Data Analytics Project - Bank Defaulters' Dataframe

Hamza Bin Riaz

- Email: [hamzaa.riaaz@gmail.com (mailto:hamzaa.riaaz@gmail.com)](mailto:hamzaa.riaaz@gmail.com)
- [https://www.linkedin.com/in/hamza-bin-riaz/ (https://www.linkedin.com/in/hamza-bin-riaz/)](https://www.linkedin.com/in/hamza-bin-riaz/)

About Project ¶

I have taken the dataframe of bank defaulters, and performed several Data Analytics tasks.

- Data Summary / Description: I have performed various functions to understand dimensions, missing values, variables, types of variables, and much more about it.
- Data Visualisation - I have tried to visually represent different variables of the dataframe. Some of them like: Surname, Customer ID, and row number were not considered.
- Data Cleaning - I have added another column of "IsDefaulter" that was helpful in different visualisation techniques, and removed some rows where Surnames had inappropriate values.
- Hypothesis Testing - There is no significant relationship between 'Tenure' and 'Exited'
- Predictive Analysis - After how much time customers are likely to exit the bank

```
LET THE FUN(WORK) BEGIN :)
```

Libraries used throughout the project

In [1]:
```python
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plotting
import sys
import scipy.stats as stats
import seaborn as sn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [2]:
```python
os.getcwd()
```

Out[2]:
```
'C:\\Users\\L380\\PythonAssignment_BankLoanDefaulters'
```

Part 1 - Importing DataFrame

Importing dataframe into the object df

In [3]:
```
df = pd.read_csv("BankLoanDefaulters.csv")
```

Showing data set using its object.

In [4]:
```
df
```

Out[4]:

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

10000 rows × 14 columns

Part 2 - Data Summary

Showing the number of observations(rows) and variables(columns).

In [5]:
```
df.shape
```

Out[5]: (10000, 14)

Checking the total values, null or missing values, and datatype of each variable(column) of the dataframe.

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
RowNumber          10000 non-null int64
CustomerId         10000 non-null int64
Surname            10000 non-null object
CreditScore        10000 non-null int64
Geography          10000 non-null object
Gender             10000 non-null object
Age                10000 non-null int64
Tenure             10000 non-null int64
Balance            10000 non-null float64
NumOfProducts      10000 non-null int64
HasCrCard          10000 non-null int64
IsActiveMember     10000 non-null int64
EstimatedSalary    10000 non-null float64
Exited             10000 non-null int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

Checking the following for each variable(column) of the dataframe:
- Number of values
- Mean
- Standard deviation
- Minimum value
- Quartile values for (Q1, Q2, and Q3)
- Maximum Values

In [7]:

```
df.describe()
```

Out[7]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000

Checking names of each variable(column) of the dataframe.

In [8]:
```
df.columns
```

Out[8]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCar
d',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

Checking first 10 observations(rows) of the dataframe.

In [9]:
```
df.head(10)
```

Out[9]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	838
2	3	15619304	Onio	502	France	Female	42	8	1596
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1255
5	6	15574012	Chu	645	Spain	Male	44	8	1137
6	7	15592531	Bartlett	822	France	Male	50	7	
7	8	15656148	Obinna	376	Germany	Female	29	4	1150
8	9	15792365	He	501	France	Male	44	4	1420
9	10	15592389	H?	684	France	Male	27	2	1346

Checking last 10 observations(rows) of the dataframe.

In [10]:
```
df.tail(10)
```

Out[10]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenur
9990	9991	15798964	Nkemakonam	714	Germany	Male	33	
9991	9992	15769959	Ajuluchukwu	597	France	Female	53	
9992	9993	15657105	Chukwualuka	726	Spain	Male	36	
9993	9994	15569266	Rahman	644	France	Male	28	
9994	9995	15719294	Wood	800	France	Female	29	
9995	9996	15606229	Obijiaku	771	France	Male	39	
9996	9997	15569892	Johnstone	516	France	Male	35	1
9997	9998	15584532	Liu	709	France	Female	36	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	
9999	10000	15628319	Walker	792	France	Female	28	

Extending the limit of compiler to show the values, we will need this for checking the unique values of each column.

In [11]:
```
#as we have 10 thousand values in each column, so we are going to extend t

pd.options.display.max_rows = 10000
```

Checking Unique Values of Each Column

In [12]:
```
print("Names of Columns are :", df.columns)
```

```
Names of Columns are : Index(['RowNumber', 'CustomerId', 'Surname', 'Cred
itScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCar
d',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [13]:
```
df["CustomerId"].unique()
```

Out[13]:
```
array([15634602, 15647311, 15619304, ..., 15584532, 15682355, 15628319],
      dtype=int64)
```

In [14]:
```
df["Surname"].unique()
```

Out[14]:
```
array(['Hargrave', 'Hill', 'Onio', ..., 'Kashiwagi', 'Aldridge',
       'Burbidge'], dtype=object)
```

In [15]:
```
df["CreditScore"].unique()
```

Out[15]:
```
array([619, 608, 502, 699, 850, 645, 822, 376, 501, 684, 528, 497, 476,
       549, 635, 616, 653, 587, 726, 732, 636, 510, 669, 846, 577, 756,
       571, 574, 411, 591, 533, 553, 520, 722, 475, 490, 804, 582, 472,
       465, 556, 834, 660, 776, 829, 637, 550, 698, 585, 788, 655, 601,
       656, 725, 511, 614, 742, 687, 555, 603, 751, 581, 735, 661, 675,
       738, 813, 657, 604, 519, 664, 678, 757, 416, 665, 777, 543, 506,
       493, 652, 750, 729, 646, 647, 808, 524, 769, 730, 515, 773, 814,
       710, 413, 623, 670, 622, 785, 605, 479, 685, 538, 562, 721, 628,
       668, 828, 674, 625, 432, 770, 758, 795, 686, 789, 589, 461, 584,
       579, 663, 682, 793, 691, 485, 650, 754, 535, 716, 539, 706, 586,
       631, 717, 800, 683, 704, 615, 667, 484, 480, 578, 512, 606, 597,
       778, 514, 525, 715, 580, 807, 521, 759, 516, 711, 618, 643, 671,
       689, 620, 676, 572, 695, 592, 567, 694, 547, 594, 673, 610, 767,
       763, 712, 703, 662, 659, 523, 772, 545, 634, 739, 771, 681, 544,
       696, 766, 727, 693, 557, 531, 498, 651, 791, 733, 811, 707, 714,
       782, 775, 799, 602, 744, 588, 747, 583, 627, 731, 629, 438, 642,
       806, 474, 559, 429, 680, 749, 734, 644, 626, 649, 805, 718, 840,
       630, 654, 762, 568, 613, 522, 737, 648, 443, 640, 540, 460, 593,
       801, 611, 802, 745, 483, 690, 492, 709, 705, 560, 752, 701, 537,
       487, 596, 702, 486, 724, 548, 464, 790, 534, 748, 494, 590, 468,
       509, 818, 816, 536, 753, 774, 621, 569, 658, 798, 641, 542, 692,
       639, 765, 570, 638, 599, 632, 779, 527, 564, 833, 504, 842, 508,
       417, 598, 741, 607, 761, 848, 546, 439, 755, 760, 526, 713, 700,
       666, 566, 495, 688, 612, 477, 427, 839, 819, 720, 459, 503, 624,
       529, 563, 482, 796, 445, 746, 786, 554, 672, 787, 499, 844, 450,
       815, 838, 803, 736, 633, 600, 679, 517, 792, 743, 488, 421, 841,
       708, 507, 505, 456, 435, 561, 518, 565, 728, 784, 552, 609, 764,
       697, 723, 551, 444, 719, 496, 541, 830, 812, 677, 420, 595, 617,
       809, 500, 826, 434, 513, 478, 797, 363, 399, 463, 780, 452, 575,
       837, 794, 824, 428, 823, 781, 849, 489, 431, 457, 768, 831, 359,
       820, 573, 576, 558, 817, 449, 440, 415, 821, 530, 350, 446, 425,
       740, 481, 783, 358, 845, 451, 458, 469, 423, 404, 836, 473, 835,
       466, 491, 351, 827, 843, 365, 532, 414, 453, 471, 401, 810, 832,
       470, 447, 422, 825, 430, 436, 426, 408, 847, 418, 437, 410, 454,
       407, 455, 462, 386, 405, 383, 395, 467, 433, 442, 424, 448, 441,
       367, 412, 382, 373, 419], dtype=int64)
```

In [16]:
```
df["CreditScore"].min()
```

Out[16]:
```
350
```

In [17]:
```
df["CreditScore"].max()
```

Out[17]:
```
850
```

In [18]:
```
df["Geography"].unique()
```

Out[18]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [19]:
```
df["Gender"].unique()
```
Out[19]:
```
array(['Female', 'Male'], dtype=object)
```

In [20]:
```
df["Age"].unique()
```
Out[20]:
```
array([42, 41, 39, 43, 44, 50, 29, 27, 31, 24, 34, 25, 35, 45, 58, 32, 3
8,
       46, 36, 33, 40, 51, 61, 49, 37, 19, 66, 56, 26, 21, 55, 75, 22, 3
0,
       28, 65, 48, 52, 57, 73, 47, 54, 72, 20, 67, 79, 62, 53, 80, 59, 6
8,
       23, 60, 70, 63, 64, 18, 82, 69, 74, 71, 76, 77, 88, 85, 84, 78, 8
1,
       92, 83], dtype=int64)
```

In [21]:
```
df["Age"].min()
```
Out[21]:
```
18
```

In [22]:
```
df["Age"].max()
```
Out[22]:
```
92
```

In [23]:
```
df["Tenure"].unique()
```
Out[23]:
```
array([ 2,  1,  8,  7,  4,  6,  3, 10,  5,  9,  0], dtype=int64)
```

In [24]:
```
df["Balance"].unique()
```
Out[24]:
```
array([     0.  ,  83807.86, 159660.8 , ...,  57369.61,  75075.31,
       130142.79])
```

In [25]:
```
df["Balance"].min()
```
Out[25]:
```
0.0
```

In [26]:
```
df["Balance"].max()
```
Out[26]:
```
250898.09
```

In [27]:
```
df["NumOfProducts"].unique()
```
Out[27]:
```
array([1, 3, 2, 4], dtype=int64)
```

In [28]:
```
df["HasCrCard"].unique()
```
Out[28]:
```
array([1, 0], dtype=int64)
```

In [29]:
```
df["IsActiveMember"].unique()
```
Out[29]:
```
array([1, 0], dtype=int64)
```

In [30]:
```
df["EstimatedSalary"].unique()
```
Out[30]:
```
array([101348.88, 112542.58, 113931.57, ...,  42085.58,  92888.52,
        38190.78])
```

In [31]:
```
df["EstimatedSalary"].min()
```
Out[31]:
```
11.58
```

In [32]:
```
df["EstimatedSalary"].max()
```
Out[32]:
```
199992.48
```

In [33]:
```
df["Exited"].unique()
```
Out[33]:
```
array([1, 0], dtype=int64)
```

Part 3 - Data Cleaning

Adding A New Column of "IsDefaulter"

```
Adding this column for safe side to use it for data visualisation or
hypothesis testing. Sometimes, to check a variable, we need atleast 2
variables to show them on any kind of plot. For that, we are adding a new
column of "IsDefaulter" with value as "1" for all rows.
```

In [34]:
```
df['IsDefaulter'] = pd.Series([1] * len(df))
```

In [35]:
```
df
```

Out[35]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender
0	1	15634602	Hargrave	619	France	Female
1	2	15647311	Hill	608	Spain	Female
2	3	15619304	Onio	502	France	Female
3	4	15701354	Boni	699	France	Female
4	5	15737888	Mitchell	850	Spain	Female
5	6	15574012	Chu	645	Spain	Male
6	7	15592531	Bartlett	822	France	Male
7	8	15656148	Obinna	376	Germany	Female
8	9	15792365	He	501	France	Male
9	10	15592389	H?	684	France	Male

Removing Rows with InAppropriate Sur Names

```
When we checked the unique values of each column, some "Surnames" had
special characters in them, we know for a fact that some special characters
are acceptable like single quotation O'neil, but question mark is
inappropriate, so we have removed the rows with Surnames having question
marks in them.
```

In [36]:
```
#Some Surnames have inappropriate character in them i.e "?" - removing suc

refined_df = df[~df['Surname'].str.contains('\?')]
```

In [37]:
```
refined_df
```

Out[37]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender
0	1	15634602	Hargrave	619	France	Female
1	2	15647311	Hill	608	Spain	Female
2	3	15619304	Onio	502	France	Female
3	4	15701354	Boni	699	France	Female
4	5	15737888	Mitchell	850	Spain	Female
5	6	15574012	Chu	645	Spain	Male
6	7	15592531	Bartlett	822	France	Male
7	8	15656148	Obinna	376	Germany	Female
8	9	15792365	He	501	France	Male
10	11	15767821	Bearce	528	France	Male

In [38]:
```
refined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9908 entries, 0 to 9999
Data columns (total 15 columns):
RowNumber          9908 non-null int64
CustomerId         9908 non-null int64
Surname            9908 non-null object
CreditScore        9908 non-null int64
Geography          9908 non-null object
Gender             9908 non-null object
Age                9908 non-null int64
Tenure             9908 non-null int64
Balance            9908 non-null float64
NumOfProducts      9908 non-null int64
HasCrCard          9908 non-null int64
IsActiveMember     9908 non-null int64
EstimatedSalary    9908 non-null float64
Exited             9908 non-null int64
IsDefaulter        9908 non-null int64
dtypes: float64(2), int64(10), object(3)
memory usage: 1.2+ MB
```

After removing rows and adding another column of "IsDefaulter" the number of rows have been reduced to 9908, initially they were 10,000. And the no. of columns are 15 now, previously they were 14.

In [39]:
```
refined_df.shape
```

Out[39]: (9908, 15)

Data has been cleaned now, and new dataframe is saved with file name as "Refined_DataFrame" AND the new object that we will use for further coding will be "refined_df"

In [40]:

```
refined_df.to_csv("Refined_DataFrame.csv", index = False)
```

Part 4 - Data Visualization

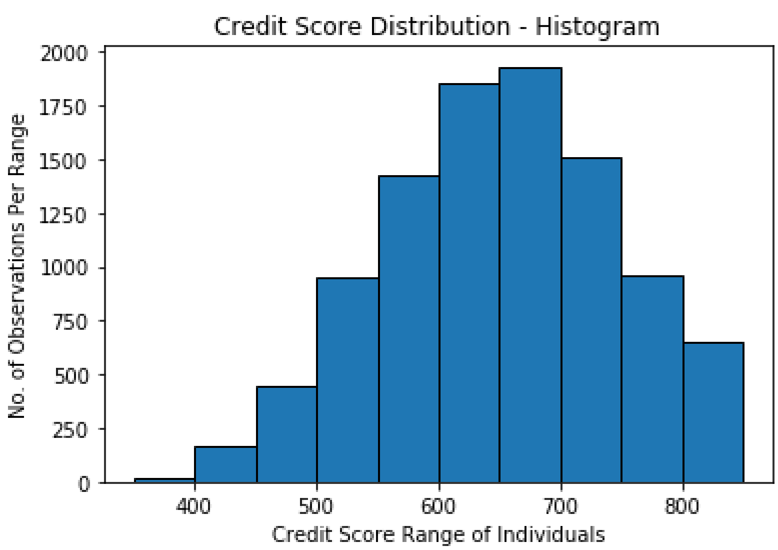
In the data visualisation section, we are expressing almost all of variables of our dataframe except ROw Number, Customer ID, and Surname. As they don't require any representation.

Credit Score Distribution

Below we have histogram with 10 bins, along x-axis, we have different ranges of Credit Scores of each individual and on y-axis we have the toltal number of observations that fall in each range.

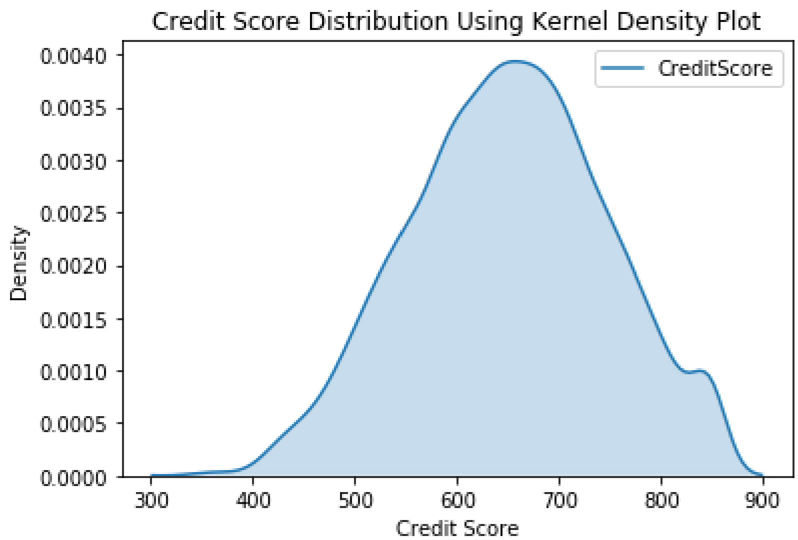
In [41]:

```
#setting the plotting type as histogram, selecting the number of bins as 1
plotting.hist(refined_df['CreditScore'], bins = 10, edgecolor = 'black')

#Setting Tile and labels of the histogram
plotting.xlabel('Credit Score Range of Individuals')
plotting.ylabel('No. of Observations Per Range')
plotting.title('Credit Score Distribution - Histogram')

#showing the histogram
plotting.show()
```



Below is just another representation of Credit Score variable using Kernal Density plot

In [42]:
```
#Using kernal density plot to represent credit score
sn.kdeplot(df['CreditScore'], shade = True)

# Setting title and labels of plot
plotting.xlabel('Credit Score')
plotting.ylabel('Density')
plotting.title('Credit Score Distribution Using Kernel Density Plot')

#showing the plot
plotting.show()
```



Region Based Distribution of Defaulters

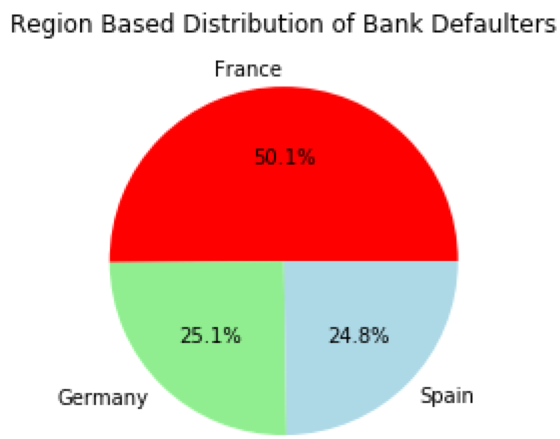
```
Below is a pie chart that represents the distribution of bank defaulters
based on their region. We have data of defaulters, from 3 countries as
France, Germany, and Spain. From the percentages it's evident that Most of
the defaulters are from france, then from Germany and last from Spain.
```

In [43]:

```
#counting all the values from the Geography column
no_of_defaulters = refined_df["Geography"].value_counts()

#using pie as plotting method, setting labels of each section of pie chart
plotting.pie(no_of_defaulters, labels = no_of_defaulters.index, autopct =

#setting title of pie chart
plotting.title("Region Based Distribution of Bank Defaulters")
```

Out[43]:

```
Text(0.5, 1.0, 'Region Based Distribution of Bank Defaulters')
```

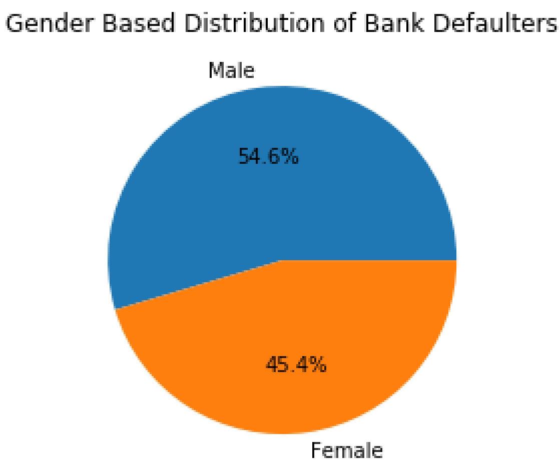


Gender Based Distribution of Defaulters

In the pie chart below, we are showing the gender based distribution of bank defaulters.

In [44]:
```
#counting all the values from the Gender column
no_of_defaulters = refined_df["Gender"].value_counts()

#using pie as plotting method, setting labels of each section of pie chart
plotting.pie(no_of_defaulters, labels = no_of_defaulters.index, autopct =

#setting title of pie chart
plotting.title("Gender Based Distribution of Bank Defaulters")
```

Out[44]:
```
Text(0.5, 1.0, 'Gender Based Distribution of Bank Defaulters')
```

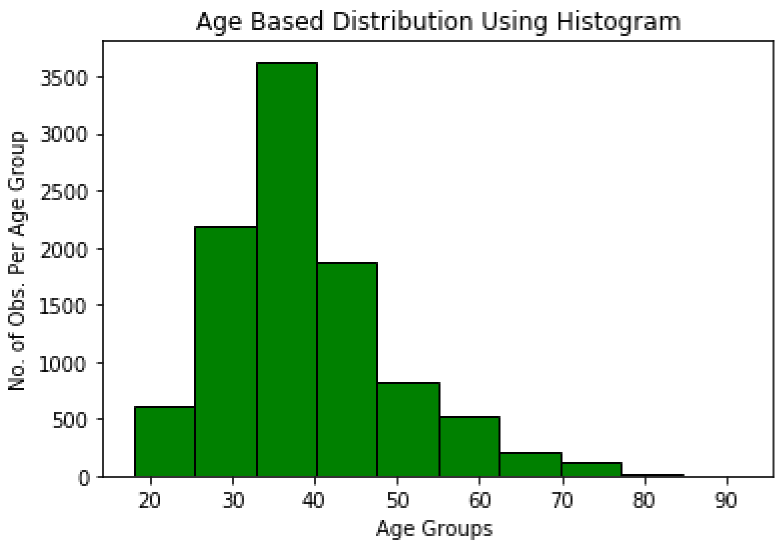


Age Groups of Defaulters

```
Difining age groups for defaulters and showing their distribution using the
"Age" variable.
```

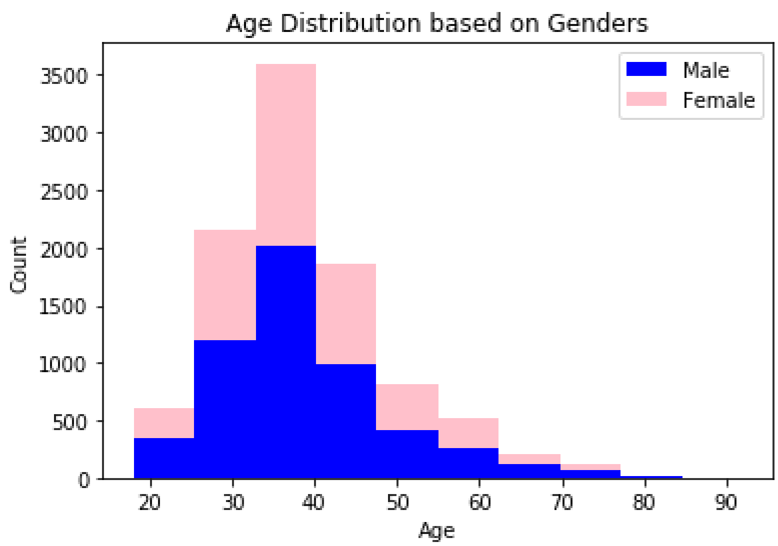
In [45]:

```
#using the histogram as visualisation technique, and choosing the no of bi
plotting.hist(df['Age'], bins = 10, edgecolor = 'black', color = 'green')

#Choosing the title and label of histogram
plotting.xlabel('Age Groups ')
plotting.ylabel('No. of Obs. Per Age Group')
plotting.title('Age Based Distribution Using Histogram')

#showing the histogram
plotting.show()
```



Age Distribution Based on Genders

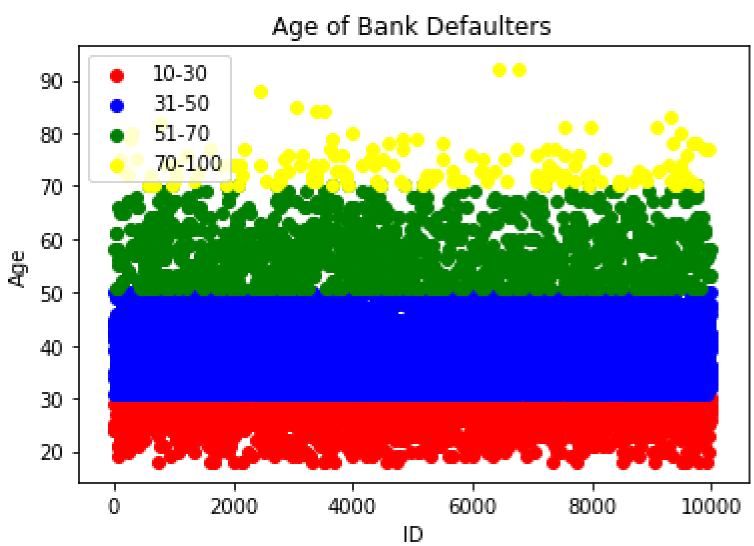
In [46]:
```
#showing age based distribution as per each gender

#getting the ages for both the genders ans setting the visualisation optio
plotting.hist([refined_df[refined_df['Gender'] == 'Male']['Age'], refined_
         bins = 10, stacked = True, color =['blue', 'pink'], label = ['Mal

#Setting title anf labels
plotting.xlabel('Age')
plotting.ylabel('Count')
plotting.title('Age Distribution based on Genders')
plotting.legend()
plotting.show()
```



In the scatter plot below, we are representing the age of bank defaulters by defining the custom age groups, as we know the minimum and maximum values of Age variable.

In [47]:

```
#defining the custom age groups by ourselves and their colors
age_groups = [(10, 30, 'red'), (31, 50, 'blue'), (51, 70, 'green'), (70, 1

#creatoing the objects for plot and axis
fig, age_plot = plotting.subplots()

#looping through the age groups and plotting the scatter points with the c
for age_group in age_groups:
    x = refined_df[refined_df['Age'].between(age_group[0], age_group[1])][
    y = refined_df[refined_df['Age'].between(age_group[0], age_group[1])][
    age_plot.scatter(x, y, c = age_group[2], label = '{}-{}'.format(age_gr


age_plot.set_title('Age of Bank Defaulters')
age_plot.set_xlabel('ID')
age_plot.set_ylabel('Age')

age_plot.legend()

plotting.show()
```

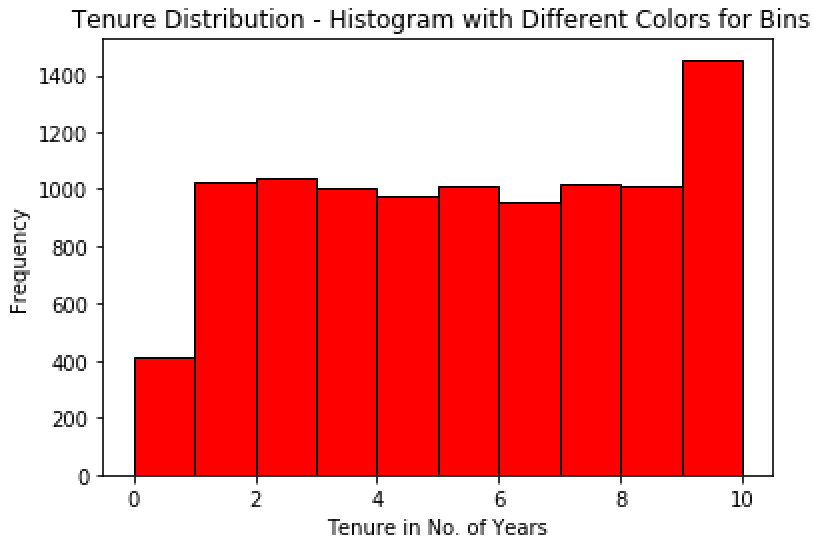


Tenure Representation Using Histogram

In the histogram below, we are showing the tenure based distribution. Tenure represent the no of years an individual has stayed with the bank as a customer.

In [48]:

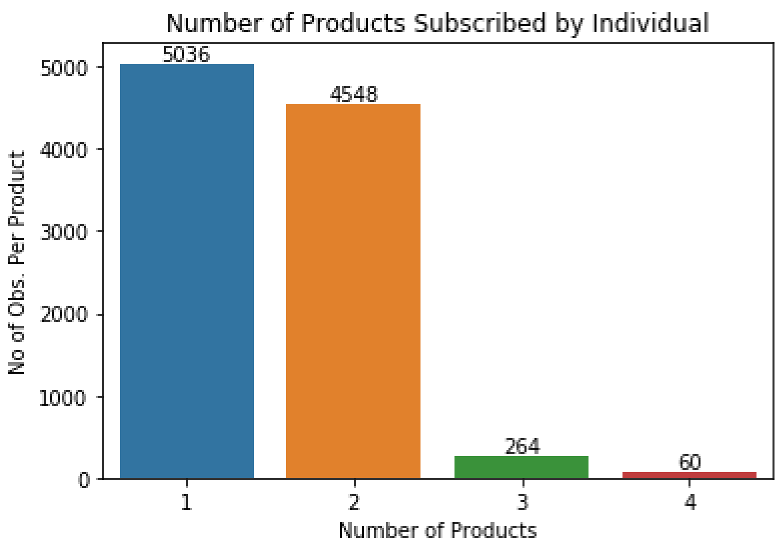
```
plotting.hist(refined_df['Tenure'], bins = 10, edgecolor = 'black', color

plotting.xlabel('Tenure in No. of Years')
plotting.ylabel('Frequency')
plotting.title(' Tenure Distribution - Histogram with Different Colors for

plotting.show()
```



No. of Products Distribution

In the count plot below, we are showing the distribution based on the products subscribed by each customer. There were maximum of 4 products, and each bar represents the no of people who were subscribed to that no. of products.

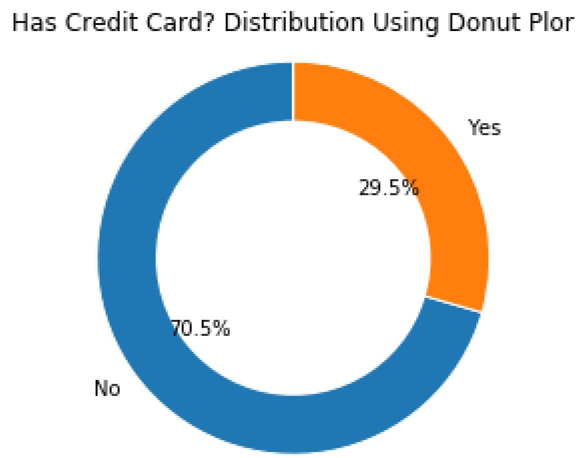
In [49]:
```
produtcs_plot = sn.countplot(x = 'NumOfProducts', data = refined_df)

#Adding count labels to the countplot
for p in produtcs_plot.patches:
    count = p.get_height()
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    produtcs_plot.annotate(count, (x, y), ha = 'center', va = 'bottom')


plotting.xlabel('Number of Products')
plotting.ylabel('No of Obs. Per Product')
plotting.title('Number of Products Subscribed by Individual')

plotting.show()
```



Has Credit Card Distribution

In the circle/donut plot below, we have shown the percentage based division of people with Credit Cards and Without Credit Cards.

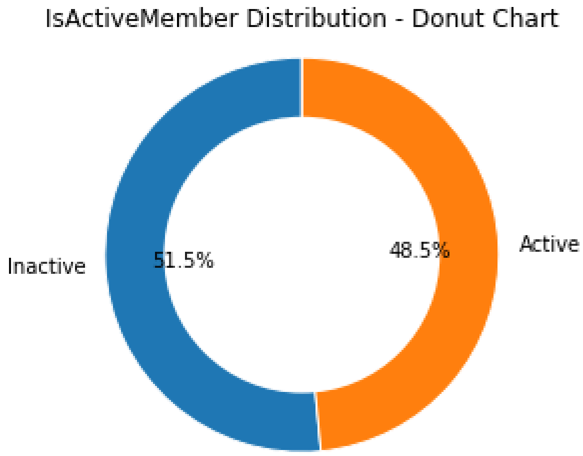
In [50]:
```
counts = refined_df['HasCrCard'].value_counts()
labels = ['No', 'Yes']

fig, ax = plotting.subplots()
ax.pie(counts, labels = labels, autopct = '%1.1f%%', startangle = 90, wedg

center_circle = plotting.Circle((0, 0), 0.70, fc = 'white')
fig.gca().add_artist(center_circle)

ax.axis('equal')
ax.set_title('Has Credit Card? Distribution Using Donut Plor')

plotting.show()
```



Is Active Member Distribution

```
Again, using the donut plot method, we are showing the percentage
distribution of people who are currently the active members of the bank and
people who are not active members at the moment.
```

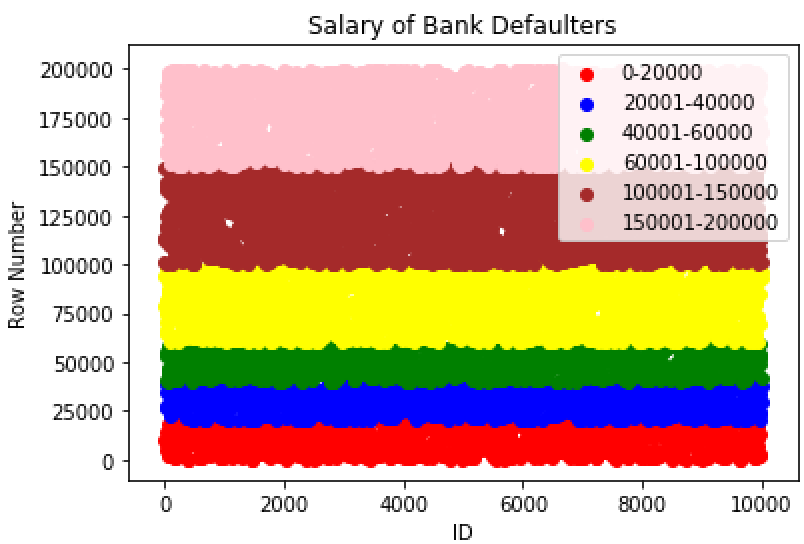
In [51]:
```
counts = refined_df['IsActiveMember'].value_counts()
labels = ['Inactive', 'Active']

plotting.pie(counts, labels = labels, autopct = '%1.1f%%', startangle = 90

# Draw a white circle at the center to create a donut chart
center_circle = plotting.Circle((0, 0), 0.70, fc = 'white')
fig = plotting.gcf()
fig.gca().add_artist(center_circle)

plotting.axis('equal')
plotting.title('IsActiveMember Distribution - Donut Chart')

plotting.show()
```



Understanding the Salary Spans

```
In the scatter plot below we have tried to define custom salary spans and
represented the observations in different salary spans.
```

In [52]:
```
#defining the custom salary spans and their colors
salary_spans = [(0, 20000, 'red'), (20001, 40000, 'blue'), (40001, 60000,

#creating objects
fig, salary_image = plotting.subplots()

#looping through the age groups and plotting the scatter points
for salary_span in salary_spans:
    x = refined_df[refined_df['EstimatedSalary'].between(salary_span[0], s
    y = refined_df[refined_df['EstimatedSalary'].between(salary_span[0], s
    salary_image.scatter(x, y, c=salary_span[2], label = '{}-{}'.format(sa

salary_image.set_title('Salary of Bank Defaulters')
salary_image.set_xlabel('ID')
salary_image.set_ylabel('Row Number')

salary_image.legend()
plotting.show()
```



Part 5 - Hypothesis Testing

In [60]:
```
# Null Hypothesis (H0): There is no significant relationship between 'Tenu

# Alternative Hypothesis (H1): There is a significant relationship between
```

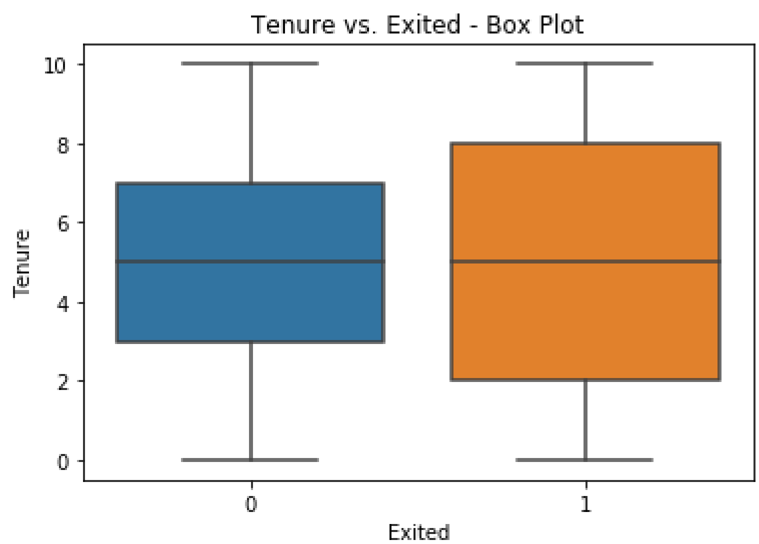
In [61]:
```
#separating the data into two groups based on 'Exited' values (0 or 1)
group_0 = refined_df[refined_df['Exited'] == 0]['Tenure']
group_1 = refined_df[refined_df['Exited'] == 1]['Tenure']

#perform two-sample t-test
t_statistic, p_value = stats.ttest_ind(group_0, group_1, equal_var = False
```

In [62]:
```
#print the results
print('T-Statistic: ', t_statistic)
print('P-Value: ', p_value)

#comparing the p-value with the significance level(0.05)
if p_value < 0.05:
    print("Null hypothesis has been rejected. There is a significant relat
else:
    print("Failed to reject the null hypothesis. There is no significant r
```

```
T-Statistic:  1.3196271681627358
P-Value:  0.1870572447799254
Failed to reject the null hypothesis. There is no significant relationshi
p between 'Tenure' and 'Exited'.
```

Visual Representation of Exited and Tenure

In [63]:
```
#using boxplot
sn.boxplot(x = 'Exited', y = 'Tenure', data = refined_df)

plotting.xlabel('Exited')
plotting.ylabel('Tenure')
plotting.title('Tenure vs. Exited - Box Plot')

plotting.show()
```

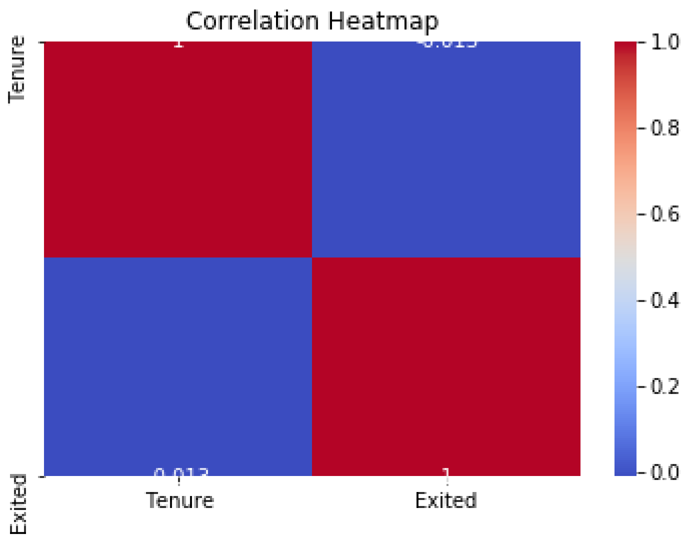


Correlation between "Tenure" and "Exited" Variables

In [64]:
```
correlation = refined_df['Tenure'].corr(refined_df['Exited'])

print("Correlation coefficient: ", correlation)
```

```
Correlation coefficient:  -0.013407416162277981
```

In [65]:
```
#calculating the correlation matrix
corr_matrix = refined_df[['Tenure', 'Exited']].corr()

#using heatmap as visual method
sn.heatmap(corr_matrix, annot=True, cmap = 'coolwarm')

plotting.title('Correlation Heatmap')

plotting.show()
```



Part 6 - Predictive Analysis

```
In this section, we are going to use logistic regression model to to
predict, after how much time, customers are predicted to exit(leave) the
bank.
```

In [69]:
```
#selecting the features and target variable
X = refined_df[['Tenure']]
y = refined_df['Exited']

#creating an instance of the logistic regression model
model = LogisticRegression()

#fiting model to data
model.fit(X, y)

#getting he coefficients of the model
coefficients = model.coef_[0]

#finding index of the maximum coefficient
max_coefficient_index = np.argmax(np.abs(coefficients))

#getting the corresponding tenure value
tenure_threshold = X.iloc[max_coefficient_index]['Tenure']

#printing outcome
print("People are more likely to exit the bank after ", tenure_threshold,
```

```
People are more likely to exit the bank after  2  years.

C:\Users\L380\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22.
Specify a solver to silence this warning.
  FutureWarning)
```

Part 7 - References

Source of the dataframe:
https://www.kaggle.com/datasets/vanshikagupta1136/artificial-neural-network-case-study-data

In [ ]: