ax.hist(x_minmax [:,0])

In [173]:

```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
import sklearn.metrics as metrics

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Import Data
df = pd.read_csv("E:\IBM projects Assignment Sona
College\Churn_Modelling.csv")
x = df[['Age', 'Tenure']].values
```

```
y = df['Balance'].values

# Split into a training and testing set
X_train, X_test, Y_train, Y_test = train_test_split(x, y)

# Define the pipeline for scaling and model fitting
pipeline = Pipeline([
    ("MinMax Scaling", MinMaxScaler()),
    ("SGD Regression", SGDRegressor())
])

# Scale the data and fit the model
pipeline.fit(X_train, Y_train)

# Evaluate the model
Y_pred = pipeline.predict(X_test)
print('Mean Absolute Error: ', mean_absolute_error(Y_pred, Y_test))
print('Score', pipeline.score(X_test, Y_test))
```

```
Mean Absolute Error:  56212.80728015005
Score 0.0015999284466322594
```

10. Split the data into training and testing

In [175]:

```
dataset = pd.read_csv("E:\IBM projects Assignment Sona 
College\Churn_Modelling.csv")
print(dataset)
```

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

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0          2       0.00              1          1               1   
1          1   83807.86              1          0               1   
2          8  159660.80              3          1               0   
3          1       0.00              2          0               0   
4          2  125510.82              1          1               1   
...      ...        ...            ...        ...             ...   
9995       5       0.00              2          1               0   
9996      10   57369.61              1          1               1   
9997       7       0.00              1          0               1   
9998       3   75075.31              2          1               0   
```

```
9999       4  130142.79               1           1               0

      EstimatedSalary  Exited
0           101348.88       1
1           112542.58       0
2           113931.57       1
3            93826.63       0
4            79084.10       0
...               ...     ...
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0

[10000 rows x 14 columns]
```

In [176]:

```
dataset.drop(["HasCrCard"],axis=1,inplace=True)
```

In [177]:

```
print(dataset.shape)#no. of rows and colume
print(dataset.head(10))
```

```
(10000, 13)
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43
5          6    15574012       Chu          645     Spain    Male   44
6          7    15592531  Bartlett          822    France    Male   50
7          8    15656148    Obinna          376   Germany  Female   29
8          9    15792365        He          501    France    Male   44
9         10    15592389        H?          684    France    Male   27

   Tenure    Balance  NumOfProducts  IsActiveMember  EstimatedSalary  Exite
d
0       2       0.00              1               1        101348.88
1
1       1   83807.86              1               1        112542.58
0
2       8  159660.80              3               0        113931.57
1
3       1       0.00              2               0         93826.63
0
4       2  125510.82              1               1         79084.10
0
5       8  113755.78              2               0        149756.71
1
6       7       0.00              2               1         10062.80
0
7       4  115046.74              4               0        119346.88
1
8       4  142051.07              2               1         74940.50
0
9       2  134603.88              1               1         71725.73
0
```

In [178]:

```
X=dataset.iloc[:,:-1].values
X
```

Out[178]:

```
array([[1, 15634602, 'Hargrave', ..., 1, 1, 101348.88],
       [2, 15647311, 'Hill', ..., 1, 1, 112542.58],
       [3, 15619304, 'Onio', ..., 3, 0, 113931.57],
       ...,
       [9998, 15584532, 'Liu', ..., 1, 1, 42085.58],
       [9999, 15682355, 'Sabbatini', ..., 2, 0, 92888.52],
       [10000, 15628319, 'Walker', ..., 1, 0, 38190.78]], dtype=object)
```

In [179]:

```
Y=dataset.iloc[:,-1].values
Y
```

Out[179]:

```
array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
```

In [185]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split( X, Y, test_size = 0.25,
random_state = 0 )
```

In [186]:

```
print(X_test)
```

```
[[9395 15615753 'Upchurch' ... 1 1 192852.67]
 [899 15654700 'Fallaci' ... 1 0 128702.1]
 [2399 15633877 'Morrison' ... 1 1 75732.25]
 ...
 [2042 15709846 'Yeh' ... 1 0 84487.62]
 [1109 15678886 'Golubev' ... 2 0 46522.68]
 [3333 15720508 'Hsing' ... 1 0 72927.68]]
```

In [ ]: