In [1]:

```
## Importing data handeling libraries

import pandas as pd
import numpy as np
from collections import OrderedDict

## Machine Learning , Model importing library

from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardScaler


## importing the KNNimputer class

from sklearn.impute import KNNImputer


## Data visualisation libraries

import matplotlib.pyplot as plt
import seaborn as sns

## importing library to stop warnings

import warnings
warnings.filterwarnings ("ignore")

##preprocessing library

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

## model bulding libraries

from sklearn.model_selection import train_test_split,GridSearchCV,learning_curve,Stra

## model evaluation libraries

from sklearn.metrics import accuracy_score,f1_score

## machien learning model libraries

from sklearn.linear_model import LogisticRegression,RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoosti
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

import xgboost
from xgboost import XGBClassifier

## importing clustering
from sklearn.cluster import KMeans
```

executed in 1.47s, finished 09:16:14 2022-06-06

Creating DataFrame

In [2]:

```
df = pd.read_csv("Abhudaya_dummy.csv")
df.head()
```

executed in 39ms, finished 09:16:14 2022-06-06

Out[2]:

	Cust No.	First Name	Surname	Credit Score	Geography	Gender	Age	Tenure	Balance	Num Of Policies	Crec Cai
0	1	Walter	Hargrave	619	France	Female	42	2.0	NaN	1.0	Y
1	2	Daniel	Hill	608	Spain	Female	41	1.0	83807.86	1.0	N
2	3	Melissa	Onio	502	France	Female	42	8.0	159660.80	3.0	Y
3	4	Miley	Boni	699	France	Female	39	1.0	NaN	2.0	N
4	5	James	Mitchell	850	Spain	Female	43	2.0	125510.82	1.0	Y

In [3]:

```
df.drop(["Cust No.","First Name","Surname"],axis=1,inplace=True)
df.head()
```

executed in 66ms, finished 09:16:14 2022-06-06

Out[3]:

	Credit Score	Geography	Gender	Age	Tenure	Balance	Num Of Policies	Credit Card	Active Member	Salary	Exit
0	619	France	Female	42	2.0	NaN	1.0	Yes	Yes	101349	
1	608	Spain	Female	41	1.0	83807.86	1.0	No	Yes	112543	
2	502	France	Female	42	8.0	159660.80	3.0	Yes	No	113932	
3	699	France	Female	39	1.0	NaN	2.0	No	No	93827	
4	850	Spain	Female	43	2.0	125510.82	1.0	Yes	Yes	79084	

Data PreProcessing

In [4]:

```
df.isna().sum()
```

executed in 132ms, finished 09:16:14 2022-06-06

Out[4]:

```
Credit Score          0
Geography            72
Gender               68
Age                   0
Tenure               18
Balance            3620
Num Of Policies     167
Credit Card           0
Active Member         0
Salary                0
Exited                0
dtype: int64
```

In [5]:

```
df.info()
```

executed in 94ms, finished 09:16:15 2022-06-06

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10071 entries, 0 to 10070
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Credit Score     10071 non-null  int64  
 1   Geography        9999 non-null   object 
 2   Gender           10003 non-null  object 
 3   Age              10071 non-null  int64  
 4   Tenure           10053 non-null  float64
 5   Balance          6451 non-null   float64
 6   Num Of Policies  9904 non-null   float64
 7   Credit Card      10071 non-null  object 
 8   Active Member    10071 non-null  object 
 9   Salary           10071 non-null  int64  
 10  Exited           10071 non-null  int64  
dtypes: float64(3), int64(4), object(4)
memory usage: 865.6+ KB
```

In [6]:

```
## around 33% of data in "balance" column is missing we can not
## simply drop those null values in this case to do further process
## we need to impute those values by using appropriate methods
### other than balance column if we consider "num of policies" Or "Geography" column
### We can simply drop them
```

executed in 96ms, finished 09:16:15 2022-06-06

In [7]:

```
new_df = df
new_df = new_df[new_df['Geography'].isnull()==False]
new_df = new_df[new_df['Num Of Policies'].isnull()==False]
```

executed in 102ms, finished 09:16:15 2022-06-06

In [8]:

```
df = new_df
df.info()
```

executed in 126ms, finished 09:16:15 2022-06-06

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9832 entries, 0 to 10069
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Credit Score     9832 non-null   int64  
 1   Geography        9832 non-null   object 
 2   Gender           9771 non-null   object 
 3   Age              9832 non-null   int64  
 4   Tenure           9828 non-null   float64
 5   Balance          6311 non-null   float64
 6   Num Of Policies  9832 non-null   float64
 7   Credit Card      9832 non-null   object 
 8   Active Member    9832 non-null   object 
 9   Salary           9832 non-null   int64  
 10  Exited           9832 non-null   int64  
dtypes: float64(3), int64(4), object(4)
memory usage: 921.8+ KB
```

In [9]:

```
## We lost around 3% of data to remove null values
## from all the features but dropping null values from "balance" will not be appropri
## hence we are going to impute values where balance is null
```

executed in 78ms, finished 09:16:15 2022-06-06

In [10]:

```
## we would do null value impution by two different methods one is by using linear re
## and other is by using KNN imputation and we will check accuracy for both the metho
## whichever will work fine we will go with that method.
```

executed in 83ms, finished 09:16:15 2022-06-06

First method for null value imputation - BY using Linear Regression

In [11]:

```
## To do Linear regression imputation we will create two different dataframes by usin
## In first data frame we will consider "balance true values" and we will use that da
## In second data frame we will consider "balance false values" and we will use that
```

executed in 93ms, finished 09:16:15 2022-06-06

Preprocessing Data to use it for null value imputation by using Linear regrssion

In [12]:

```
le = LabelEncoder()
for i in df.columns:
    if df[i].dtype == "O" :
        df[i] = le.fit_transform(df[i])
```

executed in 144ms, finished 09:16:15 2022-06-06

In [13]:

```
## As we are using linear regression just going through discriptive stats
```

executed in 101ms, finished 09:16:15 2022-06-06

In [14]:

```
df
```

executed in 116ms, finished 09:16:15 2022-06-06

Out[14]:

	Credit Score	Geography	Gender	Age	Tenure	Balance	Num Of Policies	Credit Card	Active Member	Salary
0	619	0	0	42	2.0	NaN	1.0	1	1	101349
1	608	2	0	41	1.0	83807.86	1.0	0	1	112543
2	502	0	0	42	8.0	159660.80	3.0	1	0	113932
3	699	0	0	39	1.0	NaN	2.0	0	0	93827
4	850	2	0	43	2.0	125510.82	1.0	1	1	79084
...	...	...	...	...	...	...	...	...	...	...
10064	711	2	1	86	1.0	161425.00	2.0	1	0	51518
10065	447	0	1	31	8.0	79591.00	3.0	0	1	159857
10066	777	0	0	35	3.0	80701.00	2.0	1	1	156120
10068	450	0	1	60	9.0	1053.00	4.0	1	1	86961
10069	819	1	1	45	3.0	94661.00	3.0	1	1	89433

9832 rows × 11 columns

In [15]:

```
df[["Credit Score","Age","Tenure","Balance","Num Of Policies","Salary"]].describe()
```

executed in 144ms, finished 09:16:16 2022-06-06

Out[15]:

	Credit Score	Age	Tenure	Balance	Num Of Policies	Salary
count	9832.000000	9832.000000	9828.000000	6311.000000	9832.000000	9832.000000
mean	650.292921	39.007018	5.008140	119907.324999	1.531428	100133.187449
std	96.908705	10.600976	2.895932	30562.821953	0.591560	57560.414197
min	350.000000	18.000000	0.000000	1053.000000	1.000000	12.000000
25%	583.750000	32.000000	2.000000	100116.745000	1.000000	50974.500000
50%	652.000000	37.000000	5.000000	119839.690000	1.000000	100238.000000
75%	718.000000	44.000000	7.000000	139642.525000	2.000000	149458.250000
max	850.000000	92.000000	10.000000	250898.090000	4.000000	199992.000000

In [16]:

```
df[["Credit Score","Age","Tenure","Balance","Num Of Policies","Salary"]].skew()
```

executed in 80ms, finished 09:16:16 2022-06-06

Out[16]:

```
Credit Score      -0.076301
Age                1.042858
Tenure             0.011162
Balance            0.038253
Num Of Policies    0.822019
Salary            -0.000096
dtype: float64
```

In [17]:

```
df[["Credit Score","Age","Tenure","Balance","Num Of Policies","Salary"]].kurt()
```

executed in 96ms, finished 09:16:16 2022-06-06

Out[17]:

```
Credit Score      -0.426717
Age                1.549701
Tenure            -1.165715
Balance            0.365265
Num Of Policies    0.815534
Salary            -1.183271
dtype: float64
```

In [18]:

```
df[["Credit Score","Age","Tenure","Balance","Num Of Policies","Salary"]].corr()
```

executed in 147ms, finished 09:16:16 2022-06-06

Out[18]:

	Credit Score	Age	Tenure	Balance	Num Of Policies	Salary
Credit Score	1.000000	-0.005140	-0.001289	-0.009320	0.008193	0.000862
Age	-0.005140	1.000000	-0.010486	-0.012861	-0.014252	-0.008129
Tenure	-0.001289	-0.010486	1.000000	0.002718	0.010364	0.008331
Balance	-0.009320	-0.012861	0.002718	1.000000	0.005639	-0.001756
Num Of Policies	0.008193	-0.014252	0.010364	0.005639	1.000000	0.013025
Salary	0.000862	-0.008129	0.008331	-0.001756	0.013025	1.000000

No linear relationship is there in "Balance" and other coulmns that we are suppose to use as an independant variables

Data is not distributed normally

two important considerations of Linear regression are not being followed by data

Imputation By using linear regression will not be a good choice in any condition.

Second method for null value imputation - BY using KNN Imputation

In [19]:

```
imputer = KNNImputer(n_neighbors=3)
df = imputer.fit_transform(df)
```

executed in 2.56s, finished 09:16:19 2022-06-06

Type *Markdown* and LaTeX: α^2

In [20]:

```
df = pd.DataFrame(df)
```

executed in 4ms, finished 09:16:19 2022-06-06

In [21]:

```
df["Credit_Score"] = df.iloc[0:,0]
df["Geography"] = df.iloc[0:,1]
df["Gender"] = df.iloc[0:,2]
df["Age"] = df.iloc[0:,3]
df["Tenure"] = df.iloc[0:,4]
df["Balance"] = df.iloc[0:,5]
df["Num_Of_Policies"] = df.iloc[0:,6]
df["Credit_Card"] = df.iloc[0:,7]
df["Active_Member"] = df.iloc[0:,8]
df["Salary"] = df.iloc[0:,9]
df["Exited"] = df.iloc[0:,10]
```

executed in 121ms, finished 09:16:19 2022-06-06

In [22]:

```
df = df.iloc[0:,11:]
df
```

executed in 155ms, finished 09:16:19 2022-06-06

Out[22]:

	Credit_Score	Geography	Gender	Age	Tenure	Balance	Num_Of_Policies	Credit_Carc
0	619.0	0.0	0.0	42.0	2.0	144321.21	1.0	1.0
1	608.0	2.0	0.0	41.0	1.0	83807.86	1.0	0.0
2	502.0	0.0	0.0	42.0	8.0	159660.80	3.0	1.0
3	699.0	0.0	0.0	39.0	1.0	124499.37	2.0	0.0
4	850.0	2.0	0.0	43.0	2.0	125510.82	1.0	1.0
...	...	...	...	...	...	...	...	..
9827	711.0	2.0	1.0	86.0	1.0	161425.00	2.0	1.0
9828	447.0	0.0	1.0	31.0	8.0	79591.00	3.0	0.0
9829	777.0	0.0	0.0	35.0	3.0	80701.00	2.0	1.0
9830	450.0	0.0	1.0	60.0	9.0	1053.00	4.0	1.0
9831	819.0	1.0	1.0	45.0	3.0	94661.00	3.0	1.0

9832 rows × 11 columns

In [23]:

```
df.info()
```

executed in 63ms, finished 09:16:19 2022-06-06

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9832 entries, 0 to 9831
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Credit_Score     9832 non-null   float64
 1   Geography        9832 non-null   float64
 2   Gender           9832 non-null   float64
 3   Age              9832 non-null   float64
 4   Tenure           9832 non-null   float64
 5   Balance          9832 non-null   float64
 6   Num_Of_Policies  9832 non-null   float64
 7   Credit_Card      9832 non-null   float64
 8   Active_Member    9832 non-null   float64
 9   Salary           9832 non-null   float64
 10  Exited           9832 non-null   float64
dtypes: float64(11)
memory usage: 845.1 KB
```

- This is how we can impute null values by using KNN Imputer
- instead of using mean or median of the data this type will be more impactfull
- As here we are using nearest neighbors to predict the null value.

EDA

In [24]:

```
def summary_function(df):

    result = []

    for col in df.columns:

        if df[col].dtype != 'O':

            stats = OrderedDict ({
                'Feature_Name' : col,
                'Count':df[col].count(),
                'Minimum':df[col].min(),
                'Quarter 1':df[col].quantile(0.25),
                "Mean":df[col].mean(),
                'Median':df[col].median(),
                'Quarter 3':df[col].quantile(0.75),
                'Maximum':df[col].max(),
                "Variance":df[col].var(),
                'Standard Deviation':df[col].std(),
                "Kurtosis":df[col].kurt(),
                'Skewness':df[col].skew() ,
                'IQR':df[col].quantile(0.75) - df[col].quantile(0.25)
            })

            result.append(stats)


    result = pd.DataFrame(result)

    ##skewness type

    skewtype =[]

    for i in result['Skewness']:
        if i<=-1:
            skewtype.append('Highly Negatively Skewed')
        elif i<= -0.5:
            skewtype.append('Moderately Negatively Skewed')
        elif -0.5 < i < 0 :
            skewtype.append('Approx Normal Distribution (-ve)')
        elif 0 <= i < 0.5:
            skewtype.append('Approx Normal Distribution (+ve)')
        elif 0.5<= i < 1:
            skewtype.append('Moderately Positively Skewed')
        elif i >= 1:
            skewtype.append('Highly Positively Skewed')

    result['Skew_Type'] = skewtype

    ## Kurtosis Type
    k_type = []

    for i in result['Kurtosis']:
        if i <=  -1:
            k_type.append('Highly Platykurtic Curve')
        elif -1 < i <= -0.5:
            k_type.append('Moderately Platykurtic Curve')
        elif -0.5 < i <= 0.5:
            k_type.append('Mesokurtic Curve')
```

```
        elif 0.5<= i < 1:
            k_type.append('Moderately Leptokurtic Curve')
        elif i >= 1:
            k_type.append('Highly Leptokurtic Curve')

    result['Kurtosis_Type'] = k_type


    #Outlier detection

    Upper_limit = stats['Quarter 3'] + 1.5*stats['IQR']

    lower_limit = stats['Quarter 1'] -1.5*stats['IQR']

    if len([x for x in df[col] if (x < lower_limit) or (x > Upper_limit)]) > 0:

        outlier_comment = 'has outliers'
        outlier_percentage = len([x for x in df[col] if (x < lower_limit) or (x > Upp
    else:
        oulier_comment = 'no outliers'

        outlier_percentage = 0

    result['outlier_comment'] = outlier_comment

    result['outlier_percentage'] = outlier_percentage

    return result
```

executed in 90ms, finished 09:16:19 2022-06-06

In [25]:

```
summary_function(df)
```

executed in 207ms, finished 09:16:19 2022-06-06

Out[25]:

	Feature_Name	Count	Minimum	Quarter 1	Mean	Median	Quarter
0	Credit_Score	9832	350.0	583.750000	650.292921	652.000000	718.00(
1	Geography	9832	0.0	0.000000	0.743897	0.000000	1.00(
2	Gender	9832	0.0	0.000000	0.555228	1.000000	1.00(
3	Age	9832	18.0	32.000000	39.007018	37.000000	44.00(
4	Tenure	9832	0.0	2.000000	5.007730	5.000000	7.00(
5	Balance	9832	1053.0	103492.438333	119725.604851	119455.953333	135800.397
6	Num_Of_Policies	9832	1.0	1.000000	1.531428	1.000000	2.00(
7	Credit_Card	9832	0.0	0.000000	0.705248	1.000000	1.00(
8	Active_Member	9832	0.0	0.000000	0.514341	1.000000	1.00(
9	Salary	9832	12.0	50974.500000	100133.187449	100238.000000	149458.25(
10	Exited	9832	0.0	0.000000	0.206672	0.000000	0.00(

Some Important observations from by using custom summary function and visualisations are as follows

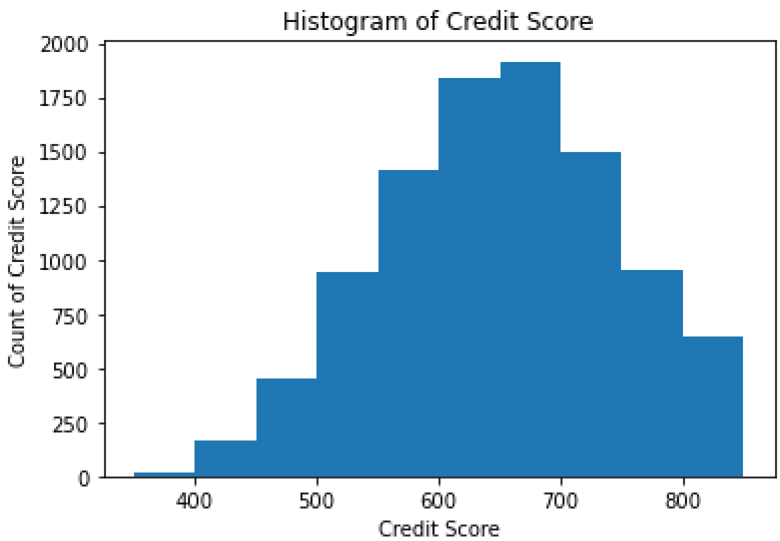
In [26]:

```
plt.hist(df["Credit_Score"])
plt.title("Histogram of Credit Score")
plt.xlabel("Credit Score")
plt.ylabel("Count of Credit Score")
```

executed in 650ms, finished 09:16:20 2022-06-06

Out[26]:

```
Text(0, 0.5, 'Count of Credit Score')
```



- Credit score is distributed almost normally at the same time peakedness is also not present in the credit score though mean is not representing the data as well median is somewhat similar to mean so credit score is distributed throughout the range similarly

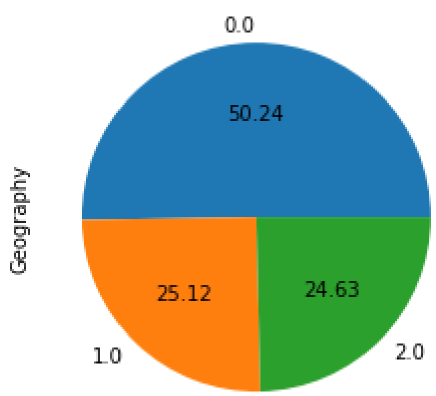
In [27]:

```
df["Geography"].value_counts().plot(kind="pie",autopct = "%0.2f")
```

executed in 109ms, finished 09:16:20 2022-06-06

Out[27]:

```
<AxesSubplot:ylabel='Geography'>
```



- Around 50 % of clients belongs to same geographical location and total catgories are three other two locations have 25% clients each

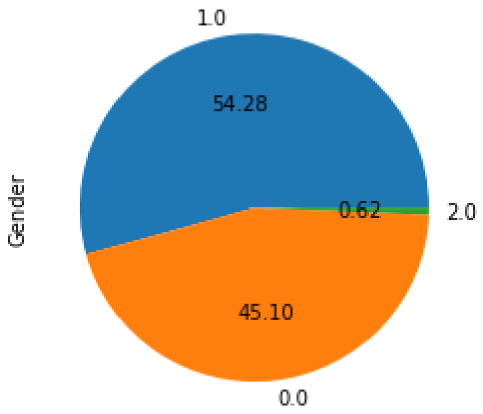
In [28]:

```
df["Gender"].value_counts().plot(kind="pie",autopct = "%0.2f")
```

executed in 97ms, finished 09:16:20 2022-06-06

Out[28]:

```
<AxesSubplot:ylabel='Gender'>
```



- We have data of 55% male and 45% female clients, we can also so that proportion of male and female clients in bank is somewhat similar

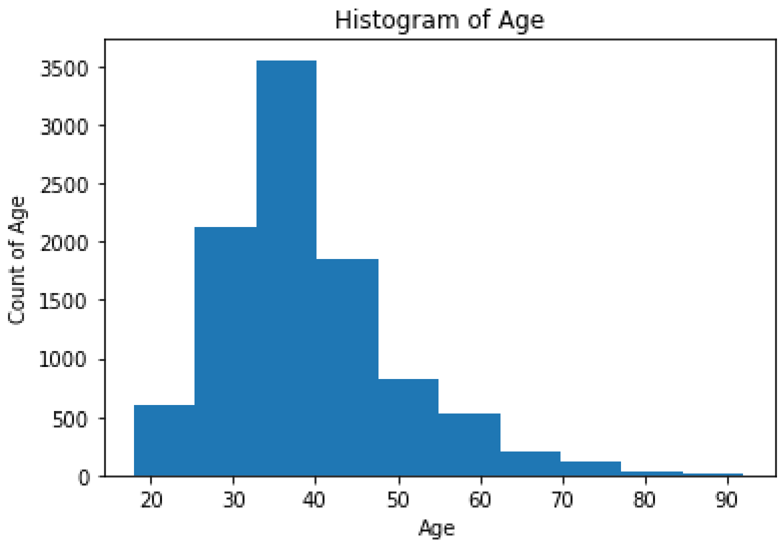
In [29]:

```
plt.hist(df["Age"])
plt.title("Histogram of Age")
plt.xlabel("Age")
plt.ylabel("Count of Age")
```

executed in 281ms, finished 09:16:20 2022-06-06

Out[29]:

```
Text(0, 0.5, 'Count of Age')
```



- Great thing is 50% of clients belongs to the range from 32 to 44 which is a good thing as this is the most active category in financial perspective, mean and median are almost similar as well as kurtosis is so high it is highly leptokurtic curve so we can say that mean is definately representing the data. Age is highly positively skewed so we can say that qurtile 3 is very lengthy and only 25% clients belongs to this category from 44-92.

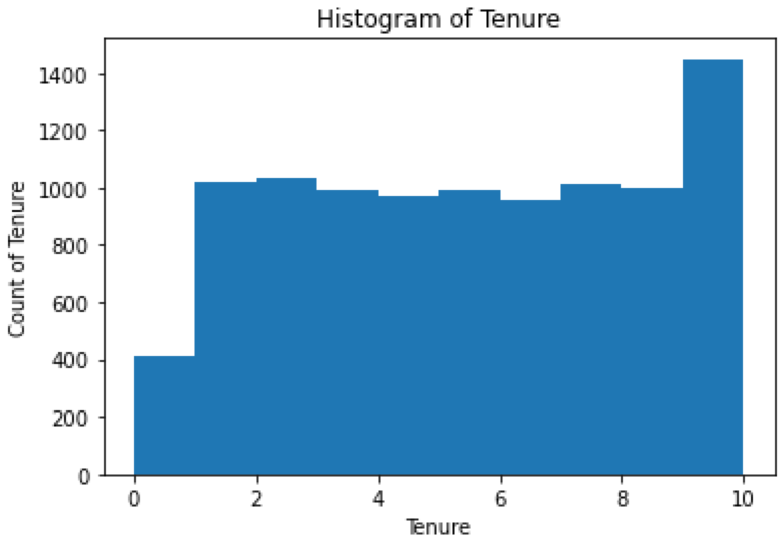
In [30]:

```
plt.hist(df["Tenure"])
plt.title("Histogram of Tenure")
plt.xlabel("Tenure")
plt.ylabel("Count of Tenure")
```

executed in 156ms, finished 09:16:20 2022-06-06

Out[30]:

```
Text(0, 0.5, 'Count of Tenure')
```



- When it comes to tenure it is distributed so normally in the range between 0 to 10. Mean and median are same that is 5 Years and kurtosis is highly platykurtic so mean is not representing the data at all. Inter - Quartile distribution is also similar. All four qurtiles have almost same range that is 2,3,2,3 years. We have clients from 0 to 10 years of tenure having almost similar counts.

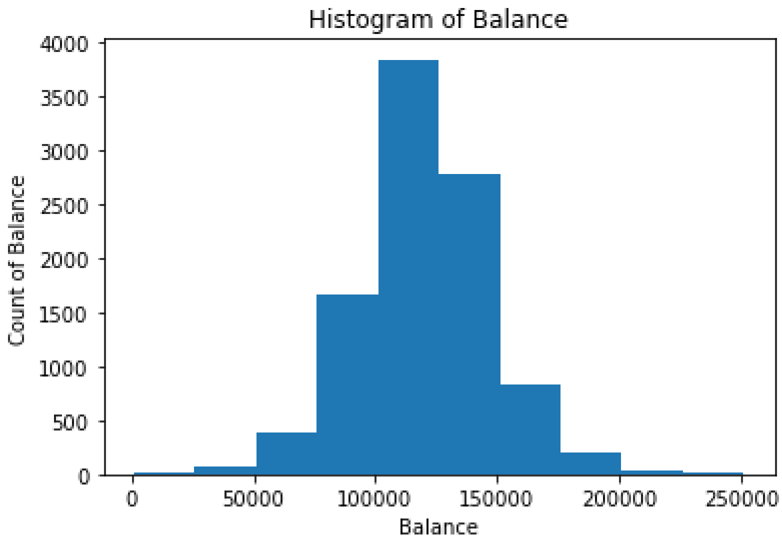
In [31]:

```
plt.hist(df["Balance"])
plt.title("Histogram of Balance")
plt.xlabel("Balance")
plt.ylabel("Count of Balance")
```

executed in 164ms, finished 09:16:21 2022-06-06

Out[31]:

```
Text(0, 0.5, 'Count of Balance')
```



- Balance is distributed normally and have leptokuric curve so mean is good it is representing the data. Interqurtile range is very small as compared to minimum and maximum values so there is concentration in balance thats why mean is representing the data. Data is highly concentrated in between inter-quartile range.

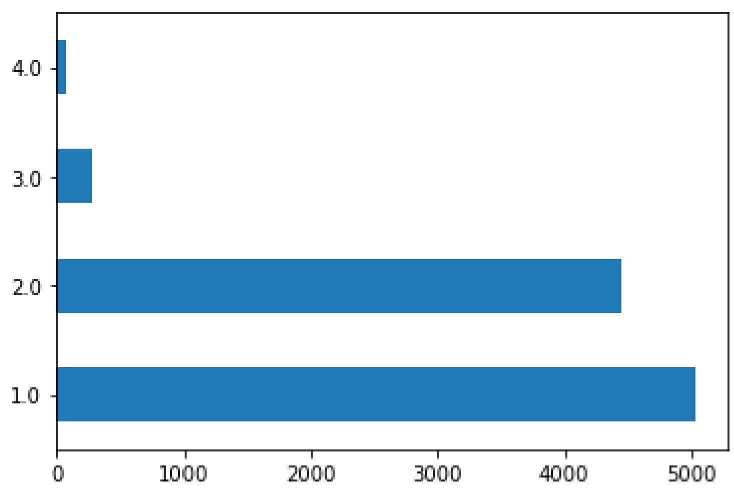
In [32]:

```
df["Num_Of_Policies"].value_counts().plot(kind="barh")
```

executed in 149ms, finished 09:16:21 2022-06-06

Out[32]:

```
<AxesSubplot:>
```



- 52 % of peoples have only 1 policy , 45% peoples have 2 policies, 2.5 % peoples have 3 policies and and only 0.5% have 4 policies need to promote policies throughout various branches.

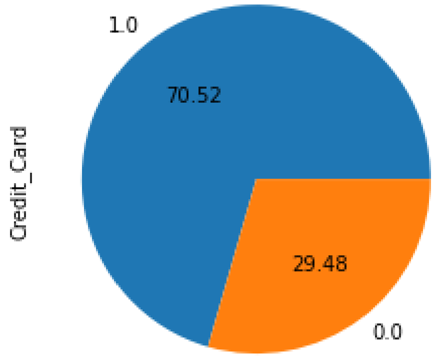
In [33]:

```
df["Credit_Card"].value_counts().plot(kind="pie",autopct = "%0.2f")
```

executed in 163ms, finished 09:16:21 2022-06-06

Out[33]:

```
<AxesSubplot:ylabel='Credit_Card'>
```



- Around 70% of clients have credit card which is a great number still there is scope for improvment

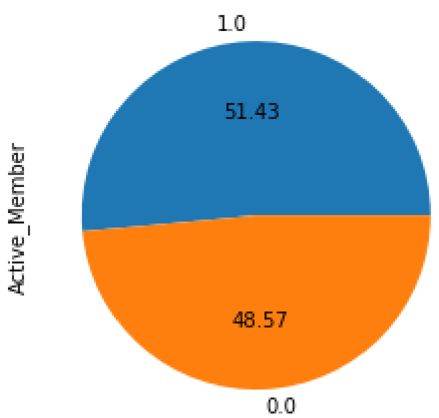
In [34]:

```
df["Active_Member"].value_counts().plot(kind="pie",autopct = "%0.2f")
```

executed in 116ms, finished 09:16:21 2022-06-06

Out[34]:

```
<AxesSubplot:ylabel='Active_Member'>
```



- Out of above available data only 50 % members are active which not a good thing at all. Business improvment and service and support team must have to go in deep to find out reasons as well as to find out remedial actions to improve above numbers.

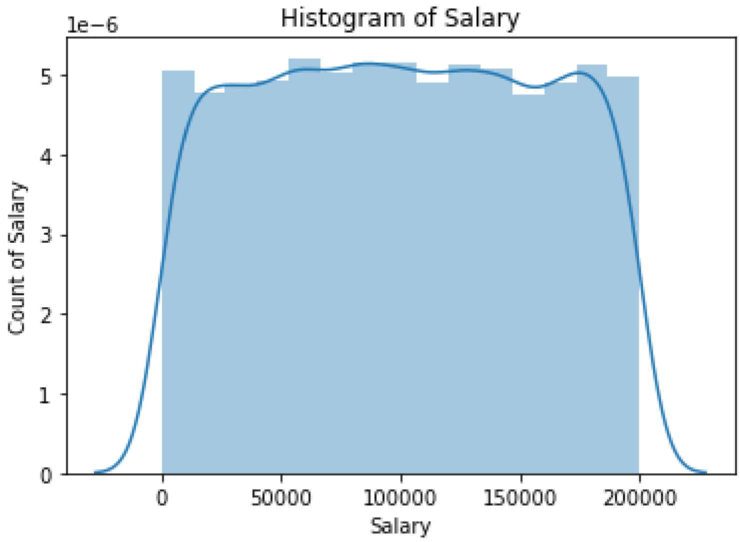
In [35]:

```
sns.distplot(df["Salary"],bins=15)
plt.title("Histogram of Salary")
plt.xlabel("Salary")
plt.ylabel("Count of Salary")
```

executed in 226ms, finished 09:16:21 2022-06-06

Out[35]:

```
Text(0, 0.5, 'Count of Salary')
```



- When it comes to salary it is distributed so normally in the range between 0 to 200000. Mean and median are same that is both are around 100000 and kurtosis is moderately platykurtic so mean is not representing the data. Inter - Quartile distribution is also similar. All four qurtiles have almost same range that is around 50000.

In [36]:

```
df["Exited"].value_counts()
```

executed in 6ms, finished 09:16:21 2022-06-06

Out[36]:

```
0.0    7800
1.0    2032
Name: Exited, dtype: int64
```

Around 22% of clients has exited from bank which is big number.

Every feature contains outliers and the % of them are exactly similar with each other so there is a strong possibility that some particular records from the data are noisy

Lets visualise outliers by using outlier detection plots

In [37]:

```
def outlier_treatment(df,col,method="quartile",strategy = "median"):
    col_data = df[col]

    # using quartile method to find outliers

    if method == "quartile":
        q2 = df[col].median()
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3-q1
        lowerlimit = q1 - 1.5*iqr
        upperlimit = q3 + 1.5*iqr

     # using std dav method to find outliers

    elif method == "standerd_daviation":
        col_mean = df[col].mean()
        col_std = df[col].std()
        lowerlimit = col_mean - 2*col_std
        upperlimit = col_mean + 2*col_std

    else:
        print("Pass a correct method")

    # printing outliers

    outliers = df.loc[( col_data < lowerlimit ) | (col_data > upperlimit) ,col]
    outlier_density = round(len(outliers)/len(df),4)

    if len(outliers) == 0 :
        print(f"the column {col} has no outliers")
    else:
        print(f"the column {col} has outliers")
        print("the outlier percentage is", outlier_density)
        print("outliers of column are : ")
        display(df[( col_data < lowerlimit ) | (col_data > upperlimit)])

    ## replacing outliers
    if strategy == "median":
        df.loc[( col_data < lowerlimit ) | (col_data > upperlimit) ,col] = df[col].me

    elif strategy == "mean":
        df.loc[( col_data < lowerlimit ) | (col_data > upperlimit) ,col] = df[col].me

    else:
        print("Pass a correct strategy")

    return (df)
```

executed in 116ms, finished 09:16:21 2022-06-06

ODT plots : Outlier detection plot

- in this we plot three graphs

- 1 Boxplot for descriptive statistics
- 2 histogram with outliers
- 3 histogram without outliers

In [38]:

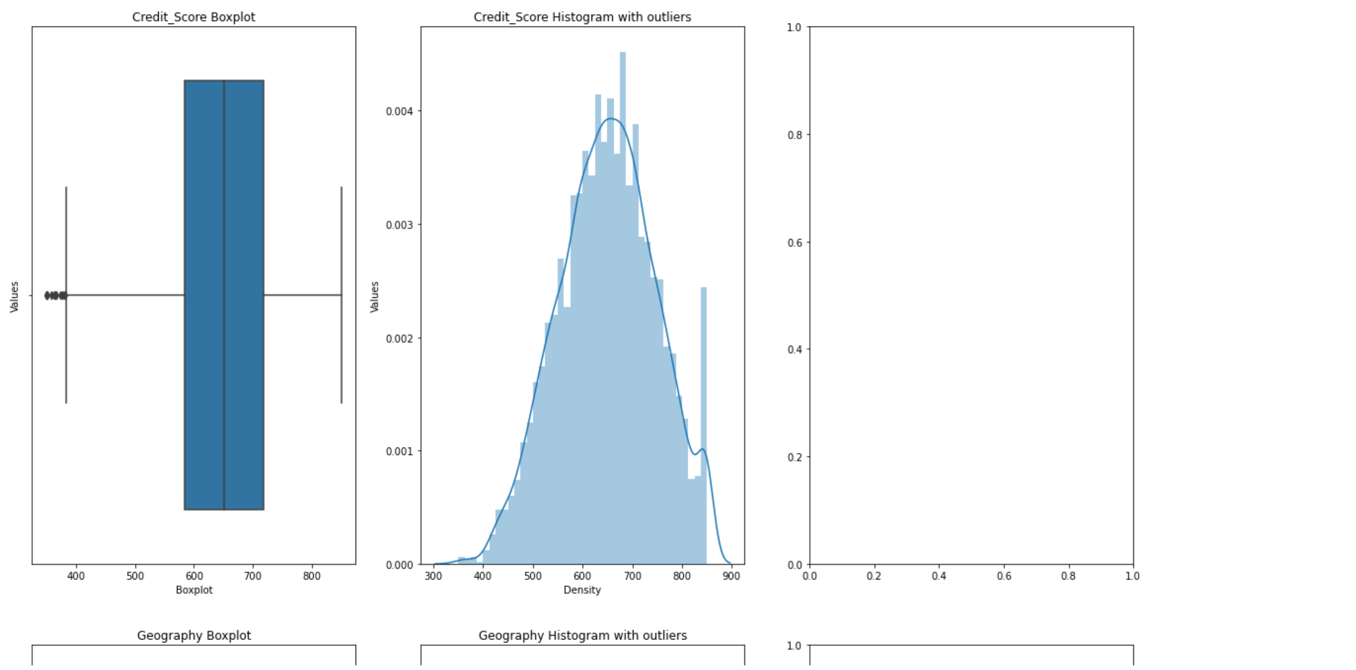
```
def odt_plot(df,col):
    f,(ax1,ax2,ax3) = plt.subplots(1,3,figsize=(20,10))

    # plotting boxplot

    sns.boxplot(df[col],ax=ax1)
    ax1.set_title(col + " Boxplot")
    ax1.set_xlabel("Boxplot")
    ax1.set_ylabel("Values")

    # plotting Histogram with outliers

    sns.distplot(df[col],ax=ax2)
    ax2.set_title(col + " Histogram with outliers")
    ax2.set_xlabel("Density")
    ax2.set_ylabel("Values")


    # plotting Histogram without outliers

    #y = outlier_treatment(df,col)

    #sns.distplot(y[col],ax=ax3)
    #ax3.set_title(col + " Histogram without outliers")
   # ax3.set_xlabel("Density")
    #ax3.set_ylabel("Values")
```

executed in 125ms, finished 09:16:22 2022-06-06

In [39]:

```
for col in df.columns:
    odt_plot(df,col)
```

executed in 4.12s, finished 09:16:26 2022-06-06



In [40]:

```
df
```

executed in 24ms, finished 09:16:26 2022-06-06

Out[40]:

	Credit_Score	Geography	Gender	Age	Tenure	Balance	Num_Of_Policies	Credit_Carc
0	619.0	0.0	0.0	42.0	2.0	144321.21	1.0	1.(
1	608.0	2.0	0.0	41.0	1.0	83807.86	1.0	0.(
2	502.0	0.0	0.0	42.0	8.0	159660.80	3.0	1.(
3	699.0	0.0	0.0	39.0	1.0	124499.37	2.0	0.(
4	850.0	2.0	0.0	43.0	2.0	125510.82	1.0	1.(
...	...	...	...	...	...	...	...	..
9827	711.0	2.0	1.0	86.0	1.0	161425.00	2.0	1.(
9828	447.0	0.0	1.0	31.0	8.0	79591.00	3.0	0.(
9829	777.0	0.0	0.0	35.0	3.0	80701.00	2.0	1.(
9830	450.0	0.0	1.0	60.0	9.0	1053.00	4.0	1.(
9831	819.0	1.0	1.0	45.0	3.0	94661.00	3.0	1.(

9832 rows × 11 columns

Not treating outliers as we are working on the target variable in the classification format.

so we are going to apply machine learning models which does not get affected by outliers.

There is issue with logistic regrssion as higher range outliers affect it.

But in this case we dont have higher range outliers so it is fine to avoid outlier treatment.

Correlation Plot

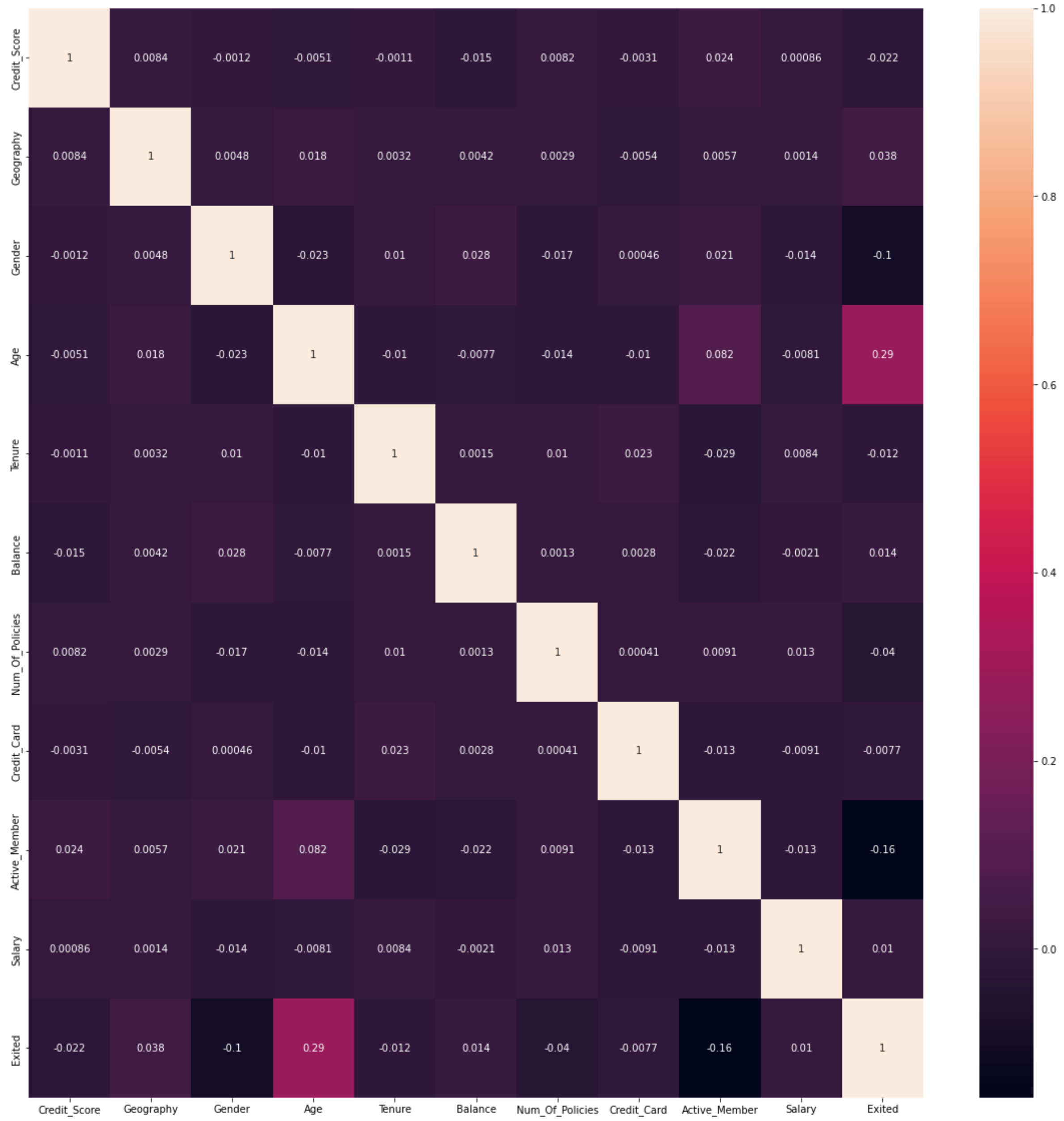
In [41]:

```python
## checking for multi-collinearity

cor = df.corr()
f,ax = plt.subplots(figsize=(20,20))
sns.heatmap(cor,annot=True)
```

executed in 772ms, finished 09:16:26 2022-06-06

Out[41]:

<AxesSubplot:>



- Multi-collinearity is not present in the data
- Age has a good positive correlation with target variable which is exited
- Activeness of a person and target variable has week negative correlation
- multicollinearity is not present in the data

Model Selection

- Lets use pairplots for data visualisation and by using them we will try to decide which models will be the best fitting ones for above data.

In [42]:

```
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df.drop(["Exited"],axis=1))
```

executed in 10ms, finished 09:16:27 2022-06-06

In [43]:

```
df_scaled = pd.DataFrame(df_scaled,columns=df.columns[:-1])
```

executed in 92ms, finished 09:16:27 2022-06-06

In [44]:

```
df_scaled
```

executed in 138ms, finished 09:16:27 2022-06-06

Out[44]:

	Credit_Score	Geography	Gender	Age	Tenure	Balance	Num_Of_Policies	Cr
0	-0.322928	-0.899999	-1.090237	0.282345	-1.038819	0.922432	-0.898396	
1	-0.436442	1.519687	-1.090237	0.188009	-1.384202	-1.347057	-0.898396	
2	-1.530311	-0.899999	-1.090237	0.282345	1.033479	1.497727	2.482668	
3	0.502633	-0.899999	-1.090237	-0.000662	-1.384202	0.179035	0.792136	
4	2.060880	1.519687	-1.090237	0.376681	-1.038819	0.216968	-0.898396	
...	...	...	...	...	...	...	...	
9827	0.626468	1.519687	0.873348	4.433118	-1.384202	1.563892	0.792136	
9828	-2.097884	-0.899999	0.873348	-0.755348	1.033479	-1.505206	2.482668	
9829	1.307556	-0.899999	-1.090237	-0.378005	-0.693436	-1.463577	0.792136	
9830	-2.066926	-0.899999	0.873348	1.980388	1.378862	-4.450691	4.173201	
9831	1.740975	0.309844	0.873348	0.565352	-0.693436	-0.940022	2.482668	

9832 rows × 10 columns

- Did standardisation of data as some algorithms such as KNN, SVM gets affected due to scales of data.
- Variables with higher scales have more impact on target variables.
- So to avoid such impacts we do standardisation.

Pair Plot

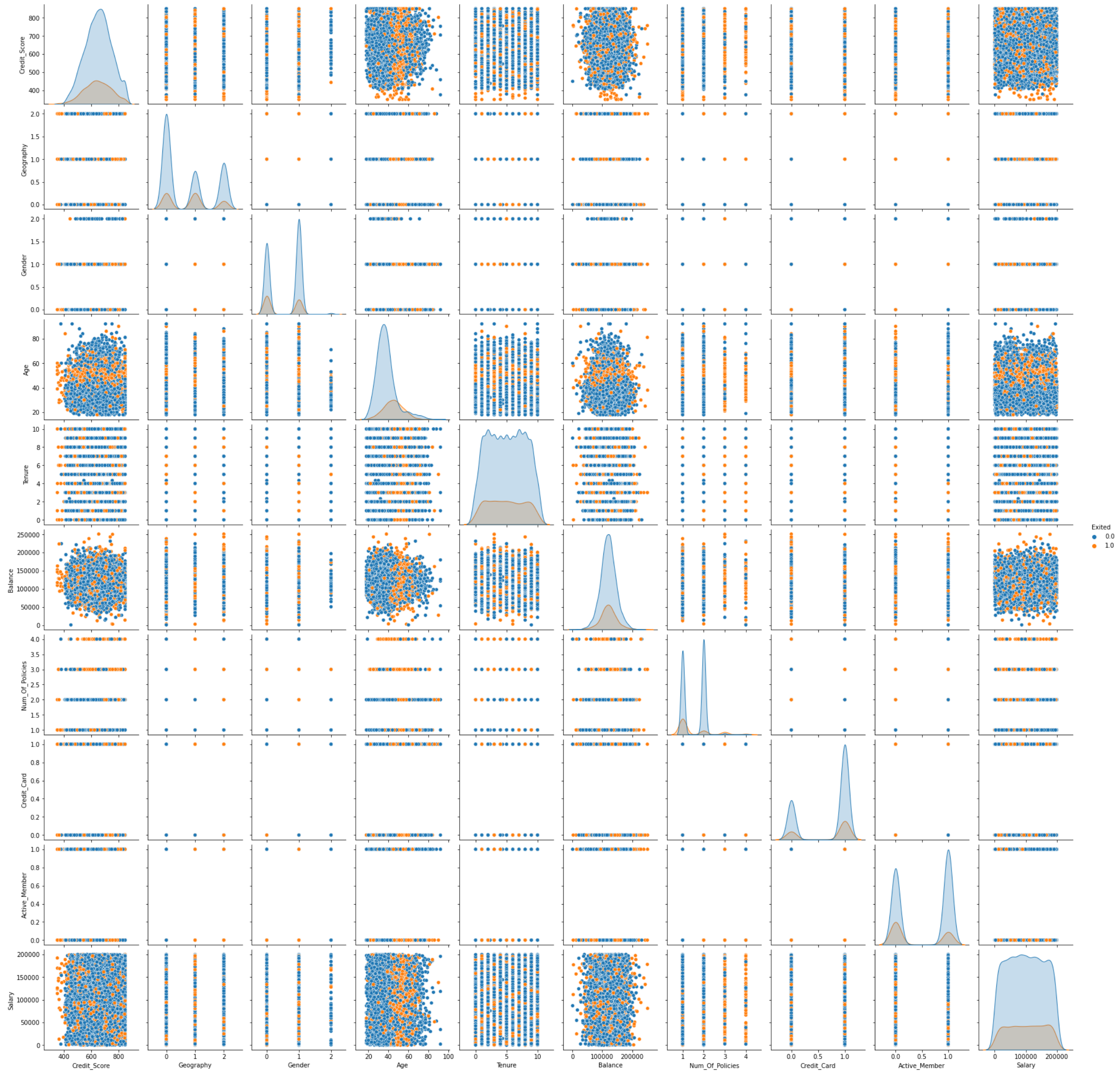
In [45]:

```
sns.pairplot(df,hue="Exited")
```

executed in 56.6s, finished 09:17:23 2022-06-06

Out[45]:

<seaborn.axisgrid.PairGrid at 0x2c27ae07700>



- 1. From above regplots we can say that there is lot of overlapping so there are vary less chances that logistic regression will work properly as it may have so much errrors and accurcay will be low.
- 2. There is high chances that Decision Tree classifier or Random forest classifier may work better in this case beacuase data is non-linear and above mentioned algorithms doesnt get affected by non-linearity in the features. As well we can see typical grouping in graph so ensamble techniques may work well.
- 3. Lets check out which algorithm works best on this data.

Creating datframe for dependant and independant variables

In [46]:

```
X = df.drop("Exited",axis=1)
Y = df[["Exited"]]
```

executed in 5ms, finished 09:17:23 2022-06-06

Model Building

- 1. Normal model building
- 2. Cross validation
- 3. Hyper parameter tunning

In [47]:

```
def train_and_test_split(x,y,test_size=0.3):
    return train_test_split(x,y,test_size=test_size,random_state=40)
```

executed in 1.26s, finished 09:17:25 2022-06-06

In [48]:

```
def build_model(model_name,estimator,x,y):
    x_train,x_test,y_train,y_test = train_and_test_split(x,y)
    estimator.fit(x_train,y_train)
    y_pred = estimator.predict(x_test)
    acc_score = accuracy_score(y_test,y_pred)
    f1_Scoree = f1_score(y_test,y_pred)
    temp = [model_name,acc_score]

    return temp
```

executed in 101ms, finished 09:17:25 2022-06-06

In [49]:

```
build_model("LogisticRegression",LogisticRegression(),X,Y)
```

executed in 205ms, finished 09:17:25 2022-06-06

Out[49]:

```
['LogisticRegression', 0.7677966101694915]
```

In [96]:

```
def build_multiple_model(x,y):
    result_df = pd.DataFrame(columns=["Model_Name","Accuracy_Score"])

    result_df.loc[len(result_df)] = build_model("Logistic_Regression",LogisticRegressi
    result_df.loc[len(result_df)] = build_model("Ridge_Classifier",RidgeClassifier(alp
    result_df.loc[len(result_df)] = build_model("KNN_Classifier",KNeighborsClassifier(
    result_df.loc[len(result_df)] = build_model("Decision_Tree_Classifier",DecisionTre
    result_df.loc[len(result_df)] = build_model("Random_Forest_Classifier",RandomFores
    result_df.loc[len(result_df)] = build_model("Adaboost_Classifier",AdaBoostClassifi
    result_df.loc[len(result_df)] = build_model("GBoost_Classifier",GradientBoostingCl
    result_df.loc[len(result_df)] = build_model("XGB_Classifier",XGBClassifier(max_dep
    result_df.loc[len(result_df)] = build_model("Support_Vector_Classifier",SVC(),x,y)

    return result_df.sort_values("Accuracy_Score",ascending=False)
```

executed in 9ms, finished 12:36:47 2022-06-06

In [51]:

```
build_multiple_model(X,Y)
```

executed in 4.83s, finished 09:17:30 2022-06-06

Out[51]:

	Model_Name	Accuracy_Score
6	GBoost_Classifier	0.854915
4	Random_Forest_Classifier	0.853898
5	Adaboost_Classifier	0.852542
7	XGB_Classifier	0.840339
1	Ridge_Classifier	0.784746
8	Support_Vector_Classifier	0.782373
3	Decision_Tree_Classifier	0.775593
0	Logistic_Regression	0.767797
2	KNN_Classifier	0.753559

Cross Validation

In [52]:

```
lr = LogisticRegression()
```

executed in 3ms, finished 09:17:30 2022-06-06

In [53]:

```
def cross_val(model_name,estimator,x,y):
    stf = StratifiedKFold(n_splits=10,shuffle=True,random_state=40)
    acc = []
    for train_index,test_index in stf.split(X,y):
        X_train,X_test = X.iloc[train_index],X.iloc[test_index]
        y_train,y_test = y.iloc[train_index],y.iloc[test_index]
        estimator.fit(X_train,y_train)
        y_pred = estimator.predict(X_test)
        accuracy_score(y_pred,y_test)
        acc.append(accuracy_score(y_pred,y_test))


    return np.mean(acc)
```

executed in 485ms, finished 09:17:30 2022-06-06

In [54]:

```
cross_val("Logistic_Regression",LogisticRegression(),X,Y)
```

executed in 579ms, finished 09:17:31 2022-06-06

Out[54]:

```
0.7812245159582827
```

In [55]:

```
def stratified_k_fold(x,y):
    lr_score = cross_val("Logistic_Regression",LogisticRegression(),x,y)
    rd_score = cross_val("Ridge_Classifier",RidgeClassifier(),x,y)
    dtc_score = cross_val("Decision_Tree",DecisionTreeClassifier(),x,y)
    knn_score = cross_val("KNN_Classifier",KNeighborsClassifier(),x,y)
    rf_score = cross_val("Random_Forest_Classifier",RandomForestClassifier(),x,y)
    ad_score = cross_val("Adaboost_Classifier",AdaBoostClassifier(),x,y)
    g_score = cross_val("GBoost_Classifier",GradientBoostingClassifier(),x,y)
    xgb_score = cross_val("XGB_Classifier",XGBClassifier(),x,y)
    svc_score = cross_val("SVC_Classifier",SVC(),x,y)

    score = [lr_score,rd_score,dtc_score,knn_score,rf_score,ad_score,g_score,xgb_scor
    models = ["Logistic_Regression","Ridge_Classifier","Decision_Tree","KNN_Classifie

    result = []

    for i in range(0,len(models)):
        score_ = score[i]
        model_name = models[i]
        temp = [model_name,score_]
        result.append(temp)

    result_df = pd.DataFrame(result,columns=["model_name","score"])

    return result_df.sort_values("score",ascending=False)
```

executed in 7ms, finished 09:17:31 2022-06-06

In [56]:

```
stratified_k_fold(X,Y)
```

executed in 55.6s, finished 09:18:27 2022-06-06

Out[56]:

	model_name	score
6	GBoost_Classifier	0.859133
4	Random_Forest_Classifier	0.856998
5	Adaboost_Classifier	0.851097
7	XGB_Classifier	0.850285
1	Ridge_Classifier	0.797091
8	SVC_Classifier	0.793328
0	Logistic_Regression	0.781225
2	Decision_Tree	0.779293
3	KNN_Classifier	0.757729

Hyper Parameter Tunning

In [57]:

```
def hyperparameter_tunning(x,y,fold=10):

    param_rd = {"alpha": [1e-15, 1e-13,1e-11,1e-9,1e-7,1e-5, 1e-3,0.001,0.01,0.1,1,2,
    param_knn = {"n_neighbors":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}
    param_adb = {"n_estimators" : [1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100,2
         "learning_rate" : [0.00001,0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8
    param_gb = {"learning_rate" : [0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]}
    param_xgb = {'learning_rate' : [0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.
         'reg_lambda' : [1,2,3,4,5,6,7,8,9,10],
         'max_depth': [1,2,3,4,5,6,7,8,9,10]}
    param_rf = {"n_estimators":[10,20,30,40,50,60,70,80,90,110,130,150,170,190],'min_
               "bootstrap":[True,False]}



    tunned_ridge=GridSearchCV(RidgeClassifier(),param_rd,cv=fold)
    tunned_knn=GridSearchCV(KNeighborsClassifier(),param_knn,cv=fold)
    tunned_adb=GridSearchCV(AdaBoostClassifier(),param_adb,cv=fold)
    tunned_gb=GridSearchCV(GradientBoostingClassifier(),param_gb,cv=fold)
    tunned_xgb=GridSearchCV(XGBClassifier(),param_xgb,cv=fold)
    tunned_rf=GridSearchCV(RandomForestClassifier(),param_rf,cv=fold)


    tunned_ridge.fit(x,y)
    tunned_knn.fit(x,y)
    tunned_adb.fit(x,y)
    tunned_gb.fit(x,y)
    tunned_xgb.fit(x,y)
    tunned_rf.fit(x,y)


    tunned = [tunned_ridge,tunned_knn,tunned_adb,tunned_gb,tunned_xgb,tunned_rf]
    models = ["Ridge","KNeighborClassifier","AdaBoostClassifier","GradientBoostingCla

    for i in range (0,len(tunned)):
        print("model",models[i])
        print("best_parameters",tunned[i].best_params_)

```

executed in 12ms, finished 09:18:27 2022-06-06

In [58]:

```
#hyperparameter_tunning(X,Y,fold=10)
```

executed in 2h 37m 33s, finished 11:56:00 2022-06-06

```
model Ridge
best_parameters {'alpha': 1e-15}
model KNeighborClassifier
best_parameters {'n_neighbors': 14}
model AdaBoostClassifier
best_parameters {'learning_rate': 0.4, 'n_estimators': 40}
model GradientBoostingClassifier
best_parameters {'learning_rate': 0.1}
model XGBClassifier
best_parameters {'learning_rate': 0.1, 'max_depth': 5, 'reg_lambda': 2}
model RandomForestClassifier
best_parameters {'bootstrap': True, 'min_impurity_decrease': 0.0, 'n_estimat
ors': 70}
```

In [59]:

```
d_k_fold_tunned(x,y):
= cross_val("Logistic_Regression",LogisticRegression(),x,y)
= cross_val("Ridge_Classifier",RidgeClassifier(alpha=1e-15),x,y)
 = cross_val("Decision_Tree",DecisionTreeClassifier(),x,y)
 = cross_val("KNN_Classifier",KNeighborsClassifier(n_neighbors=14),x,y)
= cross_val("Random_Forest_Classifier",RandomForestClassifier(n_estimators=70,bootstrap=Tr
= cross_val("Adaboost_Classifier",AdaBoostClassifier(learning_rate=0.4,n_estimators=40),x,
cross_val("GBoost_Classifier",GradientBoostingClassifier(learning_rate=0.1),x,y)
 = cross_val("XGB_Classifier",XGBClassifier(max_depth=5,reg_lambda=2,learning_rate=0.1),x,
 = cross_val("SVC_Classifier",SVC(),x,y)

lr_score,rd_score,dtc_score,knn_score,rf_score,ad_score,g_score,xgb_score,svc_score]
["Logistic_Regression","Ridge_Classifier","Decision_Tree","KNN_Classifier","Random_Forest_

[]

range(0,len(models)):
_ = score[i]
_name = models[i]
= [model_name,score_]
lt.append(temp)

 = pd.DataFrame(result,columns=["model_name","score"])

sult_df.sort_values("score",ascending=False)
```

executed in 9ms, finished 12:06:47 2022-06-06

In [60]:

```
stratified_k_fold_tunned(X,Y)
```

executed in 49.5s, finished 12:07:57 2022-06-06

Out[60]:

	model_name	score
6	GBoost_Classifier	0.859133
7	XGB_Classifier	0.856794
5	Adaboost_Classifier	0.855471
4	Random_Forest_Classifier	0.854657
1	Ridge_Classifier	0.797091
8	SVC_Classifier	0.793328
3	KNN_Classifier	0.791802
2	Decision_Tree	0.782852
0	Logistic_Regression	0.781225

Clustering

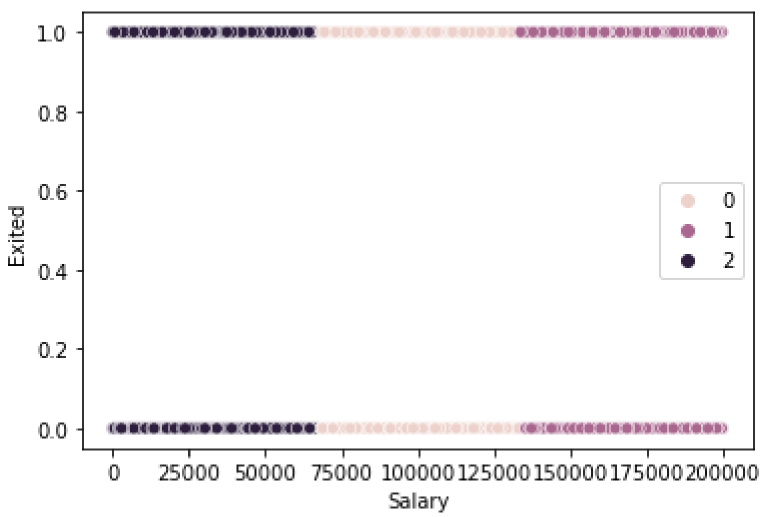
In [81]:

```
labels = KMeans(n_clusters=3,random_state=42).fit_predict(X)
sns.scatterplot(x=X.Salary,y=Y.Exited,hue=labels)
```

executed in 625ms, finished 12:22:08 2022-06-06

Out[81]:

```
<AxesSubplot:xlabel='Salary', ylabel='Exited'>
```



- Value of salary forming three different clusters with our dependant variable
- lets include this cluster as an independant variable
- And we will check out if any change is there in accucracy of our model.

In [93]:

```
df_salary_labels = pd.DataFrame(labels,columns=["Salary_Labels"])
df_salary_labels.head()
```

executed in 10ms, finished 12:30:39 2022-06-06

Out[93]:

	Salary_Labels
0	1
1	1
2	1
3	0
4	0

In [94]:

```
X = X.join(df_salary_labels,how="inner")
X.head()
```

executed in 36ms, finished 12:31:23 2022-06-06

Out[94]:

	Credit_Score	Geography	Gender	Age	Tenure	Balance	Num_Of_Policies	Credit_Card
0	619.0	0.0	0.0	42.0	2.0	144321.21	1.0	1.0
1	608.0	2.0	0.0	41.0	1.0	83807.86	1.0	0.0
2	502.0	0.0	0.0	42.0	8.0	159660.80	3.0	1.0
3	699.0	0.0	0.0	39.0	1.0	124499.37	2.0	0.0
4	850.0	2.0	0.0	43.0	2.0	125510.82	1.0	1.0

- Lets build models and check accuracy by using newly developed Dataframe of independant variables.

In [97]:

```
build_multiple_model(X,Y)
```

executed in 4.47s, finished 12:36:58 2022-06-06

Out[97]:

	Model_Name	Accuracy_Score
6	GBoost_Classifier	0.854915
7	XGB_Classifier	0.853898
4	Random_Forest_Classifier	0.852881
5	Adaboost_Classifier	0.850847
1	Ridge_Classifier	0.784068
8	Support_Vector_Classifier	0.782373
3	Decision_Tree_Classifier	0.781695
2	KNN_Classifier	0.781017
0	Logistic_Regression	0.767797

In [98]:

```
stratified_k_fold_tunned(X,Y)
```

executed in 49.4s, finished 12:38:47 2022-06-06

Out[98]:

	model_name	score
6	GBoost_Classifier	0.859235
7	XGB_Classifier	0.856997
5	Adaboost_Classifier	0.855471
4	Random_Forest_Classifier	0.853132
1	Ridge_Classifier	0.797192
8	SVC_Classifier	0.793328
3	KNN_Classifier	0.791802
2	Decision_Tree	0.781429
0	Logistic_Regression	0.781225

Learning Curves for models

In [99]:

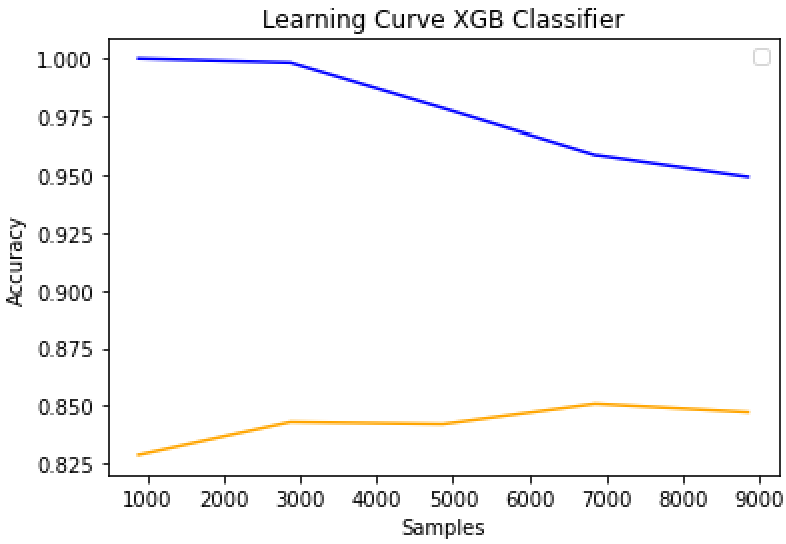
```
def genrate_learning_curve(model_name,estimator,x,Y,fold=10):
    train_size,train_score,test_score = learning_curve(estimator=estimator,X=x,y=Y,cv
    train_score_mean = np.mean(train_score,axis=1)
    test_score_mean = np.mean(test_score,axis=1)
    plt.plot(train_size,train_score_mean,color="blue")
    plt.plot(train_size,test_score_mean,color="orange")

    plt.xlabel("Samples")
    plt.ylabel("Accuracy")
    plt.title("Learning Curve " + model_name)
    plt.legend("Training accurcy","Testing accurcy")
```

executed in 6ms, finished 12:43:18 2022-06-06

In [100]:

```
genrate_learning_curve("XGB Classifier",XGBClassifier(),X,Y)
```

executed in 15.5s, finished 12:44:23 2022-06-06



Feature Importance Using XGBoost Package

In [101]:

```
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.2,random_state=42)

xgb = XGBClassifier()
xgb.fit(x_train,y_train)
```

executed in 505ms, finished 12:47:08 2022-06-06

Out[101]:

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=
1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=
0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [102]:

```
xgboost.plot_importance(xgb)
```

executed in 176ms, finished 12:47:30 2022-06-06

Out[102]:

```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel
='Features'>
```