ASA GROUP ASSIGNMENT- GROUP-17

1. ARCHIT SRIVASTAVA_12120052

2. SRIYA PARUCHURI_12120019

3. JHANVI SHARMA_12120086

In [61]:
```
#Importing basic libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [62]:
```
#Importing libraries for visualization
import plotly.offline as py # visualization
py.init_notebook_mode(connected=True) # visualization
import plotly.graph_objs as go # visualization
from plotly.subplots import make_subplots
import plotly.figure_factory as ff # visualization
import seaborn as sns
from matplotlib.ticker import MultipleLocator
import matplotlib.pyplot as plt
%matplotlib inline
```

In [27]:
```
#Importing Libraries for execution & data cleaning.
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score

from sklearn.metrics import make_scorer

from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import StandardScaler
from scipy.stats import pearsonr

from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, roc_auc_score
from sklearn.metrics import precision_recall_curve, auc, f1_score, plot_confusion_matrix, precision_score, recall_sc

import lifelines
from lifelines import KaplanMeierFitter
from lifelines import CoxPHFitter
from imblearn.over_sampling import SMOTE
```

In [118…
```
df = pd.read_csv('C:/ISB/Term-04/Stats-03/Assignment/Group/file-17.csv')
df
```

Out[118]:

	Unnamed: 0	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	...	De
0	1255	4030-VPZBD	Female	0	No	No	2	No	No phone service	DSL	...	
1	1138	0774-IFUVM	Male	0	Yes	Yes	11	Yes	Yes	DSL	...	
2	1249	2833-SLKDQ	Male	0	No	No	1	Yes	No	DSL	...	
3	4263	2121-JAFOM	Female	0	Yes	No	72	Yes	Yes	Fiber optic	...	
4	5871	0733-VUNUW	Male	0	No	No	24	Yes	Yes	DSL	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
4996	4211	2034-CGRHZ	Male	1	No	No	24	Yes	Yes	Fiber optic	...	
4997	968	3429-IFLEM	Female	0	No	No	71	Yes	No	DSL	...	
4998	5890	0383-CLDDA	Female	0	No	No	69	Yes	Yes	DSL	...	
4999	714	4312-KFRXN	Male	0	Yes	No	72	Yes	Yes	No	...	
5000	5364	9050-QLROH	Male	0	No	No	18	Yes	Yes	Fiber optic	...	

5001 rows × 22 columns

In [6]:
```
# Checking for dataset type & counts in each field.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0        5001 non-null   int64  
 1   customerID        5001 non-null   object 
 2   gender            5001 non-null   object 
 3   SeniorCitizen     5001 non-null   int64  
 4   Partner           5001 non-null   object 
 5   Dependents        5001 non-null   object 
 6   tenure            5001 non-null   int64  
 7   PhoneService      5001 non-null   object 
 8   MultipleLines     5001 non-null   object 
 9   InternetService   5001 non-null   object 
 10  OnlineSecurity    5001 non-null   object 
 11  OnlineBackup      5001 non-null   object 
 12  DeviceProtection  5001 non-null   object 
 13  TechSupport       5001 non-null   object 
 14  StreamingTV       5001 non-null   object 
 15  StreamingMovies   5001 non-null   object 
 16  Contract          5001 non-null   object 
 17  PaperlessBilling  5001 non-null   object 
 18  PaymentMethod     5001 non-null   object 
 19  MonthlyCharges    5001 non-null   float64
 20  TotalCharges      5001 non-null   object 
 21  Churn             5001 non-null   object 
dtypes: float64(1), int64(3), object(18)
memory usage: 859.7+ KB
```

Datatype of Columns such as TotalCharges is in object, hence converting into float.

In [119…
```
#df['TotalCharges'] = df['TotalCharges'].astype(float)
#df["TotalCharges"] = [float(str(i).replace("", "")) for i in df["TotalCharges"]]
#df["TotalCharges"] = df['TotalCharges'].str.replace(' ', '').astype(float)
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
```

In [65]:
```
# Checking the count of Unique values in each column (will be checking for categorical columns)
df.astype('object').describe(include='all').loc['unique', :]
```

Out[65]:
```
Unnamed: 0            5001
customerID            5001
gender                   2
SeniorCitizen            2
Partner                  2
Dependents               2
tenure                  73
PhoneService             2
MultipleLines            3
InternetService          3
OnlineSecurity           3
OnlineBackup             3
DeviceProtection         3
TechSupport              3
StreamingTV              3
StreamingMovies          3
Contract                 3
PaperlessBilling         2
PaymentMethod            4
MonthlyCharges      1450.0
TotalCharges        4710.0
Churn                    2
Name: unique, dtype: object
```

In [9]:
```
# Getting the unique values of each column for further processing
for col in df:
    print(col)
    print(df[col].unique())
    print('\n')
```

```
Unnamed: 0
[1255 1138 1249 ... 5890  714 5364]


customerID
['4030-VPZBD' '0774-IFUVM' '2833-SLKDQ' ... '0383-CLDDA' '4312-KFRXN'
 '9050-QLROH']


gender
['Female' 'Male']


SeniorCitizen
[0 1]


Partner
['No' 'Yes']


Dependents
['No' 'Yes']


tenure
[ 2 11  1 72 24 38  6 64 46 50 52 23  4 30 17 71 37 57 55 10 54 69 15 26
 16 68 51 70 42 61 43 29 18 20 62 53 25 36 67 14 13 47 31  5  7 33 12 41
 65 32 28 34 63 35 59  3 19 40 39 45 66 58 27 21 60 44 49 48  8 22  9 56
  0]


PhoneService
['No' 'Yes']


MultipleLines
['No phone service' 'Yes' 'No']


InternetService
['DSL' 'Fiber optic' 'No']


OnlineSecurity
['No' 'Yes' 'No internet service']


OnlineBackup
['No' 'Yes' 'No internet service']


DeviceProtection
['No' 'Yes' 'No internet service']


TechSupport
['Yes' 'No' 'No internet service']


StreamingTV
['No' 'Yes' 'No internet service']


StreamingMovies
['No' 'Yes' 'No internet service']


Contract
['Month-to-month' 'Two year' 'One year']


PaperlessBilling
['Yes' 'No']


PaymentMethod
['Electronic check' 'Bank transfer (automatic)' 'Mailed check'
 'Credit card (automatic)']


MonthlyCharges
[30.9  65.15 45.05 ... 30.05 51.45 88.6 ]
```

```
TotalCharges
[  59.05  723.35   45.05 ... 5897.4  1710.9  1597.25]


Churn
['Yes' 'No']
```

In [120…
```
# Renaming the string values of "PaymentMethod" for using in further analysis
payment_column = {'Electronic check': 'E-Check' , 'Bank transfer (automatic)': 'BT', 'Mailed check': 'Mailed-Check',
df["PaymentMethod"].replace(payment_column, inplace=True)
```

In [121…
```
#checking for null values
df.isnull().sum()
```

Out[121]:
```
Unnamed: 0          0
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        7
Churn               0
dtype: int64
```

Dropping the columns in TotalCharges as there are 7 null values.

In [122…
```
df.dropna(how='any',inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4994 entries, 0 to 5000
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        4994 non-null   int64
 1   customerID        4994 non-null   object
 2   gender            4994 non-null   object
 3   SeniorCitizen     4994 non-null   int64
 4   Partner           4994 non-null   object
 5   Dependents        4994 non-null   object
 6   tenure            4994 non-null   int64
 7   PhoneService      4994 non-null   object
 8   MultipleLines     4994 non-null   object
 9   InternetService   4994 non-null   object
 10  OnlineSecurity    4994 non-null   object
 11  OnlineBackup      4994 non-null   object
 12  DeviceProtection  4994 non-null   object
 13  TechSupport       4994 non-null   object
 14  StreamingTV       4994 non-null   object
 15  StreamingMovies   4994 non-null   object
 16  Contract          4994 non-null   object
 17  PaperlessBilling  4994 non-null   object
 18  PaymentMethod     4994 non-null   object
 19  MonthlyCharges    4994 non-null   float64
 20  TotalCharges      4994 non-null   float64
 21  Churn             4994 non-null   object
dtypes: float64(2), int64(3), object(17)
memory usage: 897.4+ KB
```

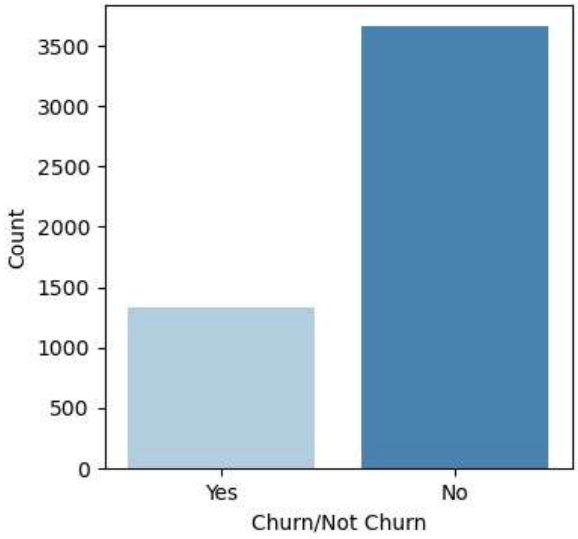
EDA

In [123…
```
#Checking the distibution of numerical values
df.describe().T
```

Out[123]:

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	4994.0	3541.682219	2028.105660	0.0	1805.2500	3550.500	5296.750	7042.0
SeniorCitizen	4994.0	0.161594	0.368115	0.0	0.0000	0.000	0.000	1.0
tenure	4994.0	32.149579	24.561042	1.0	8.2500	28.000	55.000	72.0
MonthlyCharges	4994.0	64.993542	30.019107	18.7	36.7000	70.300	90.100	118.6
TotalCharges	4994.0	2274.009592	2263.095125	18.8	382.9875	1408.925	3778.175	8684.8

In [124…
```
plt.figure(figsize=(4, 4))
ax = sns.countplot(x=df['Churn'], palette="Blues", linewidth=1).set(xlabel='Churn/Not Churn', ylabel='Count')
```

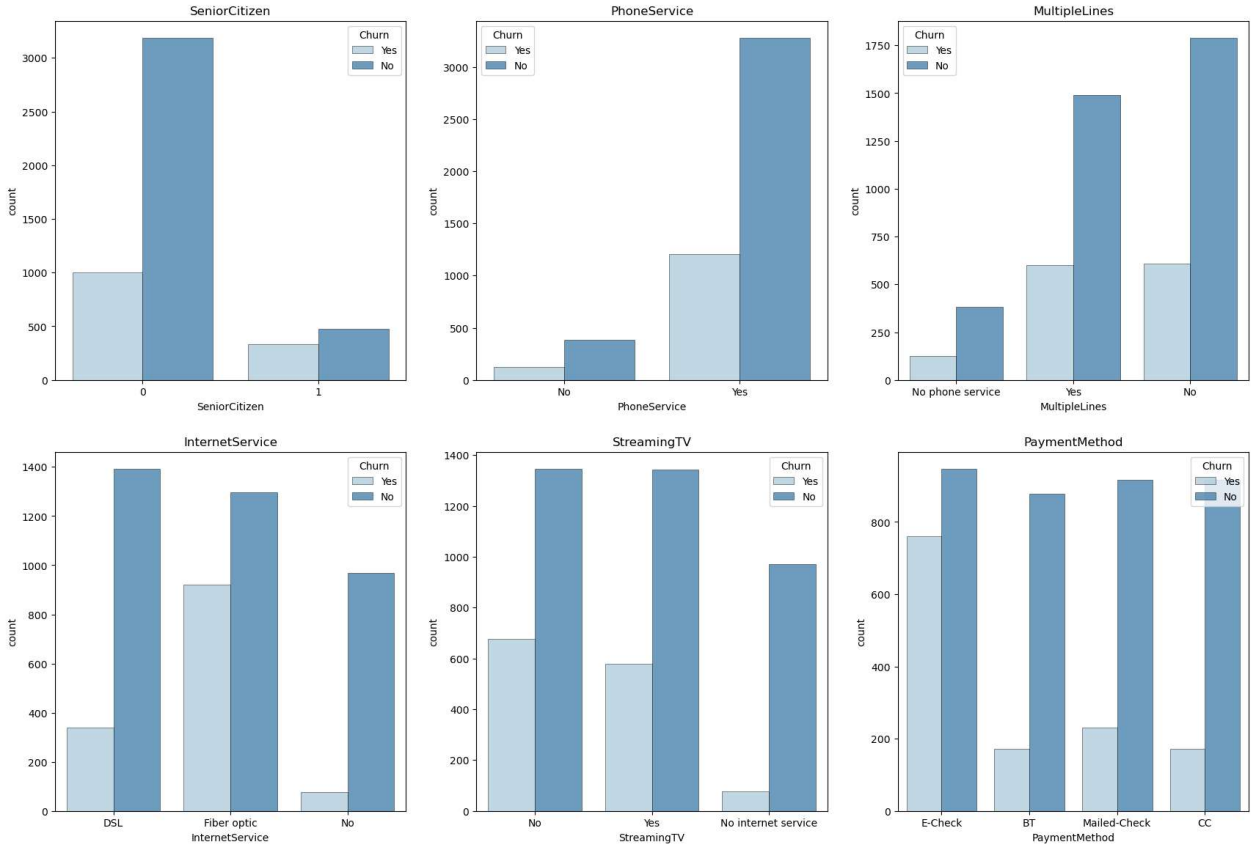


In [125…
```
non_active = len(df[df['Churn']=="Yes"])
active = len(df[df['Churn']=="No"])
per_non_active= round((non_active/(non_active+active))*100,2)
per_active= round((active/(non_active+active))*100,2)

print("Non-Churn Users & Churn Users are ",per_active,"& ", per_non_active,"respectively")
```

```
Non-Churn Users & Churn Users are  73.29 &  26.71 respectively
```

In [126…
```
## Defining multiple count bar plots of Churn of different parameters.

def countplot(x, y, df):

    rows = int(str(plots[len(y)][0])[0])
    columns = int(str(plots[len(y)][0])[1])

    plt.figure(figsize=(7*columns, 7*rows))

    for i, j in enumerate(y):
        plt.subplot(plots[len(y)][i])
        ax = sns.countplot(x=j, hue=x, data=df, palette='Blues', alpha=0.8, linewidth=0.4, edgecolor='black')
        ax.set_title(j)

    return plt.show()
```

In [127…
```
plots = {1 : [111], 2: [121, 122], 3: [131, 132, 133], 4: [221, 222, 223, 224], 5: [231, 232, 233, 234, 235], 6: [23
countplot("Churn", ['SeniorCitizen', 'PhoneService', 'MultipleLines', 'InternetService', 'StreamingTV', 'PaymentMeth
```

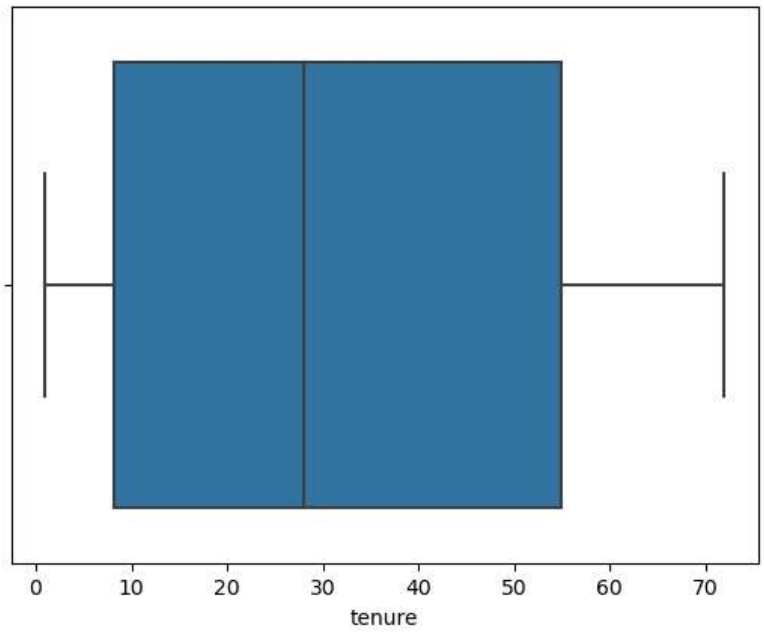


From the above observations, senior citizens have higher rate of churn.This may also be due to the death rate.
Customers using Fiberoptic services as internet are having very high proportion of churn.
Customers paying with e-check are most suspectible to churn.

In [18]:
```
sns.boxplot(df.tenure)
```

```
C:\Users\archi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data
`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```

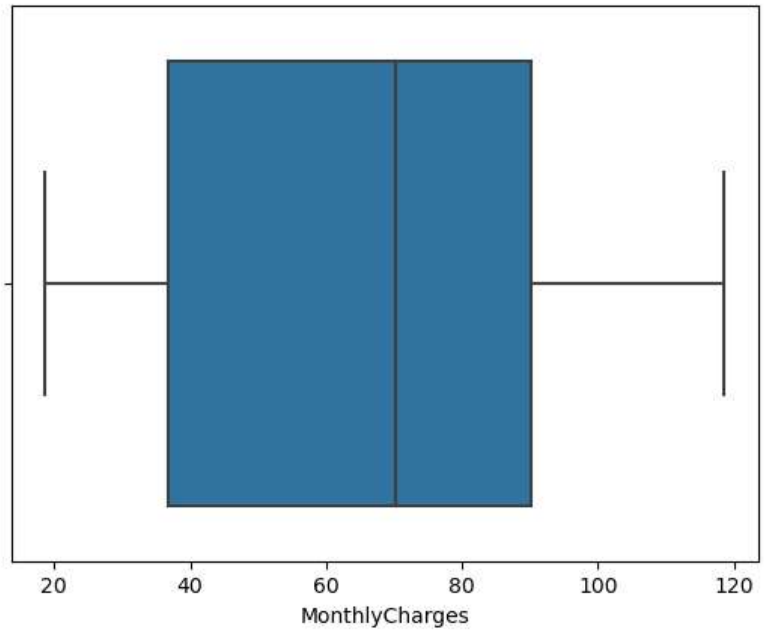
Out[18]:
```
<AxesSubplot:xlabel='tenure'>
```



In [20]:
```
sns.boxplot(df.MonthlyCharges)
```

```
C:\Users\archi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data
`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```

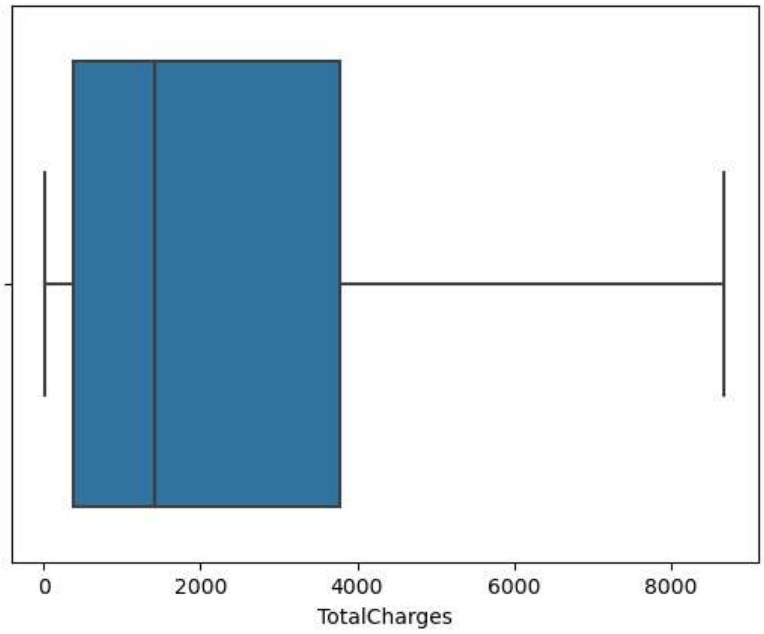
Out[20]:
```
<AxesSubplot:xlabel='MonthlyCharges'>
```



In [19]:
```
sns.boxplot(df.TotalCharges)
```

```
C:\Users\archi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data
`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```

Out[19]:
```
<AxesSubplot:xlabel='TotalCharges'>
```



There seems to be no outliers in the data set for tenure, totalcharges & monthly charges

In [128…
```
#Dropping columns of no use
df = df.drop(columns=['customerID','Unnamed: 0'])
df_1 = df.copy(deep=True)
df_2 = df.copy(deep=True)
```

LABEL ENCODING OF BINARY VARIABLE & REST CATAGORICAL VARIABLE TO OHE

In [129…
```
# Label encoding for identified columns.

categorical_cols=['Partner','Dependents','PhoneService','PaperlessBilling','Churn']
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df[categorical_cols] = df[categorical_cols].apply(lambda col: le.fit_transform(col))
df['gender'] = df['gender'].map({'Female': 1, 'Male': 0})
```

In [130…
```
df_1['gender'] = df_1['gender'].map({'Female': 1, 'Male': 0})
```

Out[130]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
0	1	0	No	No	2	No	No phone service	DSL	No	No
1	0	0	Yes	Yes	11	Yes	Yes	DSL	Yes	No
2	0	0	No	No	1	Yes	No	DSL	No	No
3	1	0	Yes	No	72	Yes	Yes	Fiber optic	Yes	Yes
4	0	0	No	No	24	Yes	Yes	DSL	Yes	No
...	...	...	...	...	...	...	...	...	...	...
4996	0	1	No	No	24	Yes	Yes	Fiber optic	No	Yes
4997	1	0	No	No	71	Yes	No	DSL	Yes	Yes
4998	1	0	No	No	69	Yes	Yes	DSL	Yes	No
4999	0	0	Yes	No	72	Yes	Yes	No	No internet service	No internet service
5000	0	0	No	No	18	Yes	Yes	Fiber optic	No	No

4994 rows × 20 columns

In [131…
```
df
```

Out[131]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
0	1	0	0	0	2	0	No phone service	DSL	No	No
1	0	0	1	1	11	1	Yes	DSL	Yes	No
2	0	0	0	0	1	1	No	DSL	No	No
3	1	0	1	0	72	1	Yes	Fiber optic	Yes	Yes
4	0	0	0	0	24	1	Yes	DSL	Yes	No
...	...	...	...	...	...	...	...	...	...	...
4996	0	1	0	0	24	1	Yes	Fiber optic	No	Yes
4997	1	0	0	0	71	1	No	DSL	Yes	Yes
4998	1	0	0	0	69	1	Yes	DSL	Yes	No
4999	0	0	1	0	72	1	Yes	No	No internet service	No internet service
5000	0	0	0	0	18	1	Yes	Fiber optic	No	No

4994 rows × 20 columns

In [132…
```
# Checking after label encoding, what is the value of YES & NO.

non_active = len(df[df['Churn']==1])
active = len(df[df['Churn']==0])
per_non_active= round((non_active/(non_active+active))*100,2)
per_active= round((active/(non_active+active))*100,2)

print("Non-Churn Users =1 & Churn Users =0 are ",per_active,"& ", per_non_active,"respectively")
```

```
Non-Churn Users =1 & Churn Users =0 are  73.29 &  26.71 respectively
```

In [133…
```
# One-Hot-Encoding for other categorical columns columns.
OHE_features = ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupp
df = pd.get_dummies(df, columns=OHE_features)
```

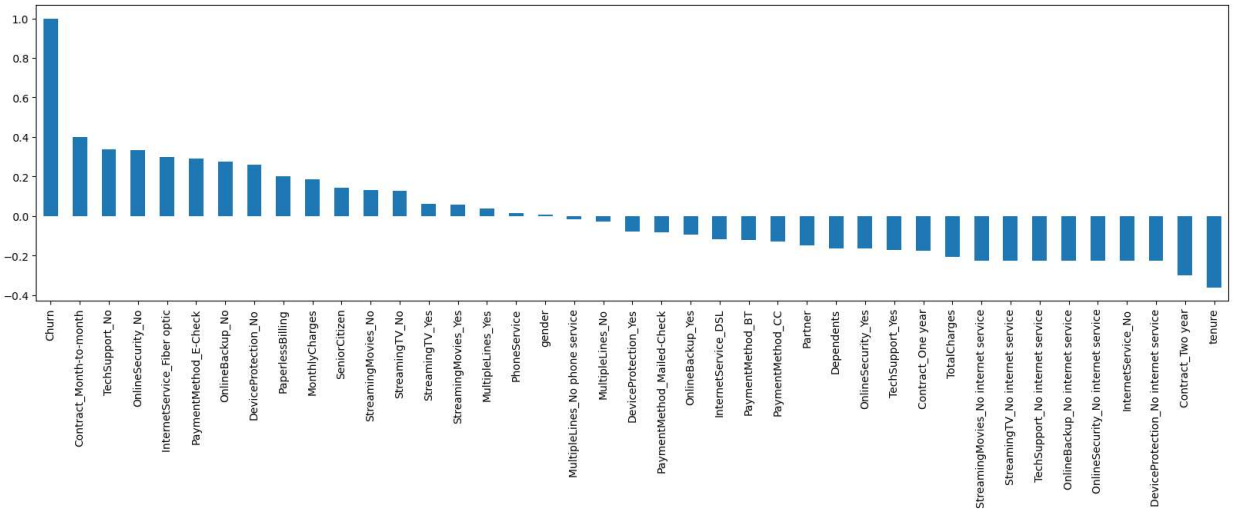
In [ ]:
```
# Checking all new columns for if any column have data less than 4994
df
```

STANDARDIZING the variables with Min Max Scalar & Checking for Correlation

In [134…
```
# Min-Max-Scaling for identified columns.
from sklearn.preprocessing import MinMaxScaler
col_mm = ['tenure', 'MonthlyCharges', 'TotalCharges']
df_col_mm = pd.DataFrame(df, columns=col_mm)
df_remaining_col = df.drop(columns=col_mm)

mms = MinMaxScaler()
rescaled_col = mms.fit_transform(df_col_mm)

df_scaled_col = pd.DataFrame(rescaled_col, columns=col_mm, index=df_remaining_col.index)

df = pd.concat([df_remaining_col, df_scaled_col], axis=1)
```

In [135…
```
# Checking relationship of other features with Churn feature.
plt.figure(figsize=(16,10))
df.corr(method='pearson', min_periods=1)
df.corr(method='pearson')['Churn'].sort_values(ascending=False).plot(kind='bar', figsize=(20,5))
```

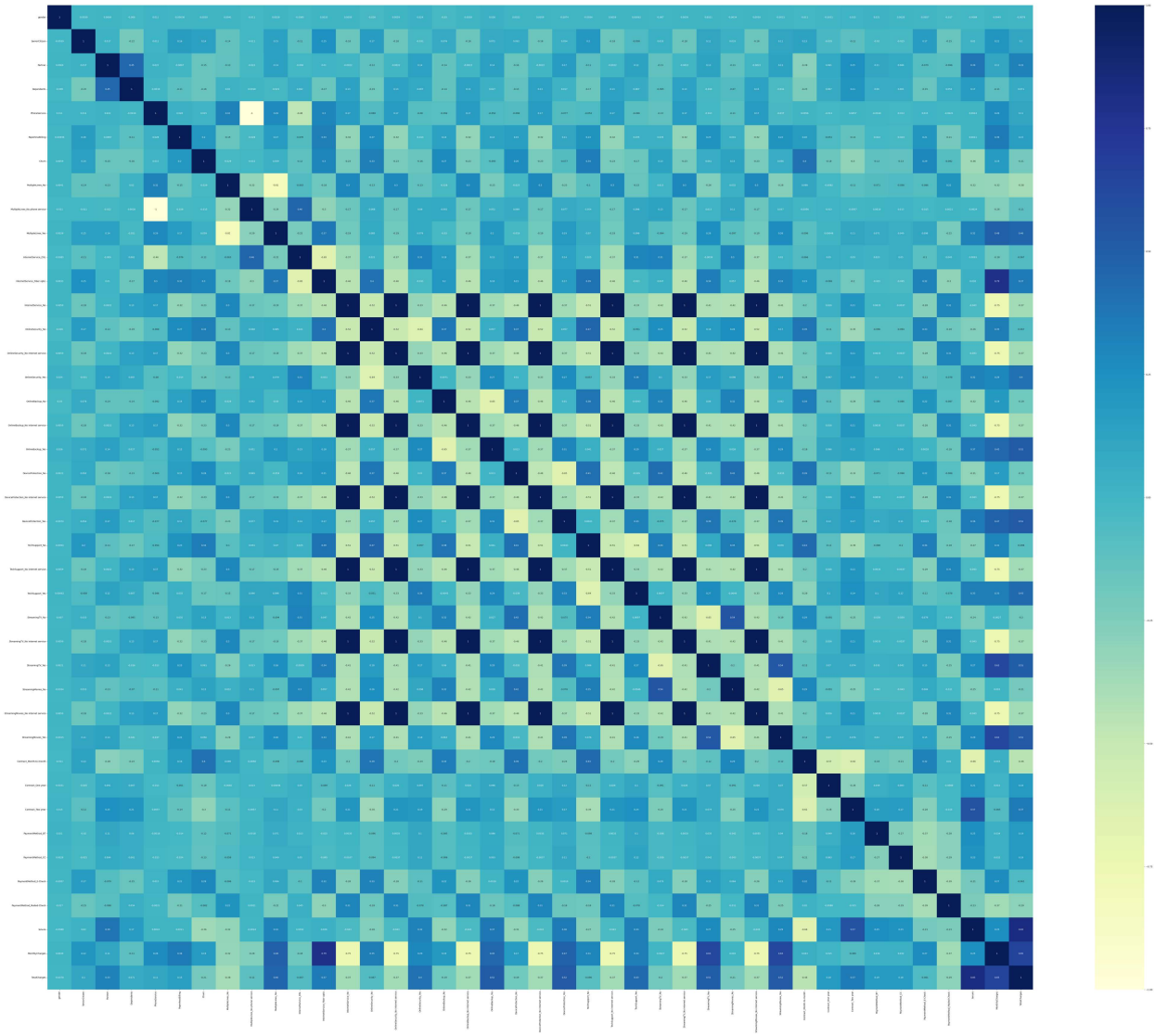
Out[135]: <AxesSubplot:>



Multiple Lines, Streaming TV & Gender seems to be least correlated to Chrun.

In [139…
```
plt.figure(figsize=(100,80))
sns.heatmap(df.corr(),square=True,cmap="YlGnBu",annot=True)
```

Out[139]: <AxesSubplot:>



Working on Questions Now

Q1.

Build a prediction model for predicting churn, using both discriminant analysis and logistic regression. You may consider using the appropriate explanatory variables. If necessary, create additional variables using the existing variables.

I. LDA

In [140…
```
# Splitting the dataset with all features as X except the target feature as y

import sklearn
X_ = df.drop("Churn", axis=1)
y_ = df.Churn

#Doing over sampling as data is highly imbalanced.

oversample  = SMOTE()
X,y=oversample.fit_resample(X_,y_)
print(X.shape, y.shape)
```

```
(7320, 40) (7320,)
```

In [141…
```
model_lda = LinearDiscriminantAnalysis(n_components=1)
lda_data = model_lda.fit(X, y)
lda_data.coef_
```

Out[141]:
```
array([[-0.26306772, -0.12965585, -0.2605492 , -0.50028264, -2.49715616,
         0.20178884, -2.70404491, -4.81551617, -2.40300463, -3.21527425,
        -2.1528381 , -2.61493747, -2.03912704, -2.61493747, -2.75295858,
        -2.3985959 , -2.61493747, -2.78029509, -2.44165587, -2.61493747,
        -2.61854607, -2.12708156, -2.61493747, -2.97295121, -2.25343073,
        -2.61493747, -2.03435828, -2.55343738, -2.61493747, -2.33673217,
        -2.73224605, -3.87949798, -3.53813683, -2.99497279, -3.09412378,
        -2.22417734, -2.94276313, -2.72091417,  0.48736883, -0.09631182]])
```

In [142…
```
lda_df1=pd.DataFrame(model_lda.coef_[0].reshape(-1,1),X.columns,columns=["LD1"])
lda_df2=pd.DataFrame(model_lda.intercept_[0].reshape(-1,1),["Bias"],columns=["LD1"])
lda_df = pd.concat([lda_df2, lda_df1], axis=0)
print ('Coefficients of LDA')
lda_df.sort_values(by=['LD1'], ascending=False)
```

```
Coefficients of LDA
```

Out[142]:

	LD1
Bias	28.354531
MonthlyCharges	0.487369
PaperlessBilling	0.201789
TotalCharges	-0.096312
SeniorCitizen	-0.129656
Partner	-0.260549
gender	-0.263068
Dependents	-0.500283
StreamingTV_Yes	-2.034358
OnlineSecurity_No	-2.039127
TechSupport_No	-2.127082
InternetService_Fiber optic	-2.152838
PaymentMethod_E-Check	-2.224177
StreamingTV_No	-2.253431
StreamingMovies_Yes	-2.336732
OnlineBackup_No	-2.398596
MultipleLines_Yes	-2.403005
DeviceProtection_No	-2.441656
PhoneService	-2.497156
StreamingMovies_No	-2.553437
InternetService_No	-2.614937
StreamingTV_No internet service	-2.614937
StreamingMovies_No internet service	-2.614937
OnlineSecurity_No internet service	-2.614937
TechSupport_No internet service	-2.614937
DeviceProtection_No internet service	-2.614937
OnlineBackup_No internet service	-2.614937
DeviceProtection_Yes	-2.618546
MultipleLines_No	-2.704045
tenure	-2.720914
Contract_Month-to-month	-2.732246
OnlineSecurity_Yes	-2.752959
OnlineBackup_Yes	-2.780295
PaymentMethod_Mailed-Check	-2.942763
TechSupport_Yes	-2.972951
PaymentMethod_BT	-2.994973
PaymentMethod_CC	-3.094124
InternetService_DSL	-3.215274
Contract_Two year	-3.538137
Contract_One year	-3.879498
MultipleLines_No phone service	-4.815516

Explained in Q2.

In [143…

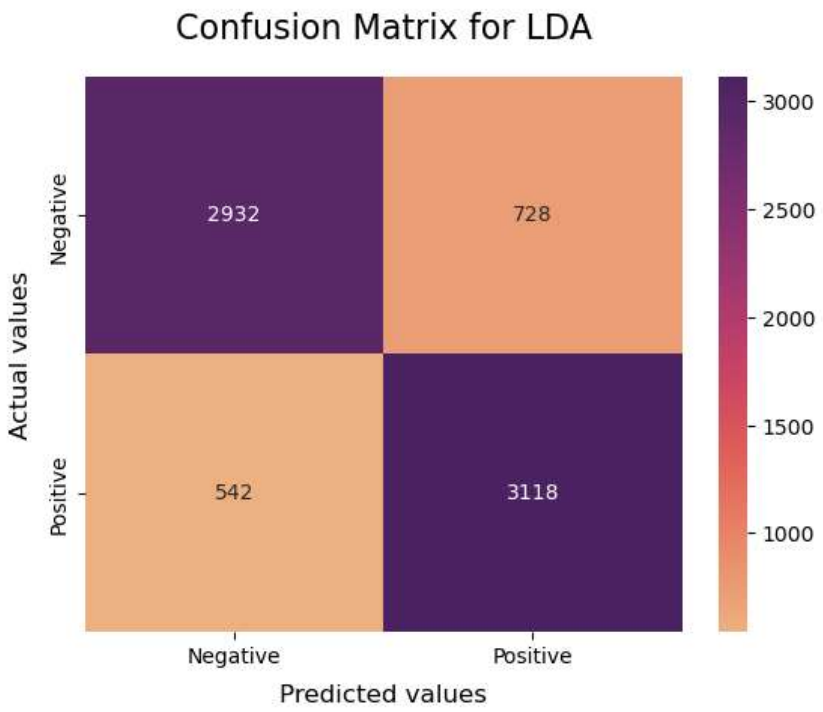
```
## Checking the accuracy of the model

ypred_lda = model_lda.predict(X)
ypred  = pd.DataFrame({"actual":y,"predicted":ypred_lda})

print('Accuracy of LDA model: {:.2f}'.format(accuracy_score(ypred.actual, ypred.predicted)))
```

```
Accuracy of LDA model: 0.83
```

Accuracy of the model is good with 83% accuracy. But to really verify the model we need to have other parameters like recall and precision aswell.

In [144…
```
# Printing classification report
print(classification_report(y, ypred_lda))

# Plotting confusion Matrix
cf_matrix = confusion_matrix(y, ypred_lda)
categories = ['Negative','Positive']
sns.heatmap(cf_matrix, annot =True, cmap = 'flare',fmt = '',
xticklabels = categories, yticklabels = categories)
plt.xlabel("Predicted values", fontdict = {'size':12}, labelpad = 8)
plt.ylabel("Actual values" , fontdict = {'size':12}, labelpad = 8)
plt.title ("Confusion Matrix for LDA", fontdict = {'size':16}, pad = 18)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.84      0.80      0.82      3660
           1       0.81      0.85      0.83      3660

    accuracy                           0.83      7320
   macro avg       0.83      0.83      0.83      7320
weighted avg       0.83      0.83      0.83      7320
```



Non-Churn Users =1 & Churn Users =0

As observed, Overall accuracy is found to be 83 % but Our recall is good for non-churn users & precision is good for churn users.

As our major concern is about the customers who churn, thus precision is important to us.

II. LOGISTIC REGRESSION

In [145…
```
logreg = LogisticRegression()
model_LR = logreg.fit(X, y)
print("Coeff of model are \n",model_LR.coef_)
print("Intercept of model are \n",model_LR.intercept_)
lr_df1=pd.DataFrame(model_LR.coef_[0].reshape(-1,1),X.columns,columns=["Coeff"])
lr_df2=pd.DataFrame(model_LR.intercept_[0].reshape(-1,1),["Bias"],columns=["Coeff"])
```

```
lr_df = pd.concat([lr_df2, lr_df1], axis=0)
print ('\nLogistic Regression Coeff\n')
print(lr_df.sort_values(by=['Coeff'], ascending=False))
```

```
Coeff of model are
 [[-0.20434266 -0.12155001 -0.2368355  -0.48302747  0.22956117  0.23621606
  -2.50131699 -1.49986555 -2.33434054 -2.23448472 -1.73621455 -2.19114016
  -2.38924851 -2.19114016 -2.99399063 -2.2540091  -2.19114016 -2.60405474
  -2.27644531 -2.19114016 -2.44812729 -2.13394701 -2.19114016 -2.84867351
  -1.61863872 -2.19114016 -1.52794311 -1.94280872 -2.19114016 -1.85581352
  -2.39641805 -3.17780996 -3.69690832 -3.28782488 -3.38778204 -2.81627823
  -3.24867244 -4.25729703  1.35746047  2.70542791]]
Intercept of model are
 [22.87417883]

Logistic Regression Coeff

                                      Coeff
Bias                              22.874179
TotalCharges                       2.705428
MonthlyCharges                     1.357460
PaperlessBilling                   0.236216
PhoneService                       0.229561
SeniorCitizen                     -0.121550
gender                            -0.204343
Partner                           -0.236835
Dependents                        -0.483027
MultipleLines_No phone service    -1.499866
StreamingTV_Yes                   -1.527943
StreamingTV_No                    -1.618639
InternetService_Fiber optic       -1.736215
StreamingMovies_Yes               -1.855814
StreamingMovies_No                -1.942809
TechSupport_No                    -2.133947
StreamingTV_No internet service   -2.191140
TechSupport_No internet service   -2.191140
StreamingMovies_No internet service -2.191140
DeviceProtection_No internet service -2.191140
OnlineSecurity_No internet service -2.191140
OnlineBackup_No internet service  -2.191140
InternetService_No                -2.191140
InternetService_DSL               -2.234485
OnlineBackup_No                   -2.254009
DeviceProtection_No               -2.276445
MultipleLines_Yes                 -2.334341
OnlineSecurity_No                 -2.389249
Contract_Month-to-month           -2.396418
DeviceProtection_Yes              -2.448127
MultipleLines_No                  -2.501317
OnlineBackup_Yes                  -2.604055
PaymentMethod_E-Check             -2.816278
TechSupport_Yes                   -2.848674
OnlineSecurity_Yes                -2.993991
Contract_One year                 -3.177810
PaymentMethod_Mailed-Check        -3.248672
PaymentMethod_BT                  -3.287825
PaymentMethod_CC                  -3.387782
Contract_Two year                 -3.696908
tenure                            -4.257297
```

```
C:\Users\archi\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Explained in Q2.

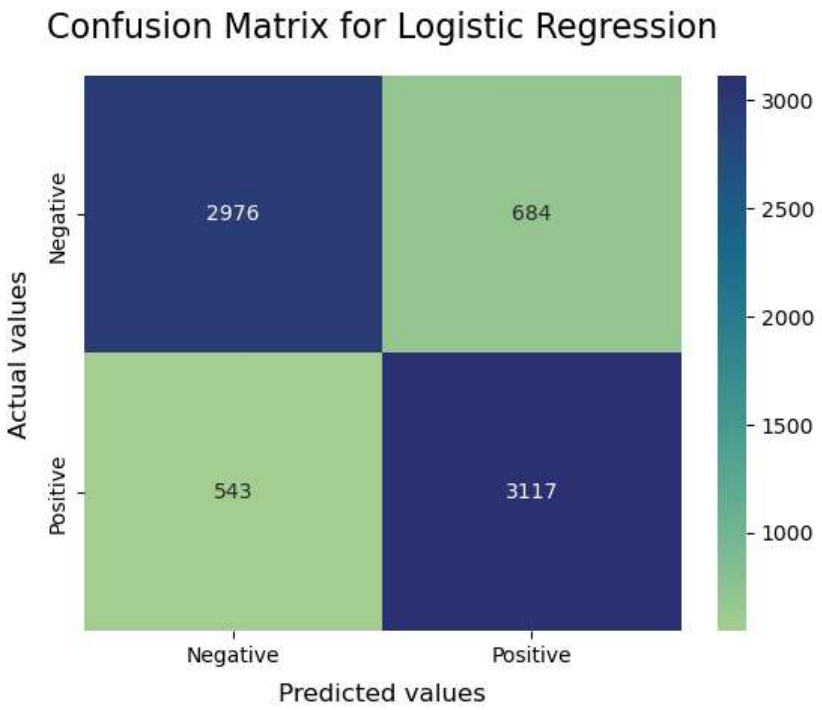
In [146…

```
# Accuracy & other metrics of LR

ypred_lr = logreg.predict(X)
y_pred_lr  = pd.DataFrame({"actual":y,"predicted":ypred_lr})
print('\nAccuracy of Logistic Regression model: {:.2f}\n'.format(accuracy_score(y_pred_lr.actual, y_pred_lr.predict

print('\n\n')
# Evaluation of LR
print(classification_report(y, ypred_lr))

# Compute and plot the Confusion matrix
```

```
cf_matrix = confusion_matrix(y, ypred_lr)
categories = ['Negative','Positive']
sns.heatmap(cf_matrix, annot = True, cmap = 'crest',fmt = '',
xticklabels = categories, yticklabels = categories)
plt.xlabel("Predicted values", fontdict = {'size':12}, labelpad = 8)
plt.ylabel("Actual values" , fontdict = {'size':12}, labelpad = 8)
plt.title ("Confusion Matrix for Logistic Regression", fontdict = {'size':16}, pad = 18)
plt.show()
```

```
Accuracy of Logistic Regression model: 0.83
```

	precision	recall	f1-score	support
0	0.85	0.81	0.83	3660
1	0.82	0.85	0.84	3660
accuracy			0.83	7320
macro avg	0.83	0.83	0.83	7320
weighted avg	0.83	0.83	0.83	7320



Non-Churn Users =1 & Churn Users =0

Over all accuracy is 83 % but Our recall is good for non-churn users & precision is good for churn users.

As our major concern are the customers who churn, thus precision of churn users are also important to us, and this LR model is giving better precision.

Q2.

Comment on the predictive accuracy and the impact of each of the explanatory variables on churn.

In [147…

```
# For explanatory variables taking stats Logistic Regression
import statsmodels.api as stm
log_reg = stm.Logit(y, X).fit()
print(log_reg.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.378637
         Iterations: 35
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Churn   No. Observations:                 7320
Model:                          Logit   Df Residuals:                     7286
Method:                           MLE   Df Model:                           33
Date:                Sun, 13 Nov 2022   Pseudo R-squ.:                  0.4537
Time:                        14:43:40   Log-Likelihood:                -2771.6
converged:                      False   LL-Null:                       -5073.8
Covariance Type:            nonrobust   LLR p-value:                     0.000
=========================================================================================================
                                            coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
gender                                   -0.1727      0.068     -2.556      0.011      -0.305      -0.040
SeniorCitizen                            -0.1156      0.092     -1.253      0.210      -0.296       0.065
Partner                                  -0.2134      0.082     -2.587      0.010      -0.375      -0.052
Dependents                               -0.4678      0.098     -4.784      0.000      -0.659      -0.276
PhoneService                             38.9495   3320.237      0.012      0.991   -6468.595    6546.494
PaperlessBilling                          0.2920      0.077      3.807      0.000       0.142       0.442
MultipleLines_No                        -20.2709   3320.238     -0.006      0.995   -6527.818    6487.277
MultipleLines_No phone service           22.5657        nan        nan        nan         nan         nan
MultipleLines_Yes                       -20.8833   3320.238     -0.006      0.995   -6528.431    6486.664
InternetService_DSL                      -1.4206      0.651     -2.182      0.029      -2.697      -0.145
InternetService_Fiber optic              -4.8251      0.838     -5.756      0.000      -6.468      -3.182
InternetService_No                       -1.6189   1.67e+07  -9.67e-08      1.000   -3.28e+07    3.28e+07
OnlineSecurity_No                        -2.6646      0.311     -8.573      0.000      -3.274      -2.055
OnlineSecurity_No internet service       -1.6189        nan        nan        nan         nan         nan
OnlineSecurity_Yes                       -4.0670      0.748     -5.435      0.000      -5.534      -2.600
OnlineBackup_No                          -2.4460      0.624     -3.921      0.000      -3.669      -1.223
OnlineBackup_No internet service         -1.6189    6.7e+06  -2.41e-07      1.000   -1.31e+07    1.31e+07
OnlineBackup_Yes                         -3.6037      0.357    -10.093      0.000      -4.304      -2.904
DeviceProtection_No                      -2.5455      0.566     -4.500      0.000      -3.654      -1.437
DeviceProtection_No internet service     -1.6189   9.16e+06  -1.77e-07      1.000   -1.79e+07    1.79e+07
DeviceProtection_Yes                     -3.5111      0.582     -6.036      0.000      -4.651      -2.371
TechSupport_No                           -2.5593        nan        nan        nan         nan         nan
TechSupport_No internet service          -1.6189   9.16e+06  -1.77e-07      1.000   -1.79e+07    1.79e+07
TechSupport_Yes                          -4.1210      0.982     -4.196      0.000      -6.046      -2.196
StreamingTV_No                           -1.3039      0.571     -2.284      0.022      -2.423      -0.185
StreamingTV_No internet service          -1.6189   9.16e+06  -1.77e-07      1.000   -1.79e+07    1.79e+07
StreamingTV_Yes                          -2.7794      0.489     -5.687      0.000      -3.737      -1.822
StreamingMovies_No                       -1.7085      0.573     -2.983      0.003      -2.831      -0.586
StreamingMovies_No internet service      -1.6189   1.11e+07  -1.46e-07      1.000   -2.18e+07    2.18e+07
StreamingMovies_Yes                      -3.1928      0.578     -5.527      0.000      -4.325      -2.061
Contract_Month-to-month                  -3.3739      0.538     -6.275      0.000      -4.428      -2.320
Contract_One year                        -4.1916      0.541     -7.746      0.000      -5.252      -3.131
Contract_Two year                        -4.8175      0.556     -8.662      0.000      -5.908      -3.727
PaymentMethod_BT                         -4.5271      0.523     -8.655      0.000      -5.552      -3.502
PaymentMethod_CC                         -4.6281      0.524     -8.829      0.000      -5.656      -3.601
PaymentMethod_E-Check                    -4.0500      0.516     -7.842      0.000      -5.062      -3.038
PaymentMethod_Mailed-Check               -4.5139      0.520     -8.680      0.000      -5.533      -3.495
tenure                                   -5.6522      0.481    -11.745      0.000      -6.595      -4.709
MonthlyCharges                           16.5790        nan        nan        nan         nan         nan
TotalCharges                              4.6153      0.663      6.959      0.000       3.315       5.915
=========================================================================================================
C:\Users\archi\anaconda3\lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retvals
```

Checking top 5 & bottom 5 p-values & inferring features

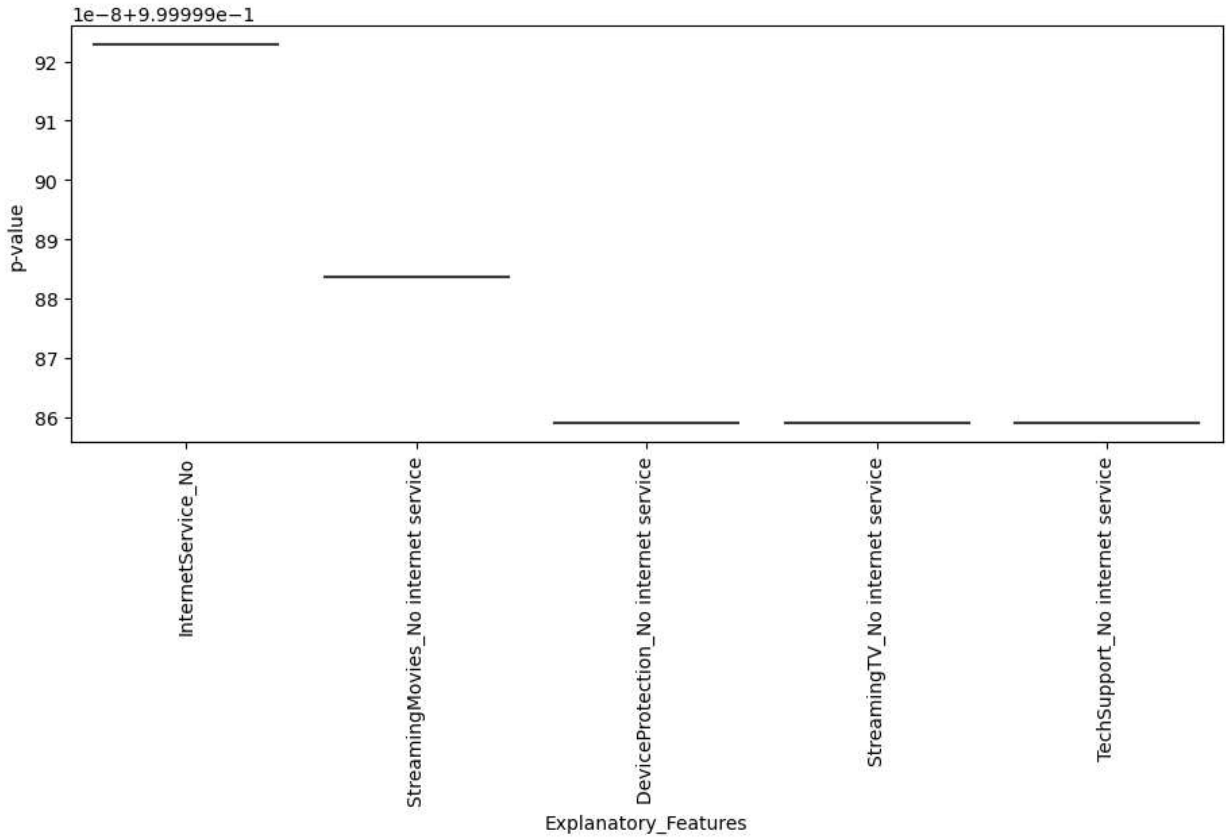
In [148…

```
# Checking feature importance on p-value
log_reg.pvalues
a = pd.DataFrame(log_reg.pvalues,columns=['p-value'])
a = a.reset_index()
a.columns = ['Explanatory_Features','p-value']
a.loc[a['p-value']>=0.05,'Sig/Non_Sig']='Not Significant'
a.loc[a['p-value']<0.05,'Sig/Non_Sig']='Significant'

print('SIGNIFICANT FEATURES \n\n',a.nsmallest(5, ['p-value']))
print('\nINSIGNIFICANT FEATURES \n\n',a.nlargest(5, ['p-value']))
```

```
SIGNIFICANT FEATURES

           Explanatory_Features       p-value  Sig/Non_Sig
37                       tenure  7.509229e-32  Significant
17             OnlineBackup_Yes  5.960994e-24  Significant
34              PaymentMethod_CC  1.060550e-18  Significant
36  PaymentMethod_Mailed-Check  3.948495e-18  Significant
32            Contract_Two year  4.653634e-18  Significant

INSIGNIFICANT FEATURES

                    Explanatory_Features  p-value      Sig/Non_Sig
11                    InternetService_No      1.0  Not Significant
28   StreamingMovies_No internet service      1.0  Not Significant
19  DeviceProtection_No internet service      1.0  Not Significant
25       StreamingTV_No internet service      1.0  Not Significant
22       TechSupport_No internet service      1.0  Not Significant
```

In [149…
```
plt.figure(figsize=(11,4))
k=sns.violinplot( x='Explanatory_Features', y='p-value', data=a.nlargest(5, 'p-value') )
k.set_xticklabels(k.get_xticklabels(), rotation=90)
```

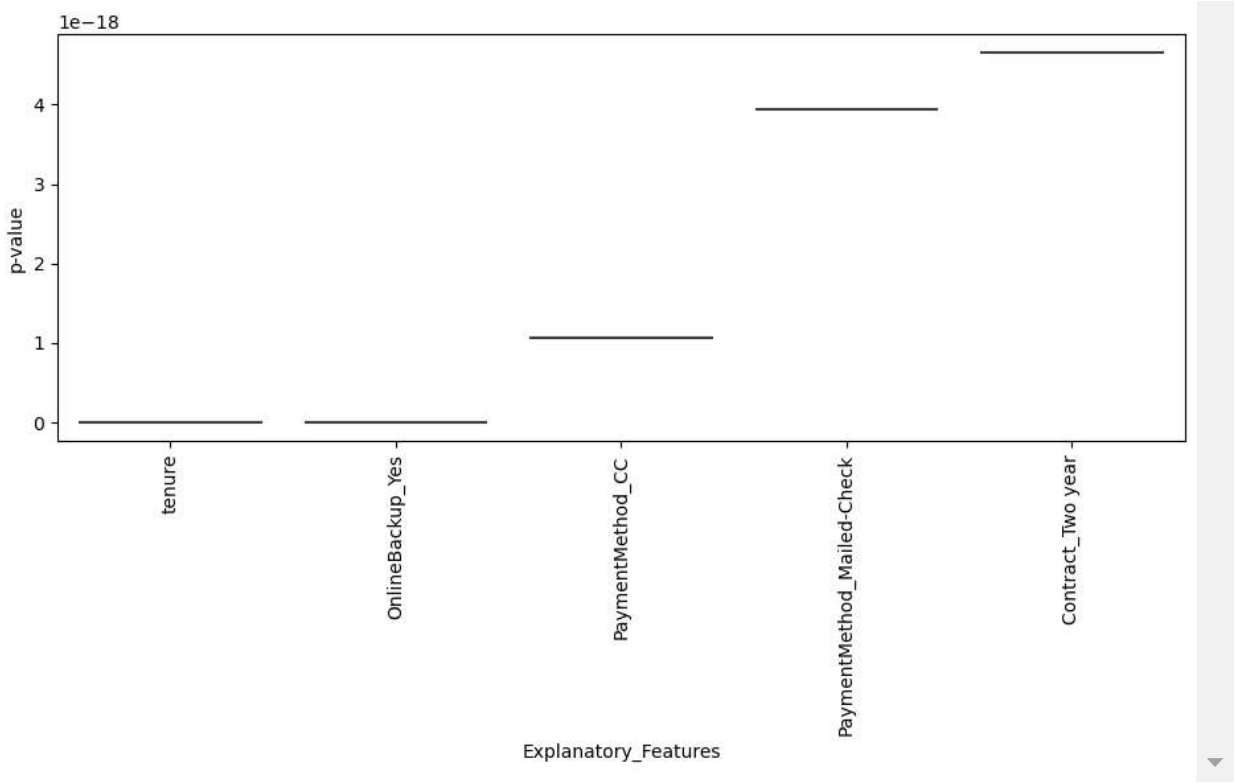
Out[149]:
```
[Text(0, 0, 'InternetService_No'),
 Text(1, 0, 'StreamingMovies_No internet service'),
 Text(2, 0, 'DeviceProtection_No internet service'),
 Text(3, 0, 'StreamingTV_No internet service'),
 Text(4, 0, 'TechSupport_No internet service')]
```



We can see that above 5 features are least significant on our LR model predictions as their p-value is very high. No Internet Service and its sub fields are important for Non-CHURN.

In [150…
```
plt.figure(figsize=(11,4))
k=sns.violinplot( x='Explanatory_Features', y='p-value', data=a.nsmallest(5, 'p-value') )
k.set_xticklabels(k.get_xticklabels(), rotation=90)
```

Out[150]:
```
[Text(0, 0, 'tenure'),
 Text(1, 0, 'OnlineBackup_Yes'),
 Text(2, 0, 'PaymentMethod_CC'),
 Text(3, 0, 'PaymentMethod_Mailed-Check'),
 Text(4, 0, 'Contract_Two year')]
```

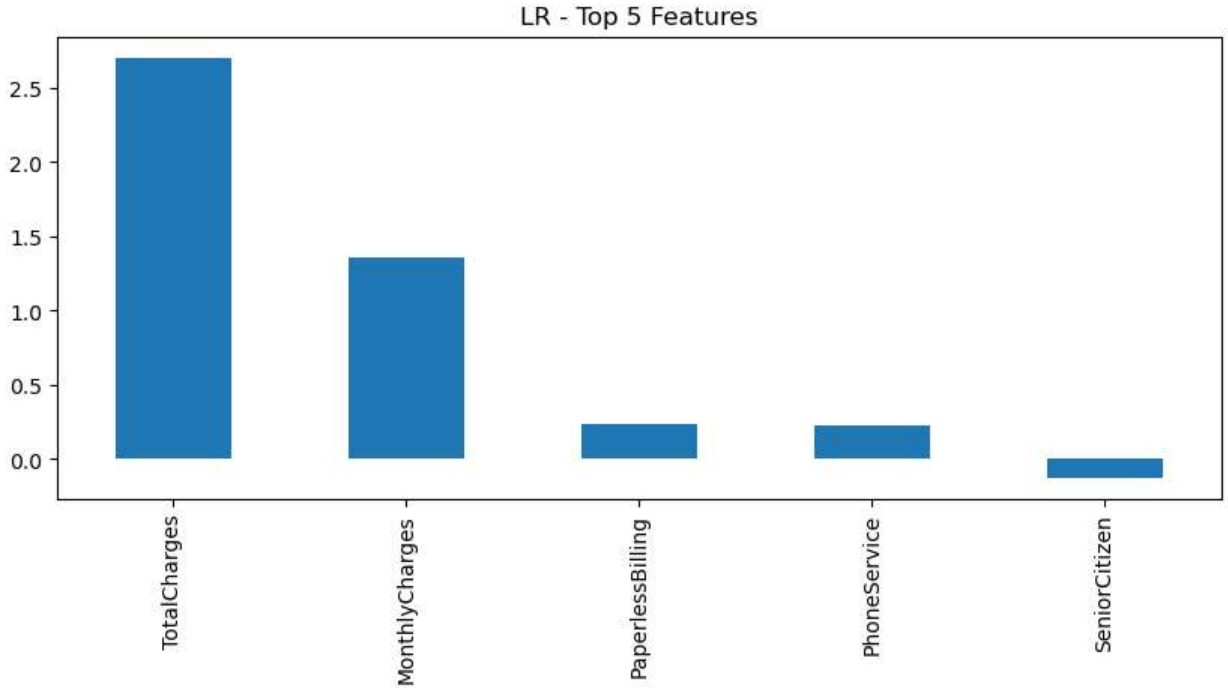


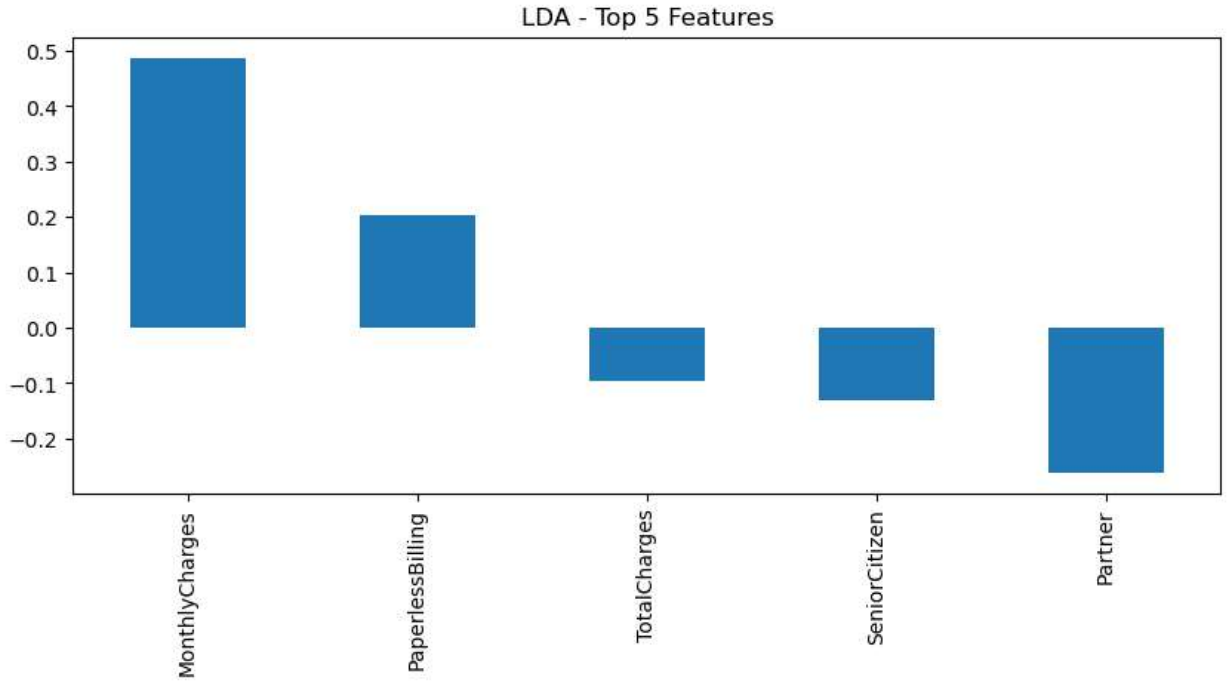
We can see that above 5 features are highly significant on our LR model predictions as their p-value is 0.

However checking with their Coeff as well

In [152…
```
def feature_weights_top10(X_df, classifier, classifier_name):
    weights = pd.Series(classifier.coef_[0], index=X_df.columns.values).sort_values(ascending=False)

    top_weights_selected = weights[:5]
    plt.figure(figsize=(10,4))
    plt.tick_params(labelsize=10)#plt.xlabel(fontsize=10)
    plt.title(f'{classifier_name} - Top 5 Features')
    top_weights_selected.plot(kind="bar")
    return print("")


def feature_weights_bottom10(X_df, classifier, classifier_name):
    weights = pd.Series(classifier.coef_[0], index=X_df.columns.values).sort_values(ascending=False)
    bottom_weights_selected = weights[-5:]
    plt.figure(figsize=(10,4))
    plt.tick_params(labelsize=10)#plt.xlabel(fontsize=10)
    plt.title(f'{classifier_name} - Bottom 5 Features')
    bottom_weights_selected.plot(kind="bar")

    return print("")
```

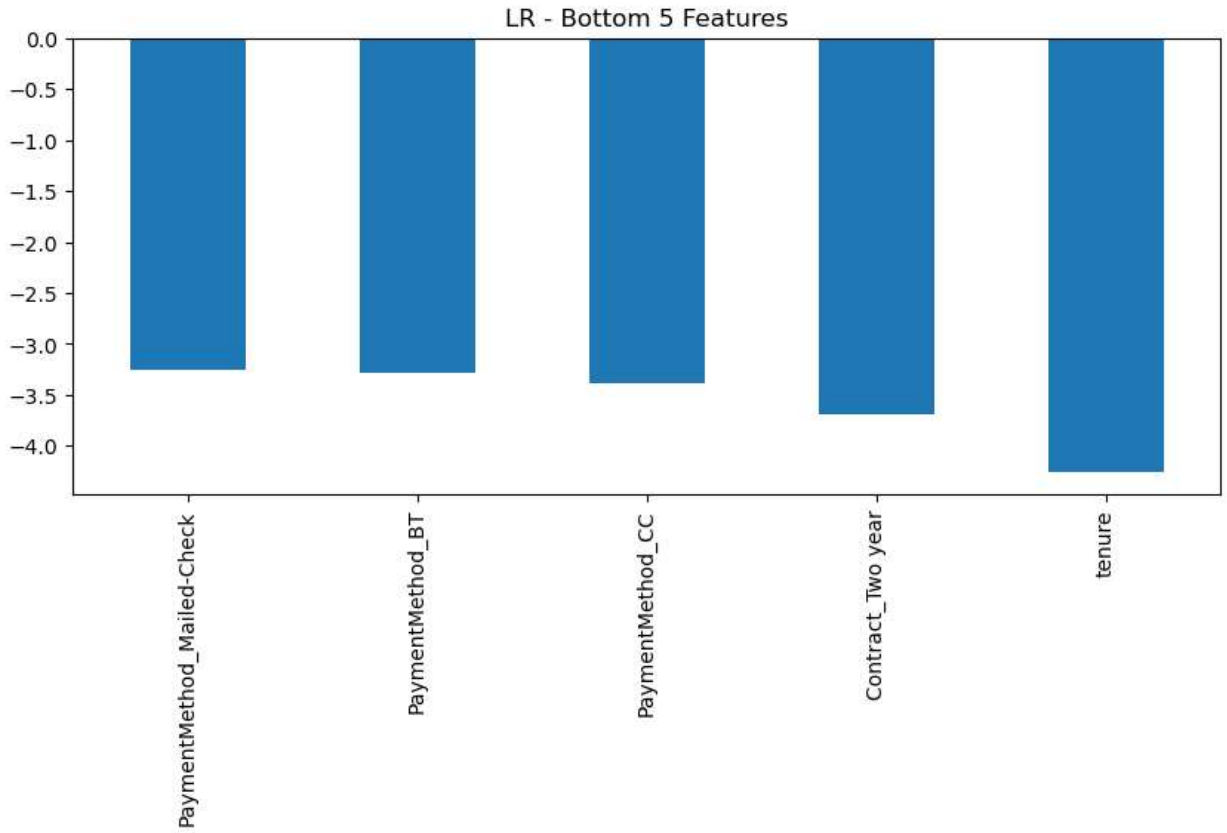
In [153…
```
feature_weights_top10(X,logreg,'LR')
feature_weights_top10(X,model_lda,'LDA')
```

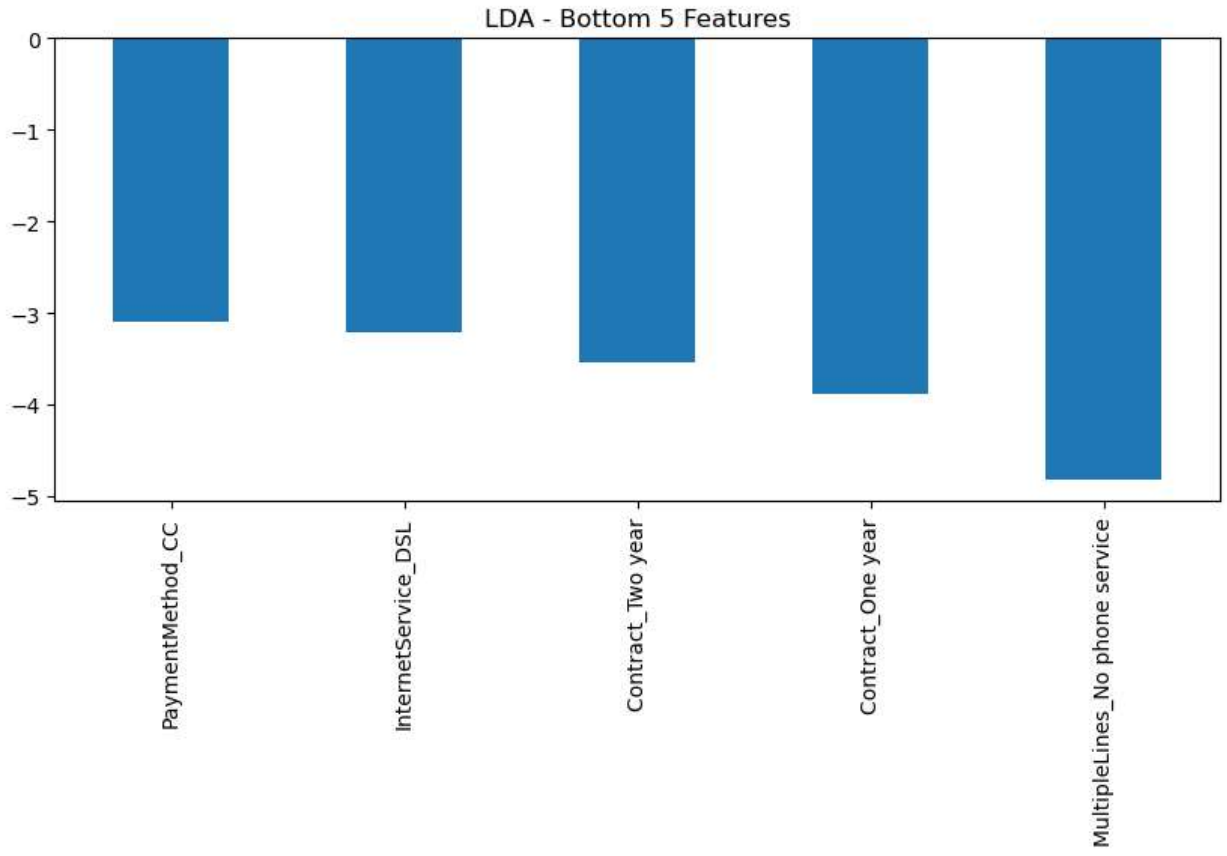




From the above observation, its clear that Top 8 features are same for both the algorithms. Total Charges & Monthly charges are playing major role.

In [154…
```
feature_weights_bottom10(X,logreg,'LR')
feature_weights_bottom10(X,model_lda,'LDA')
```





Least significant features from both models.

Again as TOP-10, in bottom-10 features we can see both models have 8/10 common features. The interesting part here is TENURE of customer is not playing any role for it's CHURN or NON-CHURN

However feature importance can only be calculated by p-value.

We have already checked the confusion matrix parameters in Q1 for both the algorithms. Along with it, we can also conclude for CHURN- monthly charges & for Non-Churn No-internet services are main features for prediction.

Q3.

Divide all the customers into 3 categories namely Low, Medium and High using the variable "TotalCharges". Let us call them "Customer Value Segments". Build prediction models to predict the category/ Value Segment. Comment on the profile of the customers in each category/ Value Segment. Identify appropriate strategies to shift customers from each value segment to the next higher segment.

In [155…
```
df.TotalCharges.describe().T
```

Out[155]:
```
count    4994.000000
mean        0.260237
std         0.261146
min         0.000000
25%         0.042025
50%         0.160411
75%         0.433807
max         1.000000
Name: TotalCharges, dtype: float64
```

As observed, Mean = .26 50% = .16 Max = 1 75% = .43

So here we are considering anything below 0.160411 as low, anything between 0.160411 and .433807 as Medium & above 0.433807 as high.

In [156…
```
df.loc[df['TotalCharges']>0.433807,'Category']='High'
df.loc[(df['TotalCharges']>0.160411) & (df['TotalCharges']<0.433807),'Category']='Medium'
df.loc[df['TotalCharges']<0.160411,'Category']='Low'
df = pd.DataFrame(df)
df
```

Out[156]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	PaperlessBilling	Churn	MultipleLines_No	MultipleLines_No phone service	Multiple
0	1	0	0	0	0	1	1	0	1	
1	0	0	1	1	1	1	0	0	0	
2	0	0	0	0	1	0	1	1	0	
3	1	0	1	0	1	1	0	0	0	
4	0	0	0	0	1	1	1	0	0	
...	...	...	...	...	...	...	...	...	...	
4996	0	1	0	0	1	1	1	0	0	
4997	1	0	0	0	1	0	0	1	0	
4998	1	0	0	0	1	1	0	0	0	
4999	0	0	1	0	1	0	0	0	0	
5000	0	0	0	0	1	1	1	0	0	

4994 rows × 42 columns

In [157…
```
# Checking if any null value in Cateory
```

```
df_na = df[df['Category'].isna()]
df_na
```

Out[157]:

gender	SeniorCitizen	Partner	Dependents	PhoneService	PaperlessBilling	Churn	MultipleLines_No	MultipleLines_No phone service	MultipleLines

0 rows × 42 columns

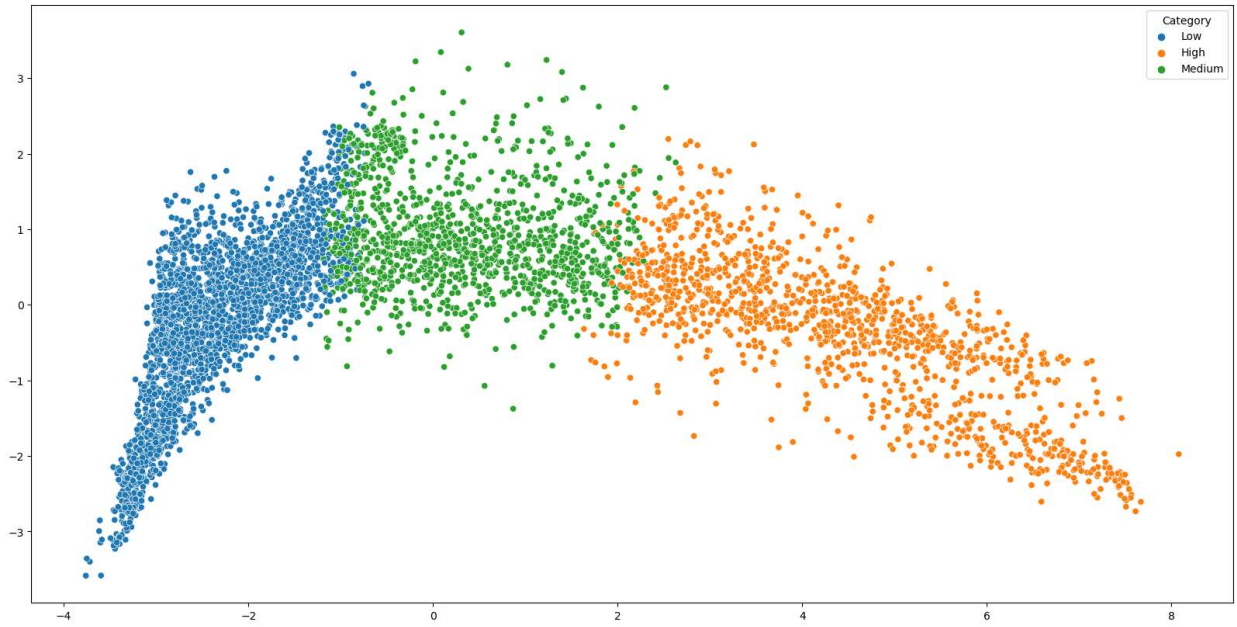
No- Null Values

USING MULTICLASS LDA

In [158…
```
X_mlda = df.drop("Category", axis=1)
y_mlda = df.Category
X_mlda.shape, y_mlda.shape
```

Out[158]: ((4994, 41), (4994,))

In [159…
```
mlda = LinearDiscriminantAnalysis(n_components=2)
m_lda = mlda.fit(X_mlda, y_mlda)
mlda_df = mlda.fit_transform(X_mlda,y_mlda)
# Scatter plot based on multiclass LDA model
plt.figure(figsize=(20,10))
sns.scatterplot(x = mlda_df[:, 0], y = mlda_df[:, 1], hue = df.Category, data=df)
plt.show()
```



The High category values are more spread across & scattered, however Low category is quite dense.

There is a very small overlapping between low & medium users and medium & high users. It cleary shows that the user class is quite seperated for high, medium & low with some exceptions.

In [161…
```
df_mlda = pd.DataFrame(m_lda.means_.T,X_mlda.columns,columns=["Mean.High","Mean.Low","Mean.Medium"])
df_mlda
```

Out[161]:

	Mean.High	Mean.Low	Mean.Medium
gender	0.493995	0.492591	0.486378
SeniorCitizen	0.210568	0.121346	0.193109
Partner	0.711769	0.339207	0.530449
Dependents	0.352282	0.277133	0.288462
PhoneService	0.967174	0.885863	0.854167
PaperlessBilling	0.699760	0.525831	0.625801
Churn	0.140112	0.348418	0.231571
MultipleLines_No	0.226581	0.662795	0.367788
MultipleLines_No phone service	0.032826	0.114137	0.145833
MultipleLines_Yes	0.740592	0.223068	0.486378
InternetService_DSL	0.295436	0.328795	0.432692
InternetService_Fiber optic	0.704564	0.307169	0.457532
InternetService_No	0.000000	0.364037	0.109776
OnlineSecurity_No	0.436349	0.507409	0.553686
OnlineSecurity_No internet service	0.000000	0.364037	0.109776
OnlineSecurity_Yes	0.563651	0.128554	0.336538
OnlineBackup_No	0.285028	0.494193	0.504006
OnlineBackup_No internet service	0.000000	0.364037	0.109776
OnlineBackup_Yes	0.714972	0.141770	0.386218
DeviceProtection_No	0.300240	0.500601	0.490385
DeviceProtection_No internet service	0.000000	0.364037	0.109776
DeviceProtection_Yes	0.699760	0.135362	0.399840
TechSupport_No	0.409127	0.508210	0.554487
TechSupport_No internet service	0.000000	0.364037	0.109776
TechSupport_Yes	0.590873	0.127753	0.335737
StreamingTV_No	0.257006	0.452944	0.458333
StreamingTV_No internet service	0.000000	0.364037	0.109776
StreamingTV_Yes	0.742994	0.183020	0.431891
StreamingMovies_No	0.243395	0.457749	0.448718
StreamingMovies_No internet service	0.000000	0.364037	0.109776
StreamingMovies_Yes	0.756605	0.178214	0.441506
Contract_Month-to-month	0.220176	0.734882	0.540064
Contract_One year	0.313851	0.133761	0.246795
Contract_Two year	0.465973	0.131358	0.213141
PaymentMethod_BT	0.317854	0.136564	0.250801
PaymentMethod_CC	0.313851	0.151782	0.254808
PaymentMethod_E-Check	0.294636	0.359632	0.353365
PaymentMethod_Mailed-Check	0.073659	0.352022	0.141026
tenure	0.847213	0.184717	0.538134
MonthlyCharges	0.748382	0.299765	0.505587
TotalCharges	0.655118	0.055654	0.274368

In [253…

```
print("\x1B[1;4m TOP 5 High Mean value features are\n\x1B[0m", df_mlda.nlargest(5, ['Mean.High']))
print("\n\x1B[1;4m TOP 5 Medium Mean value features are\n\x1B[0m", df_mlda.nlargest(5, ['Mean.Medium']))
print("\n'\x1B[1;4m TOP 5 Low Mean value features are\n\x1B[0m", df_mlda.nlargest(5, ['Mean.Low']))

print("\n\x1B[1;4m Bottom 5 High Mean value features are\n\x1B[0m", df_mlda.nsmallest(5, ['Mean.High']))
print("\n\x1B[1;4m Bottom 5 Medium Mean value features are\n\x1B[0m", df_mlda.nsmallest(5, ['Mean.Medium']))
print("\n\x1B[1;4m Bottom 5 Low Mean value features are\n\x1B[0m", df_mlda.nsmallest(5, ['Mean.Low']))
```

_TOP 5 High Mean value features are

	Mean.High	Mean.Low	Mean.Medium
PhoneService	0.967174	0.885863	0.854167
tenure	0.847213	0.184717	0.538134
StreamingMovies_Yes	0.756605	0.178214	0.441506
MonthlyCharges	0.748382	0.299765	0.505587
StreamingTV_Yes	0.742994	0.183020	0.431891

_TOP 5 Medium Mean value features are

	Mean.High	Mean.Low	Mean.Medium
PhoneService	0.967174	0.885863	0.854167
PaperlessBilling	0.699760	0.525831	0.625801
TechSupport_No	0.409127	0.508210	0.554487
OnlineSecurity_No	0.436349	0.507409	0.553686
Contract_Month-to-month	0.220176	0.734882	0.540064

' TOP 5 Low Mean value features are

	Mean.High	Mean.Low	Mean.Medium
PhoneService	0.967174	0.885863	0.854167
Contract_Month-to-month	0.220176	0.734882	0.540064
MultipleLines_No	0.226581	0.662795	0.367788
PaperlessBilling	0.699760	0.525831	0.625801
TechSupport_No	0.409127	0.508210	0.554487

_Bottom 5 High Mean value features are

	Mean.High	Mean.Low	Mean.Medium
InternetService_No	0.0	0.364037	0.109776
OnlineSecurity_No internet service	0.0	0.364037	0.109776
OnlineBackup_No internet service	0.0	0.364037	0.109776
DeviceProtection_No internet service	0.0	0.364037	0.109776
TechSupport_No internet service	0.0	0.364037	0.109776

_Bottom 5 Medium Mean value features are

	Mean.High	Mean.Low	Mean.Medium
InternetService_No	0.0	0.364037	0.109776
OnlineSecurity_No internet service	0.0	0.364037	0.109776
OnlineBackup_No internet service	0.0	0.364037	0.109776
DeviceProtection_No internet service	0.0	0.364037	0.109776
TechSupport_No internet service	0.0	0.364037	0.109776

_Bottom 5 Low Mean value features are

	Mean.High	Mean.Low	Mean.Medium
TotalCharges	0.655118	0.055654	0.274368
MultipleLines_No phone service	0.032826	0.114137	0.145833
SeniorCitizen	0.210568	0.121346	0.193109
TechSupport_Yes	0.590873	0.127753	0.335737
OnlineSecurity_Yes	0.563651	0.128554	0.336538

The HIGH category customers are using fibre optic internet than normal internet & high prone to streaming TV & Movies. It seems they are tech savvy and as they are having year on year subscription, they can be categorised to be High-class.

The MEDIUM Category customers are prone to paper less billing along with Contract of Month_to_month i.e. monthly recharge. They consume internet and online services in good amount and hence can be categorised as Middle Class.

The LOW Category customers are high users of Phone Service along with Multiple line services. They are using very less of OTT (Streaming TV/Movies) and online content. Their tenure is also least. It seems they are economically weak as they use services on need to need basis.

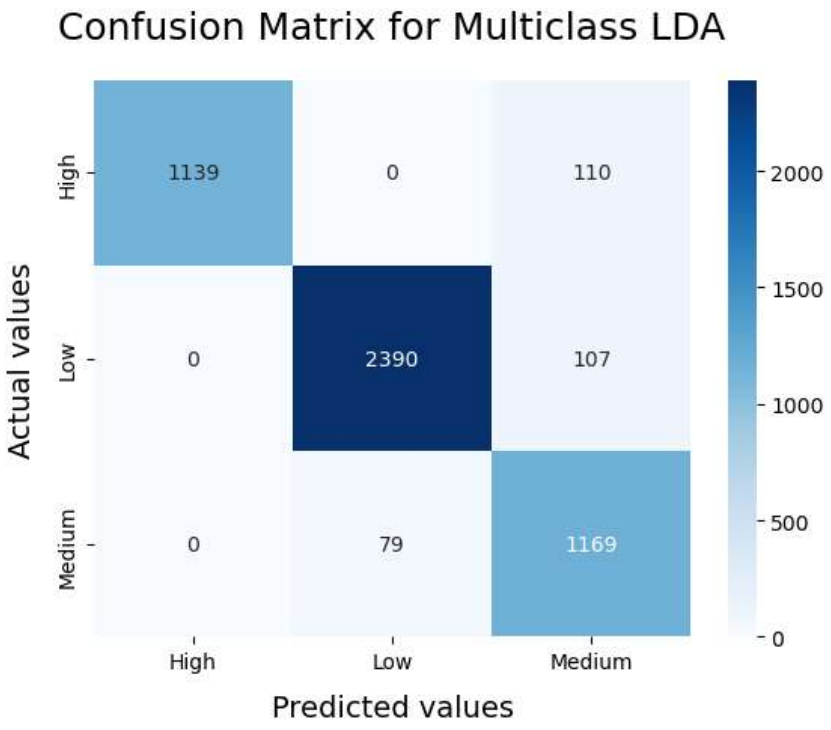
Better Fiber optic services for medium customers can make them rise to HIGH Category customers. Also for LOW category customers we need to provide more Multiple Line phone services and better resolution at Techsupport to move them up to Medium Category along with affordable and cheaper plan subscriptions.

In [163…

```
# Metrics of Multiclass Model

ypred_mlda = mlda.predict(X_mlda)
```

```
ypred_mlda1  = pd.DataFrame({"actual":y_mlda,
                             "predicted":ypred_mlda})

print('Accuracy of Multiclass LDA model: {:.2f}\n'.format(accuracy_score(ypred_mlda1.actual, ypred_mlda1.predicte
print('\n',classification_report(y_mlda, ypred_mlda))

# Compute and plot the Confusion matrix
cf_matrix = confusion_matrix(y_mlda, ypred_mlda)
cm_df = pd.DataFrame(cf_matrix,
                     index = ['High','Low','Medium'],
                     columns = ['High','Low','Medium'])
sns.heatmap(cm_df, annot = True, cmap = 'Blues',fmt = '')
plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)
plt.title ("Confusion Matrix for Multiclass LDA", fontdict = {'size':18}, pad = 20)
plt.show()
```

```
Accuracy of Multiclass LDA model: 0.94

              precision    recall  f1-score   support

        High       1.00      0.91      0.95      1249
         Low       0.97      0.96      0.96      2497
      Medium       0.84      0.94      0.89      1248

    accuracy                           0.94      4994
   macro avg       0.94      0.94      0.93      4994
weighted avg       0.94      0.94      0.94      4994
```



The model is very good as it is having very high F1 Score for all 3 categories.

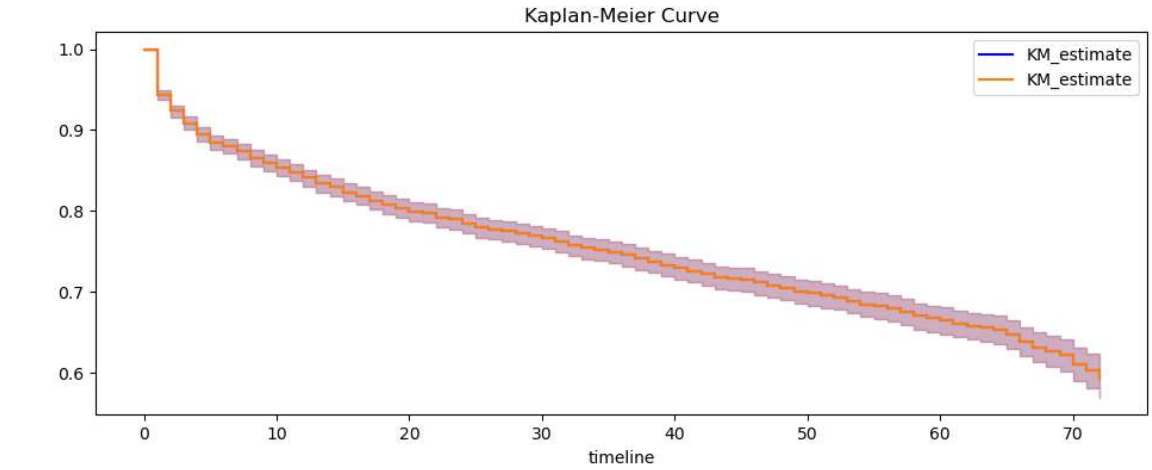
Q4.

Create an overall survival curve using the Tenure variable. Use Kaplan-Meier method.

In [173…

```
from lifelines.statistics import (logrank_test,
                                  pairwise_logrank_test,
                                  multivariate_logrank_test,
                                  survival_difference_at_fixed_point_in_time_test)
```

In [184…
```
T = df_1['tenure']
E= df_1['Churn']=="Yes"

kmf = KaplanMeierFitter()
kmf.fit(T, event_observed=E)
plt.figure(figsize=(10,5))
kmf.plot(color='blue')
kmf.plot(at_risk_counts=True)
plt.title('Kaplan-Meier Curve');
```



KM_estimate								
At risk	4994	3587	2924	2402	1933	1477	995	376
Censored	0	710	1160	1567	1931	2311	2731	3293
Events	0	697	910	1025	1130	1206	1268	1325

The survival analysis for 72 months is shown above.

In the begining of the tenure all 4994 customers are at equivalent risk.

When our tenure is 10 months of using services, we can find that approx 697/4994 have been churned & there are 710 customers who are not included in the survival analysis calculation.

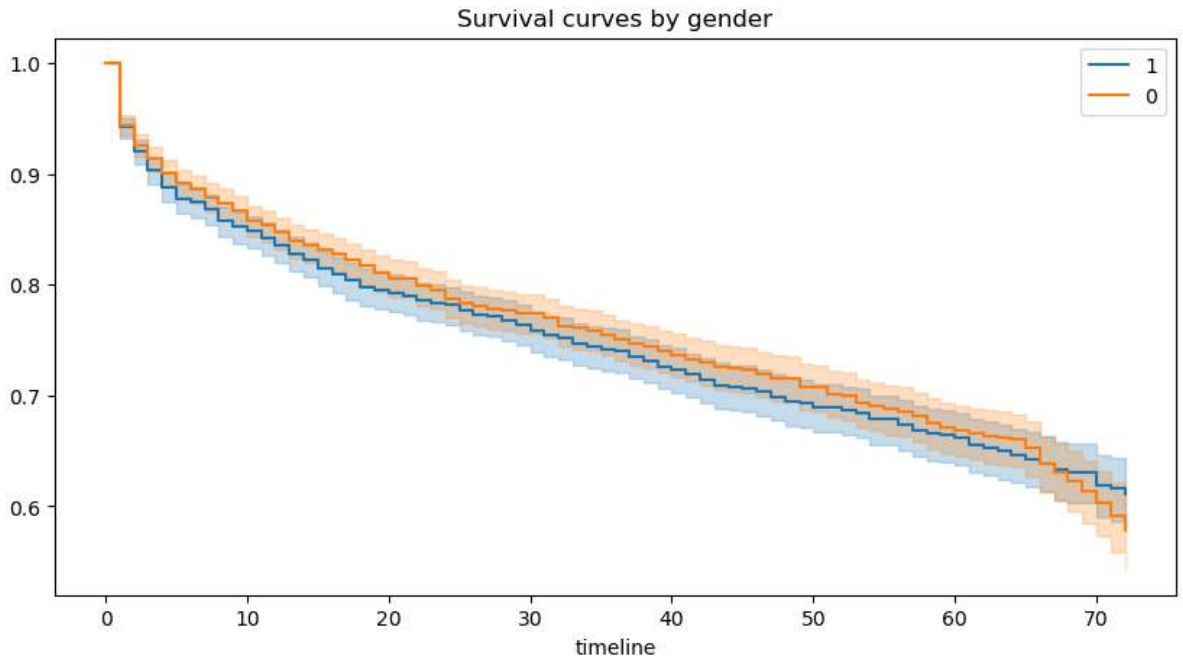
In first 10 months we are loosing i.e., Churn users are approx 50% i.e. 697/1325.

Churn also increases post 5 years of service.

Q5.

Create separate survival curves for different categories of customers (for example, Gender). Comment on the differences in these survival curves.

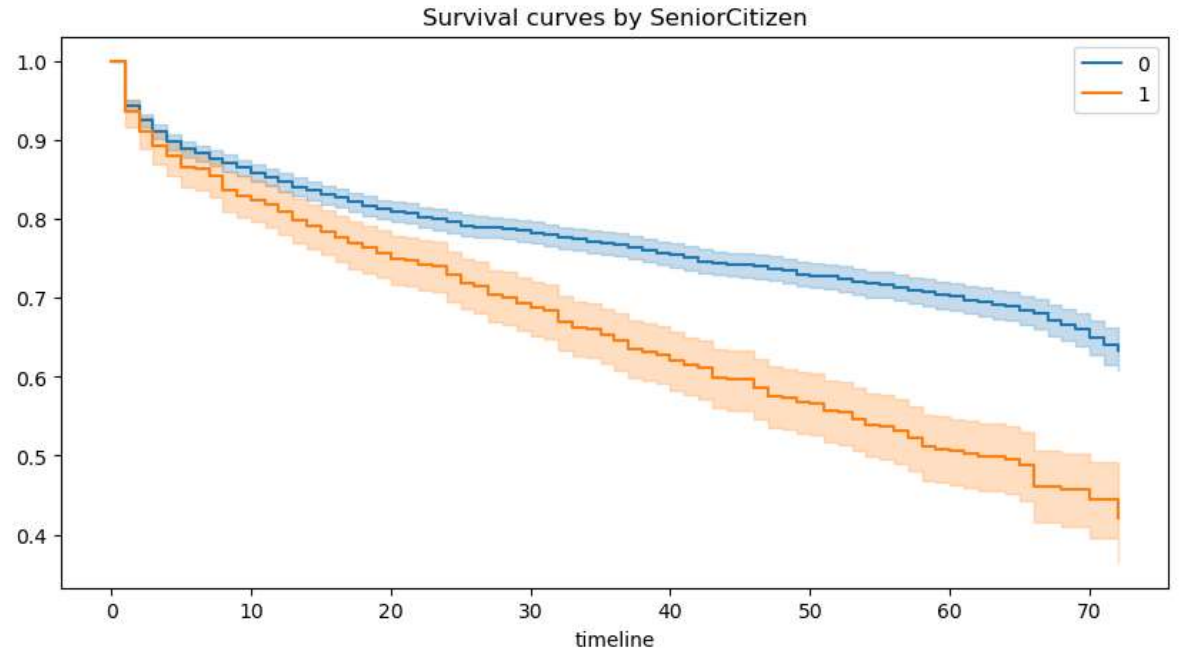
In [185…
```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)
for gender in df_1['gender'].unique():

    flag = df_1['gender'] == gender

    kmf.fit(T[flag], event_observed=E[flag], label=gender)
    kmf.plot(ax=ax)

plt.title("Survival curves by gender");
```



There is high churn rate for males as compared to females in our complete survival tenure except post 65 months.

In [197…

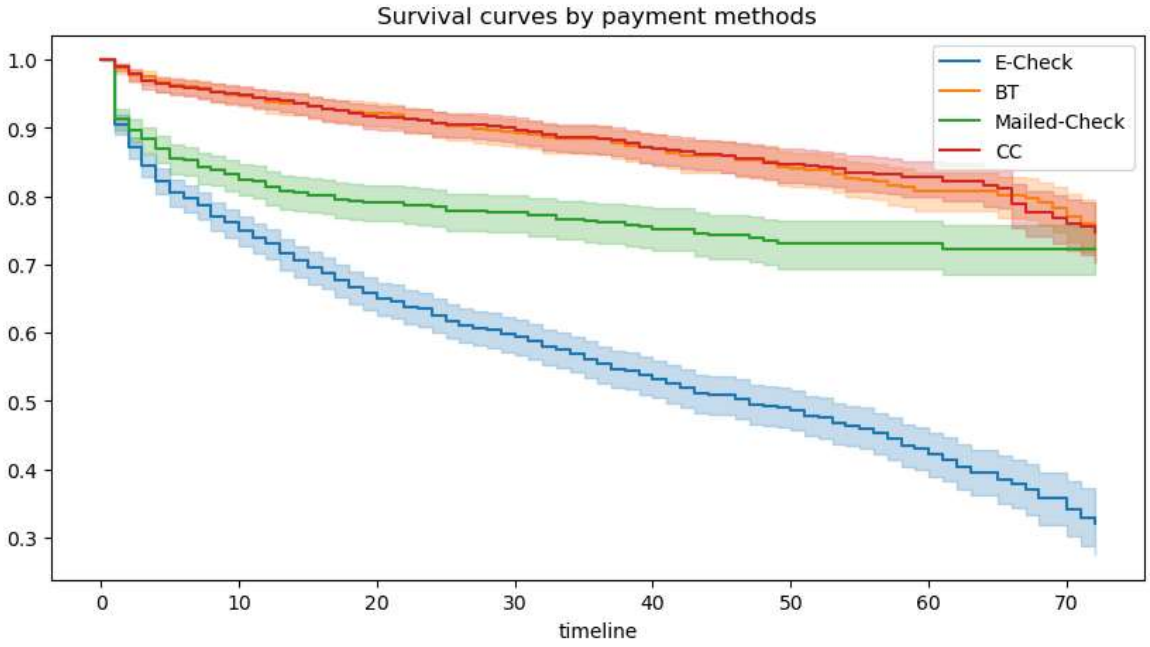
```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)

kmf = KaplanMeierFitter()

for seniorcitiz in df_1['SeniorCitizen'].unique():

    flag = df_1['SeniorCitizen'] == seniorcitiz

    kmf.fit(T[flag], event_observed=E[flag], label=seniorcitiz)
    kmf.plot(ax=ax)

plt.title("Survival curves by SeniorCitizen");
```



Non-Senior citizens are prominent in churning during complete tenure with significantly higher count. The major reason is that the count of senior citizens in our

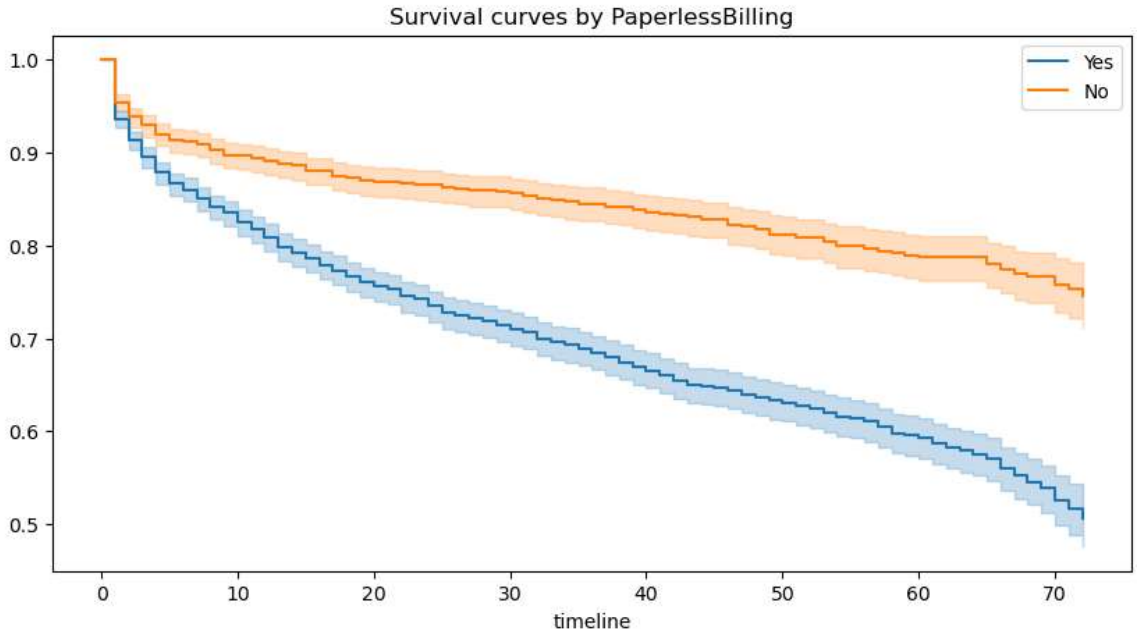
dataset is approx 16%.

In [198…
```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)

kmf = KaplanMeierFitter()

for paymentmethod in df_1['PaymentMethod'].unique():

    flag = df_1['PaymentMethod'] == paymentmethod

    kmf.fit(T[flag], event_observed=E[flag], label=paymentmethod)
    kmf.plot(ax=ax)

plt.title("Survival curves by payment methods");
```



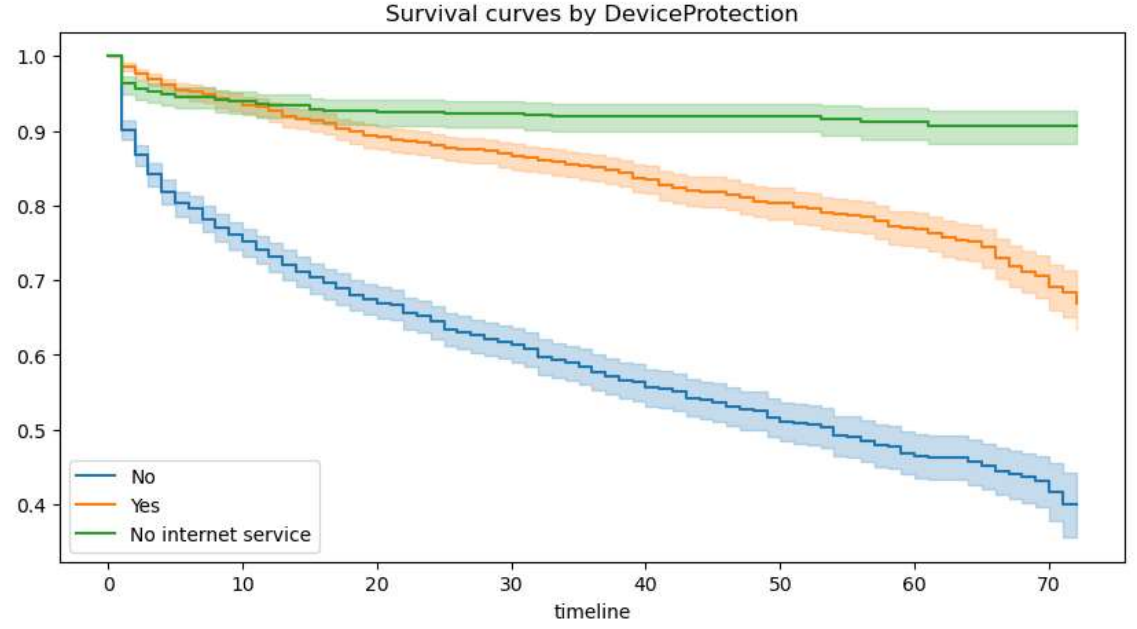
The churn of customers over the period is same for customers paying with Balance Transfer & Credit Card. However customers paying with Mailed-Check are also equally churning in oldest months.

In [199…
```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)

kmf = KaplanMeierFitter()

for plb in df_1['PaperlessBilling'].unique():

    flag = df_1['PaperlessBilling'] == plb

    kmf.fit(T[flag], event_observed=E[flag], label=plb)
    kmf.plot(ax=ax)

plt.title("Survival curves by PaperlessBilling");
```



The churn of customers over the period is high for non-paper less users.

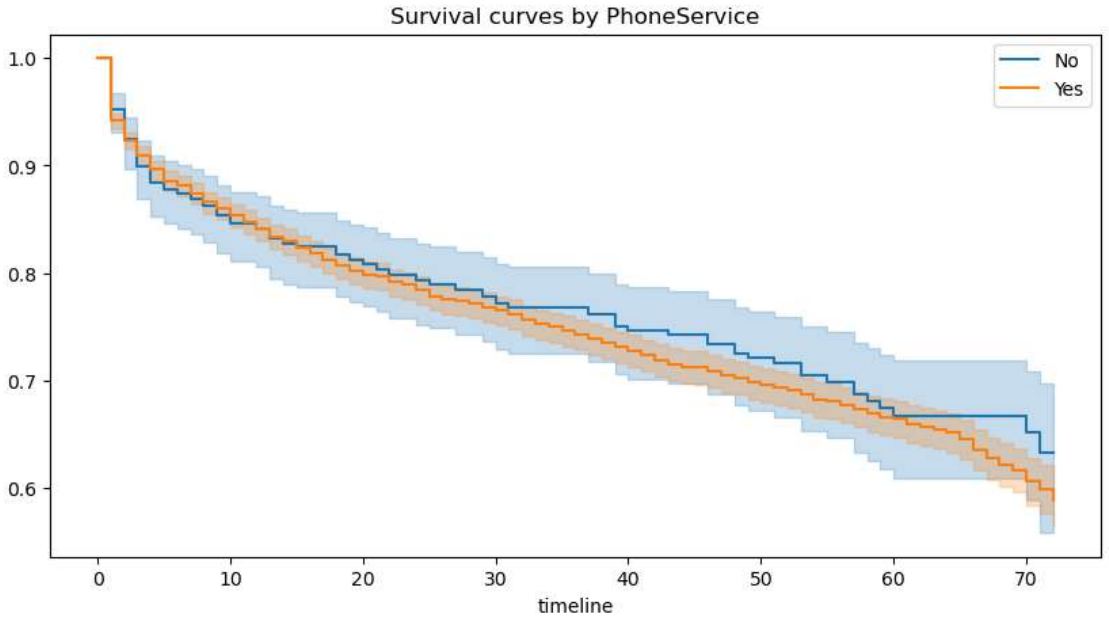
In [200…
```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)

kmf = KaplanMeierFitter()

for DP in df_1['DeviceProtection'].unique():

    flag = df_1['DeviceProtection'] == DP

    kmf.fit(T[flag], event_observed=E[flag], label=DP)
    kmf.plot(ax=ax)

plt.title("Survival curves by DeviceProtection");
```



Customers who are not using Internet Services device protection are constantly churing at same rate.

In [209…
```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)
```

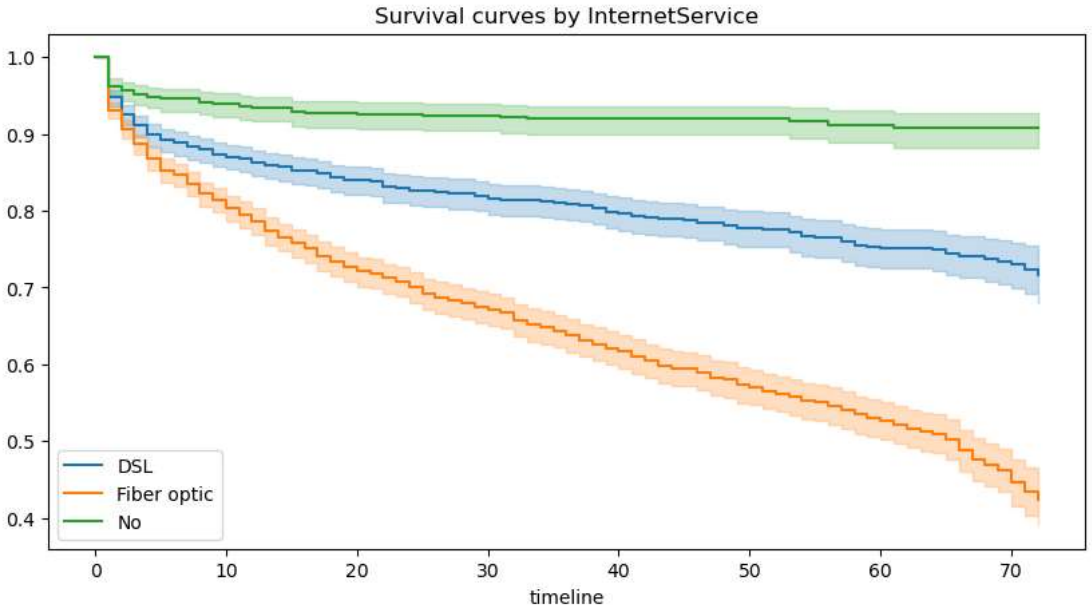
```
kmf = KaplanMeierFitter()

for PS in df_1['PhoneService'].unique():

    flag = df_1['PhoneService'] == PS

    kmf.fit(T[flag], event_observed=E[flag], label=PS)
    kmf.plot(ax=ax)

plt.title("Survival curves by PhoneService");
```



The customers with phone services no are churning more since 18 months of service.

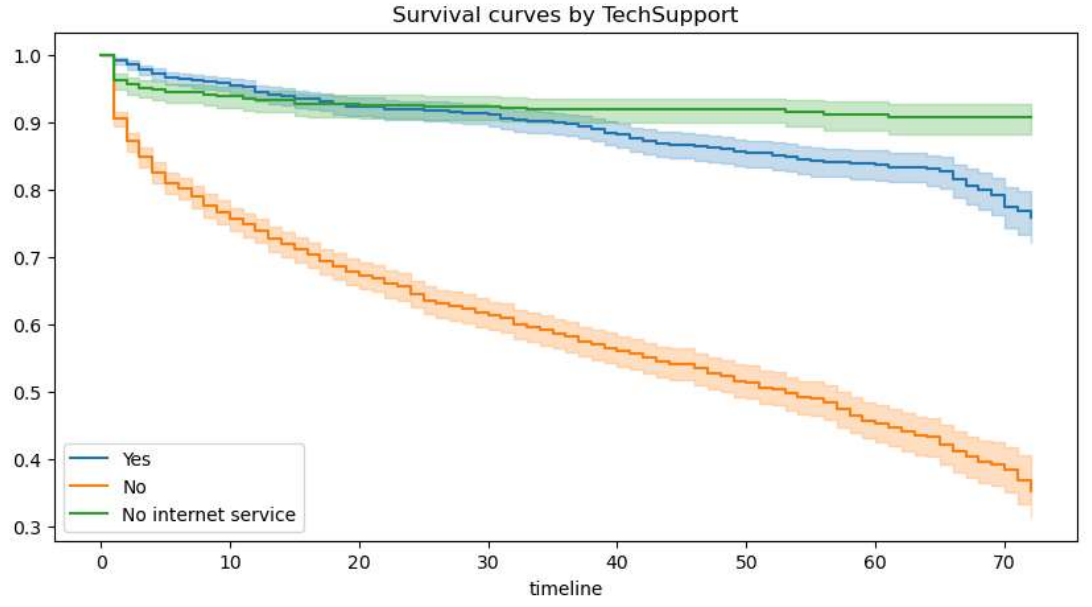
In [208…

```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)

kmf = KaplanMeierFitter()

for IS in df_1['InternetService'].unique():

    flag = df_1['InternetService'] == IS

    kmf.fit(T[flag], event_observed=E[flag], label=IS)
    kmf.plot(ax=ax)

plt.title("Survival curves by InternetService");
```



The customers No Internet services are churning at constant rate.

In [213…
```
plt.figure(figsize=(10,5))
ax = plt.subplot(111)

kmf = KaplanMeierFitter()

for TS in df_1['TechSupport'].unique():

    flag = df_1['TechSupport'] == TS

    kmf.fit(T[flag], event_observed=E[flag], label=TS)
    kmf.plot(ax=ax)

plt.title("Survival curves by TechSupport");
```



The customers who are using either techsupport via internet services or direct tech support are not satisfied and are churning at constant rate. Except there is some fall in churn for customers using direct tech support.

In [206…
```
results = multivariate_logrank_test(df_1['tenure'], df_1['PaymentMethod'], df_1['Churn'])
results.print_summary()
```

t 0	-1
null_distribution	chi squared
degrees_of_freedom	3
test_name	multivariate_logrank_test

	test_statistic	p	-log2(p)
0	778.84	<0.005	557.34

The p-value for Tenure, Payment method & Churn combined is very less, thus can conclude being major contributor for analysis.

Q6.

Build Cox's Hazard model using appropriate explanatory variables. Comment on the coefficients of the model.

In [274…
```
from lifelines import CoxPHFitter
df_3 = df_1[['Partner', 'tenure', 'PhoneService', 'PaperlessBilling','Churn', 'Dependents']]
```

Out[274]:

	Partner	Dependents	Churn	PhoneService	PaperlessBilling	tenure
0	No	No	1	No	Yes	2
1	Yes	Yes	0	Yes	Yes	11
2	No	No	1	Yes	No	1
3	Yes	No	0	Yes	Yes	72
4	No	No	1	Yes	Yes	24
...	...	...	...	...	...	...
4996	No	No	1	Yes	Yes	24
4997	No	No	0	Yes	No	71
4998	No	No	0	Yes	Yes	69
4999	Yes	No	0	Yes	No	72
5000	No	No	1	Yes	Yes	18

4994 rows × 6 columns

In [275…
```
df_3['Partner'] = df_3['Partner'].map({'Yes': 1, 'No': 0})
df_3['Dependents'] = df_3['Dependents'].map({'Yes': 1, 'No': 0})
df_3['PhoneService'] = df_3['PhoneService'].map({'Yes': 1, 'No': 0})
df_3['PaperlessBilling'] = df_3['PaperlessBilling'].map({'Yes': 1, 'No': 0})
```

Out[275]:

	Partner	Dependents	Churn	PhoneService	PaperlessBilling	tenure
0	0	0	1	0	1	2
1	1	1	0	1	1	11
2	0	0	1	1	0	1
3	1	0	0	1	1	72
4	0	0	1	1	1	24
...	...	...	...	...	...	...
4996	0	0	1	1	1	24
4997	0	0	0	1	0	71
4998	0	0	0	1	1	69
4999	1	0	0	1	0	72
5000	0	0	1	1	1	18

4994 rows × 6 columns

In [276…
```
df_4=df_3.query("Churn == 1")
```

In [277…
```
# Using COXPH Model
cph = CoxPHFitter()

# Fitting the data to model
cph.fit(df_4, 'tenure', event_col='Churn')

# Metrics summary
cph.print_summary()
```

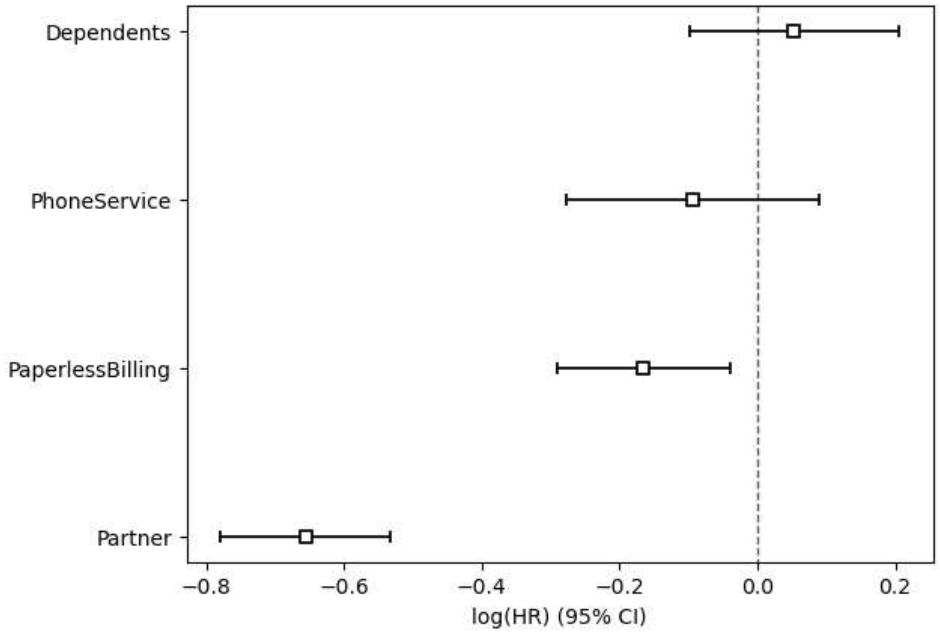
model	lifelines.CoxPHFitter
duration col	'tenure'
event col	'Churn'
baseline estimation	breslow
number of observations	1334
number of events observed	1334
partial log-likelihood	-8204.03
time fit was run	2022-11-13 14:30:19 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	cmp to	z	p	-log2(p)
Partner	-0.66	0.52	0.06	-0.78	-0.53	0.46	0.59	0.00	-10.45	<0.005	82.47
Dependents	0.05	1.05	0.08	-0.10	0.21	0.91	1.23	0.00	0.68	0.49	1.02
PhoneService	-0.09	0.91	0.09	-0.28	0.09	0.76	1.09	0.00	-1.00	0.32	1.66
PaperlessBilling	-0.17	0.85	0.06	-0.29	-0.04	0.75	0.96	0.00	-2.57	0.01	6.61

Concordance	0.61
Partial AIC	16416.07
log-likelihood ratio test	131.73 on 4 df
-log2(p) of ll-ratio test	88.96

In [278…
```
cph.plot()
```

Out[278]:
```
<AxesSubplot:xlabel='log(HR) (95% CI)'>
```



For all 4 features we can see that our coeff is having -ve lower 95% CI & for upper 95% CI only dependent shows a bit higher value. We can conclude that all features are some what providing negative effect to CHURN. However we can also say Dependents are providing some +ve effect on churn. Majorly Partner, Paperless billing is providing complete nrgative effect on Churn & Phoneservices are majorly non-effecting churn.

As we always look for exp(coef) to find whether a feature is important or not in our Target i.e. CHURN. exp(coef) less than 1 means that that feature will not effect CHURN and anything greater than that will effect. In our features only Dependents feature is effecting CHURN.