Capstone Final Report_ Rahul jha

Project Notes -1

In [1]:

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

In [2]:

```
edth=pd.read_excel("Customer Churn Data.xlsx")
```

In [3]:

```
edth.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11260 entries, 0 to 11259
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   AccountID                11260 non-null  int64  
 1   Churn                    11260 non-null  int64  
 2   Tenure                   11158 non-null  object 
 3   City_Tier                11148 non-null  float64
 4   CC_Contacted_LY          11158 non-null  float64
 5   Payment                  11151 non-null  object 
 6   Gender                   11152 non-null  object 
 7   Service_Score            11162 non-null  float64
 8   Account_user_count       11148 non-null  object 
 9   account_segment          11163 non-null  object 
 10  CC_Agent_Score           11144 non-null  float64
 11  Marital_Status           11048 non-null  object 
 12  rev_per_month            11158 non-null  object 
 13  Complain_ly              10903 non-null  float64
 14  rev_growth_yoy           11260 non-null  object 
 15  coupon_used_for_payment  11260 non-null  object 
 16  Day_Since_CC_connect     10903 non-null  object 
 17  cashback                 10789 non-null  object 
 18  Login_device             11039 non-null  object 
dtypes: float64(5), int64(2), object(12)
memory usage: 1.6+ MB
```

b) Visual inspection of data (rows, columns, descriptive details)

In [4]:

```
edth.head(20).T
```

Out[4]:

	0	1	2	3	4	5	6	
AccountID	20000	20001	20002	20003	20004	20005	20006	20(
Churn	1	1	1	1	1	1	1	
Tenure	4	0	0	0	0	0	2	
City_Tier	3	1	1	3	1	1	3	
CC_Contacted_LY	6	8	30	15	12	22	11	
Payment	Debit Card	UPI	Debit Card	Debit Card	Credit Card	Debit Card	Cash on Delivery	Cr C
Gender	Female	Male	Male	Male	Male	Female	Male	M
Service_Score	3	3	2	2	2	3	2	
Account_user_count	3	4	4	4	3	NaN	3	
account_segment	Super	Regular Plus	Regular Plus	Super	Regular Plus	Regular Plus	Super	Regu F
CC_Agent_Score	2	3	3	5	5	5	2	
Marital_Status	Single	Single	Single	Single	Single	Single	Divorced	Divor(
rev_per_month	9	7	6	8	3	2	4	
Complain_ly	1	1	1	0	0	1	0	
rev_growth_yoy	11	15	14	23	11	22	14	
coupon_used_for_payment	1	0	0	0	1	4	0	
Day_Since_CC_connect	5	0	3	3	3	7	0	
cashback	159.93	120.9	NaN	134.07	129.6	139.19	120.86	122
Login_device	Mobile	Mobile	Mobile	Mobile	Mobile	Computer	Mobile	Mol

In [5]:

```
edth.columns
```

Out[5]:

```
Index(['AccountID', 'Churn', 'Tenure', 'City_Tier', 'CC_Contacted_LY',
       'Payment', 'Gender', 'Service_Score', 'Account_user_count',
       'account_segment', 'CC_Agent_Score', 'Marital_Status', 'rev_per_mont
h',
       'Complain_ly', 'rev_growth_yoy', 'coupon_used_for_payment',
       'Day_Since_CC_connect', 'cashback', 'Login_device'],
      dtype='object')
```

In [6]:

```
edth.value_counts("Marital_Status")
```

Out[6]:

```
Marital_Status
Married     5860
Single      3520
Divorced    1668
dtype: int64
```

In [7]:

```
edth.value_counts("account_segment")
```

Out[7]:

```
account_segment
Super           4062
Regular Plus    3862
HNI             1639
Super Plus       771
Regular          520
Regular +        262
Super +           47
dtype: int64
```

In [8]:

```
edth.value_counts("Gender")
```

Out[8]:

```
Gender
Male      6328
Female    4178
M          376
F          270
dtype: int64
```

In [9]:

```
edth.value_counts("Payment")
```

Out[9]:

```
Payment
Debit Card          4587
Credit Card         3511
E wallet            1217
Cash on Delivery    1014
UPI                  822
dtype: int64
```

In [10]:

```
edth.size
```

Out[10]:

```
213940
```

In [11]:

```
edth.shape
```

Out[11]:

```
(11260, 19)
```

In [12]:

```
edth.isnull().sum()
```

Out[12]:

```
AccountID                    0
Churn                        0
Tenure                     102
City_Tier                  112
CC_Contacted_LY            102
Payment                    109
Gender                     108
Service_Score               98
Account_user_count         112
account_segment             97
CC_Agent_Score             116
Marital_Status             212
rev_per_month              102
Complain_ly                357
rev_growth_yoy               0
coupon_used_for_payment      0
Day_Since_CC_connect       357
cashback                   471
Login_device               221
dtype: int64
```

In [13]:

```
edth.count()
```

Out[13]:

```
AccountID                  11260
Churn                      11260
Tenure                     11158
City_Tier                  11148
CC_Contacted_LY            11158
Payment                    11151
Gender                     11152
Service_Score              11162
Account_user_count         11148
account_segment            11163
CC_Agent_Score             11144
Marital_Status             11048
rev_per_month              11158
Complain_ly                10903
rev_growth_yoy             11260
coupon_used_for_payment    11260
Day_Since_CC_connect       10903
cashback                   10789
Login_device               11039
dtype: int64
```

In [14]:

```
edth.duplicated().sum()
```

Out[14]:

```
0
```

In [15]:

```
edth.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%	
AccountID	11260.0	25629.500000	3250.626350	20000.0	22814.75	25629.5	28444.25	3
Churn	11260.0	0.168384	0.374223	0.0	0.00	0.0	0.00	
City_Tier	11148.0	1.653929	0.915015	1.0	1.00	1.0	3.00	
CC_Contacted_LY	11158.0	17.867091	8.853269	4.0	11.00	16.0	23.00	
Service_Score	11162.0	2.902526	0.725584	0.0	2.00	3.0	3.00	
CC_Agent_Score	11144.0	3.066493	1.379772	1.0	2.00	3.0	4.00	
Complain_ly	10903.0	0.285334	0.451594	0.0	0.00	0.0	1.00	

a) Univariate analysis (distribution and spread for every continuous attribute, distribution of data in categories

for categorical ones)

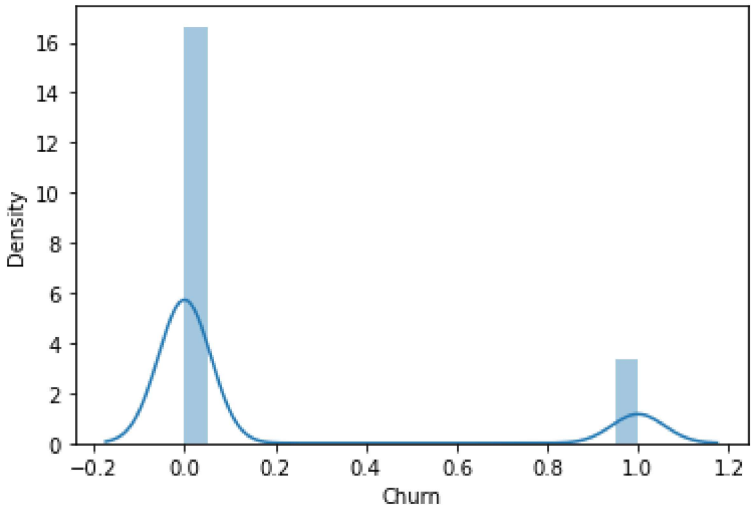
In [16]:

```
sns.distplot(edth.Churn,bins=20)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[16]:

```
<AxesSubplot:xlabel='Churn', ylabel='Density'>
```



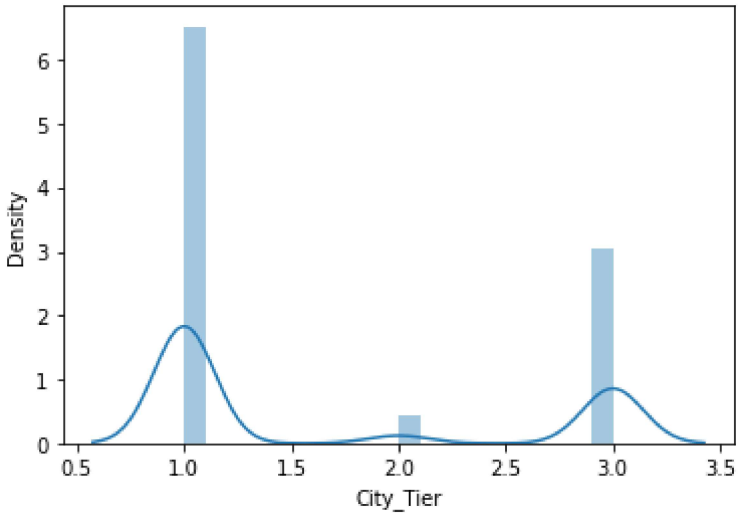
In [17]:

```
sns.distplot(edth.City_Tier,bins=20)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[17]:

```
<AxesSubplot:xlabel='City_Tier', ylabel='Density'>
```



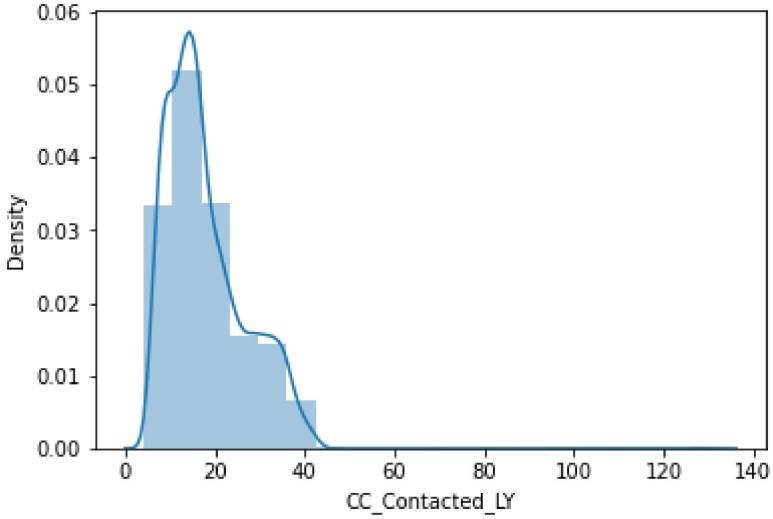
In [18]:

```
sns.distplot(edth.CC_Contacted_LY,bins=20)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[18]:

```
<AxesSubplot:xlabel='CC_Contacted_LY', ylabel='Density'>
```



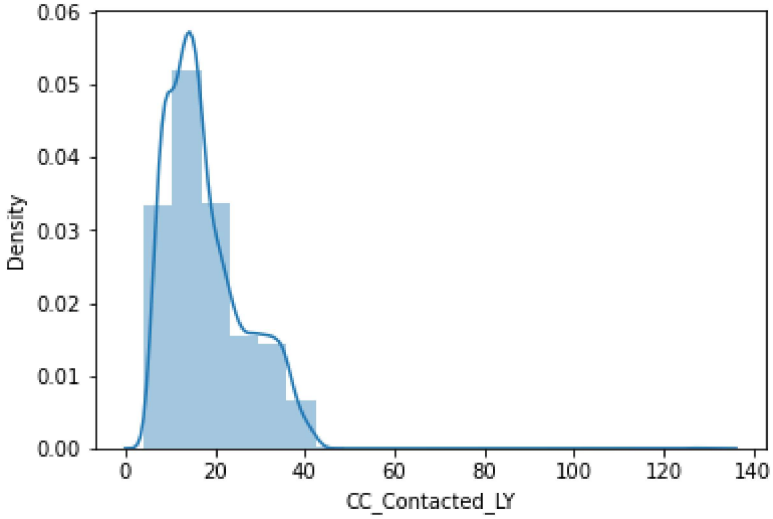
In [19]:

```
sns.distplot(edth.CC_Contacted_LY,bins=20)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[19]:

```
<AxesSubplot:xlabel='CC_Contacted_LY', ylabel='Density'>
```



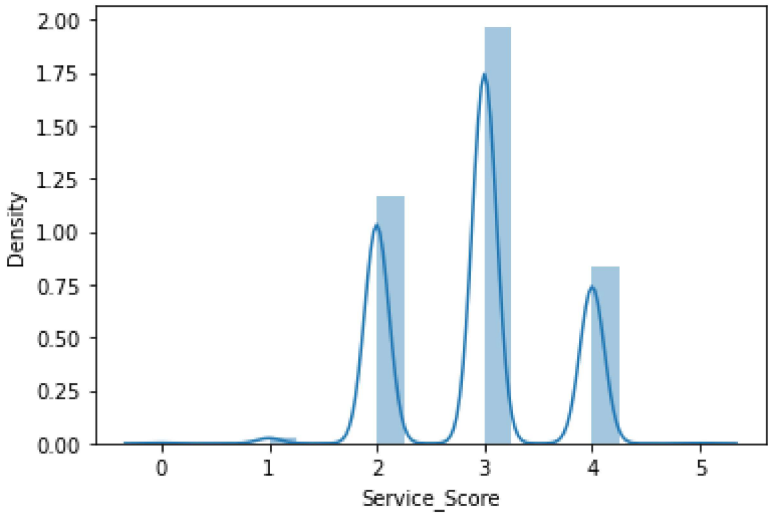
In [20]:

```
sns.distplot(edth.Service_Score,bins=20)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]:

```
<AxesSubplot:xlabel='Service_Score', ylabel='Density'>
```



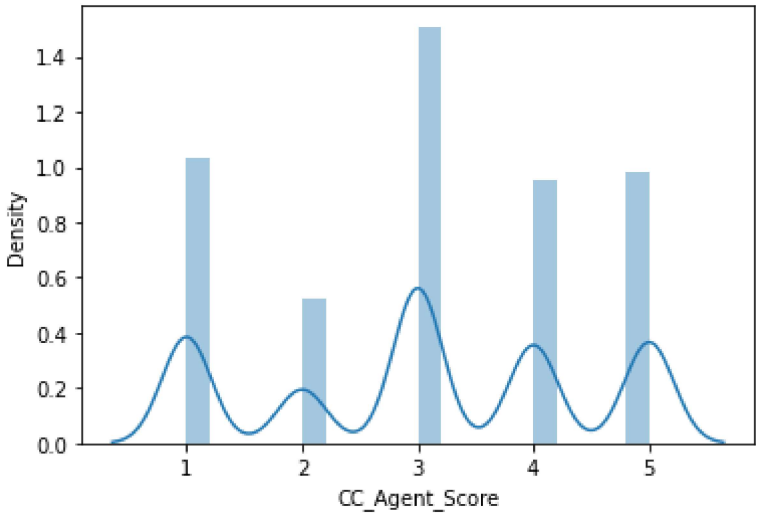
In [21]:

```
sns.distplot(edth.CC_Agent_Score,bins=20)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[21]:

```
<AxesSubplot:xlabel='CC_Agent_Score', ylabel='Density'>
```



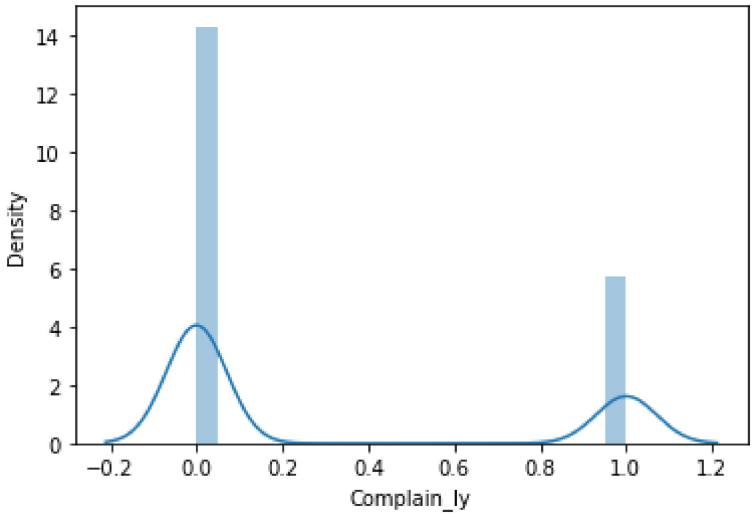
In [22]:

```
sns.distplot(edth.Complain_ly,bins=20)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[22]:

```
<AxesSubplot:xlabel='Complain_ly', ylabel='Density'>
```

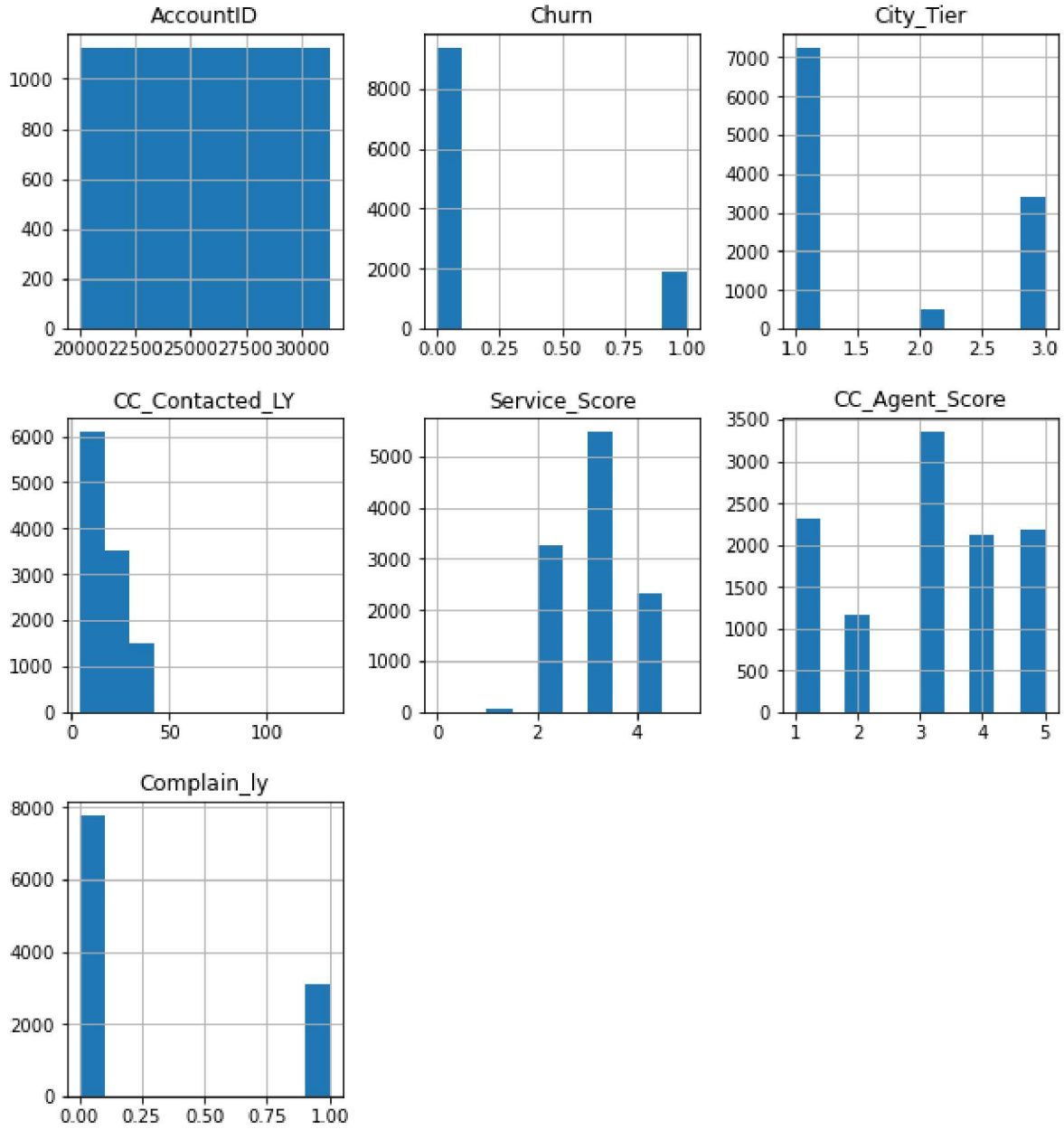


In [23]:

```
edth.hist(figsize=(10,11))
```

Out[23]:

```
array([[<AxesSubplot:title={'center':'AccountID'}>,
        <AxesSubplot:title={'center':'Churn'}>,
        <AxesSubplot:title={'center':'City_Tier'}>],
       [<AxesSubplot:title={'center':'CC_Contacted_LY'}>,
        <AxesSubplot:title={'center':'Service_Score'}>,
        <AxesSubplot:title={'center':'CC_Agent_Score'}>],
       [<AxesSubplot:title={'center':'Complain_ly'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```

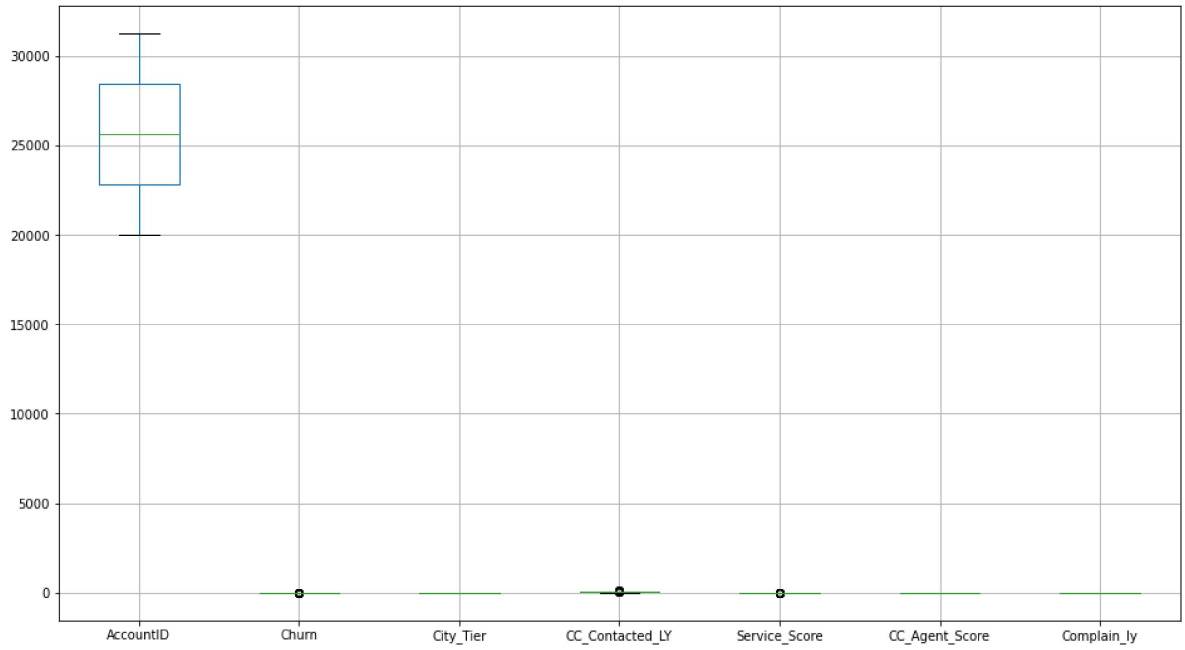


In [24]:

```
edth.boxplot(figsize=(16,9))
```

Out[24]:

```
<AxesSubplot:>
```



b) Bivariate analysis (relationship between different variables , correlations)

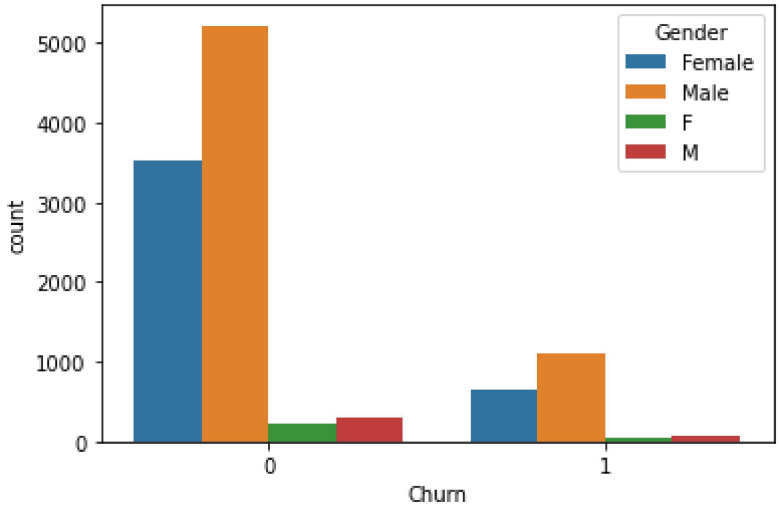
In [25]:

```
sns.countplot(edth["Churn"],hue=edth["Gender"])
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[25]:

```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



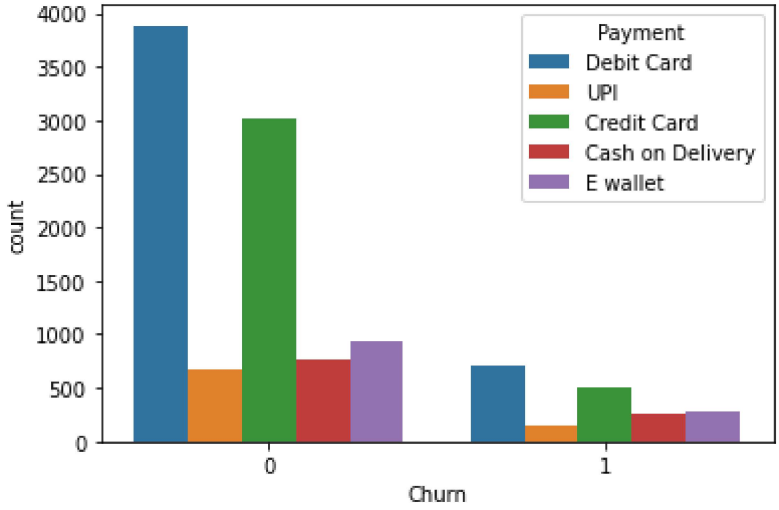
In [26]:

```
sns.countplot(edth["Churn"],hue=edth["Payment"])
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[26]:

```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



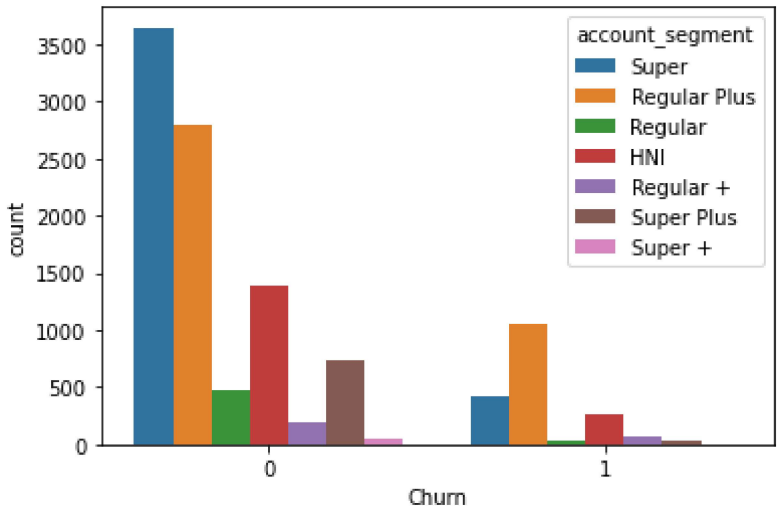
In [27]:

```
sns.countplot(edth["Churn"],hue=edth["account_segment"])
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[27]:

```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



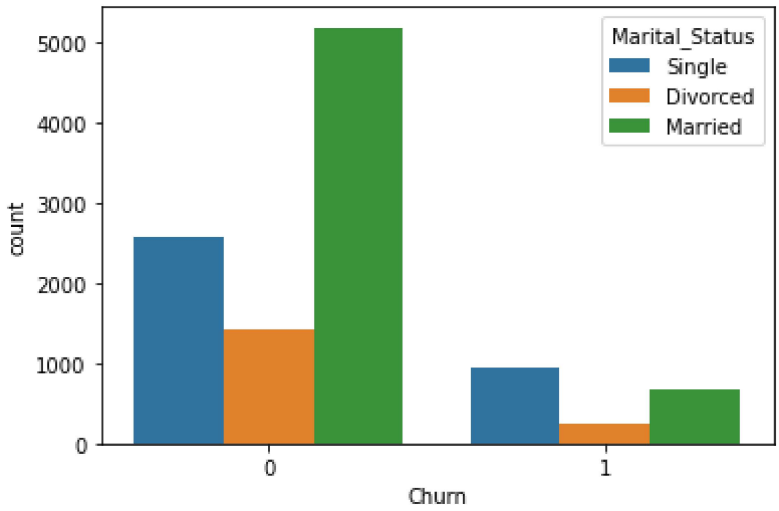
In [28]:

```
sns.countplot(edth["Churn"],hue=edth["Marital_Status"])
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[28]:

```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



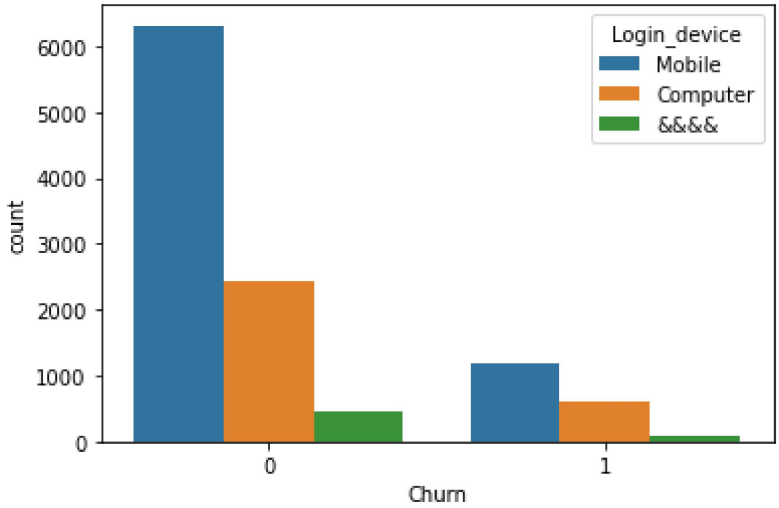
In [29]:

```
sns.countplot(edth["Churn"],hue=edth["Login_device"])
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[29]:

```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



In [30]:

```
corr=edth.corr()
corr
```

Out[30]:

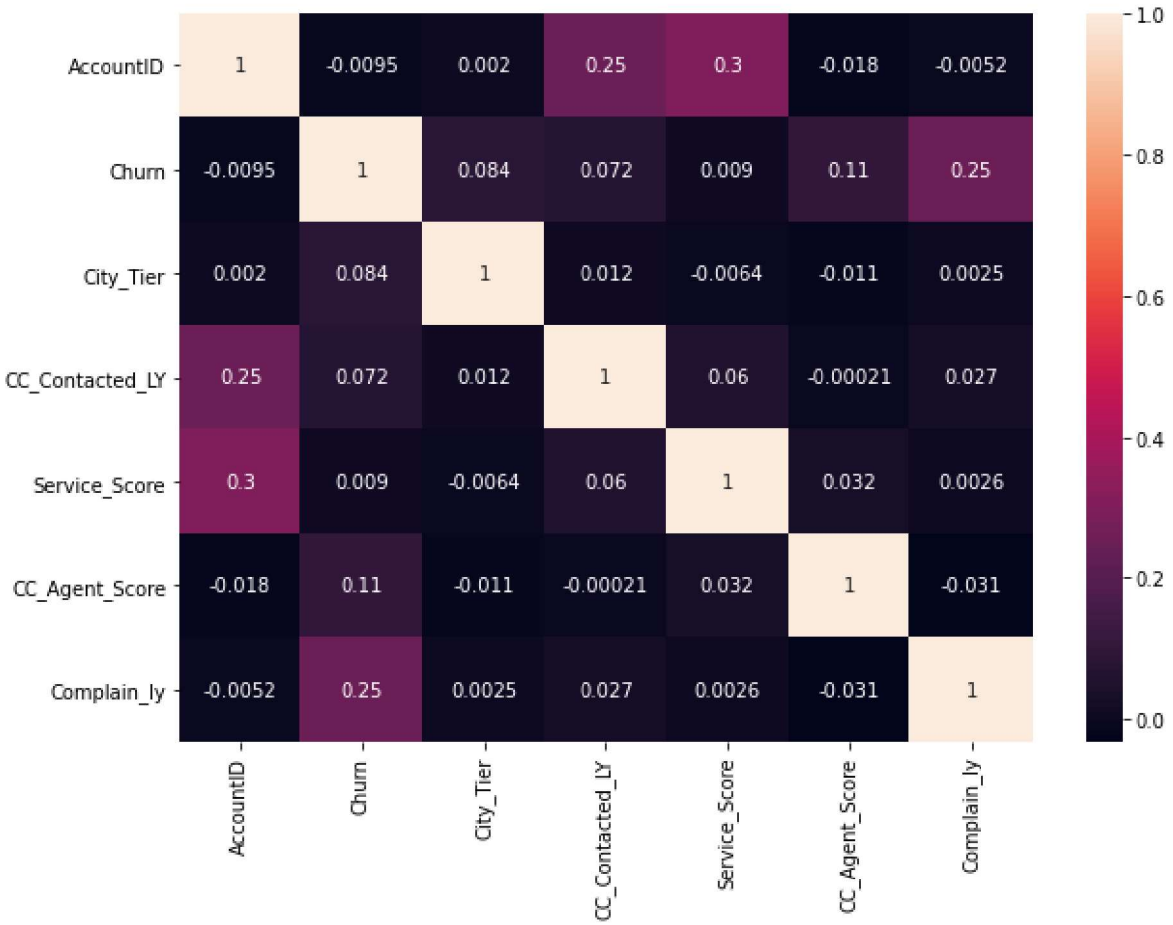
	AccountID	Churn	City_Tier	CC_Contacted_LY	Service_Score	CC_Agent
AccountID	1.000000	-0.009541	0.001983	0.252562	0.301782	-0.
Churn	-0.009541	1.000000	0.084135	0.072071	0.008991	0.
City_Tier	0.001983	0.084135	1.000000	0.011835	-0.006443	-0.
CC_Contacted_LY	0.252562	0.072071	0.011835	1.000000	0.060052	-0.
Service_Score	0.301782	0.008991	-0.006443	0.060052	1.000000	0.
CC_Agent_Score	-0.017715	0.105796	-0.011479	-0.000209	0.032135	1.
Complain_ly	-0.005168	0.251488	0.002478	0.027244	0.002643	-0.

In [31]:

```
plt.figure(figsize=(10,7))
sns.heatmap(corr,annot=True)
```

Out[31]:

```
<AxesSubplot:>
```



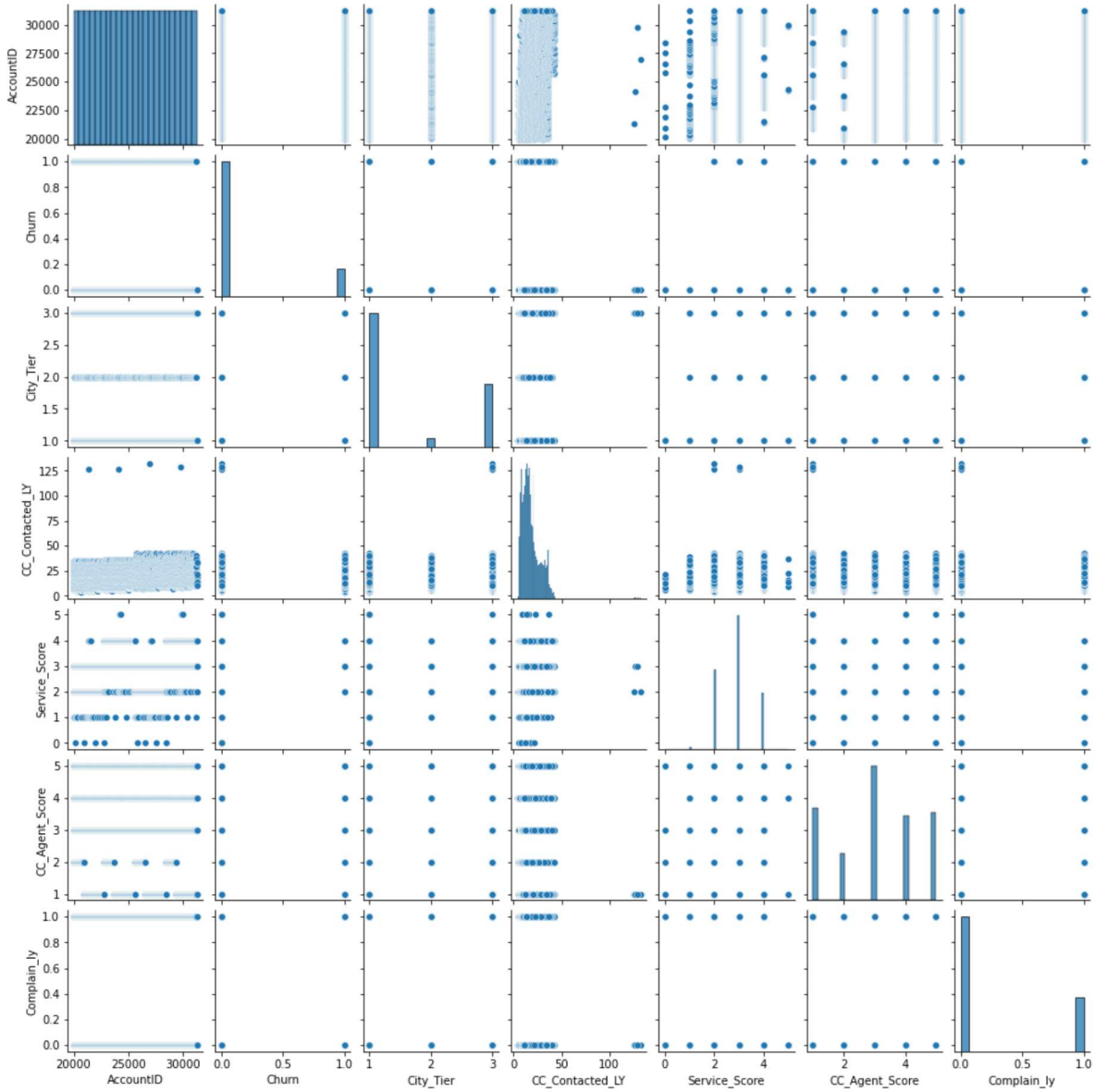
In [32]:

```
sns.pairplot(data=edth,size=2)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\axisgrid.py:1912: UserWar
ning: The `size` parameter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
```

Out[32]:

```
<seaborn.axisgrid.PairGrid at 0x22ed0bdc880>
```



3.a) Removal of unwanted variables (if applicable)

In [33]:

```
edth = edth.drop(["AccountID"], axis=1)
edth.head(5).T
```

Out[33]:

	0	1	2	3	4
Churn	1	1	1	1	1
Tenure	4	0	0	0	0
City_Tier	3	1	1	3	1
CC_Contacted_LY	6	8	30	15	12
Payment	Debit Card	UPI	Debit Card	Debit Card	Credit Card
Gender	Female	Male	Male	Male	Male
Service_Score	3	3	2	2	2
Account_user_count	3	4	4	4	3
account_segment	Super	Regular Plus	Regular Plus	Super	Regular Plus
CC_Agent_Score	2	3	3	5	5
Marital_Status	Single	Single	Single	Single	Single
rev_per_month	9	7	6	8	3
Complain_ly	1	1	1	0	0
rev_growth_yoy	11	15	14	23	11
coupon_used_for_payment	1	0	0	0	1
Day_Since_CC_connect	5	0	3	3	3
cashback	159.93	120.9	NaN	134.07	129.6
Login_device	Mobile	Mobile	Mobile	Mobile	Mobile

3.b) Missing Value & treatment

In [34]:

```
edth.isnull().sum()
```

Out[34]:

```
Churn                       0
Tenure                    102
City_Tier                 112
CC_Contacted_LY           102
Payment                   109
Gender                    108
Service_Score              98
Account_user_count        112
account_segment            97
CC_Agent_Score            116
Marital_Status            212
rev_per_month             102
Complain_ly               357
rev_growth_yoy              0
coupon_used_for_payment     0
Day_Since_CC_connect      357
cashback                  471
Login_device              221
dtype: int64
```

In [35]:

```
missing_count_edth = edth.isnull().sum()
```

In [36]:

```
value_count_edth= edth.isnull().count()
```

In [37]:

```
missing_percentage_edth = round(missing_count_edth/value_count_edth*100,2) # the percentage
missing_percentage_edth
```

Out[37]:

```
Churn                      0.00
Tenure                     0.91
City_Tier                  0.99
CC_Contacted_LY            0.91
Payment                    0.97
Gender                     0.96
Service_Score              0.87
Account_user_count         0.99
account_segment            0.86
CC_Agent_Score             1.03
Marital_Status             1.88
rev_per_month              0.91
Complain_ly                3.17
rev_growth_yoy             0.00
coupon_used_for_payment    0.00
Day_Since_CC_connect       3.17
cashback                   4.18
Login_device               1.96
dtype: float64
```

In [38]:

```
missing_edth = pd.DataFrame({'count' : missing_count_edth, 'percentage' : missing_percentag
```

In [39]:

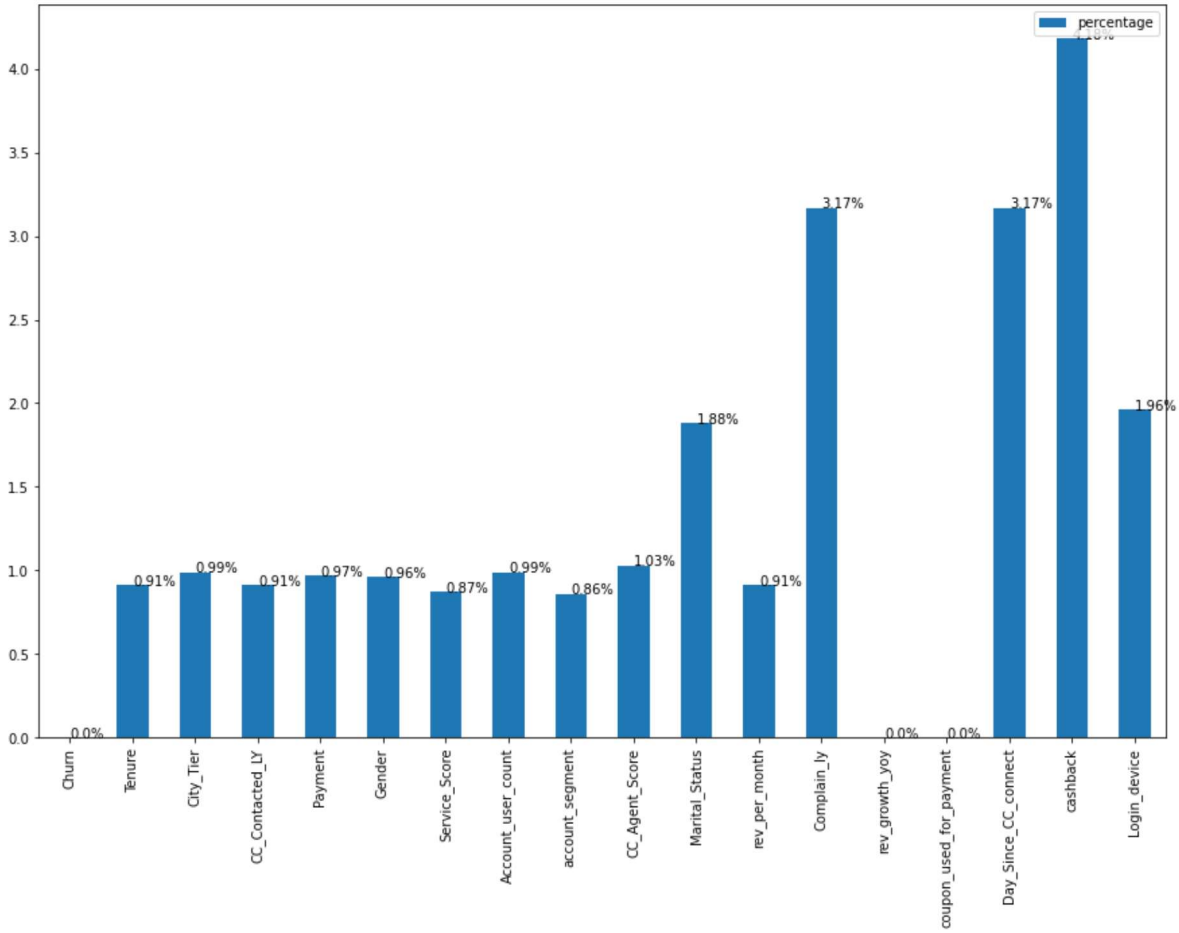
```
missing_edth
```

Out[39]:

	count	percentage
Churn	0	0.00
Tenure	102	0.91
City_Tier	112	0.99
CC_Contacted_LY	102	0.91
Payment	109	0.97
Gender	108	0.96
Service_Score	98	0.87
Account_user_count	112	0.99
account_segment	97	0.86
CC_Agent_Score	116	1.03
Marital_Status	212	1.88
rev_per_month	102	0.91
Complain_ly	357	3.17
rev_growth_yoy	0	0.00
coupon_used_for_payment	0	0.00
Day_Since_CC_connect	357	3.17
cashback	471	4.18
Login_device	221	1.96

In [40]:

```
barchart = missing_edth.plot.bar(y='percentage',rot=90,figsize=(15,10))
for index, percentage in enumerate(missing_percentage_edth):
    barchart.text(index,percentage,str(percentage) + '%')
```



Treatment Missing Values

In [41]:

```
edth["Gender"] = edth["Gender"].replace("M", 'Male').replace("F", 'Female')
```

In [42]:

```
edth['Tenure'] = edth['Tenure'].fillna("No_info")
edth['Payment'] = edth['Payment'].fillna("No_info")
edth['Gender'] = edth['Gender'].fillna("No_info")
edth['Account_user_count'] = edth['Account_user_count'].fillna("No_info")
edth['account_segment'] = edth['account_segment'].fillna("No_info")
edth['Marital_Status'] = edth['Marital_Status'].fillna("No_info")
edth['rev_per_month'] = edth['rev_per_month'].fillna("No_info")
edth['Day_Since_CC_connect'] = edth['Day_Since_CC_connect'].fillna("No_info")
edth['cashback'] =edth['cashback'].fillna("No_info")
edth['Login_device'] = edth['Login_device'].fillna("No_info")
```

In [43]:

```
edth['City_Tier'] = edth['City_Tier'].fillna(edth['City_Tier'].mean())
edth['CC_Contacted_LY'] = edth['CC_Contacted_LY'].fillna(edth['CC_Contacted_LY'].mean())
edth['Service_Score'] = edth['Service_Score'].fillna(edth['Service_Score'].mean())
edth['CC_Agent_Score'] = edth['CC_Agent_Score'].fillna(edth['CC_Agent_Score'].mean())
edth['Complain_ly'] = edth['Complain_ly'].fillna(edth['Complain_ly'].mean())
```

In [44]:

```
edth.isnull().sum()
```

Out[44]:

```
Churn                       0
Tenure                      0
City_Tier                   0
CC_Contacted_LY             0
Payment                     0
Gender                      0
Service_Score               0
Account_user_count          0
account_segment             0
CC_Agent_Score              0
Marital_Status              0
rev_per_month               0
Complain_ly                 0
rev_growth_yoy              0
coupon_used_for_payment     0
Day_Since_CC_connect        0
cashback                    0
Login_device                0
dtype: int64
```

3.2.d) Outlier treatment (if required)

In [45]:

```
def treat_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

In [46]:

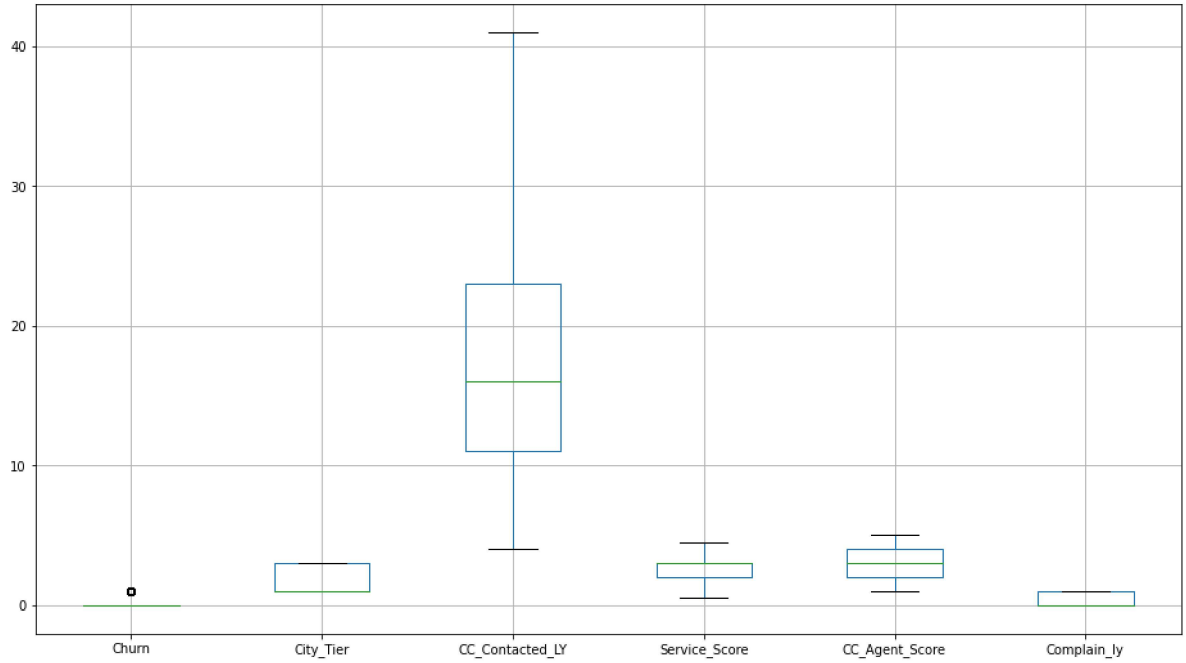
```
for feature in edth[['CC_Contacted_LY','Service_Score']]:
    lr,ur=treat_outlier(edth[feature])
    edth[feature]=np.where(edth[feature]>ur,ur,edth[feature])
    edth[feature]=np.where(edth[feature]<lr,lr,edth[feature])
```

In [47]:

```
edth.boxplot(figsize=(16,9))
```

Out[47]:

```
<AxesSubplot:>
```



In [48]:

```
edth.value_counts("Churn")
```

Out[48]:

```
Churn
0    9364
1    1896
dtype: int64
```

e) Variable transformation

In [49]:

```
for feature in edth.columns:
    if edth[feature].dtype == 'object':
        print('\n')
        print('feature:',feature)
        print(pd.Categorical(edth[feature].unique()))
        print(pd.Categorical(edth[feature].unique()).codes)
        edth[feature] = pd.Categorical(edth[feature]).codes
```

```
feature: Tenure
[4, 0, 2, 13, 11, ..., 50, 60, 31, 51, 61]
Length: 39
Categories (39, object): [0, 1, 2, 3, ..., 61, 99, '#', 'No_info']
[ 4  0  2 13 11 37  9 36 19 20 14  8 26 18  5 30  7  1 23  3 29  6 28 24
 25 16 10 15 22 38 27 12 21 17 32 34 31 33 35]


feature: Payment
['Debit Card', 'UPI', 'Credit Card', 'Cash on Delivery', 'E wallet', 'No_
info']
Categories (6, object): ['Cash on Delivery', 'Credit Card', 'Debit Card',
'E wallet', 'No_info', 'UPI']
[2 5 1 0 3 4]


feature: Gender
['Female', 'Male', 'No_info']
Categories (3, object): ['Female', 'Male', 'No_info']
[0 1 2]


feature: Account_user_count
[3, 4, 'No_info', 5, 2, '@', 1, 6]
Categories (8, object): [1, 2, 3, 4, 5, 6, '@', 'No_info']
[2 3 7 4 1 6 0 5]


feature: account_segment
['Super', 'Regular Plus', 'Regular', 'HNI', 'Regular +', 'No_info', 'Supe
r Plus', 'Super +']
Categories (8, object): ['HNI', 'No_info', 'Regular', 'Regular +', 'Regul
ar Plus', 'Super', 'Super +', 'Super Plus']
[5 4 2 0 3 1 7 6]


feature: Marital_Status
['Single', 'Divorced', 'Married', 'No_info']
Categories (4, object): ['Divorced', 'Married', 'No_info', 'Single']
[3 0 1 2]


feature: rev_per_month
[9, 7, 6, 8, 3, ..., 135, 111, 109, 100, 103]
Length: 60
Categories (60, object): [1, 2, 3, 4, ..., 139, 140, '+', 'No_info']
[ 8  6  5  7  2  1  3  9  0  4 58 48 59 15 56 23 39 55 46 42 43 35 17 45
 51 32 33 28 57 50 47 27 37 10 26 16 38 40 54 30 18 22 53 44 13 12 11 34
```

```
 19 41 36 49 25 14 20 52 31 29 21 24]


feature: rev_growth_yoy
[11, 15, 14, 23, 22, ..., 26, '$', 4, 27, 28]
Length: 20
Categories (20, object): [4, 11, 12, 13, ..., 26, 27, 28, '$']
[ 1  5  4 13 12  6  2  3  7  8 14  9 10 11 15 16 19  0 17 18]


feature: coupon_used_for_payment
[1, 0, 4, 2, 9, ..., '#', '$', 14, '*', 16]
Length: 20
Categories (20, object): [0, 1, 2, 3, ..., 16, '#', '$', '*']
[ 1  0  4  2  9  6 11  7 12 10  5  3 13 15  8 17 18 14 19 16]


feature: Day_Since_CC_connect
[5, 0, 3, 7, 2, ..., '$', 46, 18, 31, 47]
Length: 25
Categories (25, object): [0, 1, 2, 3, ..., 46, 47, '$', 'No_info']
[ 5  0  3  7  2  1  8  6  4 15 24 11 10  9 13 12 17 16 14 19 23 21 18 20
 22]


feature: cashback
[159.93, 120.9, 'No_info', 134.07, 129.6, ..., 240.54, 252.31, 227.36, 22
6.91, 191.42]
Length: 5694
Categories (5694, object): [0, 2, 3, 4, ..., 1992, 1997, '$', 'No_info']
[2222  109 5693 ... 4488 4481 3680]


feature: Login_device
['Mobile', 'Computer', '&&&&', 'No_info']
Categories (4, object): ['&&&&', 'Computer', 'Mobile', 'No_info']
[2 1 0 3]
```

In [50]:

```
edth.head().T
```

Out[50]:

	0	1	2	3	4
Churn	1.0	1.0	1.0	1.0	1.0
Tenure	4.0	0.0	0.0	0.0	0.0
City_Tier	3.0	1.0	1.0	3.0	1.0
CC_Contacted_LY	6.0	8.0	30.0	15.0	12.0
Payment	2.0	5.0	2.0	2.0	1.0
Gender	0.0	1.0	1.0	1.0	1.0
Service_Score	3.0	3.0	2.0	2.0	2.0
Account_user_count	2.0	3.0	3.0	3.0	2.0
account_segment	5.0	4.0	4.0	5.0	4.0
CC_Agent_Score	2.0	3.0	3.0	5.0	5.0
Marital_Status	3.0	3.0	3.0	3.0	3.0
rev_per_month	8.0	6.0	5.0	7.0	2.0
Complain_ly	1.0	1.0	1.0	0.0	0.0
rev_growth_yoy	1.0	5.0	4.0	13.0	1.0
coupon_used_for_payment	1.0	0.0	0.0	0.0	1.0
Day_Since_CC_connect	5.0	0.0	3.0	3.0	3.0
cashback	2222.0	109.0	5693.0	748.0	552.0
Login_device	2.0	2.0	2.0	2.0	2.0

4.a) Is the data unbalanced? If so, what can be done? Please explain in the context of the business

In [51]:

```
edth.describe().T
```

Out[51]:

	count	mean	std	min	25%	50%	75%
Churn	11260.0	0.168384	0.374223	0.0	0.0	0.0	0.00
Tenure	11260.0	10.788455	9.607642	0.0	2.0	9.0	16.00
City_Tier	11260.0	1.653929	0.910453	1.0	1.0	1.0	3.00
CC_Contacted_LY	11260.0	17.831922	8.562382	4.0	11.0	16.0	23.00
Payment	11260.0	1.854529	1.206156	0.0	1.0	2.0	2.00
Gender	11260.0	0.614565	0.506044	0.0	0.0	1.0	1.00
Service_Score	11260.0	2.902660	0.720544	0.5	2.0	3.0	3.00
Account_user_count	11260.0	2.833215	1.221705	0.0	2.0	3.0	3.00
account_segment	11260.0	3.850799	1.893734	0.0	4.0	4.0	5.00
CC_Agent_Score	11260.0	3.066493	1.372646	1.0	2.0	3.0	4.00
Marital_Status	11260.0	1.495915	1.082395	0.0	1.0	1.0	3.00
rev_per_month	11260.0	8.320249	14.417811	0.0	2.0	4.0	7.00
Complain_ly	11260.0	0.285334	0.444377	0.0	0.0	0.0	1.00
rev_growth_yoy	11260.0	6.198401	3.759120	0.0	3.0	5.0	9.00
coupon_used_for_payment	11260.0	1.794938	1.987024	0.0	1.0	1.0	2.00
Day_Since_CC_connect	11260.0	5.238277	4.923590	0.0	2.0	4.0	8.00
cashback	11260.0	2761.039964	1675.014686	0.0	1376.0	2611.0	4142.25
Login_device	11260.0	1.655861	0.600608	0.0	1.0	2.0	2.00

In [52]:

```
edth.duplicated().sum()
```

Out[52]:

264

In [53]:

```
edth.drop_duplicates(inplace=True)
```

In [54]:

```
from sklearn.preprocessing import StandardScaler
```

In [55]:

```
x=StandardScaler()
```

In [56]:

```
scaled_df=x.fit_transform(edth)
```

In [57]:

```
pd.DataFrame(scaled_df).T
```

Out[57]:

	0	1	2	3	4	5	6	7	
0	2.222740	2.222740	2.222740	2.222740	2.222740	2.222740	2.222740	2.222740	2.22
1	-0.707835	-1.122643	-1.122643	-1.122643	-1.122643	-1.122643	-0.915239	-1.122643	0.22
2	1.476649	-0.718349	-0.718349	1.476649	-0.718349	-0.718349	1.476649	-0.718349	1.47
3	-1.382990	-1.149710	1.416373	-0.333229	-0.683149	0.483252	-0.799790	-1.382990	-1.03
4	0.120636	2.615646	0.120636	0.120636	-0.711034	0.120636	-1.542704	-0.711034	0.95
5	-1.214155	0.760531	0.760531	0.760531	0.760531	-1.214155	0.760531	0.760531	0.76
6	0.137376	0.137376	-1.249690	-1.249690	-1.249690	0.137376	-1.249690	0.137376	-1.24
7	-0.682026	0.133498	0.133498	0.133498	-0.682026	3.395596	-0.682026	-0.682026	0.13
8	0.608221	0.081856	0.081856	0.608221	0.081856	0.081856	0.608221	0.081856	0.08
9	-0.770410	-0.042043	-0.042043	1.414692	1.414692	1.414692	-0.770410	-0.770410	-0.04
10	1.396318	1.396318	1.396318	1.396318	1.396318	1.396318	-1.373313	-1.373313	-1.37
11	-0.029949	-0.167280	-0.235946	-0.098615	-0.441942	-0.510608	-0.373277	-0.441942	-0.51
12	1.607372	1.607372	1.607372	-0.642992	-0.642992	1.607372	-0.642992	1.607372	1.60
13	-1.385675	-0.322297	-0.588141	1.804459	-1.385675	1.538615	-0.588141	-0.056452	-0.58
14	-0.403968	-0.905212	-0.905212	-0.905212	-0.403968	1.099766	-0.905212	0.097277	-0.90
15	-0.054141	-1.064551	-0.458305	-0.458305	-0.458305	0.350023	-1.064551	-1.064551	-0.66
16	-0.329597	-1.589744	1.740431	-1.208658	-1.325548	-1.094750	-1.591533	-1.541437	-1.41
17	0.570818	0.570818	0.570818	0.570818	0.570818	-1.086620	0.570818	0.570818	0.57

18 rows × 10996 columns

In [58]:

```
pd.DataFrame(scaled_df).describe().T
```

Out[58]:

	count	mean	std	min	25%	50%	75%	max
0	10996.0	-3.891233e-17	1.000045	-0.449895	-0.449895	-0.449895	-0.449895	2.222740
1	10996.0	2.265023e-16	1.000045	-1.122643	-0.915239	-0.189325	0.640291	2.818034
2	10996.0	2.653389e-17	1.000045	-0.718349	-0.718349	-0.718349	1.476649	1.476649
3	10996.0	-1.796076e-15	1.000045	-1.616271	-0.799790	-0.216589	0.599892	2.699415
4	10996.0	-1.262682e-16	1.000045	-1.542704	-0.711034	0.120636	0.120636	2.615646
5	10996.0	-1.750752e-17	1.000045	-1.214155	-1.214155	0.760531	0.760531	2.735217
6	10996.0	-1.356882e-14	1.000045	-3.330288	-1.249690	0.137376	0.137376	2.217975
7	10996.0	-8.203716e-15	1.000045	-2.313075	-0.682026	0.133498	0.133498	3.395596
8	10996.0	3.170335e-18	1.000045	-2.023606	0.081856	0.081856	0.608221	1.660952
9	10996.0	1.619092e-16	1.000045	-1.498778	-0.770410	-0.042043	0.686324	1.414692
10	10996.0	-2.418350e-15	1.000045	-1.373313	-0.450103	-0.450103	1.396318	1.396318
11	10996.0	-2.025708e-15	1.000045	-0.579273	-0.441942	-0.304611	-0.098615	3.471993
12	10996.0	-4.228459e-17	1.000045	-0.642992	-0.642992	-0.642992	1.607372	1.607372
13	10996.0	4.536870e-15	1.000045	-1.651519	-0.853986	-0.322297	0.741081	3.399527
14	10996.0	-2.659219e-16	1.000045	-0.905212	-0.403968	-0.403968	0.097277	8.618434
15	10996.0	8.750175e-16	1.000045	-1.064551	-0.660387	-0.256223	0.552105	3.785417
16	10996.0	-2.192347e-15	1.000045	-1.654749	-0.825933	-0.085977	0.823350	1.740431
17	10996.0	-7.501780e-17	1.000045	-2.744058	-1.086620	0.570818	0.570818	2.228256

b) Any business insights using clustering

In [59]:

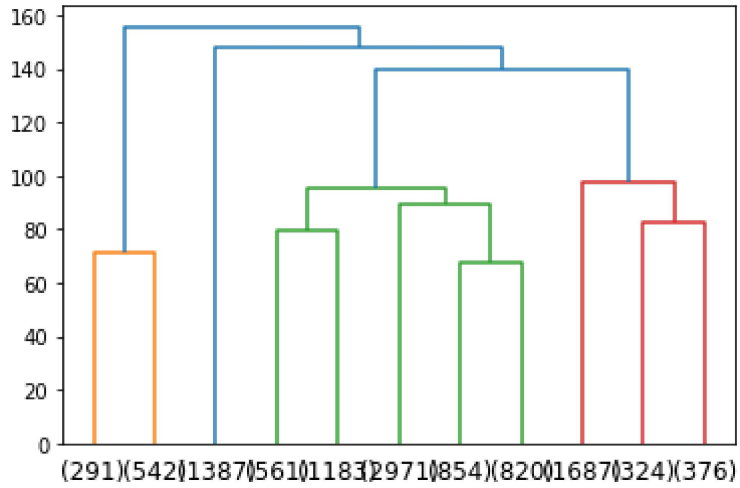
```
from scipy.cluster.hierarchy import dendrogram, linkage
```

In [60]:

```
wardlink = linkage(scaled_df, method = 'ward')
```

In [61]:

```
dendo= dendrogram(wardlink,
                  truncate_mode='lastp',
                  p = 11,
                  )
```



In [62]:

```
from scipy.cluster.hierarchy import fcluster
```

In [63]:

```
clusters = fcluster(wardlink, 3, criterion='maxclust')
clusters
```

Out[63]:

```
array([2, 2, 2, ..., 3, 3, 3], dtype=int32)
```

In [64]:

```
edth['clusters'] = clusters
```

In [65]:

```
edth.head().T
```

Out[65]:

	0	1	2	3	4
Churn	1.0	1.0	1.0	1.0	1.0
Tenure	4.0	0.0	0.0	0.0	0.0
City_Tier	3.0	1.0	1.0	3.0	1.0
CC_Contacted_LY	6.0	8.0	30.0	15.0	12.0
Payment	2.0	5.0	2.0	2.0	1.0
Gender	0.0	1.0	1.0	1.0	1.0
Service_Score	3.0	3.0	2.0	2.0	2.0
Account_user_count	2.0	3.0	3.0	3.0	2.0
account_segment	5.0	4.0	4.0	5.0	4.0
CC_Agent_Score	2.0	3.0	3.0	5.0	5.0
Marital_Status	3.0	3.0	3.0	3.0	3.0
rev_per_month	8.0	6.0	5.0	7.0	2.0
Complain_ly	1.0	1.0	1.0	0.0	0.0
rev_growth_yoy	1.0	5.0	4.0	13.0	1.0
coupon_used_for_payment	1.0	0.0	0.0	0.0	1.0
Day_Since_CC_connect	5.0	0.0	3.0	3.0	3.0
cashback	2222.0	109.0	5693.0	748.0	552.0
Login_device	2.0	2.0	2.0	2.0	2.0
clusters	2.0	2.0	2.0	2.0	2.0

In [66]:

```
edth.clusters.value_counts().sort_index()
```

Out[66]:

```
1     833
2    1387
3    8776
Name: clusters, dtype: int64
```

In [222]:

```
edth.head()
```

Out[222]:

	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count
0	4	3.0	6.0	2	0	3.0	2
1	0	1.0	8.0	5	1	3.0	3
2	0	1.0	30.0	2	1	2.0	3
3	0	3.0	15.0	2	1	2.0	3
4	0	1.0	12.0	1	1	2.0	2

Scoring guide (Rubric) - Note 2 Rubric (1)

Q 1. Model building and interpretation.

a.Ensemble modelling, wherever applicable b. Any other model tuning measures(if applicable) c. Interpretation of the most optimum model and its implication on the business

In [67]:

```
X = edth.drop("Churn", axis=1)
y = edth.pop("Churn")
```

In [68]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, train_labels, test_labels = train_test_split(X, y, test_size=.3, random_st
```

In [69]:

```
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('train_labels',train_labels.shape)
print('test_labels',test_labels.shape)
```

```
X_train (7697, 18)
X_test (3299, 18)
train_labels (7697,)
test_labels (3299,)
```

Feature Selection

In [70]:

```
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [71]:

```
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix
```

It can shows the most important and significant features between all of the features which can mark importance in the performance of the model and get most required business insights. For this we have gone through the following procedure of feature selection :

(1) VIF & Multicollinearity (dropping features having VIF>5) (2) Checking P values (dropping features having p values > 0.05) (3) Chi-square Test (Find most influenced features in ascending order) (4) Boruta Algorithm

In [72]:

```
vif_df = pd.DataFrame()
vif_df["feature"] = X_train.columns
```

In [73]:

```
vif_df["VIF"] = [variance_inflation_factor(X_train.values,i)for i in range(len(X_train.colu
print(vif_df)
```

```
                    feature        VIF
0                    Tenure   2.805496
1                 City_Tier   4.470069
2           CC_Contacted_LY   5.308218
3                   Payment   3.487847
4                    Gender   2.445188
5             Service_Score  18.360716
6        Account_user_count   7.306133
7           account_segment   5.210792
8            CC_Agent_Score   5.704792
9            Marital_Status   2.817915
10            rev_per_month   2.412830
11              Complain_ly   1.420836
12           rev_growth_yoy   3.779502
13  coupon_used_for_payment   2.109952
14     Day_Since_CC_connect   2.478015
15                 cashback   5.111848
16             Login_device   9.090818
17                 clusters  30.904804
```

Dropping Variables having VIF>5 and Recheck the VIF Values

In [74]:

```
X_train = X_train.drop(["rev_per_month","Login_device","Service_Score","Account_user_count"
X_test = X_test.drop(["rev_per_month","Login_device","Service_Score","Account_user_count"],
print('X_train',X_train.shape)
print('X_test',X_test.shape)
```

```
X_train (7697, 14)
X_test (3299, 14)
```

In [75]:

```
vif_df = pd.DataFrame()
vif_df["feature"] = X_train.columns
#
# Calculate VIF for each feature
vif_df["VIF"] = [variance_inflation_factor(X_train.values,i)
                        for i in range(len(X_train.columns))]
print(vif_df)
```

```
                    feature        VIF
0                    Tenure   2.643033
1                 City_Tier   4.373263
2           CC_Contacted_LY   5.018024
3                   Payment   3.419291
4                    Gender   2.400075
5           account_segment   4.998613
6            CC_Agent_Score   5.202223
7            Marital_Status   2.724553
8               Complain_ly   1.398316
9            rev_growth_yoy   3.602774
10  coupon_used_for_payment   2.053781
11     Day_Since_CC_connect   2.467792
12                 cashback   4.852416
13                 clusters  13.886869
```

Using p-values Check

In [76]:

```
col = X_train.columns
X_train_sm = sm.add_constant(X_train[col])
logm2 = sm.Logit(train_labels.astype(float),X_train_sm.astype(float), family = sm.families.
res = logm2.fit()
res.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.281127
         Iterations 8
```

Out[76]:

Logit Regression Results

Dep. Variable:	Churn	**No. Observations:**	7697
Model:	Logit	**Df Residuals:**	7682
Method:	MLE	**Df Model:**	14
Date:	Fri, 25 Feb 2022	**Pseudo R-squ.:**	0.3801
Time:	20:37:20	**Log-Likelihood:**	-2163.8
converged:	True	**LL-Null:**	-3490.7
Covariance Type:	nonrobust	**LLR p-value:**	0.000

Dropping the features having 'p-value' > 0.05

@ 1 Model Building

Q 2. Model Tuning

Q a. Build various models (You can choose to build models for either or all of descriptive, predictive or prescriptive purposes)

We have prepare the following model procedures to analyze and review the dataset and get the performance and importance of the features available on the dataset which can gathers more informations about the subjects. Following is the list of model building procedure can used in this project :

(1) Logistics Regression (2) Decision Tree (3) Random Forest (4) Linear Discriminant Analysis (5) K Nearest Neighbours (6) Naive Bayes (7) Gradient Boosting (8) Extreme Gradient Boosting (9) Extra Tree Classifier

For every model building precure, we have gone through the following steps get outputs generated by the alogorithm :

(a) Model Prediction (b) Model Performance (c) ROC-AUC Graph (d) Model Performance Matrics

Q b. Test your predictive model against the test set using various appropriate performance metrics c.Interpretation of the model(s)

In [77]:

```
from sklearn import tree
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import ExtraTreesClassifier
```

In [78]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

In [79]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

1 (a). Logistic Regression

In [80]:

```
model = LogisticRegression(C=1.0,
                           class_weight=None,
                           dual=False,
                           fit_intercept=True,
                           intercept_scaling=1,
                           l1_ratio=None,
                           max_iter=100,
                           n_jobs=None,
                           penalty='l2',
                           random_state=1,
                           solver='liblinear',
                           tol=0.0001,
                           verbose=0,
                           warm_start=False)
model.fit(X_train, train_labels)
```

Out[80]:

```
LogisticRegression(random_state=1, solver='liblinear')
```

1(a) Model Prediction

In [81]:

```
y_predict_train = model.predict(X_train)
log_train_acc = model.score(X_train, train_labels)
log_train_acc
```

Out[81]:

```
0.8915161751331687
```

In [82]:

```
y_predict_test = model.predict(X_test)
log_test_acc = model.score(X_test, test_labels)
log_test_acc
```

Out[82]:

```
0.8878448014549863
```

In [83]:

```
model.intercept_
```

Out[83]:

```
array([0.24634796])
```

In [84]:

```
model.coef_
```

Out[84]:

```
array([[-1.20197481e-01,  3.28921622e-01,  2.37167309e-02,
        -1.46899033e-02,  3.55256901e-01, -1.45570687e-01,
         2.63646140e-01,  3.85799595e-01,  1.63251698e+00,
        -3.25987733e-03,  1.03548958e-01, -1.47610715e-02,
        -6.86853924e-05, -1.43403121e+00]])
```

Train data confusion_matrix & classification_report

In [85]:

```
confusion_matrix(train_labels, y_predict_train)
```

Out[85]:

```
array([[6165,  235],
       [ 600,  697]], dtype=int64)
```

In [86]:

```
print(classification_report(train_labels, y_predict_train))
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.94      6400
           1       0.75      0.54      0.63      1297

    accuracy                           0.89      7697
   macro avg       0.83      0.75      0.78      7697
weighted avg       0.88      0.89      0.88      7697
```

Test data confusion_matrix & classification_report

In [87]:

```
confusion_matrix(test_labels, y_predict_test)
```

Out[87]:

```
array([[2645,  100],
       [ 270,  284]], dtype=int64)
```

In [88]:

```
print(classification_report(test_labels, y_predict_test))
```

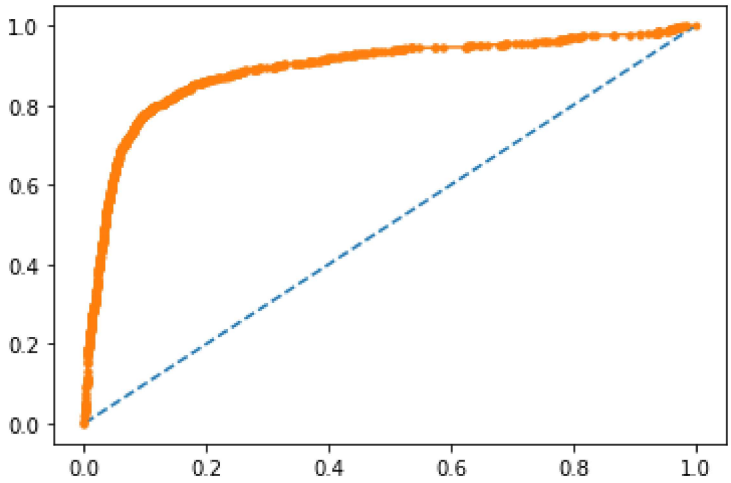
```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93      2745
           1       0.74      0.51      0.61       554

    accuracy                           0.89      3299
   macro avg       0.82      0.74      0.77      3299
weighted avg       0.88      0.89      0.88      3299
```

1(c) Train ROC-AUC Graph

In [89]:

```
# AUC and ROC for the training data

# predict probabilities
probs = model.predict_proba(X_train)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
from sklearn.metrics import roc_auc_score
log_train_auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % log_train_auc)

# calculate roc curve
from sklearn.metrics import roc_curve
log_train_fpr, log_train_tpr, train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model
plt.plot(log_train_fpr, log_train_tpr, marker='.')

# show the plot
plt.show()
```

```
AUC: 0.891
```

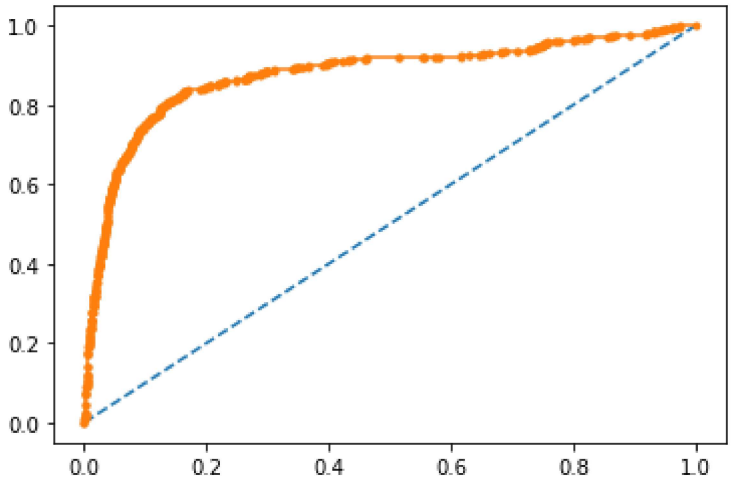


1(c) Test ROC-AUC Graph

In [90]:

```
# AUC and ROC for the test data

# predict probabilities
probs = model.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
from sklearn.metrics import roc_auc_score
log_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % log_test_auc)

# calculate roc curve
from sklearn.metrics import roc_curve
log_test_fpr, log_test_tpr, test_thresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(log_test_fpr, log_test_tpr, marker='.')

# show the plot
plt.show()
```

```
AUC: 0.878
```



1. (d) Model Performance Metrices

In [91]:

```
log_metrics=classification_report(train_labels, y_predict_train,output_dict=True)
df=pd.DataFrame(log_metrics).transpose()
log_train_precision=round(df.loc["1"][0],2)
log_train_recall=round(df.loc["1"][1],2)
log_train_f1=round(df.loc["1"][2],2)
print ('log_train_precision ',log_train_precision)
print ('log_train_recall ',log_train_recall)
print ('log_train_f1 ',log_train_f1)
```

```
log_train_precision  0.75
log_train_recall  0.54
log_train_f1  0.63
```

In [92]:

```
log_metrics=classification_report(test_labels, y_predict_test,output_dict=True)
df=pd.DataFrame(log_metrics).transpose()
log_test_precision=round(df.loc["1"][0],2)
log_test_recall=round(df.loc["1"][1],2)
log_test_f1=round(df.loc["1"][2],2)
print ('log_test_precision ',log_test_precision)
print ('log_test_recall ',log_test_recall)
print ('log_test_f1 ',log_test_f1)
```

```
log_test_precision  0.74
log_test_recall  0.51
log_test_f1  0.61
```

2. Decision Tree Classifier modal perform

In [93]:

```
dtcl = DecisionTreeClassifier(random_state=1)
dtcl.fit(X_train, train_labels)
```

Out[93]:

```
DecisionTreeClassifier(random_state=1)
```

In [94]:

```
param_grid_dt = {
    'criterion': ['gini'],
    'max_depth': [10],
    'min_samples_leaf': [10],
    'min_samples_split': [50],
}
gs_dtcl = GridSearchCV(dtcl, param_grid_dt, cv = 5, n_jobs = -1, verbose = 1)
gs_dtcl.fit(X_train, train_labels)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

Out[94]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=1), n_jobs=
-1,
             param_grid={'criterion': ['gini'], 'max_depth': [10],
                         'min_samples_leaf': [10], 'min_samples_split': [5
0]},
             verbose=1)
```

2. (a) Model Prediction

In [95]:

```
print(gs_dtcl.best_params_)
best_grid = gs_dtcl.best_estimator_
best_grid
```

```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 10, 'min_samples_
split': 50}
```

Out[95]:

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=10, min_samples_split=
50,
                       random_state=1)
```

In [96]:

```
gs_dtcl.best_score_
```

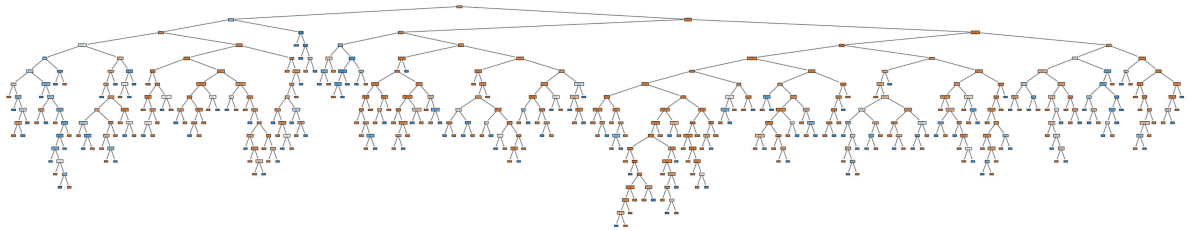
Out[96]:

```
0.9627123363965469
```

Train Decision Tree

In [97]:

```
dot_data1 = tree.export_graphviz(dtcl, out_file=None , feature_names = list(X_train))
fig = plt.figure(figsize=(50,10))
_ = tree.plot_tree(dtcl,
                   feature_names=list(X_train),
                   filled=True)
```



In [98]:

```
dt_imp = pd.DataFrame(dtcl.feature_importances_, columns = ["Imp"], index = X_train.columns
print(dt_imp)
```

```
                              Imp
Tenure                   0.079923
City_Tier                0.010117
CC_Contacted_LY          0.018573
Payment                  0.015599
Gender                   0.005560
account_segment          0.015363
CC_Agent_Score           0.022950
Marital_Status           0.009642
Complain_ly              0.019746
rev_growth_yoy           0.015152
coupon_used_for_payment  0.010426
Day_Since_CC_connect     0.024262
cashback                 0.034583
clusters                 0.718104
```

In [99]:

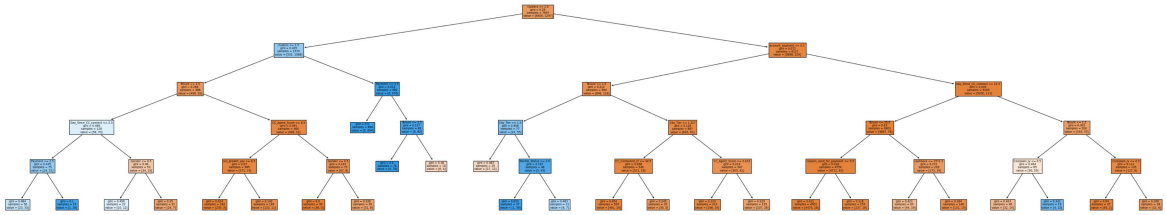
```
reg_dtcl = DecisionTreeClassifier(criterion = 'gini',max_depth = 5,min_samples_leaf=10,min_
reg_dtcl.fit(X_train, train_labels)
```

Out[99]:

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=10, min_samples_split=3
0)
```

In [100]:

```
dot_data2 = tree.export_graphviz(reg_dtcl, out_file=None , feature_names = list(X_train))
fig = plt.figure(figsize=(50,10))
_ = tree.plot_tree(reg_dtcl,
                   feature_names=list(X_train),
                   filled=True)
```



In [101]:

```
ytrain_predict = reg_dtcl.predict(X_train)
ytest_predict = reg_dtcl.predict(X_test)
ytrain_predict_prob=best_grid.predict_proba(X_train)
pd.DataFrame(ytrain_predict_prob).head()
```

Out[101]:

	0	1
0	0.928571	0.071429
1	1.000000	0.000000
2	1.000000	0.000000
3	0.997238	0.002762
4	1.000000	0.000000

Test Decision Tree

In [102]:

```
ytest_predict_prob=best_grid.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

Out[102]:

	0	1
0	0.258065	0.741935
1	0.914286	0.085714
2	1.000000	0.000000
3	0.000000	1.000000
4	1.000000	0.000000

2. (b) Model Performance Train

In [103]:

```
confusion_matrix(train_labels, ytrain_predict)
```

Out[103]:

```
array([[6357,   43],
       [ 204, 1093]], dtype=int64)
```

In [104]:

```
cart_train_acc=best_grid.score(X_train,train_labels)
cart_train_acc
```

Out[104]:

```
0.969208782642588
```

In [105]:

```
print(classification_report(train_labels, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      6400
           1       0.96      0.84      0.90      1297

    accuracy                           0.97      7697
   macro avg       0.97      0.92      0.94      7697
weighted avg       0.97      0.97      0.97      7697
```

Performance Test

In [106]:

```
confusion_matrix(test_labels, ytest_predict)
```

Out[106]:

```
array([[2719,   26],
       [ 106,  448]], dtype=int64)
```

In [107]:

```
cart_test_acc=best_grid.score(X_test,test_labels)
cart_test_acc
```

Out[107]:

```
0.9621097302212792
```

In [108]:

```
print(classification_report(test_labels, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      2745
           1       0.95      0.81      0.87       554

    accuracy                           0.96      3299
   macro avg       0.95      0.90      0.92      3299
weighted avg       0.96      0.96      0.96      3299
```

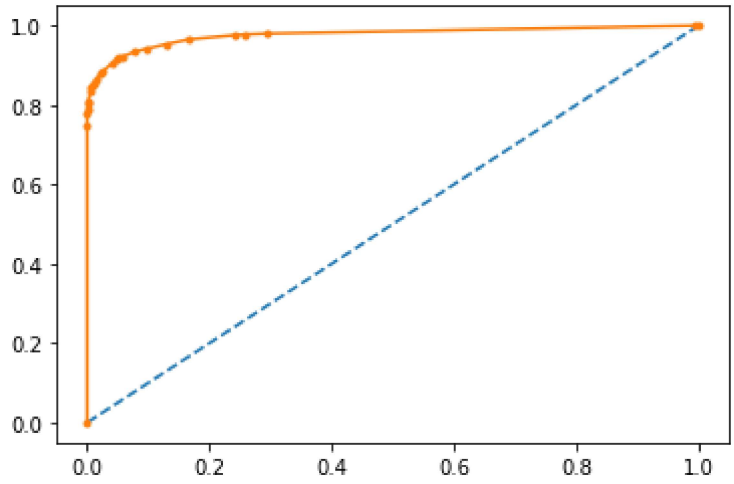
2. (c) ROC-AUC Graph

ROC -AUC Train

In [109]:

```
# AUC and ROC for the training data

# predict probabilities
probs = reg_dtcl.predict_proba(X_train)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
from sklearn.metrics import roc_auc_score
cart_train_auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % cart_train_auc)
# calculate roc curve
from sklearn.metrics import roc_curve
cart_train_fpr, cart_train_tpr, cart_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model
plt.plot(cart_train_fpr, cart_train_tpr, marker='.')

# show the plot
plt.show()
```

```
AUC: 0.977
```



In [ ]:

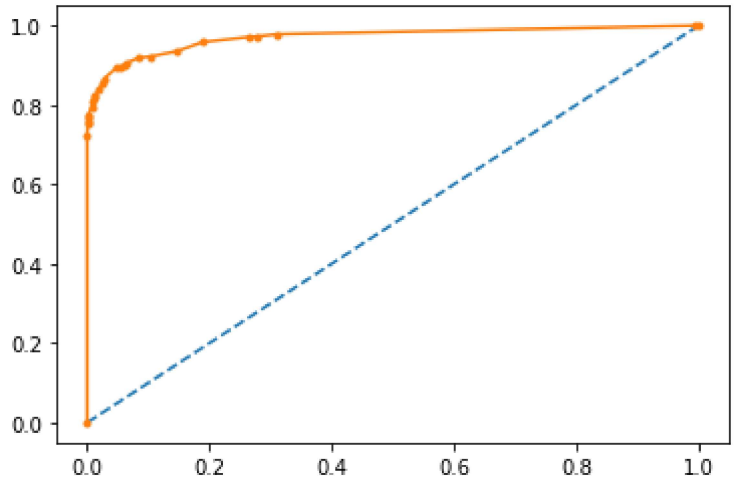
```

```

ROC -AUC Test

In [110]:

```
# AUC and ROC for the test data

# predict probabilities
probs = reg_dtcl.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
from sklearn.metrics import roc_auc_score
cart_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % cart_test_auc)

# calculate roc curve
from sklearn.metrics import roc_curve
cart_test_fpr, cart_test_tpr, cart_test_thresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model
plt.plot(cart_test_fpr, cart_test_tpr, marker='.')

# show the plot
plt.show()
```

```
AUC: 0.971
```



2. (d) Model Performance Metrices

Train Modal

In [111]:

```
cart_metrics=classification_report(train_labels, ytrain_predict,output_dict=True)
df=pd.DataFrame(cart_metrics).transpose()
cart_train_f1=round(df.loc["1"][2],2)
cart_train_recall=round(df.loc["1"][1],2)
cart_train_precision=round(df.loc["1"][0],2)
print ('cart_train_precision ',cart_train_precision)
print ('cart_train_recall ',cart_train_recall)
print ('cart_train_f1 ',cart_train_f1)
```

```
cart_train_precision  0.96
cart_train_recall  0.84
cart_train_f1  0.9
```

Test Modal

In [112]:

```
cart_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(cart_metrics).transpose()
cart_test_precision=round(df.loc["1"][0],2)
cart_test_recall=round(df.loc["1"][1],2)
cart_test_f1=round(df.loc["1"][2],2)
print ('cart_test_precision ',cart_test_precision)
print ('cart_test_recall ',cart_test_recall)
print ('cart_test_f1 ',cart_test_f1)
```

```
cart_test_precision  0.95
cart_test_recall  0.81
cart_test_f1  0.87
```

2. (e) Feature Importance

In [113]:

```
reg_dt_imp = pd.DataFrame(best_grid.feature_importances_, columns = ["Imp"],
                          index = X_train.columns).sort_values('Imp',ascending=False)
print(reg_dt_imp)
```

```
                              Imp
clusters                 0.853618
Tenure                   0.079158
Complain_ly              0.011590
Day_Since_CC_connect     0.010822
account_segment          0.010539
City_Tier                0.007382
CC_Agent_Score           0.005681
CC_Contacted_LY          0.004282
cashback                 0.003953
Gender                   0.003841
rev_growth_yoy           0.003463
Payment                  0.002675
Marital_Status           0.001890
coupon_used_for_payment  0.001106
```

3. Random Forest

In [114]:

```
param_grid = {
    'max_depth': [10],## 20,30,40
    'max_features': [11],## 7,8,9
    'min_samples_leaf': [10],## 50,100
    'min_samples_split': [50], ## 60,70
    'n_estimators': [100] ## 100,200
}
rfcl = RandomForestClassifier(random_state=1)
grid_search_rf = GridSearchCV(estimator = rfcl, param_grid = param_grid, cv = 5)
```

In [115]:

```
grid_search_rf.fit(X_train, train_labels)
```

Out[115]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [10], 'max_features': [11],
                         'min_samples_leaf': [10], 'min_samples_split': [5
0],
                         'n_estimators': [100]})
```

In [116]:

```
grid_search_rf.best_params_
```

Out[116]:

```
{'max_depth': 10,
 'max_features': 11,
 'min_samples_leaf': 10,
 'min_samples_split': 50,
 'n_estimators': 100}
```

In [117]:

```
best_grid_rf = grid_search_rf.best_estimator_
best_grid_rf
```

Out[117]:

```
RandomForestClassifier(max_depth=10, max_features=11, min_samples_leaf=10,
                       min_samples_split=50, random_state=1)
```

In [118]:

```
# To understand the differences of different random states affecting Out-of-Bag score
random_state=[0,30,64]
for i in random_state:
    rfcl=RandomForestClassifier(random_state=i,oob_score=True)
    rfcl.fit(X_train,train_labels)
    print(rfcl.oob_score_)
```

```
0.9777835520332597
0.9775237105365727
0.9783032350266337
```

Train

In [119]:

```
rfcl=RandomForestClassifier(n_estimators=500,random_state=1,oob_score=True,n_jobs=-1)
rfcl=rfcl.fit(X_train,train_labels)
rfcl.oob_score_
```

Out[119]:

```
0.9777835520332597
```

Test

In [120]:

```
rfcl=rfcl.fit(X_test,test_labels)
rfcl.oob_score_
```

Out[120]:

```
0.9736283722340103
```

3. (b) Model Performance

Train

In [121]:

```
confusion_matrix(train_labels,ytrain_predict)
```

Out[121]:

```
array([[6357,   43],
       [ 204, 1093]], dtype=int64)
```

In [122]:

```
rf_train_acc=best_grid.score(X_train,train_labels)
rf_train_acc
```

Out[122]:

```
0.969208782642588
```

In [123]:

```
print(classification_report(train_labels,ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      6400
           1       0.96      0.84      0.90      1297

    accuracy                           0.97      7697
   macro avg       0.97      0.92      0.94      7697
weighted avg       0.97      0.97      0.97      7697
```

Test

In [124]:

```
confusion_matrix(test_labels, ytest_predict)
```

Out[124]:

```
array([[2719,   26],
       [ 106,  448]], dtype=int64)
```

In [125]:

```
rf_test_acc=best_grid.score(X_test,test_labels)
rf_test_acc
```

Out[125]:

```
0.9621097302212792
```

In [126]:

```
print(classification_report(test_labels, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      2745
           1       0.95      0.81      0.87       554

    accuracy                           0.96      3299
   macro avg       0.95      0.90      0.92      3299
weighted avg       0.96      0.96      0.96      3299
```

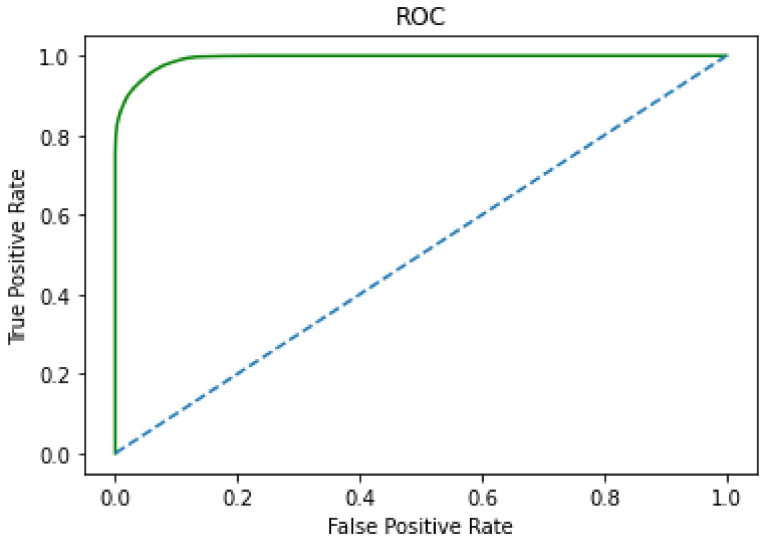
3. (c) ROC-AUC Graph

Train

In [127]:

```
rf_train_fpr, rf_train_tpr,_=roc_curve(train_labels,best_grid.predict_proba(X_train)[:,1])
plt.plot(rf_train_fpr,rf_train_tpr,color='green')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
rf_train_auc=roc_auc_score(train_labels,best_grid.predict_proba(X_train)[:,1])
print('Area under Curve is', rf_train_auc)
```

```
Area under Curve is 0.9931543947571319
```

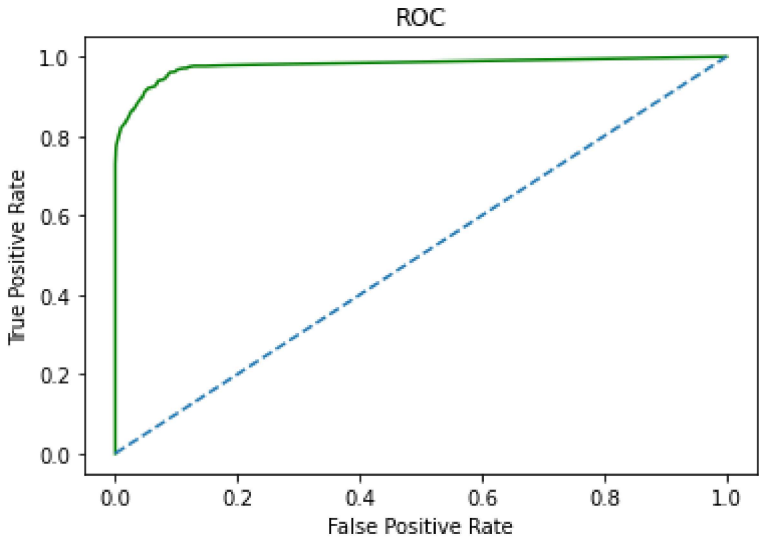


Test

In [128]:

```
rf_test_fpr, rf_test_tpr,_=roc_curve(test_labels,best_grid.predict_proba(X_test)[:,1])
plt.plot(rf_test_fpr,rf_test_tpr,color='green')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
rf_test_auc=roc_auc_score(test_labels,best_grid.predict_proba(X_test)[:,1])
print('Area under Curve is', rf_test_auc)
```

```
Area under Curve is 0.9789430076344914
```



3. (d) Model Performance Metrices

Train

In [129]:

```
rf_metrics=classification_report(train_labels, ytrain_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_train_precision=round(df.loc["1"][0],2)
rf_train_recall=round(df.loc["1"][1],2)
rf_train_f1=round(df.loc["1"][2],2)
print ('rf_train_precision ',rf_train_precision)
print ('rf_train_recall ',rf_train_recall)
print ('rf_train_f1 ',rf_train_f1)
```

```
rf_train_precision  0.96
rf_train_recall  0.84
rf_train_f1  0.9
```

Test

In [130]:

```
rf_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_test_precision=round(df.loc["1"][0],2)
rf_test_recall=round(df.loc["1"][1],2)
rf_test_f1=round(df.loc["1"][2],2)
print ('rf_test_precision ',rf_test_precision)
print ('rf_test_recall ',rf_test_recall)
print ('rf_test_f1 ',rf_test_f1)
```

```
rf_test_precision  0.95
rf_test_recall  0.81
rf_test_f1  0.87
```

3.(e) Feature Importance

In [131]:

```
rf_imp = pd.DataFrame(best_grid_rf.feature_importances_, columns = ["Imp"],
                      index = X_train.columns).sort_values('Imp',ascending=False)
print(rf_imp)
```

```
                            Imp
clusters                0.786472
Tenure                  0.133406
account_segment         0.019795
Day_Since_CC_connect    0.011940
Complain_ly             0.011162
cashback                0.007065
CC_Agent_Score          0.006463
CC_Contacted_LY         0.005383
City_Tier               0.005028
rev_growth_yoy          0.003870
Marital_Status          0.003303
Payment                 0.002731
coupon_used_for_payment 0.002494
Gender                  0.000890
```

4. Linear Discriminant Analysis

In [132]:

```
clf = LinearDiscriminantAnalysis(solver='eigen',shrinkage = 'auto',)
model=clf.fit(X_train,train_labels)
model
```

Out[132]:

```
LinearDiscriminantAnalysis(shrinkage='auto', solver='eigen')
```

4. (a) Model Prediction

In [133]:

```
# Training Data Class Prediction with a cut-off value of 0.5
pred_class_train = model.predict(X_train)
# Test Data Class Prediction with a cut-off value of 0.5
pred_class_test = model.predict(X_test)
pred_class_test
```

Out[133]:

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

In [134]:

```
# Training Data Probability Prediction
pred_prob_train = model.predict_proba(X_train)
# Test Data Probability Prediction
pred_prob_test = model.predict_proba(X_test)
pred_prob_test
```

Out[134]:

```
array([[0.04903014, 0.95096986],
       [0.86740815, 0.13259185],
       [0.98725783, 0.01274217],
       ...,
       [0.05764501, 0.94235499],
       [0.28474431, 0.71525569],
       [0.83162468, 0.16837532]])
```

4. (b) Model Performance

Train

In [135]:

```
confusion_matrix(train_labels, pred_class_train)
```

Out[135]:

```
array([[5966,  434],
       [ 495,  802]], dtype=int64)
```

In [136]:

```
lda_train_acc = model.score(X_train,train_labels)
lda_train_acc
```

Out[136]:

```
0.8793036247888788
```

In [137]:

```
print(classification_report(train_labels, pred_class_train))
```

```
              precision    recall  f1-score   support

           0       0.92      0.93      0.93      6400
           1       0.65      0.62      0.63      1297

    accuracy                           0.88      7697
   macro avg       0.79      0.78      0.78      7697
weighted avg       0.88      0.88      0.88      7697
```

Test

In [138]:

```
confusion_matrix(test_labels, pred_class_test)
```

Out[138]:

```
array([[2567,  178],
       [ 232,  322]], dtype=int64)
```

In [139]:

```
lda_test_acc = model.score(X_test,test_labels)
lda_test_acc
```

Out[139]:

```
0.8757199151257957
```

In [140]:

```
print(classification_report(test_labels, pred_class_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      2745
           1       0.64      0.58      0.61       554

    accuracy                           0.88      3299
   macro avg       0.78      0.76      0.77      3299
weighted avg       0.87      0.88      0.87      3299
```

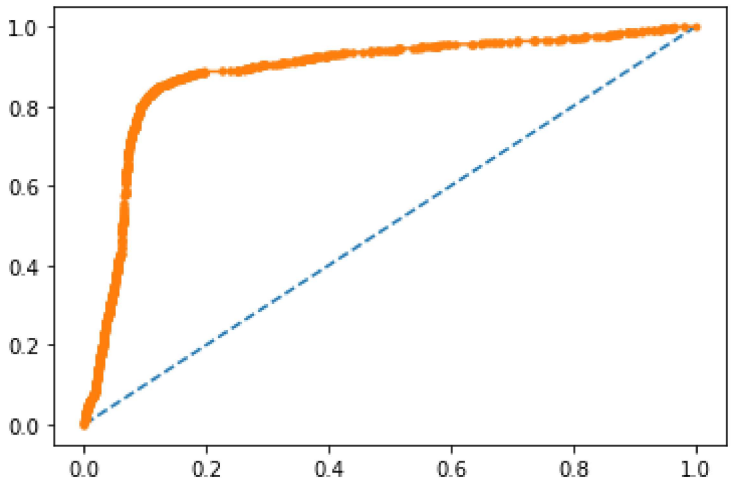
4. (c) ROC-AUC Graph

Train

In [141]:

```
# AUC and ROC for the training data

# predict probabilities
probs = model.predict_proba(X_train)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
from sklearn.metrics import roc_auc_score
lda_train_auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % cart_train_auc)

# calculate roc curve
from sklearn.metrics import roc_curve
lda_train_fpr, lda_train_tpr, lda_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model
plt.plot(lda_train_fpr, lda_train_tpr, marker='.')

# show the plot
plt.show()
```

```
AUC: 0.977
```

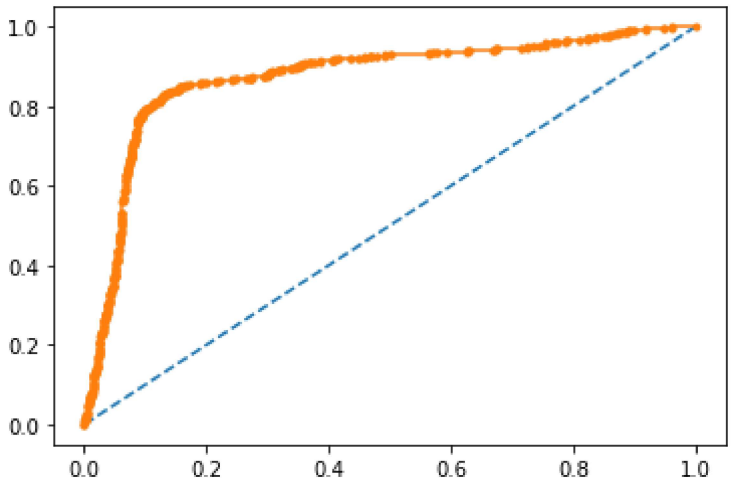


Test

In [142]:

```
# AUC and ROC for the test data

# predict probabilities
probs = model.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
from sklearn.metrics import roc_auc_score
lda_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % cart_test_auc)

# calculate roc curve
from sklearn.metrics import roc_curve
lda_test_fpr, lda_test_tpr, lda_test_thresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(lda_test_fpr, lda_test_tpr, marker='.')

# show the plot
plt.show()
```

```
AUC: 0.971
```



4. (d) Model Performance Metrices

Train

In [143]:

```
lda_metrics=classification_report(train_labels, pred_class_train,output_dict=True)
df=pd.DataFrame(lda_metrics).transpose()
lda_train_f1=round(df.loc["1"][2],2)
lda_train_recall=round(df.loc["1"][1],2)
lda_train_precision=round(df.loc["1"][0],2)
print ('lda_train_precision ',lda_train_precision)
print ('lda_train_recall ',lda_train_recall)
print ('lda_train_f1 ',lda_train_f1)
```

```
lda_train_precision  0.65
lda_train_recall  0.62
lda_train_f1  0.63
```

Test

In [144]:

```
lda_metrics=classification_report(test_labels, pred_class_test,output_dict=True)
df=pd.DataFrame(lda_metrics).transpose()
lda_test_f1=round(df.loc["0"][2],2)
lda_test_recall=round(df.loc["0"][1],2)
lda_test_precision=round(df.loc["0"][0],2)
print ('lda_test_precision ',lda_test_precision)
print ('lda_test_recall ',lda_test_recall)
print ('lda_test_f1 ',lda_test_f1)
```

```
lda_test_precision  0.92
lda_test_recall  0.94
lda_test_f1  0.93
```

----------5. K Nearest Neighbours

In [145]:

```
KNN_model=KNeighborsClassifier(n_neighbors = 15,
                               weights = 'uniform',
                               metric = 'minkowski')
KNN_model.fit(X_train,train_labels)
```

Out[145]:

```
KNeighborsClassifier(n_neighbors=15)
```

5. (a) Model Prediction

In [146]:

```
KNN_train_predict = KNN_model.predict(X_train)
KNN_train_acc = KNN_model.score(X_train, train_labels)
KNN_train_acc
```

Out[146]:

```
0.844744705729505
```

In [147]:

```
KNN_test_predict = KNN_model.predict(X_test)
KNN_test_acc = KNN_model.score(X_test, test_labels)
KNN_test_acc
```

Out[147]:

```
0.829948469233101
```

5. (b) Model Performance

Train

In [148]:

```
confusion_matrix(train_labels, KNN_train_predict)
```

Out[148]:

```
array([[6310,   90],
       [1105,  192]], dtype=int64)
```

In [149]:

```
print(classification_report(train_labels, KNN_train_predict))
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.91      6400
           1       0.68      0.15      0.24      1297

    accuracy                           0.84      7697
   macro avg       0.77      0.57      0.58      7697
weighted avg       0.82      0.84      0.80      7697
```

Test

In [150]:

```
confusion_matrix(test_labels, KNN_test_predict)
```

Out[150]:

```
array([[2703,   42],
       [ 519,   35]], dtype=int64)
```

In [151]:

```
print(classification_report(test_labels, KNN_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.84      0.98      0.91      2745
           1       0.45      0.06      0.11       554

    accuracy                           0.83      3299
   macro avg       0.65      0.52      0.51      3299
weighted avg       0.77      0.83      0.77      3299
```

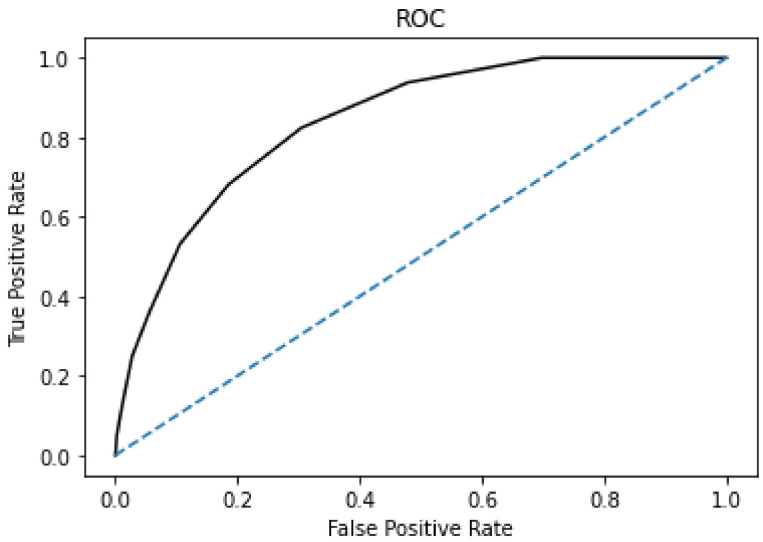
5. (c) ROC-AUC Graph

Train

In [152]:

```
KNN_train_fpr, KNN_train_tpr,_=roc_curve(train_labels,KNN_model.predict_proba(X_train)[:,1]
plt.plot(KNN_train_fpr,KNN_train_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
KNN_train_auc=roc_auc_score(train_labels,KNN_model.predict_proba(X_train)[:,1])
print('Area under Curve is', KNN_train_auc)
```

```
Area under Curve is 0.840641865844256
```

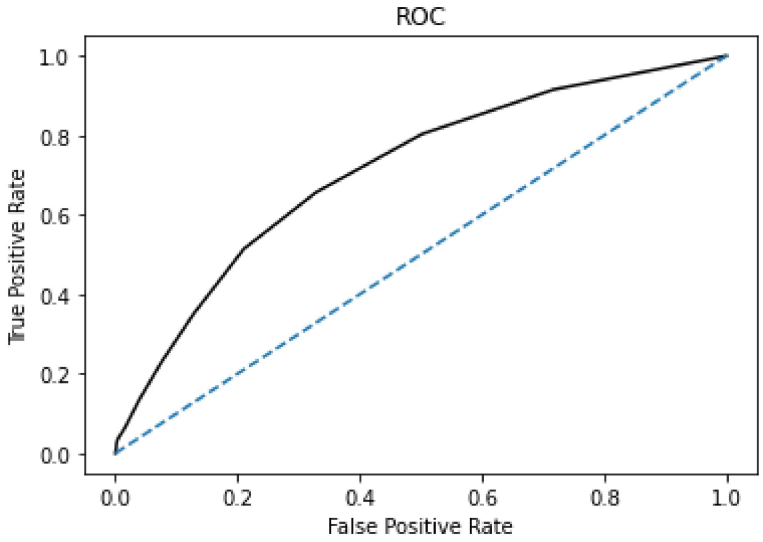


Test

In [153]:

```
KNN_test_fpr, KNN_test_tpr,_=roc_curve(test_labels,KNN_model.predict_proba(X_test)[:,1])
plt.plot(KNN_test_fpr,KNN_test_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
KNN_test_auc=roc_auc_score(test_labels,KNN_model.predict_proba(X_test)[:,1])
print('Area under Curve is', KNN_test_auc)
```

```
Area under Curve is 0.7119327559790364
```



5. (d) Model Performance Metrices

Train

In [154]:

```
KNN_metrics=classification_report(train_labels, KNN_train_predict,output_dict=True)
df=pd.DataFrame(KNN_metrics).transpose()
KNN_train_f1=round(df.loc["1"][2],2)
KNN_train_recall=round(df.loc["1"][1],2)
KNN_train_precision=round(df.loc["1"][0],2)
print ('KNN_train_precision ',KNN_train_precision)
print ('KNN_train_recall ',KNN_train_recall)
print ('KNN_train_f1 ',KNN_train_f1)
```

```
KNN_train_precision  0.68
KNN_train_recall  0.15
KNN_train_f1  0.24
```

Test

In [155]:

```
KNN_metrics=classification_report(test_labels, KNN_test_predict,output_dict=True)
df=pd.DataFrame(KNN_metrics).transpose()
KNN_test_f1=round(df.loc["1"][2],2)
KNN_test_recall=round(df.loc["1"][1],2)
KNN_test_precision=round(df.loc["1"][0],2)
print ('KNN_test_precision ',KNN_test_precision)
print ('KNN_test_recall ',KNN_test_recall)
print ('KNN_test_f1 ',KNN_test_f1)
```

```
KNN_test_precision  0.45
KNN_test_recall  0.06
KNN_test_f1  0.11
```

In [156]:

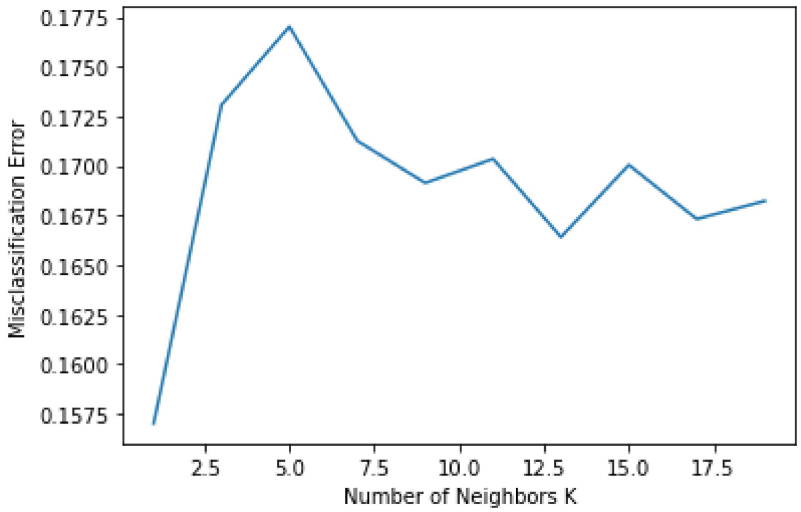
```
# empty list that will hold accuracy scores
ac_scores = []
#
# perform accuracy metrics for values from 1,3,5....19
for k in range(1,20,2):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, train_labels)
    # evaluate test accuracy
    scores = knn.score(X_test, test_labels)
    ac_scores.append(scores)
#
# changing to misclassification error
MCE = [1 - x for x in ac_scores]
MCE
```

Out[156]:

```
[0.15701727796301912,
 0.17308275234919668,
 0.17702334040618373,
 0.17126401939981817,
 0.16914216429220974,
 0.17035465292512886,
 0.1664140648681418,
 0.17005153076689905,
 0.16732343134283112,
 0.16823279781752043]
```

In [157]:

```
# plot misclassification error vs k
plt.plot(range(1,20,2), MCE)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
```



---------- 6. Gaussian Naive Bayes

In [158]:

```
NB_model = GaussianNB()
NB_model.fit(X_train, train_labels)
```

Out[158]:

```
GaussianNB()
```

6. (a) Model Prediction

Train

In [159]:

```
NB_train_predict = NB_model.predict(X_train)
NB_train_acc = NB_model.score(X_train, train_labels)
NB_train_acc
```

Out[159]:

```
0.8981421332986878
```

Test

In [160]:

```
NB_test_predict = NB_model.predict(X_test)
NB_test_acc = NB_model.score(X_test, test_labels)
NB_test_acc
```

Out[160]:

```
0.8996665656259473
```

6. (b) Model Performance

Train

In [161]:

```
confusion_matrix(train_labels, NB_train_predict)
```

Out[161]:

```
array([[6014,  386],
       [ 398,  899]], dtype=int64)
```

In [162]:

```
print(classification_report(train_labels, NB_train_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      6400
           1       0.70      0.69      0.70      1297

    accuracy                           0.90      7697
   macro avg       0.82      0.82      0.82      7697
weighted avg       0.90      0.90      0.90      7697
```

Test

In [163]:

```
confusion_matrix(test_labels, NB_test_predict)
```

Out[163]:

```
array([[2597,  148],
       [ 183,  371]], dtype=int64)
```

In [164]:

```
print(classification_report(test_labels, NB_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      2745
           1       0.71      0.67      0.69       554

    accuracy                           0.90      3299
   macro avg       0.82      0.81      0.82      3299
weighted avg       0.90      0.90      0.90      3299
```

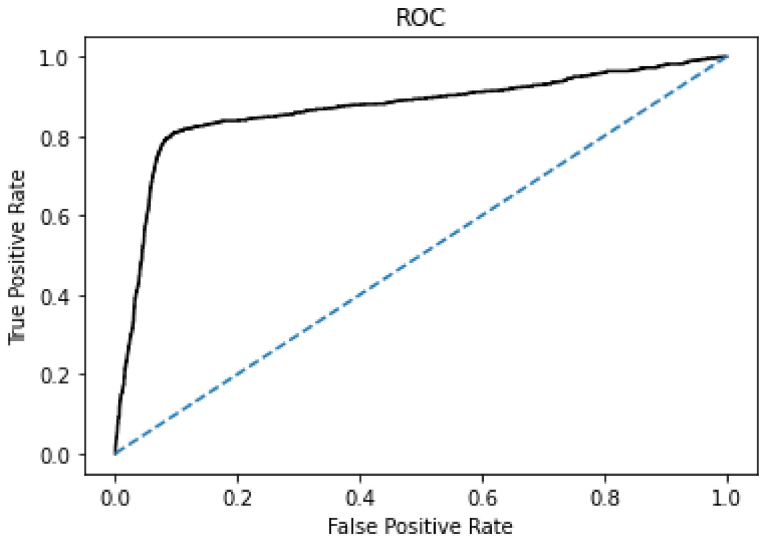
6. (c) ROC-AUC Graph

Train

In [165]:

```
NB_train_fpr, NB_train_tpr,_=roc_curve(train_labels,NB_model.predict_proba(X_train)[:,1])
plt.plot(NB_train_fpr,NB_train_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
NB_train_auc=roc_auc_score(train_labels,NB_model.predict_proba(X_train)[:,1])
print('Area under Curve is', NB_train_auc)
```

```
Area under Curve is 0.8682787201233616
```

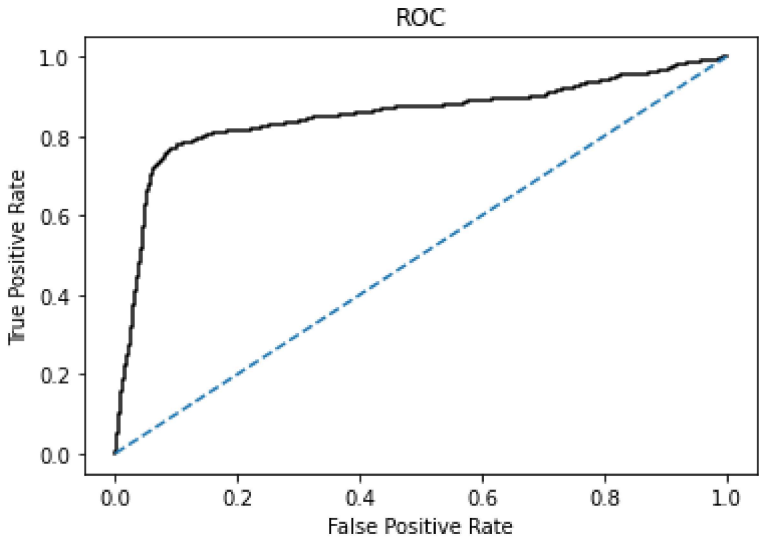


Test

In [166]:

```
NB_test_fpr, NB_test_tpr,_=roc_curve(test_labels,NB_model.predict_proba(X_test)[:,1])
plt.plot(NB_test_fpr,NB_test_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
NB_test_auc=roc_auc_score(test_labels,NB_model.predict_proba(X_test)[:,1])
print('Area under Curve is', NB_test_auc)
```

```
Area under Curve is 0.8504303854070085
```



6. (d) Model Performance Metrices

Train

In [167]:

```
NB_metrics=classification_report(train_labels, NB_train_predict,output_dict=True)
df=pd.DataFrame(NB_metrics).transpose()
NB_train_f1=round(df.loc["1"][2],2)
NB_train_recall=round(df.loc["1"][1],2)
NB_train_precision=round(df.loc["1"][0],2)
print ('NB_train_precision ',NB_train_precision)
print ('NB_train_recall ',NB_train_recall)
print ('NB_train_f1 ',NB_train_f1)
```

```
NB_train_precision  0.7
NB_train_recall  0.69
NB_train_f1  0.7
```

Test

In [168]:

```
NB_metrics=classification_report(test_labels, NB_test_predict,output_dict=True)
df=pd.DataFrame(NB_metrics).transpose()
NB_test_f1=round(df.loc["1"][2],2)
NB_test_recall=round(df.loc["1"][1],2)
NB_test_precision=round(df.loc["1"][0],2)
print ('NB_test_precision ',NB_test_precision)
print ('NB_test_recall ',NB_test_recall)
print ('NB_test_f1 ',NB_test_f1)
```

```
NB_test_precision  0.71
NB_test_recall  0.67
NB_test_f1  0.69
```

--------- 7. Gradient Boosting

In [169]:

```
params = {'loss' : 'deviance',
          'n_estimators': 100,
          'max_depth': 10,
          'criterion': 'friedman_mse',
          'subsample': 1.0,
          'learning_rate': 0.1,
          'min_samples_leaf': 10,
          'min_samples_split': 50,
          'random_state': 1,
          'ccp_alpha' : 0.5}
#
gbcl = GradientBoostingClassifier(**params)
gbcl = gbcl.fit(X_train, train_labels)
```

7. (a) Model Prediction

In [170]:

```
gbcl_train_predict = gbcl.predict(X_train)
gbcl_test_predict = gbcl.predict(X_test)
```

Train

In [171]:

```
gbcl_train_acc = gbcl.score(X_train,train_labels)
gbcl_train_acc
```

Out[171]:

```
0.8314927893984669
```

Test

In [172]:

```
gbcl_test_acc = gbcl.score(X_test,test_labels)
gbcl_test_acc
```

Out[172]:

```
0.8320703243407093
```

7. (b) Model Performance

Train

In [173]:

```
confusion_matrix(train_labels, gbcl_train_predict)
```

Out[173]:

```
array([[6400,    0],
       [1297,    0]], dtype=int64)
```

In [174]:

```
print(classification_report(train_labels, gbcl_train_predict))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91      6400
           1       0.00      0.00      0.00      1297

    accuracy                           0.83      7697
   macro avg       0.42      0.50      0.45      7697
weighted avg       0.69      0.83      0.75      7697


C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Test

In [175]:

```
confusion_matrix(test_labels, gbcl_test_predict)
```

Out[175]:

```
array([[2745,    0],
       [ 554,    0]], dtype=int64)
```

In [176]:

```
print(classification_report(test_labels, gbcl_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91      2745
           1       0.00      0.00      0.00       554

    accuracy                           0.83      3299
   macro avg       0.42      0.50      0.45      3299
weighted avg       0.69      0.83      0.76      3299


C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

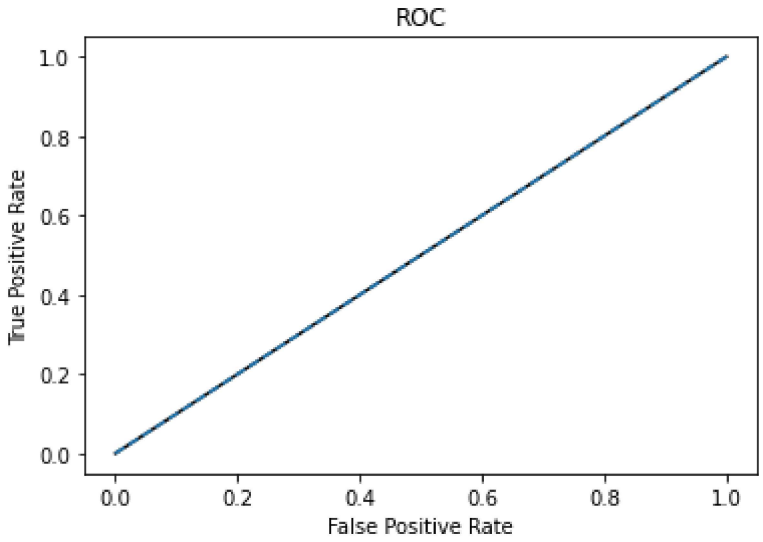
7. (c) ROC-AUC Graph

Train

In [177]:

```
gbcl_train_fpr, gbcl_train_tpr,_=roc_curve(train_labels,gbcl.predict_proba(X_train)[:,1])
plt.plot(gbcl_train_fpr,gbcl_train_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
gbcl_train_auc=roc_auc_score(train_labels,gbcl.predict_proba(X_train)[:,1])
print('Area under Curve is', gbcl_train_auc)
```

```
Area under Curve is 0.5
```

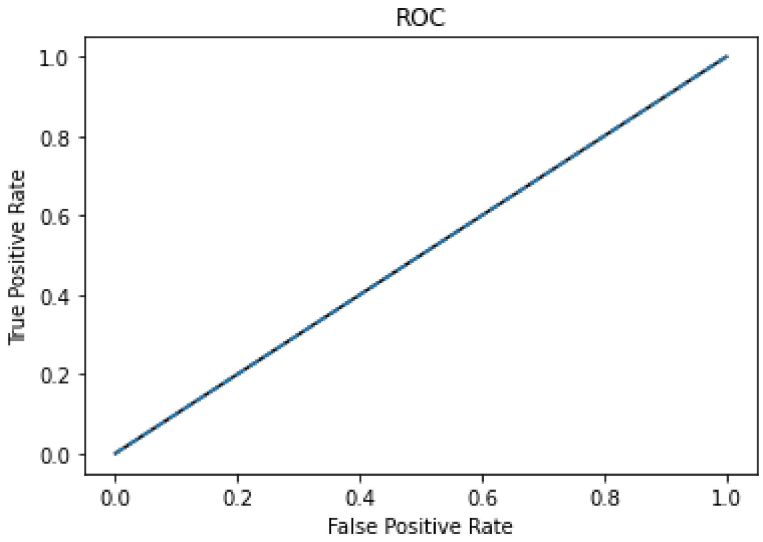


Test

In [178]:

```
gbcl_test_fpr, gbcl_test_tpr,_=roc_curve(test_labels,gbcl.predict_proba(X_test)[:,1])
plt.plot(gbcl_test_fpr,gbcl_test_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
gbcl_test_auc=roc_auc_score(test_labels,gbcl.predict_proba(X_test)[:,1])
print('Area under Curve is', gbcl_test_auc)
```

```
Area under Curve is 0.5
```



7. (d) Model Performance Metrices

Train

In [179]:

```
gbcl_metrics=classification_report(train_labels, gbcl_train_predict,output_dict=True)
df=pd.DataFrame(gbcl_metrics).transpose()
gbcl_train_f1=round(df.loc["1"][2],2)
gbcl_train_recall=round(df.loc["1"][1],2)
gbcl_train_precision=round(df.loc["1"][0],2)
print ('gbcl_train_precision ',gbcl_train_precision)
print ('gbcl_train_recall ',gbcl_train_recall)
print ('gbcl_train_f1 ',gbcl_train_f1)
```

```
gbcl_train_precision  0.0
gbcl_train_recall  0.0
gbcl_train_f1  0.0
```

```
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Test

In [180]:

```
gbcl_metrics=classification_report(test_labels, gbcl_test_predict,output_dict=True)
df=pd.DataFrame(gbcl_metrics).transpose()
gbcl_test_f1=round(df.loc["1"][2],2)
gbcl_test_recall=round(df.loc["1"][1],2)
gbcl_test_precision=round(df.loc["1"][0],2)
print ('gbcl_test_precision ',gbcl_test_precision)
print ('gbcl_test_recall ',gbcl_test_recall)
print ('gbcl_test_f1 ',gbcl_test_f1)
```

```
gbcl_test_precision  0.0
gbcl_test_recall  0.0
gbcl_test_f1  0.0

C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

8. Xtreme Gradient Boosting (XGB)

In [181]:

```
from xgboost import XGBClassifier
import optuna.integration.lightgbm as lgb
```

In [182]:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import optuna.integration.lightgbm as lgb
import optuna
```

In [183]:

```
xgb = XGBClassifier(base_score = 0.5,
                    colsample_bylevel = 1,
                    colsample_bytree = 1,
                    gamma = 0,
                    learning_rate = 0.1,
                    max_depth = 10,
                    min_child_weight = 1,
                    n_estimators = 100,
                    objective ='binary:logistic',
                    reg_alpha = 1,
                    reg_lambda = 1,
                    scale_pos_weight=1,
                    subsample = 1)
xgb = xgb.fit(X_train, train_labels)
```

```
C:\Users\rahul\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarn
ing: The use of label encoder in XGBClassifier is deprecated and will be rem
oved in a future release. To remove this warning, do the following: 1) Pass
option use_label_encoder=False when constructing XGBClassifier object; and
2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [n
um_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[20:38:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behav
ior.
```

8. (a) Model Prediction

In [184]:

```
xgb_train_predict = xgb.predict(X_train)
xgb_test_predict = xgb.predict(X_test)
```

Train

In [185]:

```
xgb_train_acc = xgb.score(X_train,train_labels)
xgb_train_acc
```

Out[185]:

```
0.9976614265298168
```

Test

In [186]:

```
xgb_test_acc = xgb.score(X_test,test_labels)
xgb_test_acc
```

Out[186]:

```
0.976659593816308
```

8. (b) Model Performance

Train

In [187]:

```
confusion_matrix(train_labels, xgb_train_predict)
```

Out[187]:

```
array([[6400,    0],
       [  18, 1279]], dtype=int64)
```

In [188]:

```
print(classification_report(train_labels, xgb_train_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6400
           1       1.00      0.99      0.99      1297

    accuracy                           1.00      7697
   macro avg       1.00      0.99      1.00      7697
weighted avg       1.00      1.00      1.00      7697
```

Test

In [189]:

```
confusion_matrix(test_labels, xgb_test_predict)
```

Out[189]:

```
array([[2726,   19],
       [  58,  496]], dtype=int64)
```

In [190]:

```
print(classification_report(test_labels, xgb_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      2745
           1       0.96      0.90      0.93       554

    accuracy                           0.98      3299
   macro avg       0.97      0.94      0.96      3299
weighted avg       0.98      0.98      0.98      3299
```

8. (c) ROC-AUC Graph

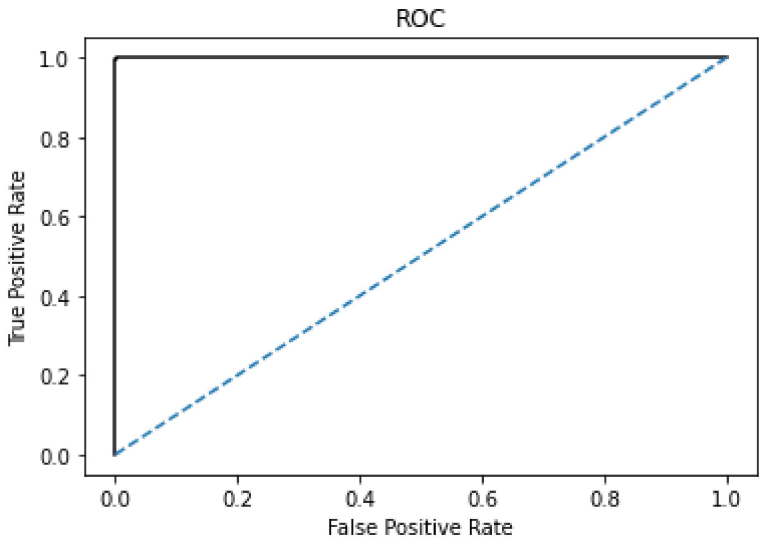
In [191]:

```
## Train
```

In [192]:

```
xgb_train_fpr, xgb_train_tpr,_=roc_curve(train_labels,xgb.predict_proba(X_train)[:,1])
plt.plot(xgb_train_fpr,xgb_train_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
xgb_train_auc=roc_auc_score(train_labels,xgb.predict_proba(X_train)[:,1])
print('Area under Curve is', xgb_train_auc)
```

```
Area under Curve is 0.9999943378951426
```



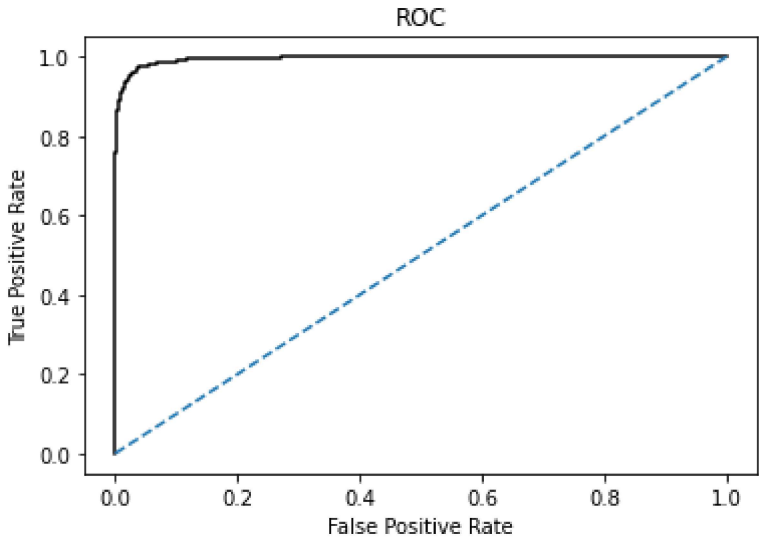
In [193]:

```
## Train
```

In [194]:

```
xgb_test_fpr, xgb_test_tpr,_=roc_curve(test_labels,xgb.predict_proba(X_test)[:,1])
plt.plot(xgb_test_fpr,xgb_test_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
xgb_test_auc=roc_auc_score(test_labels,xgb.predict_proba(X_test)[:,1])
print('Area under Curve is', xgb_test_auc)
```

```
Area under Curve is 0.9951792888941495
```



8. (d) Model Performance Metrices

Train

In [195]:

```
xgb_metrics=classification_report(train_labels, xgb_train_predict,output_dict=True)
df=pd.DataFrame(xgb_metrics).transpose()
xgb_train_f1=round(df.loc["1"][2],2)
xgb_train_recall=round(df.loc["1"][1],2)
xgb_train_precision=round(df.loc["1"][0],2)
print ('gbcl_train_precision ',xgb_train_precision)
print ('gbcl_train_recall ',xgb_train_recall)
print ('gbcl_train_f1 ',xgb_train_f1)
```

```
gbcl_train_precision  1.0
gbcl_train_recall  0.99
gbcl_train_f1  0.99
```

Test

In [196]:

```
xgb_metrics=classification_report(test_labels, xgb_test_predict,output_dict=True)
df=pd.DataFrame(xgb_metrics).transpose()
xgb_test_f1=round(df.loc["1"][2],2)
xgb_test_recall=round(df.loc["1"][1],2)
xgb_test_precision=round(df.loc["1"][0],2)
print ('gbcl_test_precision ',xgb_test_precision)
print ('gbcl_test_recall ',xgb_test_recall)
print ('gbcl_test_f1 ',xgb_test_f1)
```

```
gbcl_test_precision  0.96
gbcl_test_recall  0.9
gbcl_test_f1  0.93
```

----------9. Extra Tree Classifier

In [197]:

```
etc = ExtraTreesClassifier(n_estimators = 100,
                           criterion ='gini',
                           max_depth = 10,
                           min_samples_split = 10,
                           min_samples_leaf = 50,
                           max_features = 'auto',
                           min_impurity_decrease = 0.5,
                           random_state = 1)
etc = etc.fit(X_train, train_labels)
```

9. (a) Model Prediction

In [198]:

```
etc_train_predict = etc.predict(X_train)
etc_test_predict = etc.predict(X_test)
```

Train

In [199]:

```
etc_train_acc = etc.score(X_train,train_labels)
etc_train_acc
```

Out[199]:

```
0.8314927893984669
```

Test

In [200]:

```
etc_test_acc = etc.score(X_test,test_labels)
etc_test_acc
```

Out[200]:

```
0.8320703243407093
```

9. (b) Model Performance

Train

In [201]:

```
confusion_matrix(train_labels, etc_train_predict)
```

Out[201]:

```
array([[6400,    0],
       [1297,    0]], dtype=int64)
```

In [202]:

```
print(classification_report(train_labels, etc_train_predict))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91      6400
           1       0.00      0.00      0.00      1297

    accuracy                           0.83      7697
   macro avg       0.42      0.50      0.45      7697
weighted avg       0.69      0.83      0.75      7697


C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Train

In [203]:

```
confusion_matrix(test_labels, etc_test_predict)
```

Out[203]:

```
array([[2745,    0],
       [ 554,    0]], dtype=int64)
```

In [204]:

```
print(classification_report(test_labels, etc_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91      2745
           1       0.00      0.00      0.00       554

    accuracy                           0.83      3299
   macro avg       0.42      0.50      0.45      3299
weighted avg       0.69      0.83      0.76      3299


C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

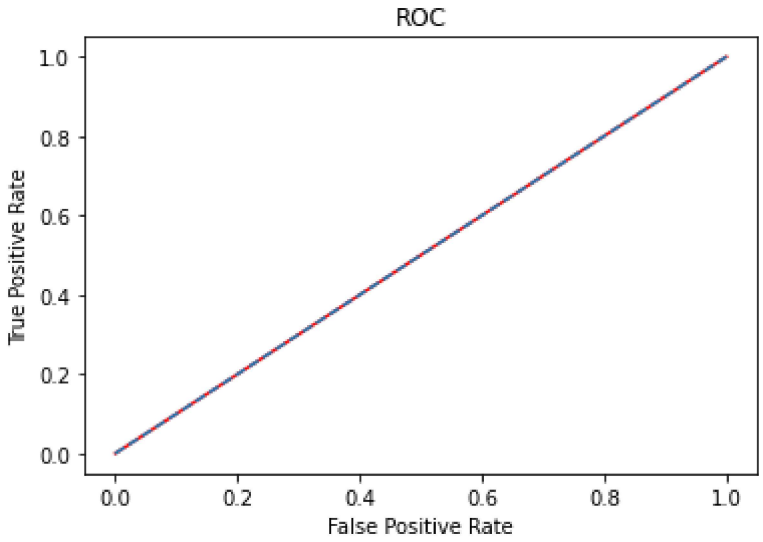
9. (c) ROC-AUC Graph

Train

In [205]:

```
etc_train_fpr, etc_train_tpr,_= roc_curve(train_labels,etc.predict_proba(X_train)[:,1])
plt.plot(etc_train_fpr,etc_train_tpr,color='red')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
etc_train_auc=roc_auc_score(train_labels,etc.predict_proba(X_train)[:,1])
print('Area under Curve is', etc_train_auc)
```

```
Area under Curve is 0.5
```

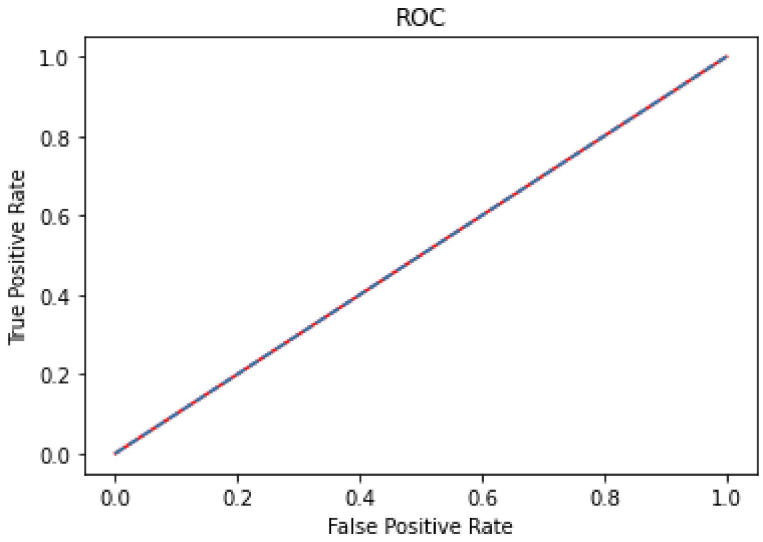


Test

In [206]:

```
etc_test_fpr, etc_test_tpr,_= roc_curve(test_labels,etc.predict_proba(X_test)[:,1])
plt.plot(etc_test_fpr,etc_test_tpr,color='red')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
etc_test_auc=roc_auc_score(test_labels,etc.predict_proba(X_test)[:,1])
print('Area under Curve is', etc_test_auc)
```

```
Area under Curve is 0.5
```



9. (d) Model Performance Metrices

Train

In [207]:

```
etc_metrics=classification_report(train_labels, etc_train_predict,output_dict=True)
df=pd.DataFrame(etc_metrics).transpose()
etc_train_f1=round(df.loc["1"][2],2)
etc_train_recall=round(df.loc["1"][1],2)
etc_train_precision=round(df.loc["1"][0],2)
print ('etc_train_precision ',etc_train_precision)
print ('etc_train_recall ',etc_train_recall)
print ('etc_train_f1 ',etc_train_f1)
```

```
etc_train_precision  0.0
etc_train_recall  0.0
etc_train_f1  0.0

C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Test

In [208]:

```
etc_metrics=classification_report(test_labels, etc_test_predict,output_dict=True)
df=pd.DataFrame(etc_metrics).transpose()
etc_test_f1=round(df.loc["1"][2],2)
etc_test_recall=round(df.loc["1"][1],2)
etc_test_precision=round(df.loc["1"][0],2)
print ('etc_test_precision ',etc_test_precision)
print ('etc_test_recall ',etc_test_recall)
print ('etc_test_f1 ',etc_test_f1)
```

```
etc_test_precision  0.0
etc_test_recall  0.0
etc_test_f1  0.0

C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Model Performance Comparison

1. Performance Matrics - on Train Data

In [209]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({
       'CART Train':[cart_train_acc,cart_train_auc,cart_train_recall,cart_train_precision,c
       'RF Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_train_precision,rf_train_f1
       'Log Train':[log_train_acc,log_train_auc,log_train_recall,log_train_precision,log_tr
       'LDA Train':[lda_train_acc,lda_train_auc,lda_train_recall,lda_train_precision,lda_tr
       'KNN Train':[KNN_train_acc,KNN_train_auc,KNN_train_recall,KNN_train_precision,KNN_tr
       'NB Train':[NB_train_acc,NB_train_auc,NB_train_recall,NB_train_precision,NB_train_f1
       'Gr.Boost Train':[gbcl_train_acc,gbcl_train_auc,gbcl_train_recall,gbcl_train_precisi
       'XG Boost Train':[xgb_train_acc,xgb_train_auc,xgb_train_recall,xgb_train_precision,x
       'Extra Tree Train':[etc_train_acc,etc_train_auc,etc_train_recall,etc_train_precision
round(data,2)
```

Out[209]:

	CART Train	RF Train	Log Train	LDA Train	KNN Train	NB Train	Gr.Boost Train	XG Boost Train	Extra Tree Train
Accuracy	0.97	0.97	0.89	0.88	0.84	0.90	0.83	1.00	0.83
AUC	0.98	0.99	0.89	0.89	0.84	0.87	0.50	1.00	0.50
Recall	0.84	0.84	0.54	0.62	0.15	0.69	0.00	0.99	0.00
Precision	0.96	0.96	0.75	0.65	0.68	0.70	0.00	1.00	0.00
F1 Score	0.90	0.90	0.63	0.63	0.24	0.70	0.00	0.99	0.00

2. Performance Matrics - on Test Data

In [210]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({
        'CART Test':[cart_test_acc,cart_test_auc,cart_test_recall,cart_test_precision,cart_
        'RF Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_precision,rf_test_f1],
        'Log Test':[log_test_acc,log_test_auc,log_test_recall,log_test_precision,log_test_f
        'LDA Test':[lda_test_acc,lda_test_auc,lda_test_recall,lda_test_precision,lda_test_f
        'KNN Test':[KNN_test_acc,KNN_test_auc,KNN_test_recall,KNN_test_precision,KNN_test_f
        'NB Test':[NB_test_acc,NB_test_auc,NB_test_recall,NB_test_precision,NB_test_f1],
        'Gr.Boost Test':[gbcl_test_acc,gbcl_test_auc,gbcl_test_recall,gbcl_test_precision,g
        'XG Boost Test':[xgb_test_acc,xgb_test_auc,xgb_test_recall,xgb_test_precision,xgb_t
        'Extra Tree Test':[etc_test_acc,etc_test_auc,etc_test_recall,etc_test_precision,etc
round(data,2)
```

Out[210]:

	CART Test	RF Test	Log Test	LDA Test	KNN Test	NB Test	Gr.Boost Test	XG Boost Test	Extra Tree Test
Accuracy	0.96	0.96	0.89	0.88	0.83	0.90	0.83	0.98	0.83
AUC	0.97	0.98	0.88	0.87	0.71	0.85	0.50	1.00	0.50
Recall	0.81	0.81	0.51	0.94	0.06	0.67	0.00	0.90	0.00
Precision	0.95	0.95	0.74	0.92	0.45	0.71	0.00	0.96	0.00
F1 Score	0.87	0.87	0.61	0.93	0.11	0.69	0.00	0.93	0.00

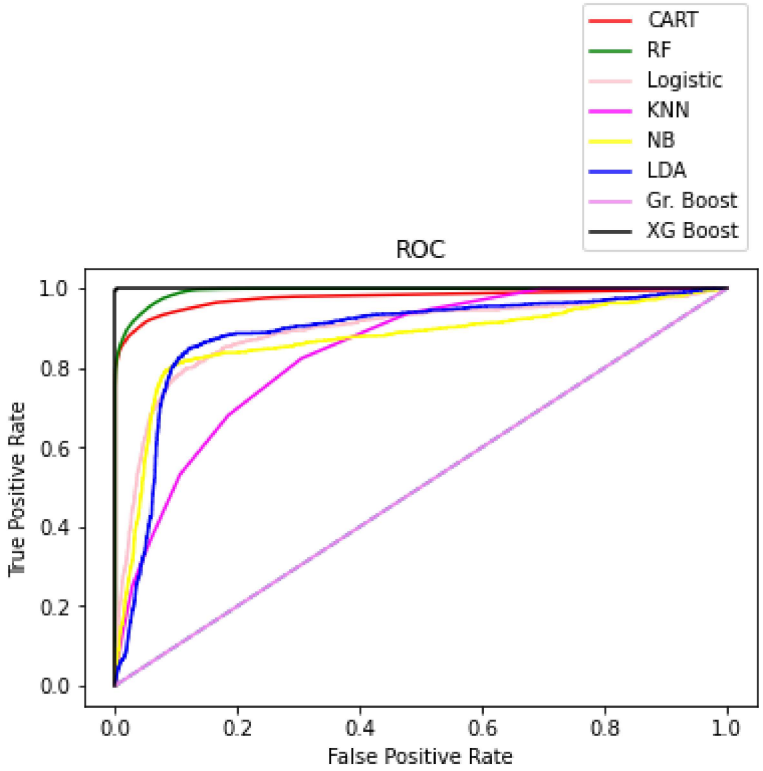
3. ROC-AUC - on Train Data

In [211]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(cart_train_fpr, cart_train_tpr,color='red',label="CART")
plt.plot(rf_train_fpr,rf_train_tpr,color='green',label="RF")
plt.plot(log_train_fpr,log_train_tpr,color='Pink',label="Logistic")
plt.plot(KNN_train_fpr,KNN_train_tpr,color='magenta',label="KNN")
plt.plot(NB_train_fpr,NB_train_tpr,color='yellow',label="NB")
plt.plot(lda_train_fpr,lda_train_tpr,color='blue',label="LDA")
plt.plot(gbcl_train_fpr,gbcl_train_tpr,color='violet',label="Gr. Boost")
plt.plot(xgb_train_fpr,xgb_train_tpr,color='black',label="XG Boost")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower right')
```

Out[211]:

```
<matplotlib.legend.Legend at 0x22e88cd9fa0>
```



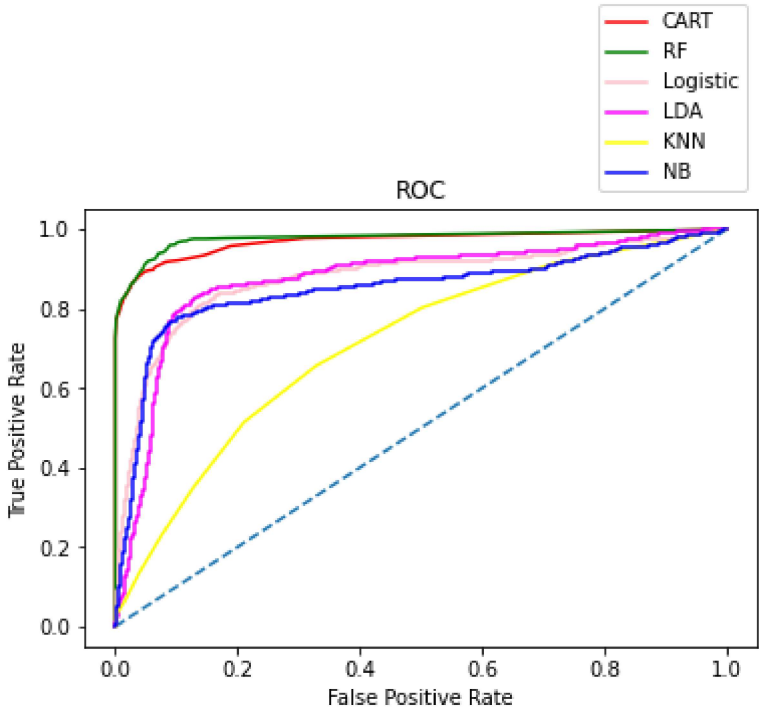
4. ROC-AUC - on Test Data

In [212]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(cart_test_fpr, cart_test_tpr,color='red',label="CART")
plt.plot(rf_test_fpr,rf_test_tpr,color='green',label="RF")
plt.plot(log_test_fpr,log_test_tpr,color='pink',label="Logistic")
plt.plot(lda_test_fpr,lda_test_tpr,color='magenta',label="LDA")
plt.plot(KNN_test_fpr,KNN_test_tpr,color='yellow',label="KNN")
plt.plot(NB_test_fpr,NB_test_tpr,color='blue',label="NB")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower right')
```

Out[212]:

```
<matplotlib.legend.Legend at 0x22e88d55a90>
```



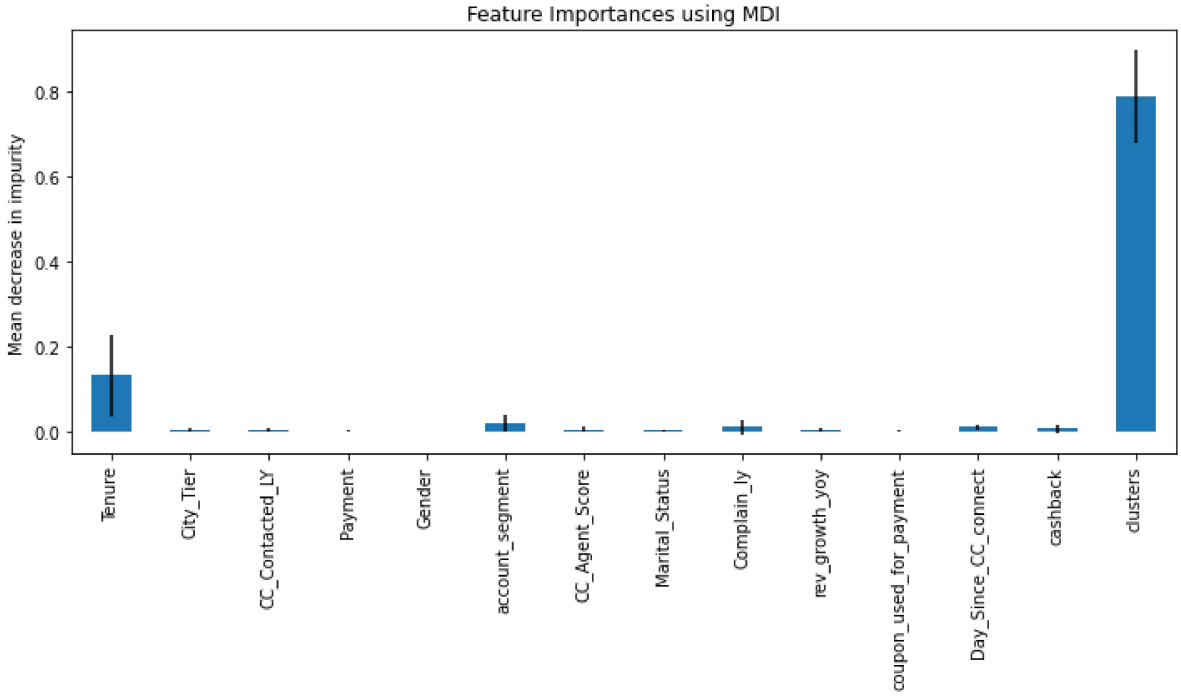
After Cheking all types of model performances, next we need to find the importances of the features used for model building.Below are the procedure used for the same :

(1) Impurity Based (Base estimator = decision tree) (2) Permutation Based (3) SHAP (shapley additive features) (4) ELI5 python Package

1. Impurity-based Importance.

In [213]:

```
forest_importances = pd.Series(best_grid_rf.feature_importances_, index=X_train.columns)
std = np.std([tree.feature_importances_ for tree in best_grid_rf.estimators_], axis=0)
#
fig, ax = plt.subplots(figsize=(10,6))
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature Importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



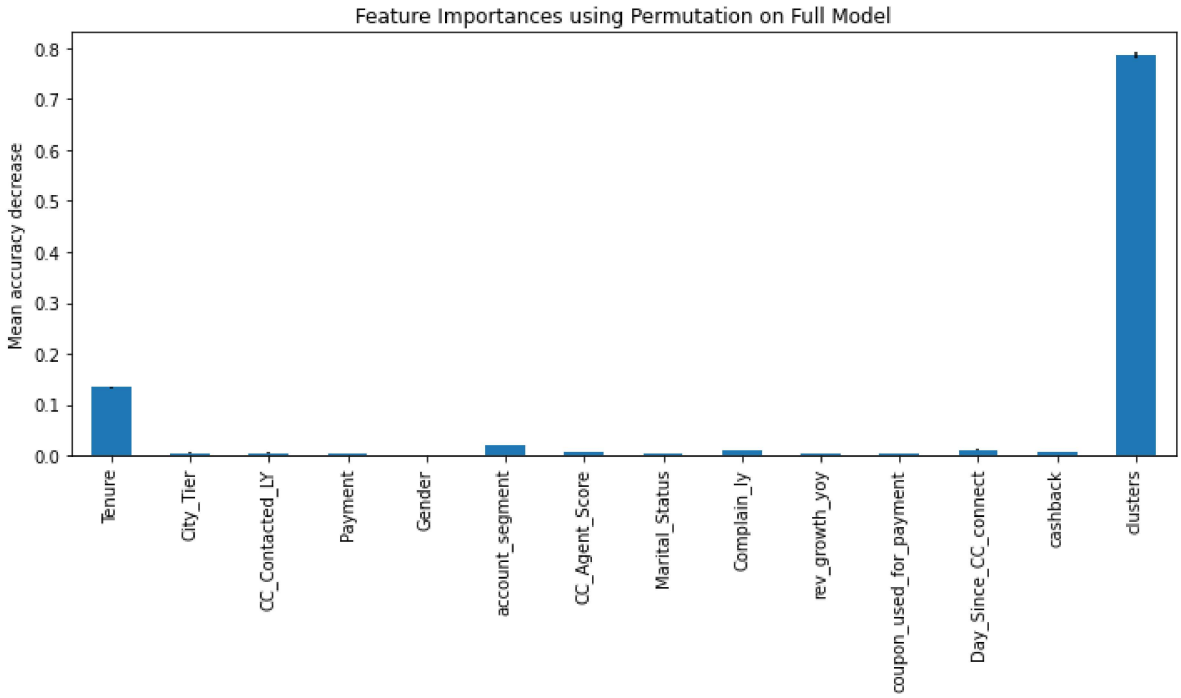
2. Permutation-based Importance.

In [214]:

```
from sklearn.inspection import permutation_importance
```

In [215]:

```
result = permutation_importance(best_grid_rf, X_test, test_labels, n_repeats=10, random_sta
#
fig, ax = plt.subplots(figsize=(10,6))
forest_importances.plot.bar(yerr=result.importances_std, ax=ax)
ax.set_title("Feature Importances using Permutation on Full Model")
ax.set_ylabel("Mean accuracy decrease")
fig.tight_layout()
plt.show()
```

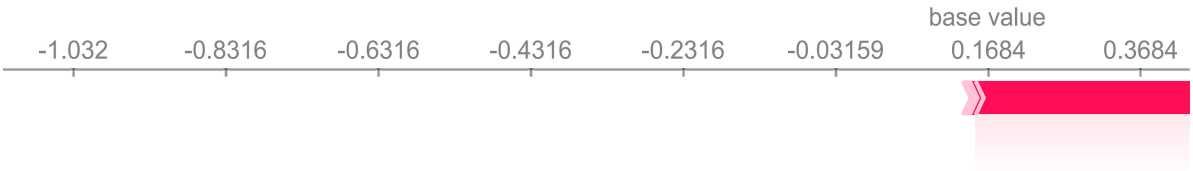


3. SHAP Value-based Importance.

In [216]:

```
import shap
row_to_show = 5
data_for_prediction = X_train.iloc[row_to_show]
explainer = shap.TreeExplainer(best_grid_rf) # Create object that can calculate shap values
shap_values = explainer.shap_values(data_for_prediction) # Calculate Shap values
#
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], data_for_prediction)
```

Out[216]:



4. ELI5 Feature Importance.

In [217]:

```
import eli5 as eli
eli.explain_weights(best_grid_rf)
```

Out[217]:

Weight	Feature
0.7865 ± 0.2209	x13
0.1334 ± 0.1917	x0
0.0198 ± 0.0370	x5
0.0119 ± 0.0124	x11
0.0112 ± 0.0335	x8
0.0071 ± 0.0170	x12
0.0065 ± 0.0103	x6
0.0054 ± 0.0065	x2
0.0050 ± 0.0084	x1
0.0039 ± 0.0061	x9
0.0033 ± 0.0068	x7
0.0027 ± 0.0057	x3
0.0025 ± 0.0047	x10
0.0009 ± 0.0023	x4

In [218]:

```
eli.explain_prediction(best_grid_rf , np.array(X_test)[1])
```

```
X does not have valid feature names, but RandomForestClassifier was fitted w
ith feature names
```

Out[218]:

y=0 (probability **0.934**) top features

Contribution?	Feature
+0.832	<BIAS>
+0.158	x0
+0.010	x8
+0.008	x2
+0.006	x6
+0.005	x1
+0.004	x11
+0.002	x3
+0.001	x10
-0.000	x7
-0.004	x5
-0.006	x12
-0.036	x13
-0.043	x9

In [219]:

```
eli.show_prediction(best_grid_rf, X_test.iloc[1],feature_names=list(X_train.columns),
                    show_feature_values=True)
```

```
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
```

In [220]:

```
eli.show_prediction(best_grid_rf, X_test.iloc[20],feature_names=list(X_train.columns),
                    show_feature_values=True)
```

```
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
e names
X has feature names, but DecisionTreeClassifier was fitted without featur
```

In [ ]:

THANKS

In [ ]: