Comparative Analysis of Machine Learning Algorithms: Visualizing Accuracy Across Models

PROJECT DESCRIPTION

This mini project aims to provide a comprehensive comparison of various machine learning algorithms by visualizing their accuracy on a single graph. The algorithms considered include logistic regression, linear regression, random forest, and decision tree.

DATASET DESCRIPTION

This dataset contains information about customers who have either stayed with or left a company The goal is to predict whether a customer will churn based on their demographic, account, and service usage information.

DATASET FIELDS DESCRIPTION:

CustomerID: Unique identifier for each customer. Gender: Customer's gender (Male/Female). SeniorCitizen: Indicates if the customer is a senior citizen (1 for Yes, 0 for No). Partner: Whether the customer has a partner (Yes/No). Dependents: Whether the customer has dependents (Yes/No). Tenure: Number of months the customer has stayed with the company. PhoneService: Whether the customer has phone service (Yes/No). MultipleLines: Whether the customer has multiple phone lines (Yes/No). InternetService: Type of internet service the customer has (DSL/Fiber optic/No). OnlineSecurity: Whether the customer has online security service (Yes/No/No internet service). OnlineBackup: Whether the customer has online backup service (Yes/No/No internet service). DeviceProtection: Whether the customer has device protection service (Yes/No/No internet service). TechSupport: Whether the customer has tech support service (Yes/No/No internet service). StreamingTV: Whether the customer has streaming TV service (Yes/No/No internet service). StreamingMovies: Whether the customer has streaming movies service (Yes/No/No internet service). Contract: The contract term of the customer (Month-to-month/One year/Two year). PaperlessBilling: Whether the customer uses paperless billing (Yes/No). PaymentMethod: The customer's payment method (Electronic check/Mailed check/Bank transfer (automatic)/Credit card (automatic)). MonthlyCharges: The amount charged to the customer monthly. TotalCharges: The total amount charged to the customer over their tenure. Churn: Whether the customer has left the company (Yes/No).

Problem Statement

create a project that aims to leverage machine learning techniques to predict customer churn and compare the accuracy of various algorithms.

Logistic regression

import libraries *Pandas-data manipulation and analysis *Numpy-mathematical calculations *seaborn-visualization *matplotlib-visualization

In [117]:
```
#import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

import the dataset

In [118]:
```
df=pd.read_csv("customer_churn.csv")
```

In [119]:
```
df
```

Out[119]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	7590-VHVEG	Female	0	Yes	No	1	No	Nc
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	Nc
4	9237-HQITU	Female	0	No	No	2	Yes	
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	Nc
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	

7043 rows × 21 columns

```
Exploratory Data Analysis
Here we explore the data and do analysis.
step1 in EDA:CHECK THE DATA TYPE
             ->here in the dataset total charges which is in numerical datatype but shown as object type so first we need to change it.
step2 in EDA:DATA CLEANSING
             ->Null values
             ->Duplicates
             ->Outliers
step3:LABLE ENCODING:
```

```
              ->object converted to numeric
              ->Label encoding assigns a unique integer to each
category
              1.One-Hot Encoding:
step4:FEATURE SELECTION:
              1.FIND CORRELATION
              2.VIF
step5:SPLITTING DATA INTO DEPENDENT AND INDEPENDENT
step6:MODEL BUILDING
              ->training the model
              ->testing part

```

In [120]:

```
df=df.drop(["customerID"],axis=1)
```

In [121]:

```
#STEP 1 in eda
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   object 
 19  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
```

In [122]:

```
#we use pd.to_numeric because It can convert different types of inputs
#has the errors parameter, which allows you to handle errors gracefully
df["TotalCharges"]=pd.to_numeric(df["TotalCharges"],errors="coerce")
```

In [123]:

```
#to check if the data set contain "?"," " we use unique()
for i in df.columns:
    print(df[i].unique())
```

```
['Female' 'Male']
[0 1]
['Yes' 'No']
['No' 'Yes']
[ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26  0
 39]
['No' 'Yes']
['No phone service' 'No' 'Yes']
['DSL' 'Fiber optic' 'No']
['No' 'Yes' 'No internet service']
['Yes' 'No' 'No internet service']
['No' 'Yes' 'No internet service']
['No' 'Yes' 'No internet service']
['No' 'Yes' 'No internet service']
['No' 'Yes' 'No internet service']
['Month-to-month' 'One year' 'Two year']
['Yes' 'No']
['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
[29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
[  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
['No' 'Yes']
```

In [124]:
```
#STEP 2.1 in eda
#CHECK NULL VALUES
'''if the null values <10% ------drop
else
   if the null values are more :
                                scattered-replace
                                clustered-dropping
 '''
df.isnull().sum()
```

Out[124]:
```
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       11
Churn               0
dtype: int64
```

In [125]:
```
#here we see null values in total charges beacuse of errors=coerce whic
#so we will drop them ....dropping 11 null values will not effect the c
#inplace=True is used because the changes are directly applied to the c
df.dropna(inplace=True)
```

In [126]:
```
df
```

Out[126]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inter
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	Yes	No	
2	Male	0	No	No	2	Yes	No	
3	Male	0	No	No	45	No	No phone service	
4	Female	0	No	No	2	Yes	No	
...	...	...	...	...	...	...	...	
7038	Male	0	Yes	Yes	24	Yes	Yes	
7039	Female	0	Yes	Yes	72	Yes	Yes	
7040	Female	0	Yes	Yes	11	No	No phone service	
7041	Male	1	Yes	No	4	Yes	Yes	
7042	Male	0	No	No	66	Yes	No	

7032 rows × 20 columns

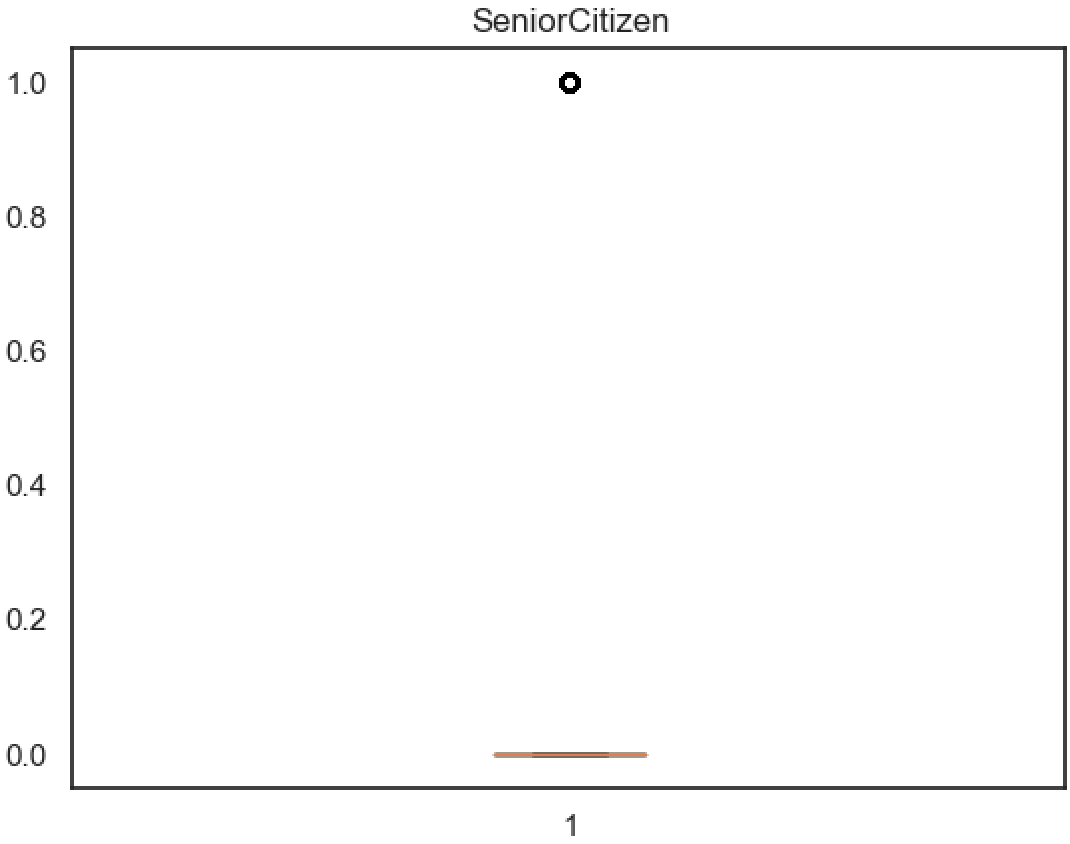
In [127]:
```
#STEP 2.2 in eda
#if there are duplicate values we drop them to reduce the data redundar
df.duplicated().sum()
```

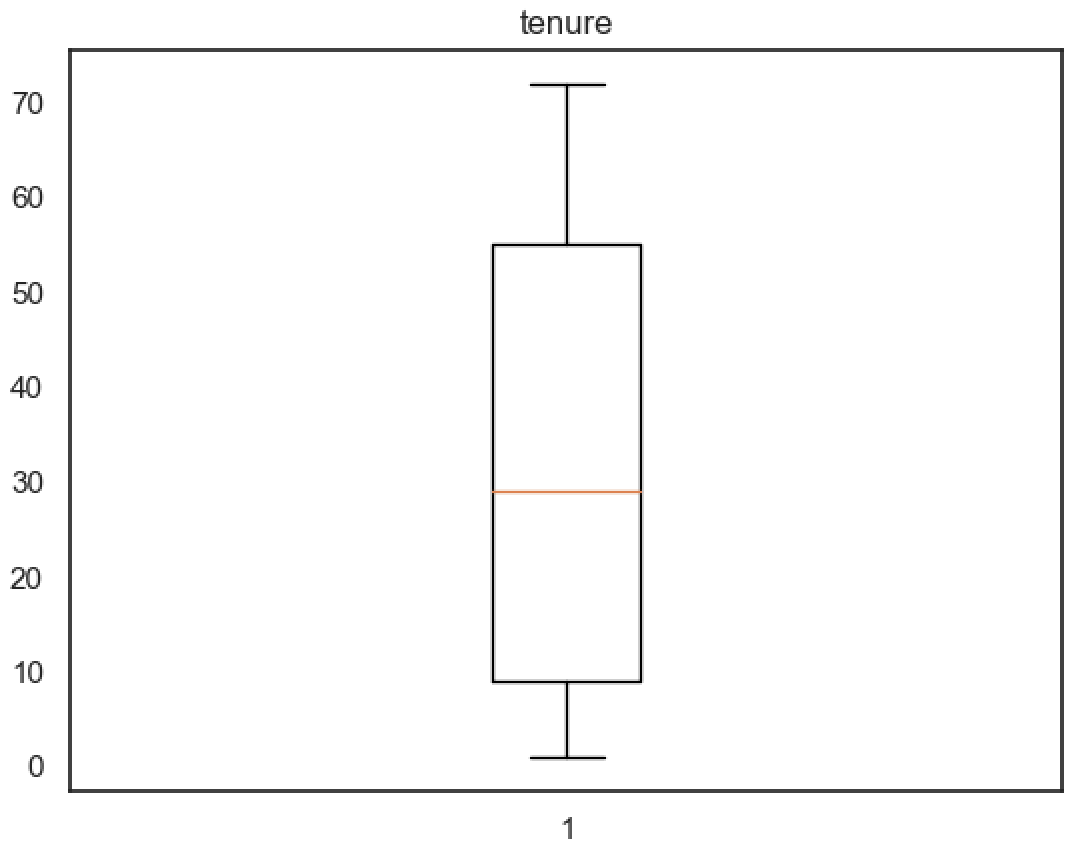
Out[127]: 22

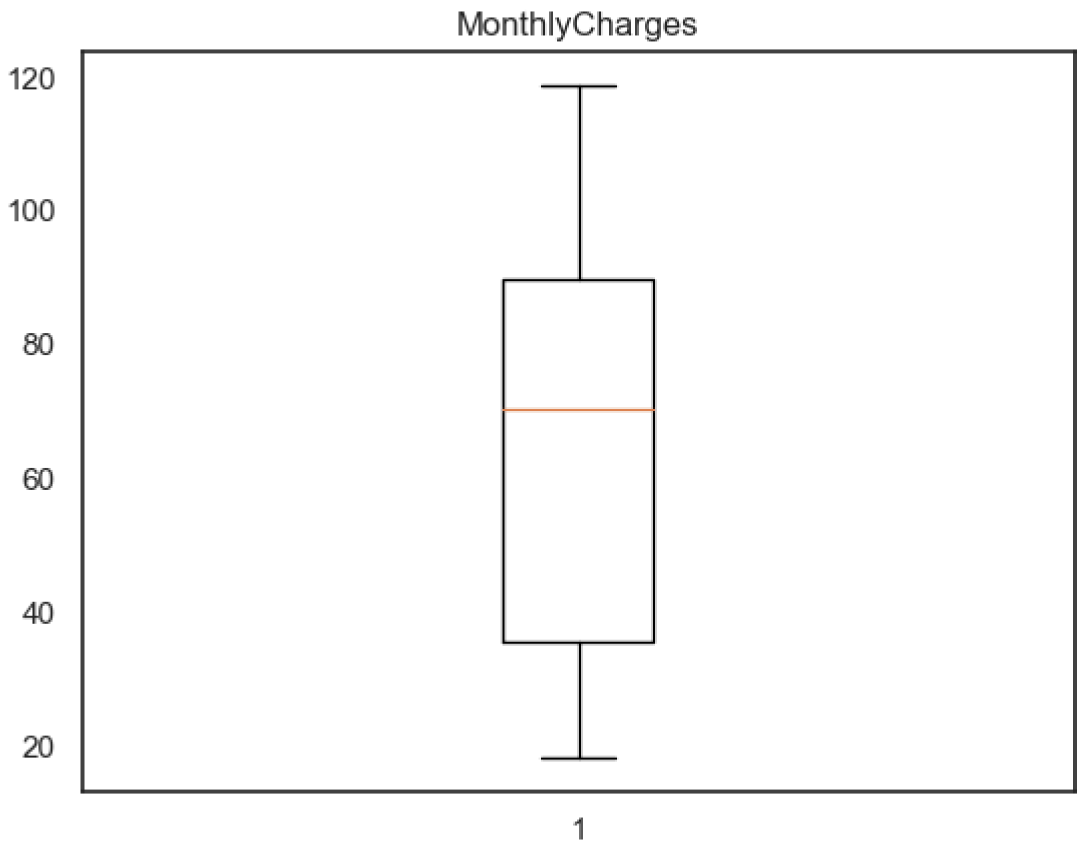
#Outlier analysis Outliers are data points that are significantly different from the rest of the dataset.Any values below lowerfence and above upperfence are outliers and should be removed. -->how to calculate Q1 and Q3 Q1=df[col].quantile(0.25) Q3=df[col].quantile(0.75) -->IQR=Q3-Q1 -->Lowelimit=Q1-1.5*IQR -->Upperlimit=Q3+1.5*IQR IQR analysis:

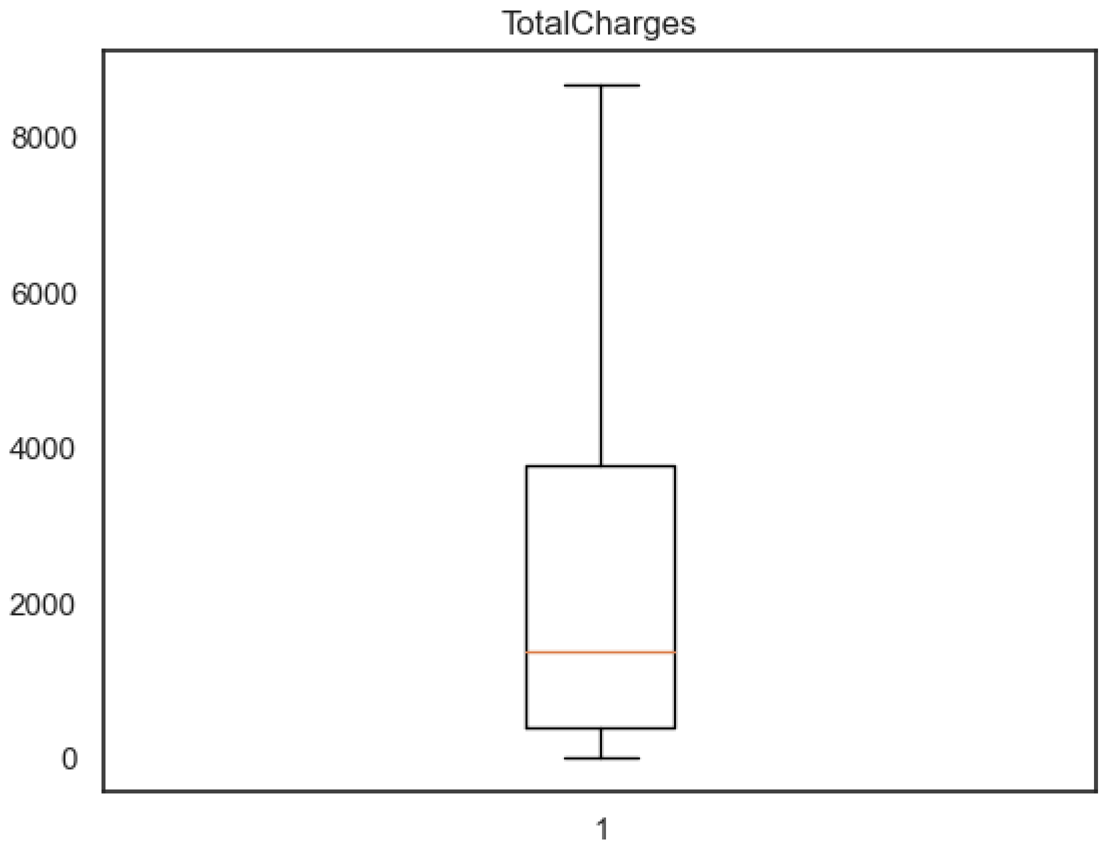
In [128]:

```
#STEP 2.3 in eda
# Outlier analysis
#We take the help of Box plot(only numerical)
for i in df.columns:
    if df[i].dtype!="object":
        plt.boxplot(df[i])
        plt.title(i)
        plt.show()
```




tenure
70
60
50
40
30
20
10
0
1


MonthlyCharges
120
100
80
60
40
20
1



In [129]:
```
#we can ignore the outlier analysis for seniorcitizen as we did not get
#in above data we don't have outliers
#some time after analysis new outliers will be formed but we dont consi
```

In [130]:
```
#STEP 3
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [131]:
```
#fit transformation: fit-used to take the values transform-change the v
for col in df.columns:
    if df[col].dtype=="object":
        df[col]=le.fit_transform(df[col])
```

In [132]:
```
df
```

Out[132]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Interi
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	0	
3	1	0	0	0	45	0	1	
4	0	0	0	0	2	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	2	
7039	0	0	1	1	72	1	2	
7040	0	0	1	1	11	0	1	
7041	1	1	1	0	4	1	2	
7042	1	0	0	0	66	1	0	

7032 rows × 20 columns

In [233]:
```
#STEP 4:
# feature Selection
df3=df
```

In [134]:

```
#helps to understanding the correlation between features (independent v
df.corr()
```

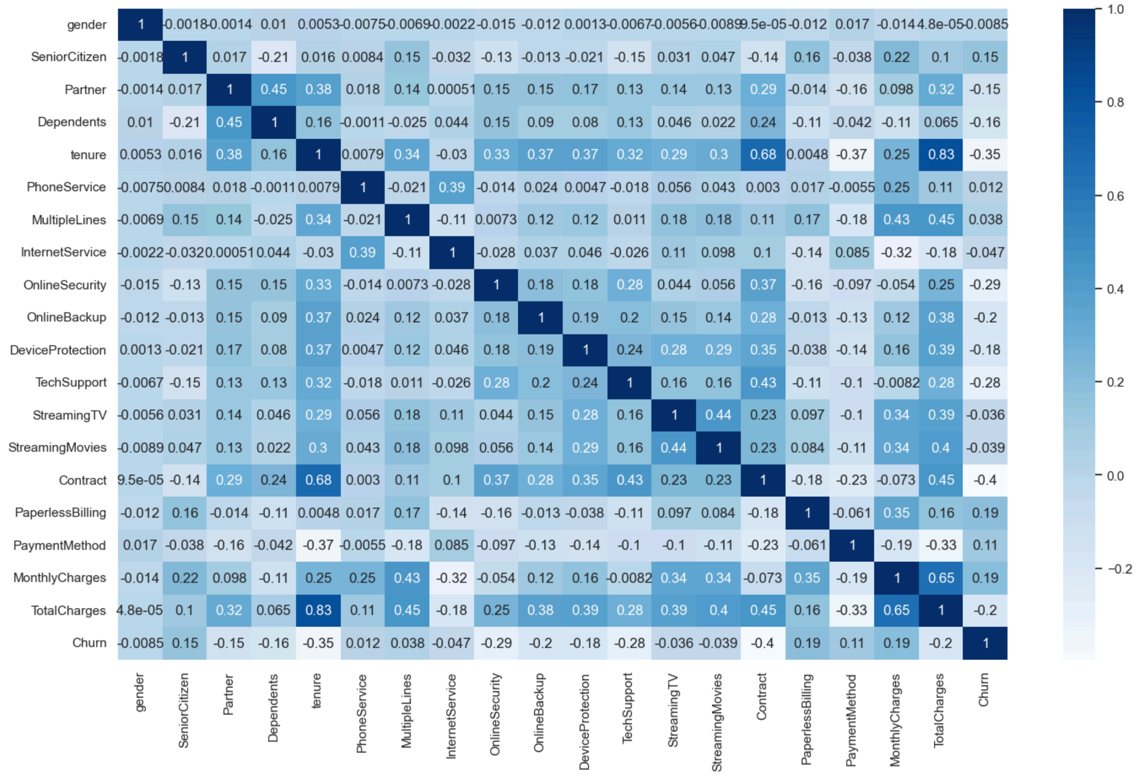
Out[134]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService
gender	1.000000	-0.001819	-0.001379	0.010349	0.005285	-0.007515
SeniorCitizen	-0.001819	1.000000	0.016957	-0.210550	0.015683	0.008392
Partner	-0.001379	0.016957	1.000000	0.452269	0.381912	0.018397
Dependents	0.010349	-0.210550	0.452269	1.000000	0.163386	-0.001078
tenure	0.005285	0.015683	0.381912	0.163386	1.000000	0.007877
PhoneService	-0.007515	0.008392	0.018397	-0.001078	0.007877	1.000000
MultipleLines	-0.006908	0.146287	0.142717	-0.024975	0.343673	-0.020504
InternetService	-0.002236	-0.032160	0.000513	0.044030	-0.029835	0.387266
OnlineSecurity	-0.014899	-0.127937	0.150610	0.151198	0.327283	-0.014163
OnlineBackup	-0.011920	-0.013355	0.153045	0.090231	0.372434	0.024040
DeviceProtection	0.001348	-0.021124	0.165614	0.079723	0.372669	0.004718
TechSupport	-0.006695	-0.151007	0.126488	0.132530	0.324729	-0.018136
StreamingTV	-0.005624	0.031019	0.136679	0.046214	0.290572	0.056393
StreamingMovies	-0.008920	0.047088	0.129907	0.022088	0.296785	0.043025
Contract	0.000095	-0.141820	0.294094	0.240556	0.676734	0.003019
PaperlessBilling	-0.011902	0.156258	-0.013957	-0.110131	0.004823	0.016696
PaymentMethod	0.016942	-0.038158	-0.156232	-0.041989	-0.370087	-0.005499
MonthlyCharges	-0.013779	0.219874	0.097825	-0.112343	0.246862	0.248033
TotalCharges	0.000048	0.102411	0.319072	0.064653	0.825880	0.113008
Churn	-0.008545	0.150541	-0.149982	-0.163128	-0.354049	0.011691

In [135]:

```
#annot is to get te vakues
#we search for good correlation values i.e correlation>0.7  but here we
plt.figure(figsize=(17,10))
sns.heatmap(df.corr(),cmap="Blues",annot=True)
```

Out[135]: <Axes: >



In [136]:

```
from statsmodels.stats.outliers_influence import variance_inflation_fac
```

In [137]:

```
#we use the vif to check multicollinearity
#here we are selecting object type col and no target col
col=[]
for i in df.columns:
    if (df[i].dtype!="object") &(i!="Churn"):
        col.append(i)
```

In [138]:
```
col
```

Out[138]:
```
['gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'tenure',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'MonthlyCharges',
 'TotalCharges']
```

In [139]:
```
#create a dataframe so that the values we get in a categorize way
x=df[col]
```

In [140]:
```
#in this data frame we perform feature selection
x
```

Out[140]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inter
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	0	
3	1	0	0	0	45	0	1	
4	0	0	0	0	2	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	2	
7039	0	0	1	1	72	1	2	
7040	0	0	1	1	11	0	1	
7041	1	1	1	0	4	1	2	
7042	1	0	0	0	66	1	0	

7032 rows × 19 columns

In [141]:

```
#manipulate and update the x continuously for every iteration of vif
'''Steps involved
Initialize an empty DataFrame
Assign feature names to the DataFrame
Calculate VIF values'''
vif_data=pd.DataFrame()
vif_data["Feature"]=x.columns
vif_data["VIF_values"]=[variance_inflation_factor(x.values,i) for i in
vif_data
```

Out[141]:

	Feature	VIF_values
0	gender	1.954535
1	SeniorCitizen	1.369954
2	Partner	2.819229
3	Dependents	1.957360
4	tenure	15.084412
5	PhoneService	15.150758
6	MultipleLines	2.756988
7	InternetService	4.350001
8	OnlineSecurity	2.247863
9	OnlineBackup	2.455913
10	DeviceProtection	2.629892
11	TechSupport	2.381046
12	StreamingTV	3.237958
13	StreamingMovies	3.265595
14	Contract	4.194484
15	PaperlessBilling	2.875010
16	PaymentMethod	3.095143
17	MonthlyCharges	20.503844
18	TotalCharges	13.869098

In [142]:

```
#we only consider the col whose vif is less than 5
#we dont drop all the values at a time because for every iteration the
#first take highest vif value col and drop
x=x.drop(["MonthlyCharges"],axis=1)
```

In [143]:

```
x
```

Out[143]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Interı
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	0	
3	1	0	0	0	45	0	1	
4	0	0	0	0	2	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	2	
7039	0	0	1	1	72	1	2	
7040	0	0	1	1	11	0	1	
7041	1	1	1	0	4	1	2	
7042	1	0	0	0	66	1	0	

7032 rows × 18 columns

In [144]:

```
vif_data=pd.DataFrame()
vif_data["Feature"]=x.columns
vif_data["VIF_values"]=[variance_inflation_factor(x.values,i) for i in
vif_data
```

Out[144]:

	Feature	VIF_values
0	gender	1.936952
1	SeniorCitizen	1.343210
2	Partner	2.814039
3	Dependents	1.957317
4	tenure	13.942277
5	PhoneService	8.202506
6	MultipleLines	2.511962
7	InternetService	3.646896
8	OnlineSecurity	2.247428
9	OnlineBackup	2.454485
10	DeviceProtection	2.617893
11	TechSupport	2.380627
12	StreamingTV	3.095955
13	StreamingMovies	3.117757
14	Contract	4.073047
15	PaperlessBilling	2.614613
16	PaymentMethod	3.021672
17	TotalCharges	9.995409

In [145]:
```
x=x.drop(["tenure"],axis=1)
```

In [146]:
```
x
```

Out[146]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetServi
0	0	0	1	0	0	1	
1	1	0	0	0	1	0	
2	1	0	0	0	1	0	
3	1	0	0	0	0	1	
4	0	0	0	0	1	0	
...	...	...	...	...	...	...	
7038	1	0	1	1	1	2	
7039	0	0	1	1	1	2	
7040	0	0	1	1	0	1	
7041	1	1	1	0	1	2	
7042	1	0	0	0	1	0	

7032 rows × 17 columns

In [147]:
```
vif_data=pd.DataFrame()
vif_data["Feature"]=x.columns
vif_data["VIF_values"]=[variance_inflation_factor(x.values,i) for i in
vif_data
```

Out[147]:

	Feature	VIF_values
0	gender	1.919674
1	SeniorCitizen	1.341260
2	Partner	2.749816
3	Dependents	1.955831
4	PhoneService	8.200629
5	MultipleLines	2.499564
6	InternetService	3.491818
7	OnlineSecurity	2.228559
8	OnlineBackup	2.441964
9	DeviceProtection	2.617616
10	TechSupport	2.380037
11	StreamingTV	3.075287
12	StreamingMovies	3.100860
13	Contract	3.014782
14	PaperlessBilling	2.605425
15	PaymentMethod	3.021575
16	TotalCharges	5.316537

In [148]:
```
x=x.drop(["PhoneService"],axis=1)
```

In [149]:
```
x
```

Out[149]:

	gender	SeniorCitizen	Partner	Dependents	MultipleLines	InternetService	OnlineSecur
0	0	0	1	0	1	0	
1	1	0	0	0	0	0	
2	1	0	0	0	0	0	
3	1	0	0	0	1	0	
4	0	0	0	0	0	1	
...	...	...	...	...	...	...	
7038	1	0	1	1	2	0	
7039	0	0	1	1	2	1	
7040	0	0	1	1	1	0	
7041	1	1	1	0	2	1	
7042	1	0	0	0	0	1	

7032 rows × 16 columns

In [150]:
```
vif_data=pd.DataFrame()
vif_data["Feature"]=x.columns
vif_data["VIF_values"]=[variance_inflation_factor(x.values,i) for i in
vif_data
```

Out[150]:

	Feature	VIF_values
0	gender	1.864278
1	SeniorCitizen	1.336778
2	Partner	2.739901
3	Dependents	1.949167
4	MultipleLines	2.492203
5	InternetService	2.529810
6	OnlineSecurity	2.196135
7	OnlineBackup	2.437734
8	DeviceProtection	2.616390
9	TechSupport	2.357595
10	StreamingTV	3.075266
11	StreamingMovies	3.100816
12	Contract	2.997897
13	PaperlessBilling	2.411725
14	PaymentMethod	2.615868
15	TotalCharges	5.075258

In [151]:
```
x
```

Out[151]:

	gender	SeniorCitizen	Partner	Dependents	MultipleLines	InternetService	OnlineSecur
0	0	0	1	0	1	0	
1	1	0	0	0	0	0	
2	1	0	0	0	0	0	
3	1	0	0	0	1	0	
4	0	0	0	0	0	1	
...	...	...	...	...	...	...	
7038	1	0	1	1	2	0	
7039	0	0	1	1	2	1	
7040	0	0	1	1	1	0	
7041	1	1	1	0	2	1	
7042	1	0	0	0	0	1	

7032 rows × 16 columns

In [152]:
```
# Splitting the data into dependent and independent data
```

In [153]:
```
x #independent
```

Out[153]:

	gender	SeniorCitizen	Partner	Dependents	MultipleLines	InternetService	OnlineSecur
0	0	0	1	0	1	0	
1	1	0	0	0	0	0	
2	1	0	0	0	0	0	
3	1	0	0	0	1	0	
4	0	0	0	0	0	1	
...	...	...	...	...	...	...	
7038	1	0	1	1	2	0	
7039	0	0	1	1	2	1	
7040	0	0	1	1	1	0	
7041	1	1	1	0	2	1	
7042	1	0	0	0	0	1	

7032 rows × 16 columns

In [154]:
```
y=df["Churn"]
```

In [155]:
```
y
```

Out[155]:
```
0       0
1       0
2       1
3       0
4       1
       ..
7038    0
7039    0
7040    0
7041    1
7042    0
Name: Churn, Length: 7032, dtype: int32
```

In [156]:
```
# split the data into train and testing
```

In [157]:
```
#sklearn.model_selection module to split a dataset into training and te
from sklearn.model_selection import train_test_split
```

In [158]:
```
#x-independent variables
#y-dependent variable
#train_size=0.70 specifies that 70% of the data should be used for trai
#Setting a random state ensures that you get the same split every time
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.70,ranc
```

In [159]:
```
x_train
```

Out[159]:

	gender	SeniorCitizen	Partner	Dependents	MultipleLines	InternetService	OnlineSecur
5841	1	1	0	0	1	0	
1513	0	1	0	0	1	0	
6238	1	1	1	0	2	1	
4579	1	0	0	0	2	1	
5601	0	0	1	1	1	0	
...	...	...	...	...	...	...	
4711	1	0	0	0	0	2	
3622	0	0	1	1	2	0	
6021	0	0	1	1	0	2	
5772	1	0	1	1	2	0	
6567	1	0	0	0	2	1	

4922 rows × 16 columns

In [160]:
```
1 y_train
```

Out[160]:
```
5841    1
1513    1
6238    0
4579    0
5601    0
       ..
4711    0
3622    0
6021    0
5772    0
6567    0
Name: Churn, Length: 4922, dtype: int32
```

In [161]:
```
1 x_test
```

Out[161]:

	gender	SeniorCitizen	Partner	Dependents	MultipleLines	InternetService	OnlineSecur
2287	1	1	1	0	2	1	
2087	0	0	0	0	1	0	
2308	1	0	1	1	0	1	
1960	0	0	1	1	2	1	
4634	0	0	0	0	0	2	
...	...	...	...	...	...	...	
6237	1	0	0	0	0	1	
1034	0	0	1	0	0	0	
6628	0	0	0	0	0	1	
6101	0	1	0	0	2	1	
713	0	0	0	1	2	1	

2110 rows × 16 columns

In [162]:
```
1 y_test
```

Out[162]:
```
2287    1
2087    1
2308    0
1960    0
4634    0
       ..
6237    1
1034    0
6628    0
6101    0
713     0
Name: Churn, Length: 2110, dtype: int32
```

In [163]:
```
# Model Implementation
#import model
from sklearn.linear_model import LogisticRegression
```

In [164]:
```
log_model=LogisticRegression()
```

In [165]:
```
# training the model
#get values-fit
log_model.fit(x_train,y_train)
```

Out[165]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [166]:
```
# testing part
log_pred=log_model.predict(x_test)
log_pred
```

Out[166]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [167]:
```
y_test
```

Out[167]:
```
2287    1
2087    1
2308    0
1960    0
4634    0
       ..
6237    1
1034    0
6628    0
6101    0
713     0
Name: Churn, Length: 2110, dtype: int32
```

In [168]:
```
from sklearn.metrics import *
```

In [169]:
```
accuracy_score(y_test,log_pred)
```

Out[169]:
```
0.7815165876777251
```

Linear Regression

steps.....

In [170]:
```
#this the data frame in which all the encoding part had been completed
dfli=pd.read_csv("customer_churn.csv")
```

In [171]:
```
1 dfli
```

Out[171]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	7590-VHVEG	Female	0	Yes	No	1	No	Nc
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	Nc
4	9237-HQITU	Female	0	No	No	2	Yes	
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	Nc
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	

7043 rows × 21 columns

In [172]:
```
#tells type of data stored
dfli.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [173]:
```
dfli["TotalCharges"]=pd.to_numeric(dfli["TotalCharges"],errors="coerce'
```

In [174]:
```
dfli.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7032 non-null   float64
 20  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

In [175]:
```
# data Cleansin
dfli.isnull().sum()
```

Out[175]:
```
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        11
Churn                0
dtype: int64
```

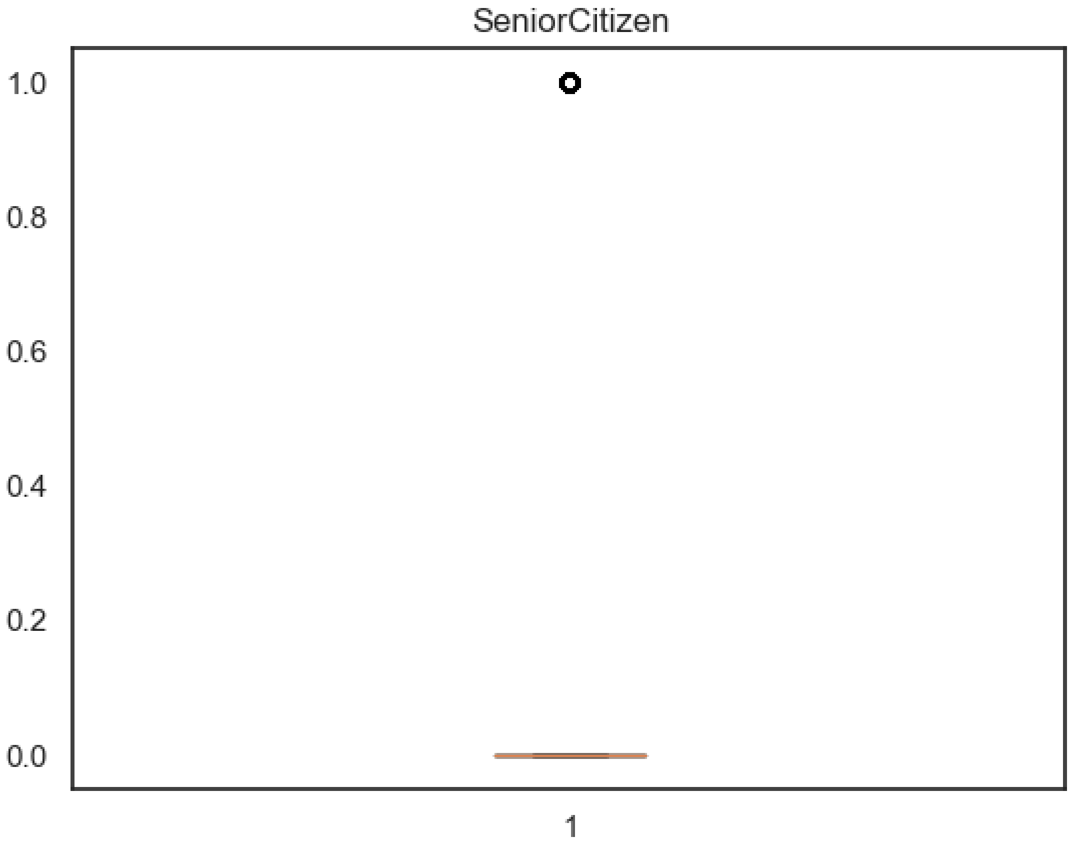
In [176]:
```
dfli.dropna(inplace=True)
```

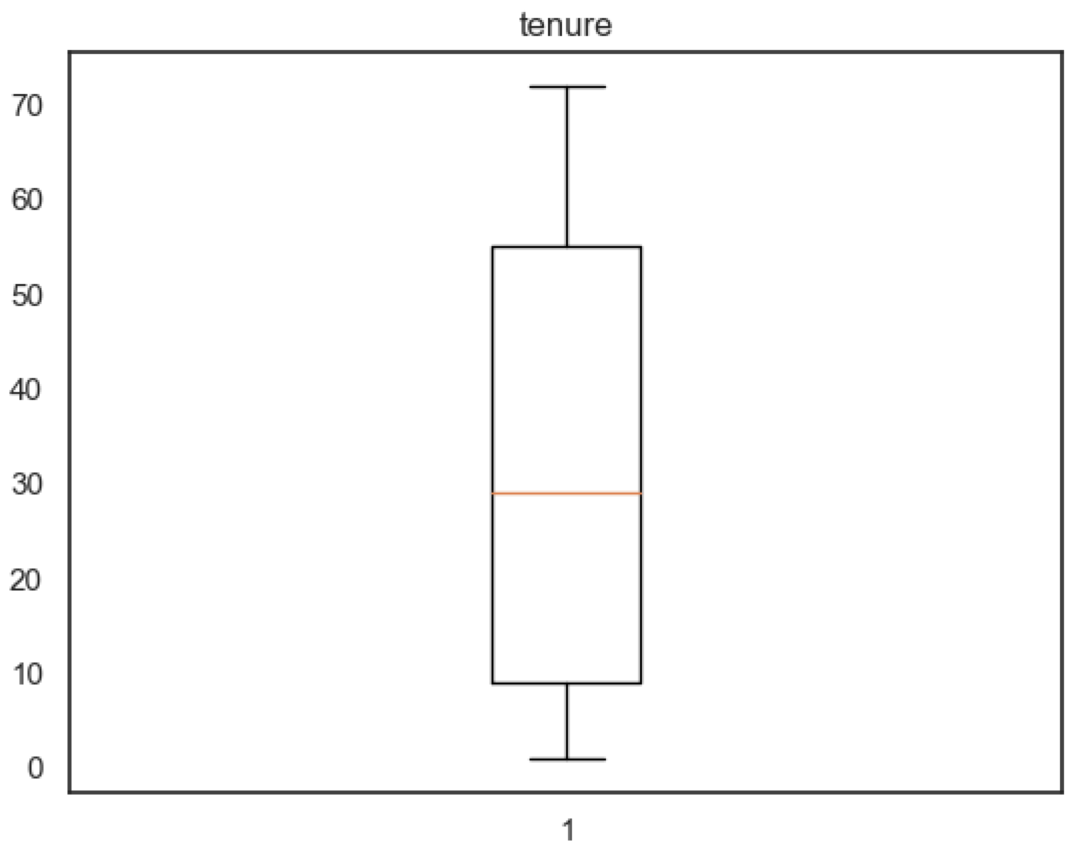
In [177]:
```
df.duplicated().sum()
```

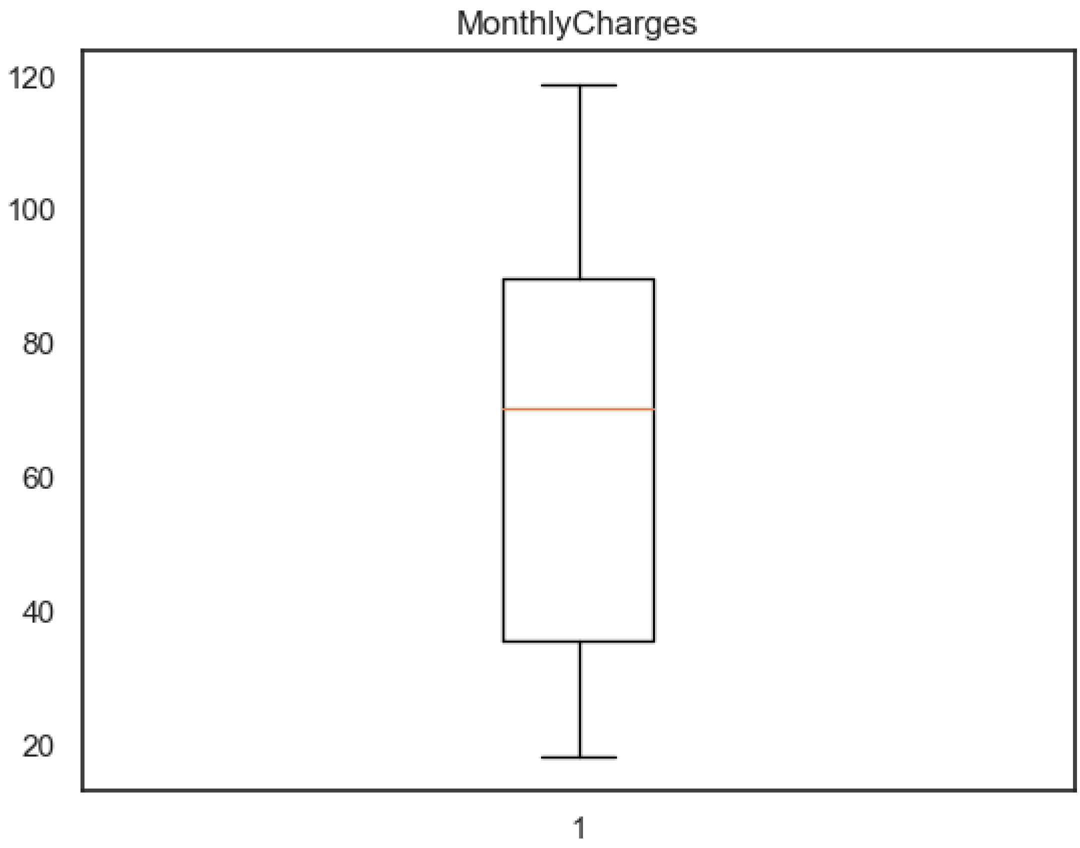
Out[177]: 22

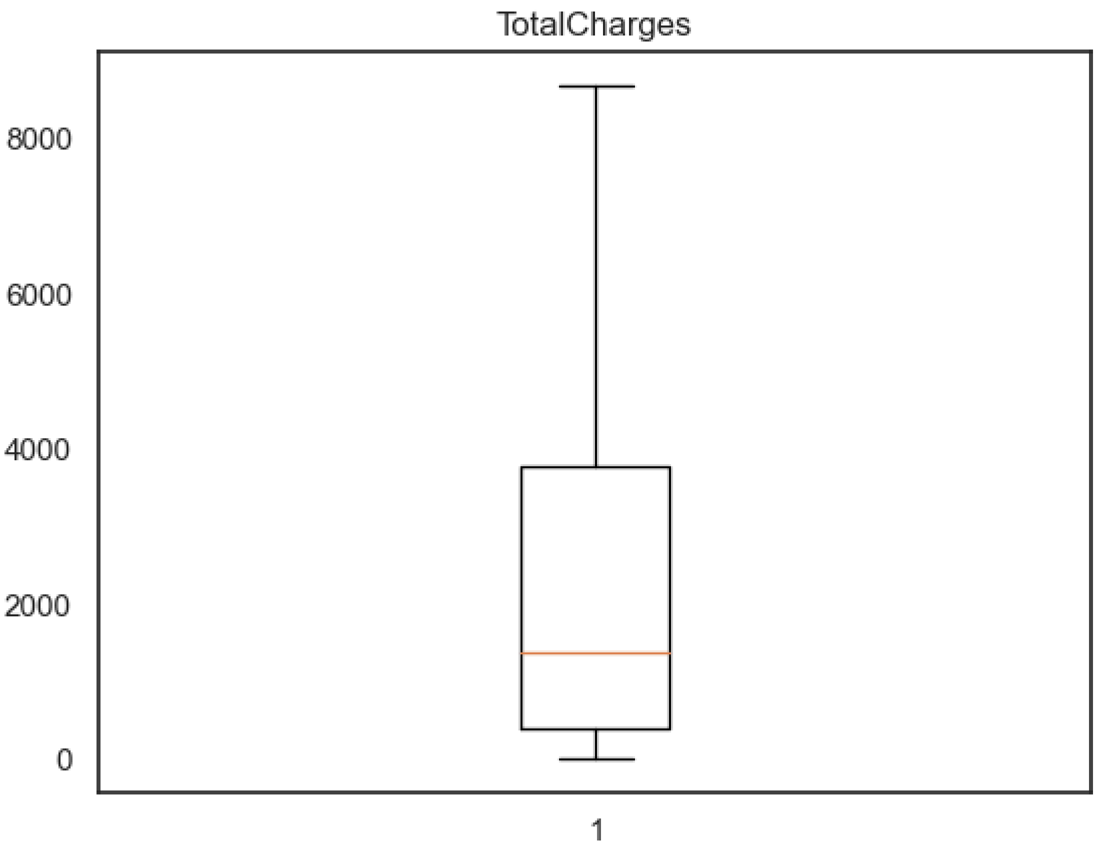
In [178]:
```
# Outliers
```

In [179]:
```
for i in dfli.columns:
    if(dfli[i].dtype=="int64" or dfli[i].dtype=="float64"):
        plt.boxplot(dfli[i])
        plt.title(i)
        plt.show()
```









In [180]:
```
from sklearn.preprocessing import LabelEncoder
```

In [181]:
```
for col in dfli.columns:
    if dfli[col].dtype=="object":
        dfli[col]=le.fit_transform(dfli[col])
```

In [182]:
```
dfli
```

Out[182]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	5365	0	0	1	0	1	0	
1	3953	1	0	0	0	34	1	
2	2558	1	0	0	0	2	1	
3	5524	1	0	0	0	45	0	
4	6500	0	0	0	0	2	1	
...	...	...	...	...	...	...	...	
7038	4843	1	0	1	1	24	1	
7039	1524	0	0	1	1	72	1	
7040	3358	0	0	1	1	11	0	
7041	5923	1	1	1	0	4	1	
7042	2221	1	0	0	0	66	1	

7032 rows × 21 columns

In [183]:
```
dfli.corr()
```

Out[183]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Ph
customerID	1.000000	0.006235	-0.002368	-0.026509	-0.011871	0.007209	
gender	0.006235	1.000000	-0.001819	-0.001379	0.010349	0.005285	
SeniorCitizen	-0.002368	-0.001819	1.000000	0.016957	-0.210550	0.015683	
Partner	-0.026509	-0.001379	0.016957	1.000000	0.452269	0.381912	
Dependents	-0.011871	0.010349	-0.210550	0.452269	1.000000	0.163386	
tenure	0.007209	0.005285	0.015683	0.381912	0.163386	1.000000	
PhoneService	-0.006987	-0.007515	0.008392	0.018397	-0.001078	0.007877	
MultipleLines	0.004497	-0.006908	0.146287	0.142717	-0.024975	0.343673	
InternetService	-0.012335	-0.002236	-0.032160	0.000513	0.044030	-0.029835	
OnlineSecurity	0.013740	-0.014899	-0.127937	0.150610	0.151198	0.327283	
OnlineBackup	-0.002960	-0.011920	-0.013355	0.153045	0.090231	0.372434	
DeviceProtection	-0.006726	0.001348	-0.021124	0.165614	0.079723	0.372669	
TechSupport	0.001763	-0.006695	-0.151007	0.126488	0.132530	0.324729	
StreamingTV	-0.007650	-0.005624	0.031019	0.136679	0.046214	0.290572	
StreamingMovies	-0.017207	-0.008920	0.047088	0.129907	0.022088	0.296785	
Contract	0.015949	0.000095	-0.141820	0.294094	0.240556	0.676734	
PaperlessBilling	-0.002225	-0.011902	0.156258	-0.013957	-0.110131	0.004823	
PaymentMethod	0.011754	0.016942	-0.038158	-0.156232	-0.041989	-0.370087	
MonthlyCharges	-0.004445	-0.013779	0.219874	0.097825	-0.112343	0.246862	
TotalCharges	-0.000263	0.000048	0.102411	0.319072	0.064653	0.825880	
Churn	-0.017858	-0.008545	0.150541	-0.149982	-0.163128	-0.354049	

21 rows × 21 columns

In [184]:
```
from statsmodels.stats.outliers_influence import variance_inflation_fac
```

In [185]:
```
col=[]
for i in dfli.columns:
    if (dfli[i].dtype!="object") &(i!="MonthlyCharges"):
        col.append(i)
```

In [186]:
```
col
```

Out[186]:
```
['customerID',
 'gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'tenure',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'TotalCharges',
 'Churn']
```

In [187]:
```
x=dfli[col]
```

In [188]:
```
x
```

Out[188]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	5365	0	0	1	0	1	0	
1	3953	1	0	0	0	34	1	
2	2558	1	0	0	0	2	1	
3	5524	1	0	0	0	45	0	
4	6500	0	0	0	0	2	1	
...	...	...	...	...	...	...	...	
7038	4843	1	0	1	1	24	1	
7039	1524	0	0	1	1	72	1	
7040	3358	0	0	1	1	11	0	
7041	5923	1	1	1	0	4	1	
7042	2221	1	0	0	0	66	1	

7032 rows × 20 columns

In [189]:

```
vif_data=pd.DataFrame()
vif_data["Feature"]=x.columns
vif_data["VIF_values"]=[variance_inflation_factor(x.values,i) for i in
vif_data
```

Out[189]:

	Feature	VIF_values
0	customerID	3.604625
1	gender	1.949827
2	SeniorCitizen	1.355221
3	Partner	2.815156
4	Dependents	1.958513
5	tenure	14.281169
6	PhoneService	9.094752
7	MultipleLines	2.566525
8	InternetService	3.664851
9	OnlineSecurity	2.267826
10	OnlineBackup	2.460921
11	DeviceProtection	2.621761
12	TechSupport	2.396986
13	StreamingTV	3.111502
14	StreamingMovies	3.133713
15	Contract	4.123100
16	PaperlessBilling	2.710889
17	PaymentMethod	3.141583
18	TotalCharges	10.199680
19	Churn	1.712721

In [190]:

```
x=x.drop(["tenure"],axis=1)
```

In [191]:
```
1 x
```

Out[191]:

	customerID	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines
0	5365	0	0	1	0	0	1
1	3953	1	0	0	0	1	0
2	2558	1	0	0	0	1	0
3	5524	1	0	0	0	0	1
4	6500	0	0	0	0	1	0
...	...	...	...	...	...	...	...
7038	4843	1	0	1	1	1	2
7039	1524	0	0	1	1	1	2
7040	3358	0	0	1	1	0	1
7041	5923	1	1	1	0	1	2
7042	2221	1	0	0	0	1	0

7032 rows × 19 columns

In [192]:

```
vif_data=pd.DataFrame()
vif_data["Feature"]=x.columns
vif_data["VIF_values"]=[variance_inflation_factor(x.values,i) for i in
vif_data
```

Out[192]:

	Feature	VIF_values
0	customerID	3.536850
1	gender	1.935379
2	SeniorCitizen	1.353125
3	Partner	2.750355
4	Dependents	1.957417
5	PhoneService	9.082872
6	MultipleLines	2.554642
7	InternetService	3.505913
8	OnlineSecurity	2.254640
9	OnlineBackup	2.451262
10	DeviceProtection	2.621703
11	TechSupport	2.396939
12	StreamingTV	3.091057
13	StreamingMovies	3.117584
14	Contract	3.114658
15	PaperlessBilling	2.702222
16	PaymentMethod	3.140300
17	TotalCharges	5.403306
18	Churn	1.702173

In [193]:

```
x=x.drop(["PhoneService"],axis=1)
```

In [194]:
```
x
```

Out[194]:

	customerID	gender	SeniorCitizen	Partner	Dependents	MultipleLines	InternetService
0	5365	0	0	1	0	1	0
1	3953	1	0	0	0	0	0
2	2558	1	0	0	0	0	0
3	5524	1	0	0	0	1	0
4	6500	0	0	0	0	0	1
...	...	...	...	...	...	...	...
7038	4843	1	0	1	1	2	0
7039	1524	0	0	1	1	2	1
7040	3358	0	0	1	1	1	0
7041	5923	1	1	1	0	2	1
7042	2221	1	0	0	0	0	1

7032 rows × 18 columns

In [195]:
```
vif_data=pd.DataFrame()
vif_data["Feature"]=x.columns
vif_data["VIF_values"]=[variance_inflation_factor(x.values,i) for i in
vif_data
```

Out[195]:

	Feature	VIF_values
0	customerID	3.313093
1	gender	1.900839
2	SeniorCitizen	1.351954
3	Partner	2.742714
4	Dependents	1.951666
5	MultipleLines	2.554471
6	InternetService	2.574290
7	OnlineSecurity	2.228371
8	OnlineBackup	2.448668
9	DeviceProtection	2.621366
10	TechSupport	2.376101
11	StreamingTV	3.090264
12	StreamingMovies	3.116970
13	Contract	3.105066
14	PaperlessBilling	2.594533
15	PaymentMethod	2.892367
16	TotalCharges	5.101130
17	Churn	1.652455

In [196]:
```
1 # Splitting the data into dependent and independent data
```

In [197]:
```
1 x #independent
```

Out[197]:

	customerID	gender	SeniorCitizen	Partner	Dependents	MultipleLines	InternetService
0	5365	0	0	1	0	1	0
1	3953	1	0	0	0	0	0
2	2558	1	0	0	0	0	0
3	5524	1	0	0	0	1	0
4	6500	0	0	0	0	0	1
...	...	...	...	...	...	...	...
7038	4843	1	0	1	1	2	0
7039	1524	0	0	1	1	2	1
7040	3358	0	0	1	1	1	0
7041	5923	1	1	1	0	2	1
7042	2221	1	0	0	0	0	1

7032 rows × 18 columns

In [198]:
```
1 y=dfli["MonthlyCharges"]
```

In [199]:
```
1 y
```

Out[199]:
```
0        29.85
1        56.95
2        53.85
3        42.30
4        70.70
         ...
7038     84.80
7039    103.20
7040     29.60
7041     74.40
7042    105.65
Name: MonthlyCharges, Length: 7032, dtype: float64
```

In [200]:
```
1 # split the data into train and testing
```

In [201]:
```
1 from sklearn.linear_model import LinearRegression
```

In [202]:
```
1 x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.70,ranc
```

In [203]:
```
1 lin_model=LinearRegression()
```

In [204]:
```
# training the model
lin_model.fit(x_train,y_train)
```

Out[204]: LinearRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [205]:
```
# testing part
lin_pred=lin_model.predict(x_test)

```

In [206]:
```
lin_pred
```

Out[206]:
```
array([117.30911993,  59.37969146, 105.52710272, ...,  52.05615548,
       102.39628966,  52.80613879])
```

In [207]:
```
#difference between our actual and predicted is error
error_pred=pd.DataFrame(columns=["Actual_data","Predicted_data"])
```

In [208]:
```
error_pred
```

Out[208]:

Actual_data	Predicted_data

In [209]:
```
error_pred["Actual_data"]=y_test
```

In [210]:
```
error_pred["Predicted_data"]=lin_pred
```

In [211]:
```
error_pred
```

Out[211]:

	Actual_data	Predicted_data
2287	108.40	117.309120
2087	33.65	59.379691
2308	104.65	105.527103
1960	88.60	64.312687
4634	18.75	36.971458
...	...	...
6237	69.95	50.577934
1034	81.85	85.845096
6628	94.05	52.056155
6101	110.25	102.396290
713	86.00	52.806139

2110 rows × 2 columns

In [212]:
```
error_pred["Error"]=error_pred["Actual_data"]-error_pred["Predicted_dat
```

In [213]:
```
error_pred
```

Out[213]:

	Actual_data	Predicted_data	Error
2287	108.40	117.309120	-8.909120
2087	33.65	59.379691	-25.729691
2308	104.65	105.527103	-0.877103
1960	88.60	64.312687	24.287313
4634	18.75	36.971458	-18.221458
...	...	...	...
6237	69.95	50.577934	19.372066
1034	81.85	85.845096	-3.995096
6628	94.05	52.056155	41.993845
6101	110.25	102.396290	7.853710
713	86.00	52.806139	33.193861

2110 rows × 3 columns

In [214]:
```
y_test
```

Out[214]:
```
2287    108.40
2087     33.65
2308    104.65
1960     88.60
4634     18.75
         ...
6237     69.95
1034     81.85
6628     94.05
6101    110.25
713      86.00
Name: MonthlyCharges, Length: 2110, dtype: float64
```

In [215]:
```
r2_score(y_test,lin_pred)
```

Out[215]:
```
0.6939559300266316
```

Random forest

In [216]:
```
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots #visualization
import warnings
warnings.filterwarnings('ignore')
```

In [217]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
#from sklearn.metrics import roc_curve
#from sklearn.metrics import recall_score, confusion_matrix, precision_
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
#from sklearn.metrics import f1_score, accuracy_score, classification_r
from sklearn.metrics import*  #if you want to import all the metric use
```

In [218]:

```
#loading data
dfra = pd.read_csv('customer_churn.csv')
```

In [219]:

```
dfra
```

Out[219]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	7590-VHVEG	Female	0	Yes	No	1	No	Nc
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	Nc
4	9237-HQITU	Female	0	No	No	2	Yes	
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	Nc
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	

7043 rows × 21 columns

In [220]:
```
1 dfra.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [221]:
```
1 df.isnull().sum()
```

Out[221]:
```
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        0
Churn               0
dtype: int64
```

In [222]:

```
#data clensing
#Since, we don't need customerID, We drop the column
dfra = dfra.drop(['customerID'], axis = 1)
dfra.head()
```

Out[222]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetS
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	Yes	No	
2	Male	0	No	No	2	Yes	No	
3	Male	0	No	No	45	No	No phone service	
4	Female	0	No	No	2	Yes	No	Fib

In [223]:

```
#Converting Object column to Numerical Column, which is actually holds
dfra['TotalCharges'] = pd.to_numeric(dfra.TotalCharges, errors='coerce'
dfra.isnull().sum()
```

Out[223]:

```
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       11
Churn               0
dtype: int64
```

In [224]:

```
# Fillna with mean values
#df.fillna(df["TotalCharges"].mean())

#or
#Removing missing values
dfra.dropna(inplace = True)
#or
#monthly charges*tenure=fill in total charges

```

In [225]:
```
dfra.isnull().sum()
```

Out[225]:
```
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        0
Churn               0
dtype: int64
```

In [226]:
```
#Removing tenure equal to 0
#axis=0-row wise drop,axis=1 col wise drop
dfra.drop(labels=dfra[dfra['tenure'] == 0].index, axis=0, inplace=True)
dfra[dfra['tenure'] == 0].index
```

Out[226]:
```
Index([], dtype='int64')
```

In [227]:
```
'''#Creating Dataframe for correlation plot
df2 = df
df2['Churn'].replace(to_replace='Yes', value=1, inplace=True)
df2['Churn'].replace(to_replace='No',  value=0, inplace=True)
df_dummies = pd.get_dummies(df2)
df_dummies.head()'''
```

Out[227]:
```
"#Creating Dataframe for correlation plot\ndf2 = df\ndf2['Churn'].replace
(to_replace='Yes', value=1, inplace=True)\ndf2['Churn'].replace(to_replace
='No',  value=0, inplace=True)\ndf_dummies = pd.get_dummies(df2)\ndf_dummi
es.head()"
```

In [228]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [229]:
```
for col in dfra.columns:
    if dfra[col].dtype=="object":
        dfra[col]=le.fit_transform(dfra[col])
```

In [230]:

```
dfra
```

Out[230]:

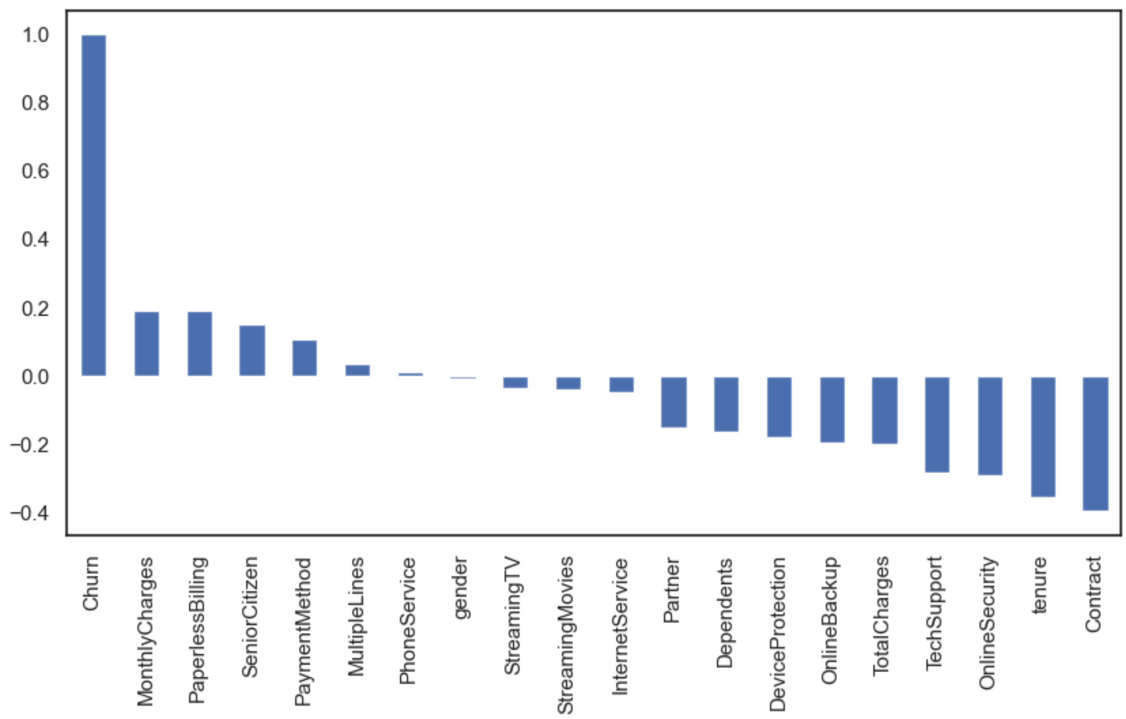
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inter
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	0	
3	1	0	0	0	45	0	1	
4	0	0	0	0	2	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	2	
7039	0	0	1	1	72	1	2	
7040	0	0	1	1	11	0	1	
7041	1	1	1	0	4	1	2	
7042	1	0	0	0	66	1	0	

7032 rows × 20 columns

In [231]:

```
#Correlation of "Churn" with Features:
plt.figure(figsize=(10,5))
sns.set(style = 'white')

dfra.corr()['Churn'].sort_values(ascending = False).plot(kind='bar')
```

Out[231]: <Axes: >



In [232]:

```
#data preprocessing
#splitting the data into training and test sets
```

Random Forest

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [3]:
```
# Load the dataset

data=pd.read_csv("customer_churn.csv")
```

In [4]:
```
# Preprocessing: Handle missing and incorrect data types
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coer
data = data.dropna()  # Drop rows with missing values

```

In [5]:
```
# Encode categorical variables
categorical_columns = [
    'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
    'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtect
    'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
    'PaperlessBilling', 'PaymentMethod', 'Churn'
]
data = pd.get_dummies(data, columns=categorical_columns, drop_first=Tru
```

In [6]:
```
# Define features and target
X = data.drop(columns=['customerID', 'Churn_Yes'])  # Exclude ID and ta
y = data['Churn_Yes']
```

In [7]:
```
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

```

In [8]:
```
# Train the Random Forest model
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)
```

Out[8]:
```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [11]:
```
# Predict on the test set
y_pred = rf_model.predict(X_test)
```

In [12]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

In [13]:
```
accuracy
```

Out[13]: 0.7853589196872779

Decision tree

In [22]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

In [23]:
```
#Load the dataset
data=pd.read_csv("customer_churn.csv")
```

In [24]:
```
# Preprocessing: Handle missing and incorrect data types
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coer
data = data.dropna()  # Drop rows with missing values

```

In [25]:
```
# Encode categorical variables
categorical_columns = [
    'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
    'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtect
    'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
    'PaperlessBilling', 'PaymentMethod', 'Churn'
]
data = pd.get_dummies(data, columns=categorical_columns, drop_first=Tru
```

In [26]:
```
# Define features and target
X = data.drop(columns=['customerID', 'Churn_Yes'])  # Exclude ID and ta
y = data['Churn_Yes']
```

In [27]:
```
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [28]:
```
# Train the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
```

Out[28]: DecisionTreeClassifier(random_state=42)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

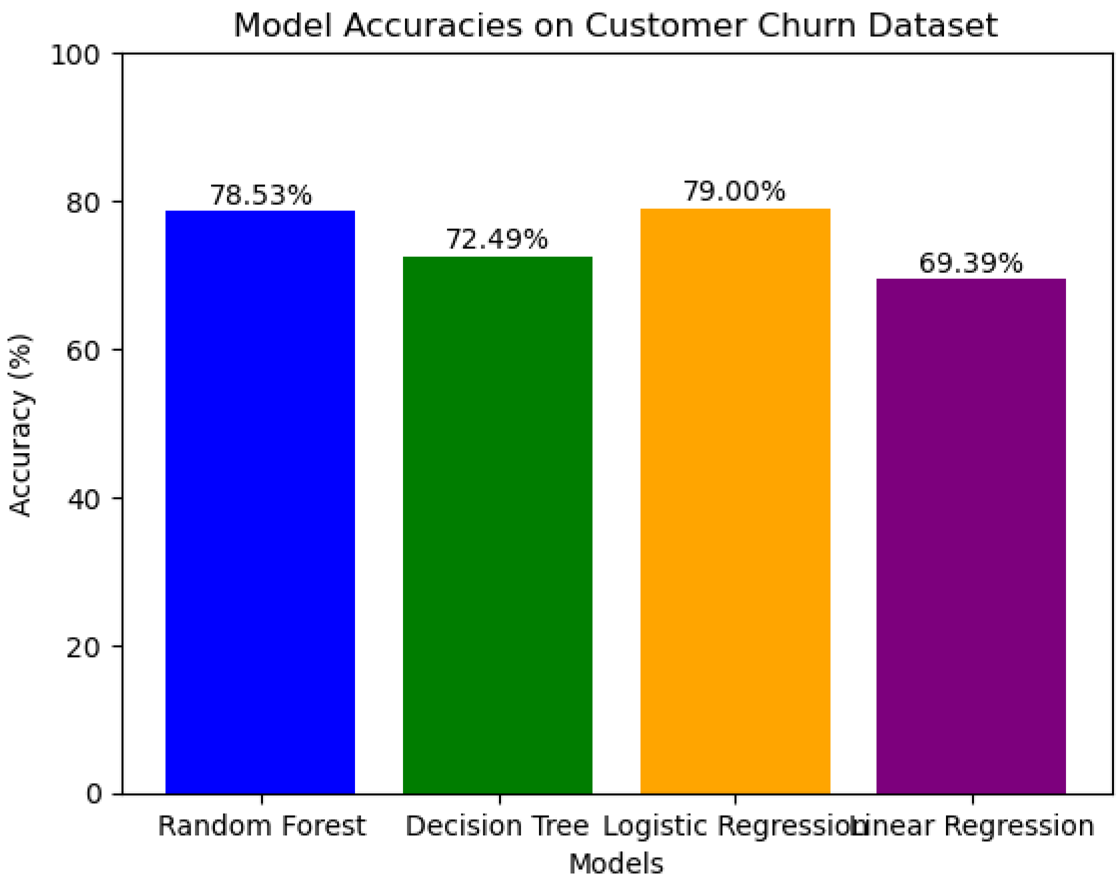
In [29]:
```
# Predict on the test set
y_pred = dt_model.predict(X_test)
```

In [30]:
```python
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

In [31]:
```python
accuracy
```

Out[31]: 0.7249466950959488

In [36]:
```python
import matplotlib.pyplot as plt

# Accuracy values
accuracies = {
    "Random Forest": 78.53,
    "Decision Tree": 72.49,
    "Logistic Regression": 79,
    "Linear Regression": 69.39
}

# Plotting
plt.bar(accuracies.keys(), accuracies.values(), color=['blue', 'green',
plt.title("Model Accuracies on Customer Churn Dataset")
plt.xlabel("Models")
plt.ylabel("Accuracy (%)")
plt.ylim(0, 100)

# Adding values on top of bars
for i, v in enumerate(accuracies.values()):
    plt.text(i, v + 1, f"{v:.2f}%", ha='center')

plt.show()

```



In [ ]:
```
1
```

In [ ]:
```
1
```