LINEAR MODELS  
Project Report (Logistics regression)

**Name and PRN:**

Akshata Satpute (22060641073)

Komal Solkar (22060641055)

**Linear Models**

**Logistic regression**

**Problem Statement:**

Predicting Churn based on different independent variable Churn, AccountWeeks, ContractRenewal, DataPlan, DataUsage, CustServCalls, DayMins, DayCalls, MonthlyCharge, OverageFee, RoamMins

**Data**: In the telecom industry, customers are able to choose from multiple service providers and actively switch from one operator to another. In this highly competitive market, the telecommunications industry experiences an average of 15-25% annual churn rate. Given the fact that it costs 5-10 times more to acquire a new customer than to retain an existing one, customer retention has now become even more important than customer acquisition. To reduce customer churn, telecom companies need to predict which customers are at high risk of churn. In this project, you will analyse customer-level data of a leading telecom firm, build predictive models to identify customers at high risk of churn, and identify the main indicators of churn. The data set consists of 1000 observations. Where it contains 11 columns and 999 rows. We consider churn as our response variable and the other 12 variables as independent variables.

**Source of data: Kaggle**

**Software used: Python**

**Python code with output:**

1. **Importing data into Python:**

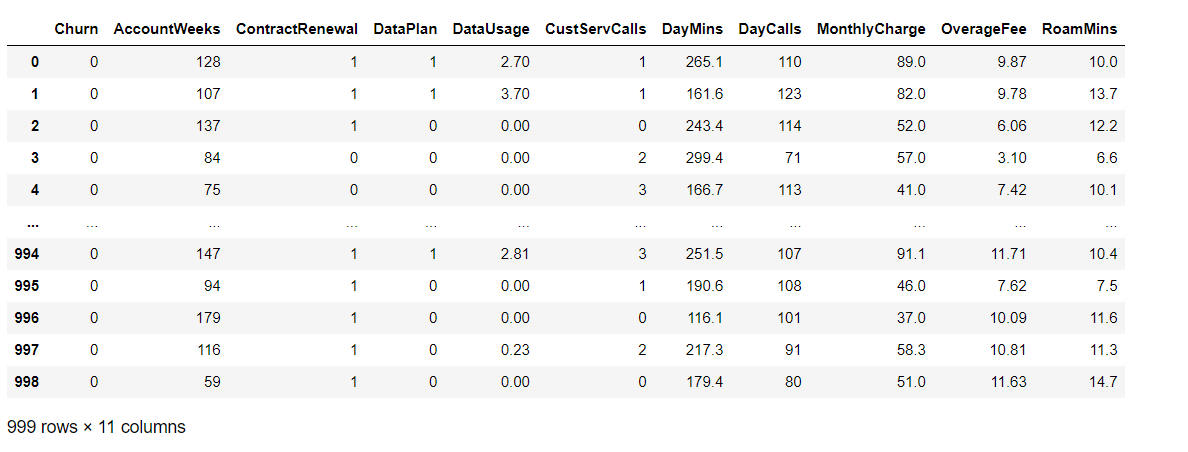
**import pandas as pd**

**import numpy as np**

**data=pd.read\_csv("C:/Users/User/Downloads/Cellphone.csv")**

**data**

**Output:**

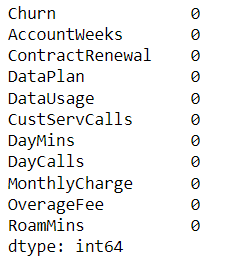
****

1. **Checking for Null Values:**

**Code:**

**data.isna().sum()**

**Output:**

****

Conclusion:

From this, we found that our data contains no null values.

1. **Showing Missing Data:**

**Code:**

**def show\_missing(data):**

**"""Return a Pandas dataframe describing the contents of a source dataframe including missing values."""**

**variables = []**

**dtypes = []**

**count = []**

**unique = []**

**missing = []**

**pc\_missing = []**

**for item in data.columns:**

**variables.append(item)**

**dtypes.append(data[item].dtype)**

**count.append(len(data[item]))**

**unique.append(len(data[item].unique()))**

**missing.append(data[item].isna().sum())**

**pc\_missing.append(round((data[item].isna().sum() / len(data[item])) \* 100, 2))**

**output = pd.DataFrame({**

**'variable': variables,**

**'dtype': dtypes,**

**'count': count,**

**'unique': unique,**

**'missing': missing,**

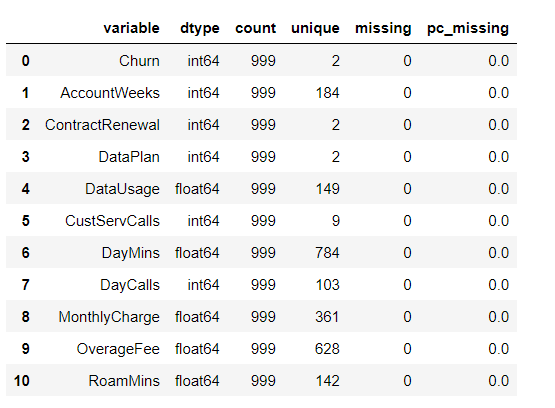
**'pc\_missing': pc\_missing**

**})**

**return output**

**show\_missing(data)**

**output:**

**show\_missing(data)**

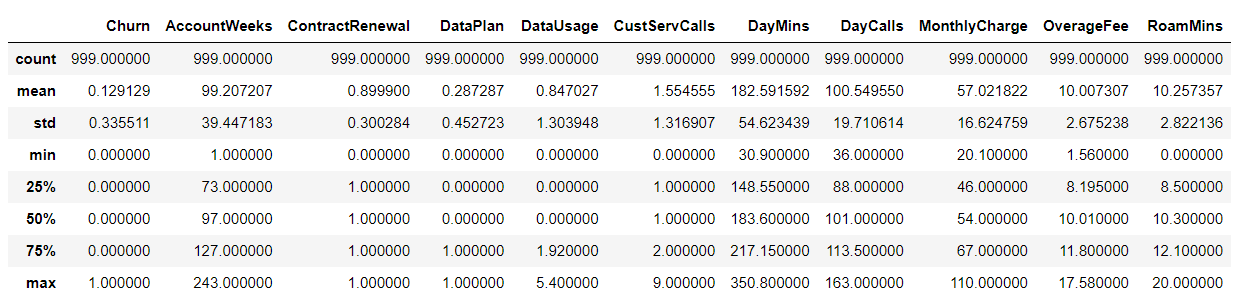
**conclusion:**

**Data has no missing values.**

1. **Describing the Data:**

**Code:** **data.describe()**

**Output:**

****

**Conclusion:**

**Here, DayMins, Daycalls, and Account weeks have higher Standard Deviations in comparison to other Independent variables.**

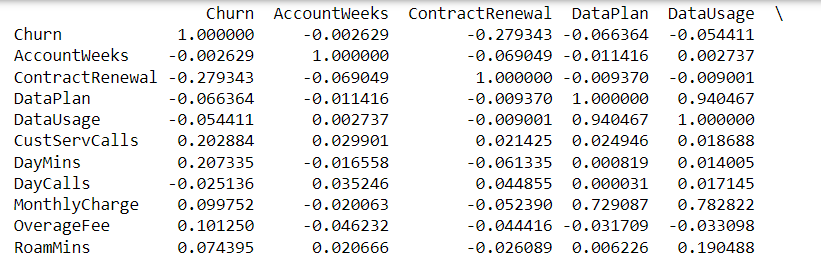
**5)Finding out the Correlation in the data**

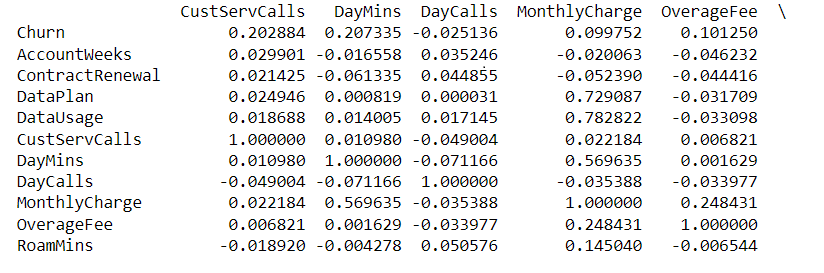
**Code:**

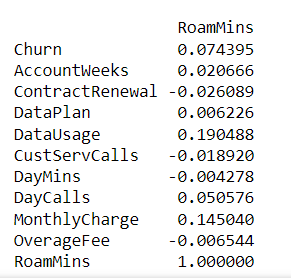
**a=(data.corr())**

**print(a)**

**Output:**

****

****

****

**6)Plotting a Heatmap for the correlation matrix:**

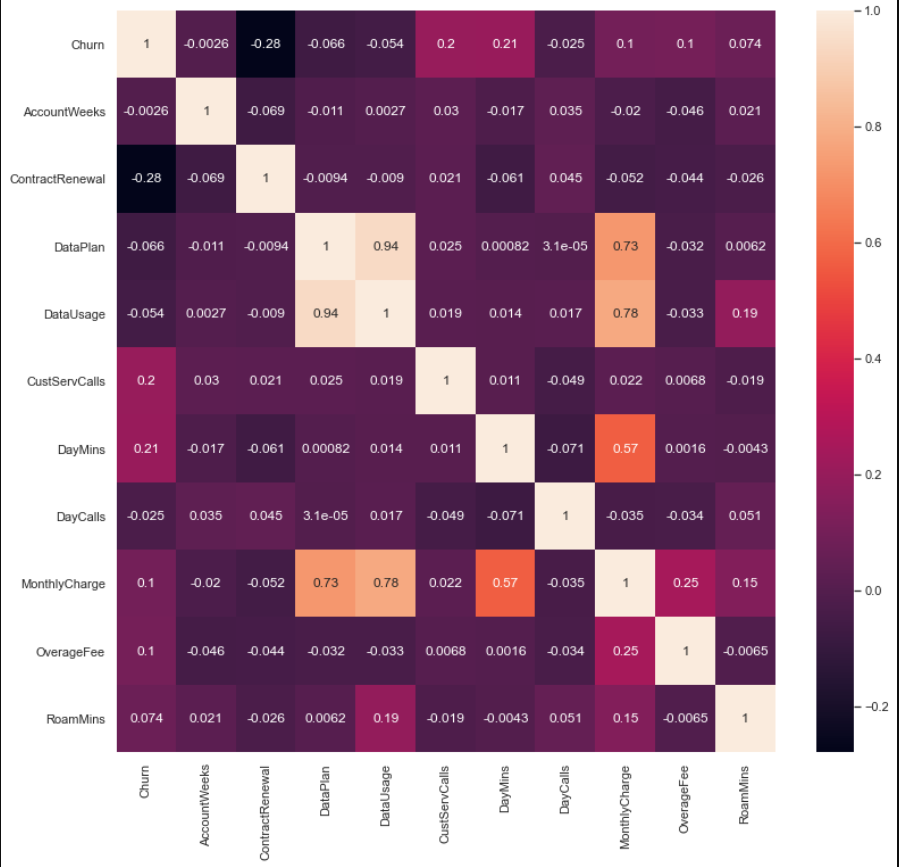
**Code:**

**import seaborn**

**seaborn.set (rc = {'figure.figsize':(13, 12)})**

**seaborn.heatmap(a,annot=True)**

**Output:**

****

**Conclusion:**

**From this heat map, we conclude that there exists less correlation between the dependent variable and other independent variables.**

**# imports**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import statsmodels.api as sm**

**import statsmodels.formula.api as smf**

**import seaborn as sns**

**# Fit a OLS regression variable**

**X=data[['AccountWeeks','DayMins','OverageFee','RoamMins','ContractRenewal','DataPlan','DataUsage','CustServCalls','DayCalls','MonthlyCharge']]**

**y=data[['Churn']]**

**x=sm.add\_constant(X)**

**results=sm.OLS(y,x).fit()**

**print(results.summary())**

**# Get different Variables for diagnostic**

**residuals = results.resid**

**fitted\_value = results.fittedvalues**

**stand\_resids = results.resid\_pearson**

**influence = results.get\_influence()**

**leverage = influence.hat\_matrix\_diag**

**# PLot different diagnostic plots**

**plt.rcParams["figure.figsize"] = (20,15)**

**fig, ax = plt.subplots(nrows=2, ncols=2)**

**plt.style.use('seaborn')**

**# Residual vs Fitted Plot**

**sns.scatterplot(x=fitted\_value, y=residuals, ax=ax[0, 0])**

**ax[0, 0].axhline(y=0, color='grey', linestyle='dashed')**

**ax[0, 0].set\_xlabel('Fitted Values')**

**ax[0, 0].set\_ylabel('Residuals')**

**ax[0, 0].set\_title('Residuals vs Fitted Fitted')**

**# Normal Q-Q plot**

**sm.qqplot(residuals, fit=True, line='45',ax=ax[0, 1], c='#4C72B0')**

**ax[0, 1].set\_title('Normal Q-Q')**

**# Scale-Location Plot**

**sns.scatterplot(x=fitted\_value, y=residuals, ax=ax[1, 0])**

**ax[1, 0].axhline(y=0, color='grey', linestyle='dashed')**

**ax[1, 0].set\_xlabel('Fitted values')**

**ax[1, 0].set\_ylabel('Sqrt(standardized residuals)')**

**ax[1, 0].set\_title('Scale-Location Plot')**

**# Residual vs Leverage Plot**

**sns.scatterplot(x=leverage, y=stand\_resids, ax=ax[1, 1])**

**ax[1, 1].axhline(y=0, color='grey', linestyle='dashed')**

**ax[1, 1].set\_xlabel('Leverage')**

**ax[1, 1].set\_ylabel('Sqrt(standardized residuals)')**

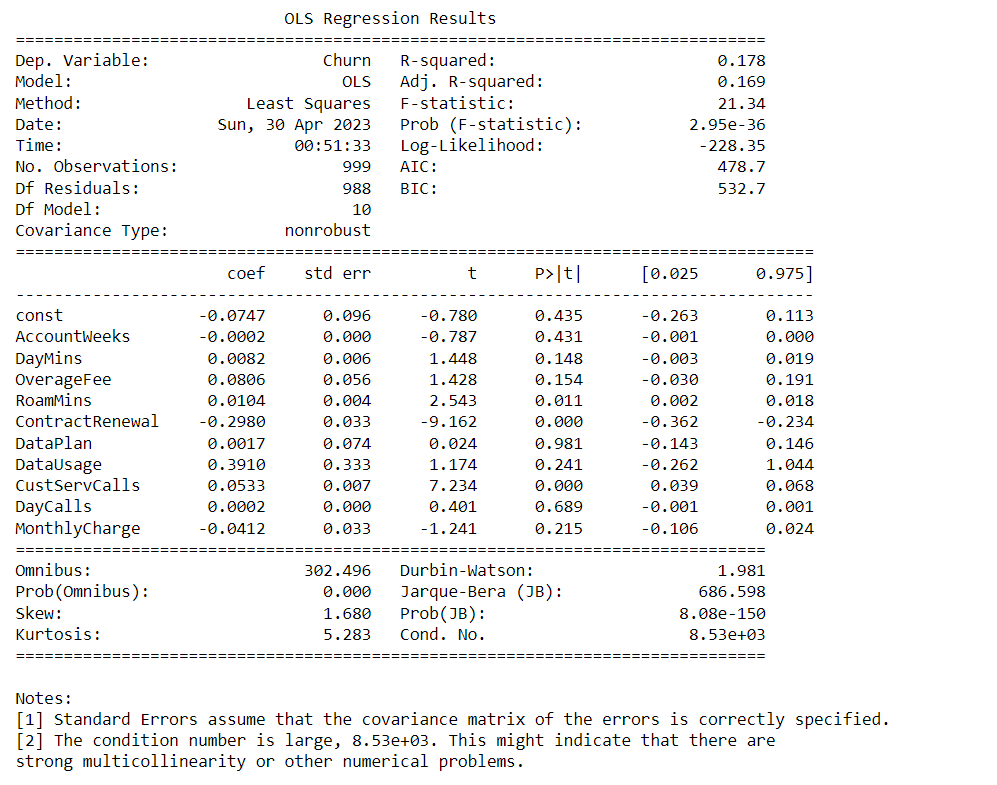
**ax[1, 1].set\_title('Residuals vs Leverage Plot')**

**plt.tight\_layout()**

**plt.show()**

**# PLot Cook's distance plot**

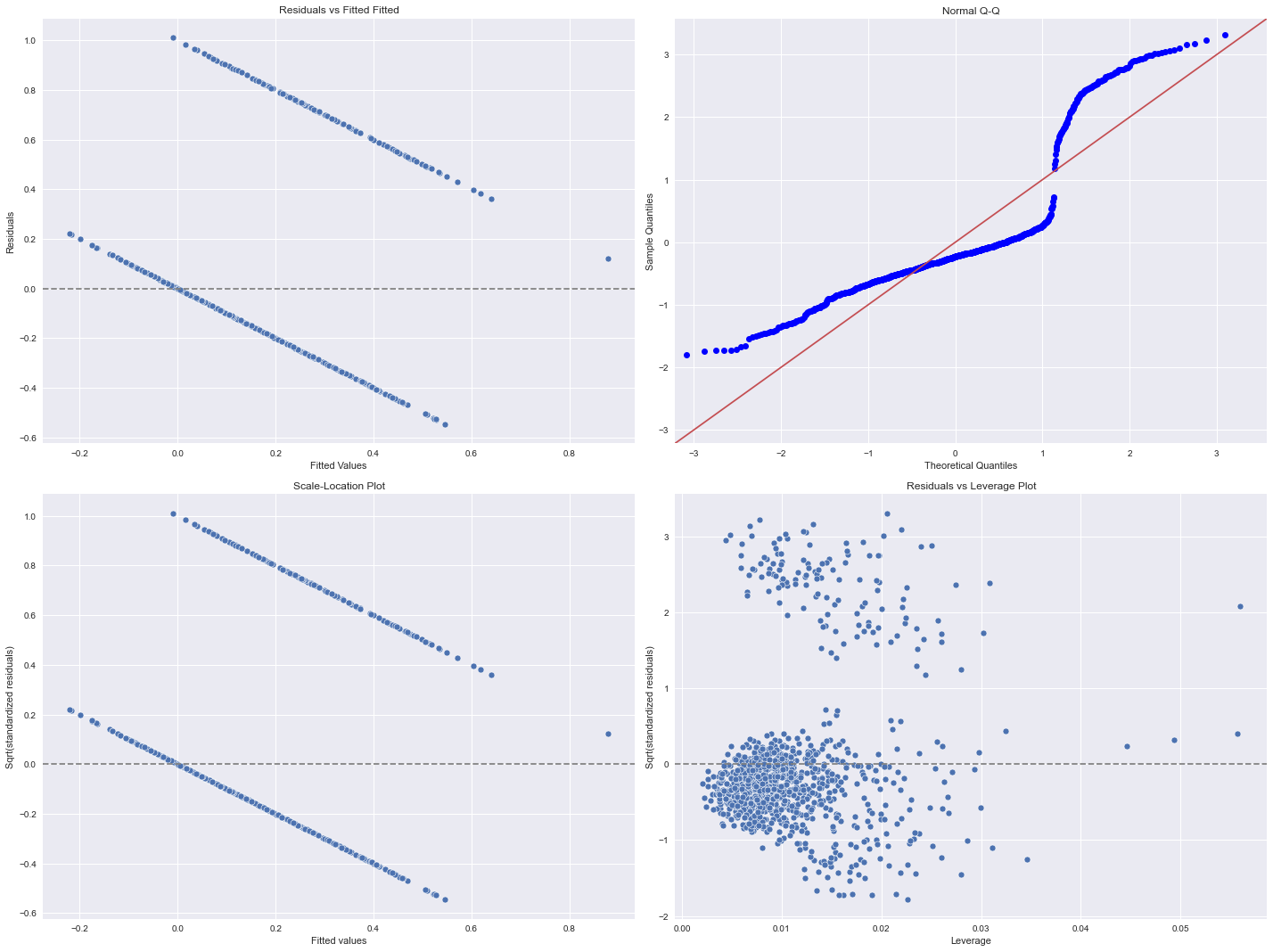
**sm.graphics.influence\_plot(results, criterion="cooks")**

****

**Conclusion:**

**Here, R-Square is 0.178 which is very less and hence it can be said that the model is inadequate. Here AIC is 478.7 which is more for a dataset with 1000 rows.**

**Hence this model is inadequate.**

****

**Conclusion:**

1. **Residual versus fitted plot**

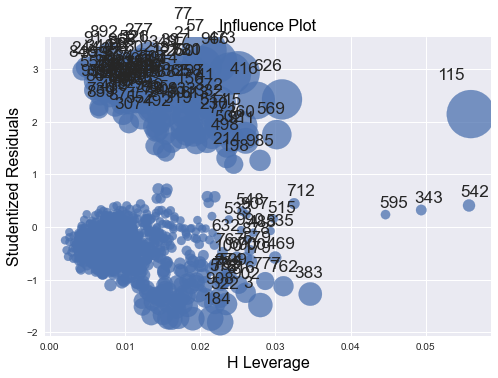
**we have those two lines of points because we predict a probability for a variable taking values 0 or 1. If the tree value is 0, residuals have to be negative and if the true value is 1, then we underestimate, and residuals are positive And of course, there is a monotone relationship except for some starting points.**

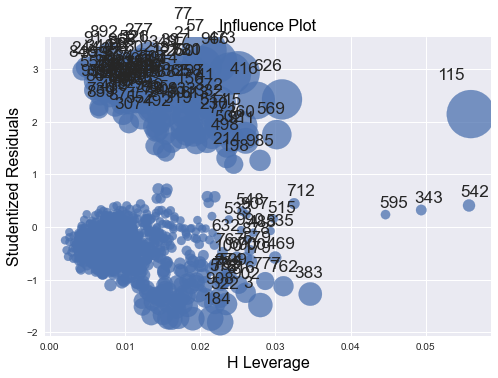
1. **From the Normal Q-Q plot it can be seen that the graph is not normal and is far more deviated from the normal line i.e. y=x which implies the number of outliers in the data needs to be fixed in order to obtain a linearity.**
2. **Scale location plot:**

The residuals are not randomly scattered around the red line with roughly equal variability at all fitted values.

1. **Residual versus leverage plot:**

**In this plot we can see too many data points are away from the cook’s distance, it implies that there are so many influential data points available.**

****

****

**To remove outliers from the normal Q-Q plot we try to detect the variables having outliers and try to reduce the data by removing the outliers and creating a new data frame with a reduced shape. Applying this to all the variables with outliers we concatenate all reduced data frames to form a new data frame i.e. data3 with all variables including the reduced one.**

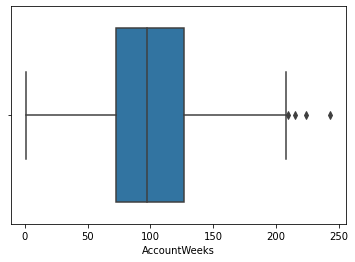
**7) Defining data frames, applying the IQR method to find outliers, and then again finding the shape of the data:**

**Command:**

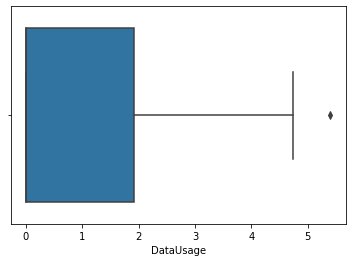
**import seaborn as sns**

**sns.boxplot(data['AccountWeeks'])**

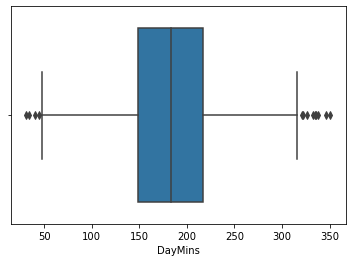
**output:**

****

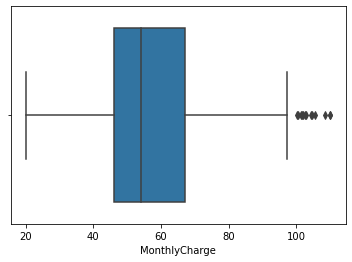
**sns.boxplot(data['DataUsage'])**

****

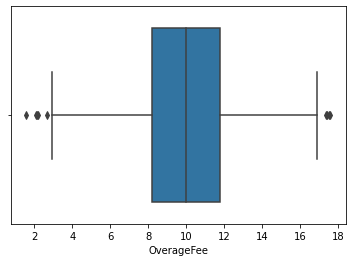
**sns.boxplot(data['DayMins'])**

****

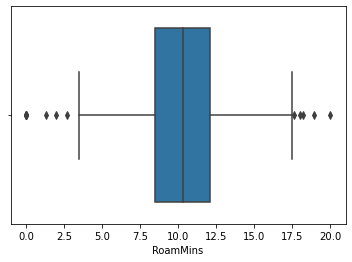
**sns.boxplot(data['MonthlyCharge'])**

****

**sns.boxplot(data['OverageFee'])**

****

**sns.boxplot(data['RoamMins'])**

****

**Q1=data['AccountWeeks'].quantile(0.25)**

**Q3=data['AccountWeeks'].quantile(0.75)**

**IQR=Q3-Q1**

**print(Q1)**

**print(Q3)**

**print(IQR)**

**Lower\_Whisker = Q1-1.5\*IQR**

**Upper\_Whisker = Q3+1.5\*IQR**

**print(Lower\_Whisker,Upper\_Whisker)**

**Output:**

73.0

127.0

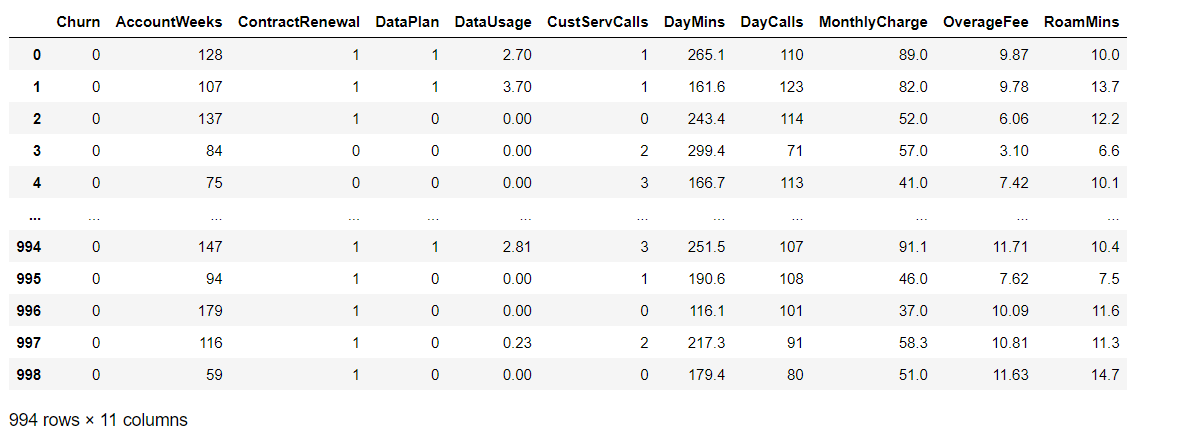
54.0

-8.0 208.0

**Command:**

**df1 = data[data['AccountWeeks']<Upper\_Whisker]**

**df1**

****

**df1.shape**

(997, 11)

**Conclusion: df1 represents the reduced size of the data after removing the outlier from the variable Account Weeks i.e. 999 is reduced to 997.**

**Q1=data['DayMins'].quantile(0.25)**

**Q3=data['DayMins'].quantile(0.75)**

**IQR=Q3-Q1**

**print(Q1)**

**print(Q3)**

**print(IQR)**

**Lower\_Whisker = Q1-1.5\*IQR**

**Upper\_Whisker = Q3+1.5\*IQR**

**print(Lower\_Whisker,Upper\_Whisker)**

**Output:**

148.55

217.14999999999998

68.59999999999997

45.65000000000006 320.04999999999995

**df2 =data[data['DayMins']<Upper\_Whisker]**

**df2=data[data['DayMins']>Lower\_Whisker]**

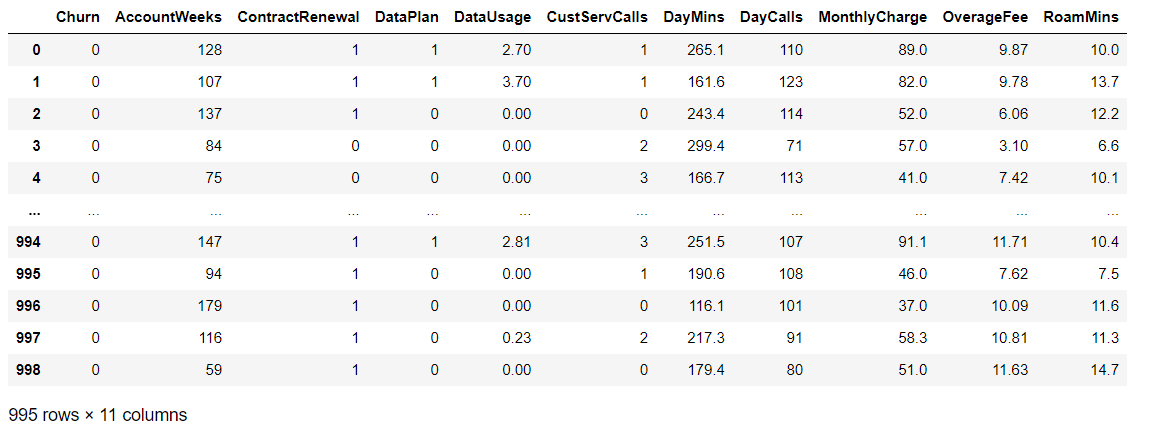
**df2.shape**

(995, 11)

**Conclusion: df2 represents the reduced size of the data after removing the outlier from the variable DayMins i.e. 999 is reduced to 995.**

**df2 = pd.DataFrame(df2)**

**df2**

****

**Q1=data['OverageFee'].quantile(0.25)**

**Q3=data['OverageFee'].quantile(0.75)**

**IQR=Q3-Q1**

**print(Q1)**

**print(Q3)**

**print(IQR)**

**Lower\_Whisker = Q1-1.5\*IQR**

**Upper\_Whisker = Q3+1.5\*IQR**

**print(Lower\_Whisker,Upper\_Whisker)**

**df3 = data[data['OverageFee']<Upper\_Whisker]**

**df3=data[data['OverageFee']>Lower\_Whisker]**

**df3.shape**

8.195

11.8

3.6050000000000004

2.7874999999999996 17.207500000000003

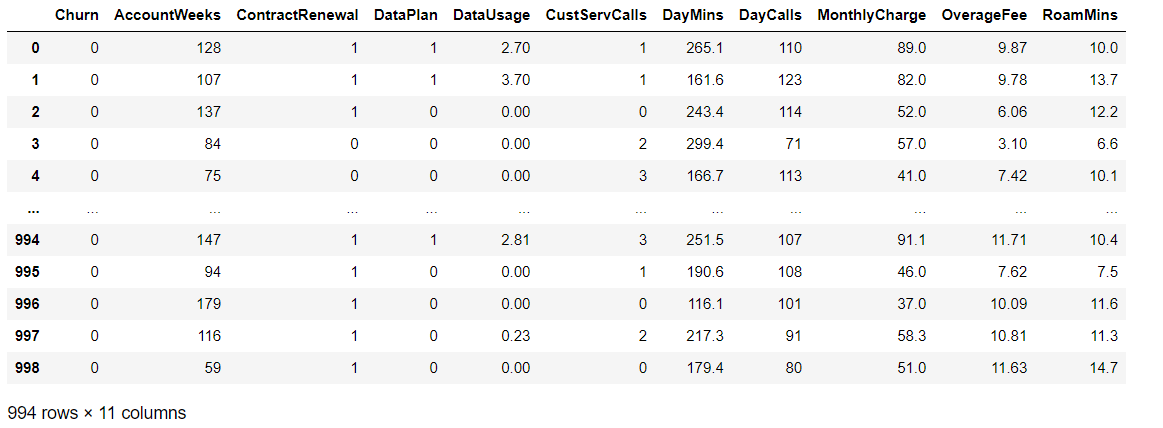
Out[37]:

(994, 11)

**Conclusion: df3 represents the reduced size of the data after removing the outlier from the variable OverageFee i.e. 999 is reduced to 994.**

**df3 = pd.DataFrame(df3)**

**df3**

****

**Q1=data['RoamMins'].quantile(0.25)**

**Q3=data['RoamMins'].quantile(0.75)**

**IQR=Q3-Q1**

**print(Q1)**

**print(Q3)**

**print(IQR)**

**Lower\_Whisker = Q1-1.5\*IQR**

**Upper\_Whisker = Q3+1.5\*IQR**

**print(Lower\_Whisker,Upper\_Whisker)**

**df4 = data[data['RoamMins']<Upper\_Whisker]**

**df4=data[data['RoamMins']>Lower\_Whisker]**

**df4.shape**

8.5

12.1

3.5999999999999996

3.1000000000000005 17.5

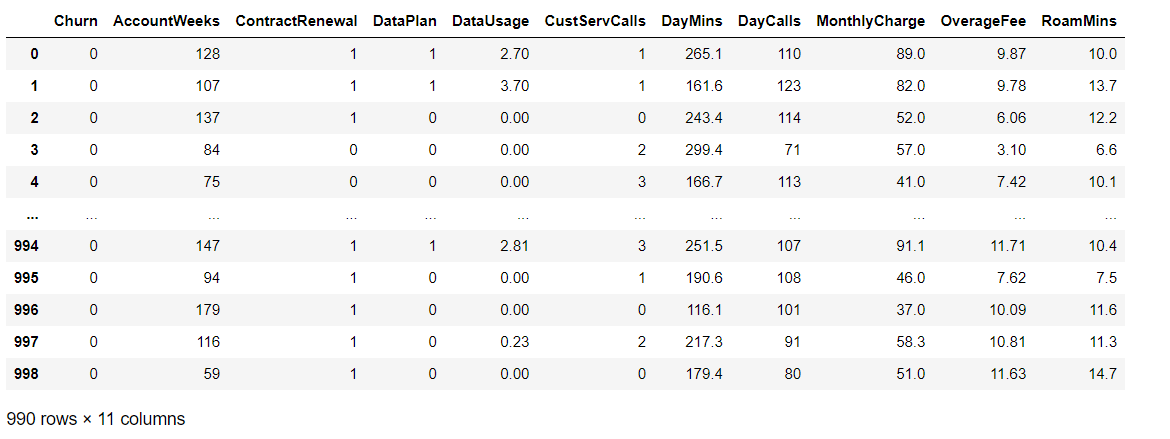
Out[39]:

(990, 11)

**Conclusion: df4 represents the reduced size of the data after removing the outlier from the variable DayMins i.e. 999 is reduced to 990.**

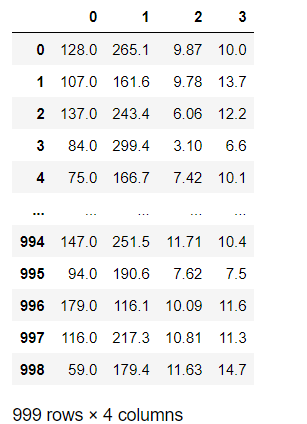
**df4 = pd.DataFrame(df4)**

**df4**

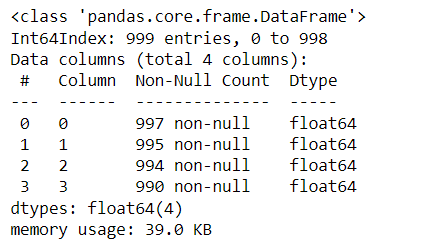
****

**data1=pd.concat([df1['AccountWeeks'],df2['DayMins'], df3['OverageFee'],df4['RoamMins']], axis=1, ignore\_index=True)**

**data1**

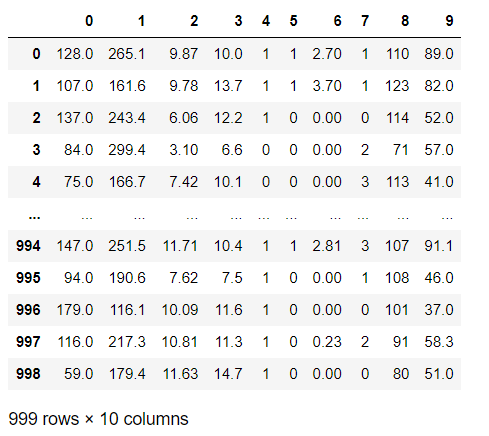
****

**data1.info()**

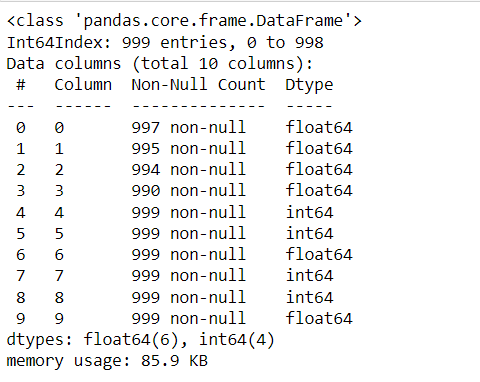
****

**data2=pd.concat([df1['AccountWeeks'],df2['DayMins'],df3['OverageFee'],df4['RoamMins'],data['ContractRenewal'],data['DataPlan'],data['DataUsage'],data['CustServCalls'],data['DayCalls'],data['MonthlyCharge']], axis=1, ignore\_index=True)**

**data2**

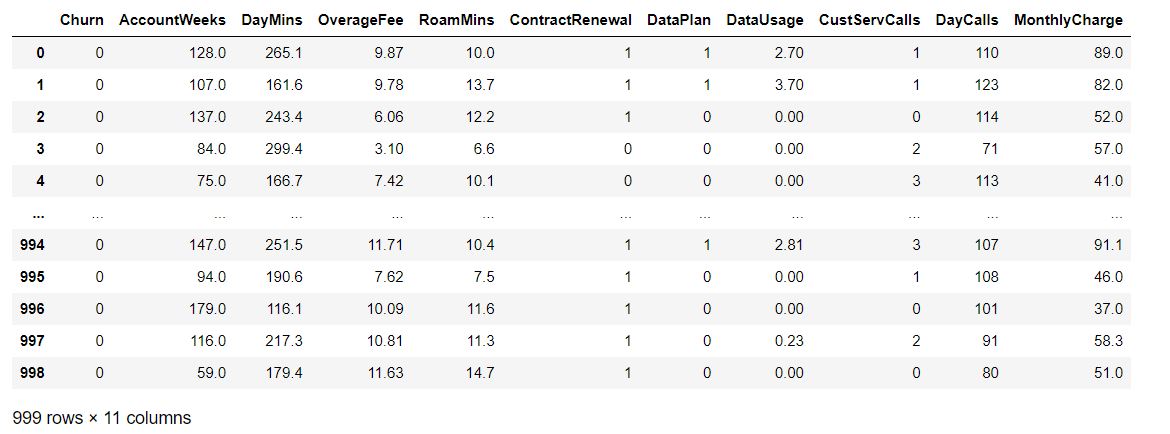
****

**data2.info()**

****

**data3=pd.concat([data['Churn'],df1['AccountWeeks'],df2['DayMins'],df3['OverageFee'],df4['RoamMins'],data['ContractRenewal'],data['DataPlan'],data['DataUsage'],data['CustServCalls'],data['DayCalls'],data['MonthlyCharge']], axis=1, ignore\_index=False)**

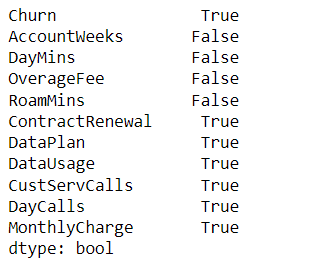
**data3**

****

**Command to find whether the data is infinite or finite**

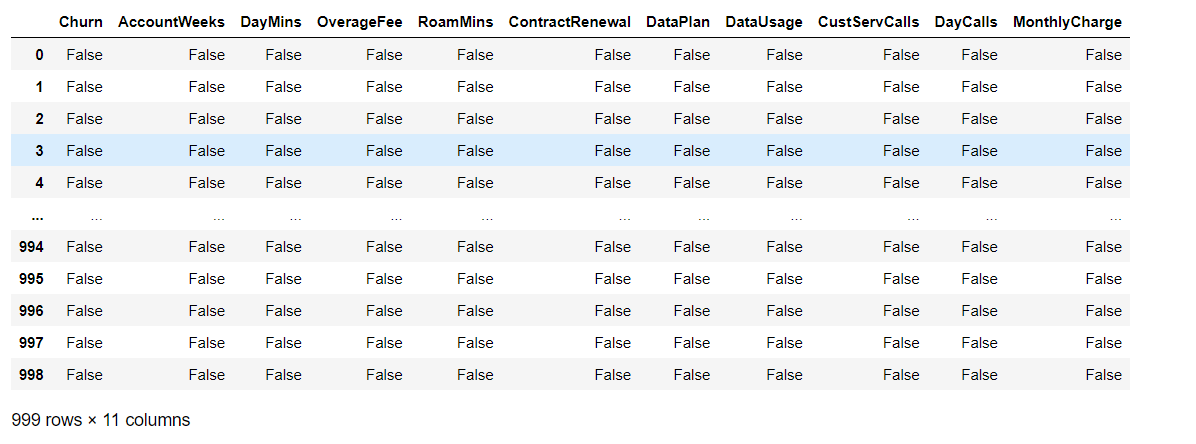
**import numpy**

**numpy.isfinite(data3).all()**

****

**Command to find whether data contains NAN values or not:**

**np.isnan(data3)**

****

**Command to replace infinite value with NAN:**

**# Replacing infinite with nan**

**data3.replace([np.inf, -np.inf], np.nan, inplace=True)**

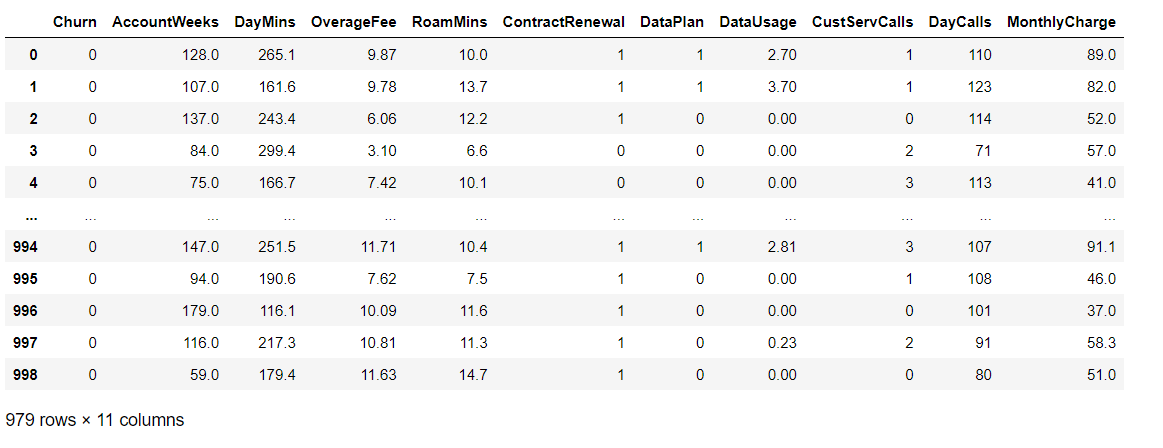
**Command to drop all the NAN values:**

**# Dropping all the rows with nan values**

**data3.dropna(inplace=True)**

**# Printing df**

**data3**

****

**Reduced size of the data after removing infinite values:**

**data3.shape**

(979, 11)

**Command to find the VIF:**

**1/(1-r\_square)**

**Output:**

8.024590163934427

**Conclusion: Also, the VIF is 8.0245 which is near 5 and hence it can be said that the variables are moderately multi-correlated.**

**Applying Logistic Regression to the Reduced Data with deleted outliers :**

**Command:**

**import numpy**

**from sklearn import linear\_model**

**#Reshaped for Logistic function.**

**X=data3[['AccountWeeks','DayMins','OverageFee','RoamMins','ContractRenewal',**

**'DataPlan','DataUsage','CustServCalls','DayCalls','MonthlyCharge']]**

**y=data3[['Churn']]**

**logr = linear\_model.LogisticRegression()**

**logr.fit(X,y)**

**Output:**

LogisticRegression()

**Command to find R square:**

**r\_square=logr.score(X,y)**

**r\_square**

**Output:**

0.8753830439223698

**Conclusion: Here, the Value of R -Square is 0.8753 which is nearly equal to 1, and hence the model is highly adequate after removing the outlier.**