**CUSTOMER CHURN PROJECT REPORT**

**By SREEKARI. I**

**INTRODUCTION**

Customer churn is when a company’s customers stop doing business with that company. Businesses are very keen on measuring churn because keeping an existing customer is far less expensive than acquiring a new customer. New business involves working leads through a sales funnel, using marketing and sales budgets to gain additional customers. Existing customers will often have a higher volume of service consumption and can generate additional customer referrals.

**Let us see why the churn is a big deal.** Customer acquisition cost (CAC) is the total cost of sales and marketing required to acquire a customer. The more customers you **churn**, the more money you must spend to recoup the loss of business by finding new ones.

Customer **churn** is an **important** metric because lost customers equal lost revenue. If a company loses enough customers it can have a serious impact on its bottom line. One of the studies by Bain & Company found that customer acquisition costs to find a new customer are five times higher than the **cost** of keeping an existing customer.

So having a robust and accurate churn prediction model helps businesses to take actions to prevent customers from leaving the company.

**PROBLEM STATEMENT**

Customer retention can be achieved with good customer service and products. But the most effective way for a company to prevent attrition of customers is to truly know them. The vast volumes of data collected about customers can be used to build churn prediction models. Knowing who is most likely to defect means that a company can prioritise focused marketing efforts on that subset of their customer base.

Preventing customer churn is critically important to the telecommunications sector, as the barriers to entry for switching services are so low.

In this problem, we are going to examine customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models.

**DATA ANALYSIS**

Let us import the required libraries that helps in data analysis.

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **sklearn**

**import** **seaborn** **as** **sns**

**import** **matplotlib.pyplot** **as** **plt**

**import** **warnings**

warnings.filterwarnings('ignore')

pd.set\_option('display.max\_columns', **None**)

I have created a new text file in jupyter notebook with the raw data and named it as ‘customer\_churn\_analysis’. Importing the file now.

df=pd.read\_csv("customer\_churn\_analysis")

**EXPLORATORY DATA ANALYSIS**

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modeling the data in order to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important.

Let us explore the data we have now

We can check the shape of the data i.e., number of row and columns using df.shape

By checking the shape, I see that there are  7043 rows and 21 columns in the dataset.

Following are the columns of the dataset.

'customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService','OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport','StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling','PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'

Here, the **target column** is ‘**Churn’**.

Upon checking the data information, df.info() we can that  there is one column of datatype float64, two columns of int64 and 18 columns of object datatype.

<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

This describes the information of the the columns of the dataset. It shows the column names, count and datatype.

Now **let us check if there are any null values** in the dataset because the null values affect the model predictions.

We can check the null values from the following function. This shows the sum of the null values in the columns of the dataset.

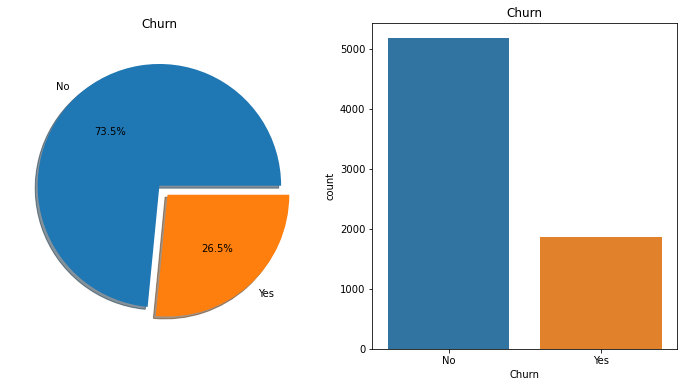
df.isnull().sum()

In the given dataset, there are no null values.

**Analysing the data**

Here I am going to analyse various columns which can help in prediction model.

We can see that 1869 customers stopped using the service(i.e., there is churn) and 5174 are continuing the services with the company(there is no churn).



* Pie chart showing the percentage of Customers who left the company and who are with the company.  
  --We can see that there is 26.5 percent churn.
* Barplot shows the number of customers who left the company and who are with the company

--🡪 Checking Churn by Gender

We can see that there are 3555 male customers and 3488 female customers.



From the above barplot, we can see that Churn is almost same in Male and Female customers.

--🡪 **Checking Churn by Partner**

From the dataset, we can see that there are partners for 3402 customers and 3641 customers have no partners.

We can see that the churn is customers with no partners is little higher than customers with partners.



-🡪 **Checking Churn by Dependents**

We can see that 2110 customers have dependants and 4933 have no dependants.



The percentage of Churn in Customers with Dependants is lesser than the customers without Depandants.

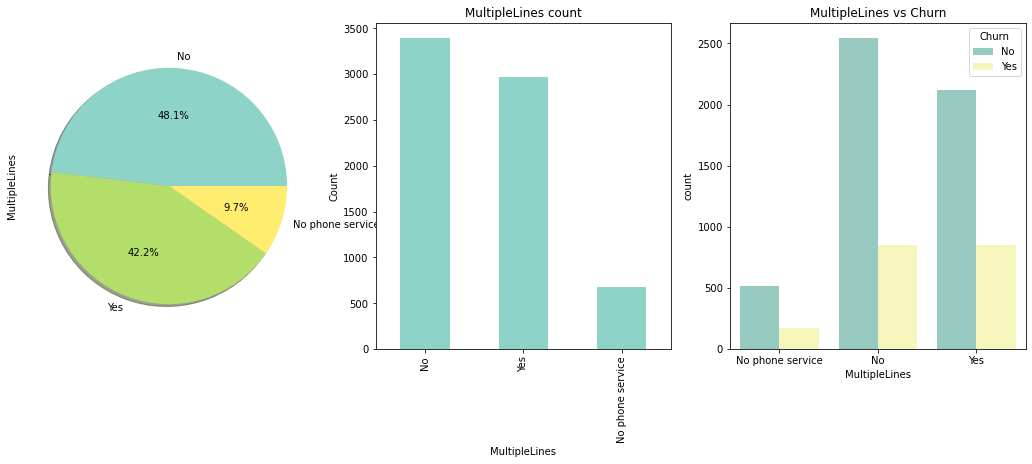
--🡪 **Checking Churn by PhoneService**

6361 customers have Phone Service and 682 customers do not have phone service.

We can see that the percentage of churn rate is almost the same in customers with phone service and without phone service.

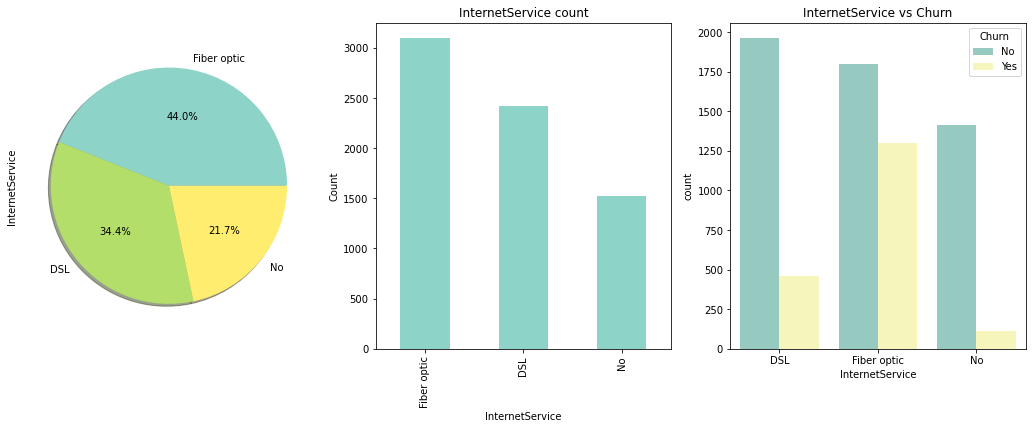


-🡪 **Checking Churn by MultipleLines**



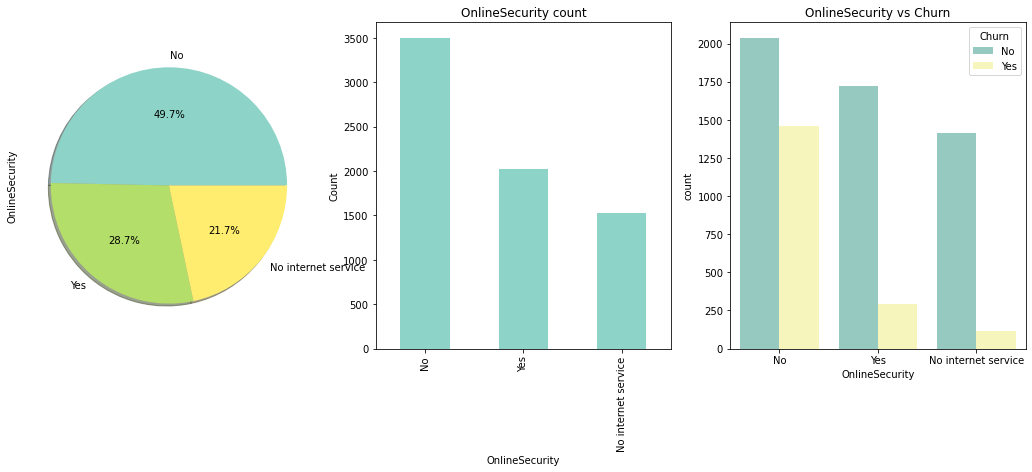
There are 9.7% customers with No Phone Service, 48.1% with phone service but no multiple lines and 42.2% with multiple lines.

-🡪 **Checking Churn by InternetService**



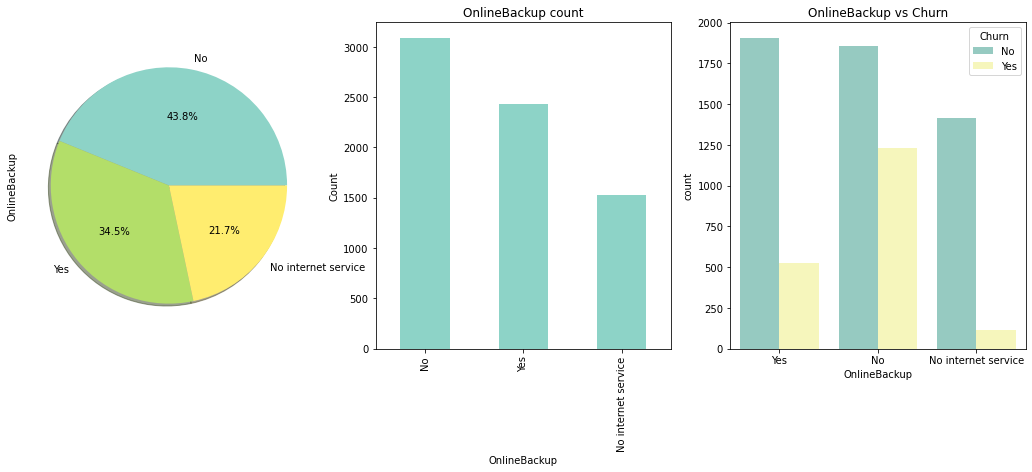
We can see that there are 44.0% customers with Fiber optic and 21.7% with no InternetService and 34.4% with DSL InternetService.

--🡪 **Checking Churn in OnlineSecurity**



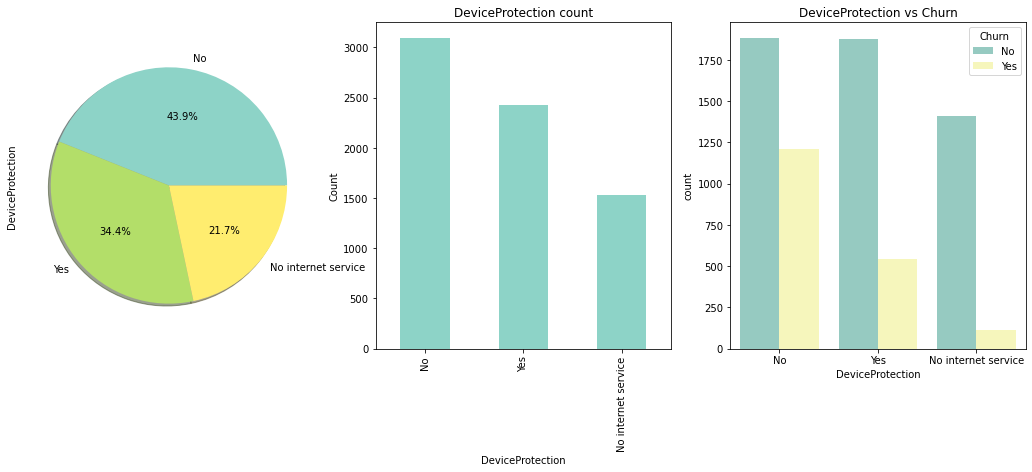
There are 49.7% customers with no OnlineSecurity, 28.7% with OnlineSecurity and 21.7% with no InternetService.

-🡪 **Checking Churn in OnlineBackup**



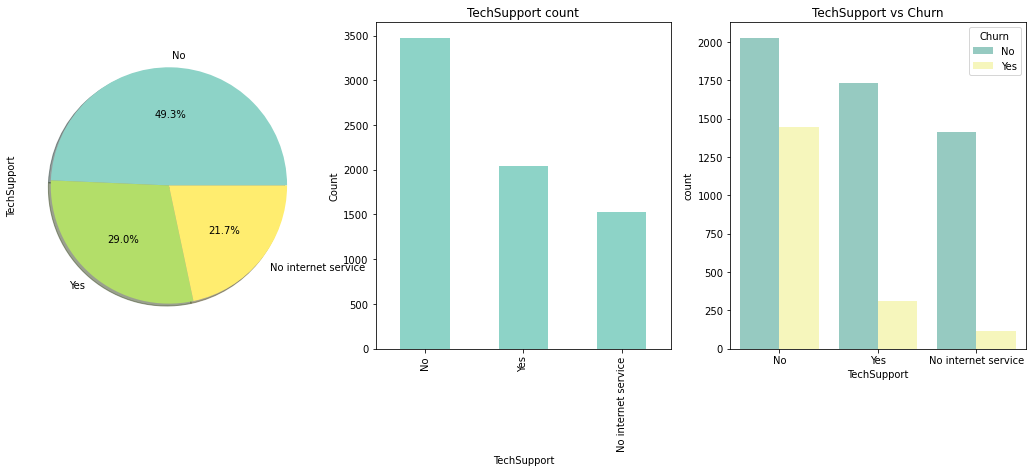
The percentage of churn is little higher in customers with no OnlineBackup when compared to the customers with OnlineBackup.

--🡪 **Checking Churn in DeviceProtection**



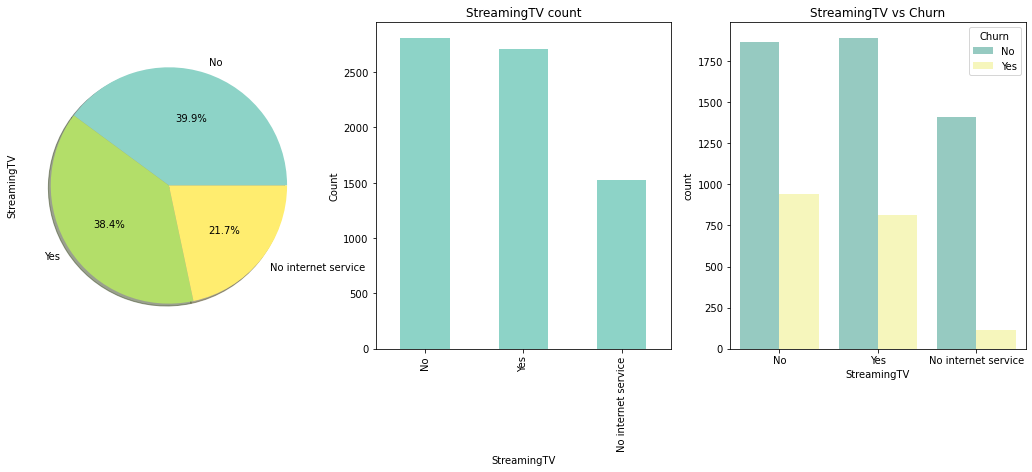
We can see that the churn rate is higher in customers with no DeviceProtection.

--🡪 **Checking Churn in TechSupport**

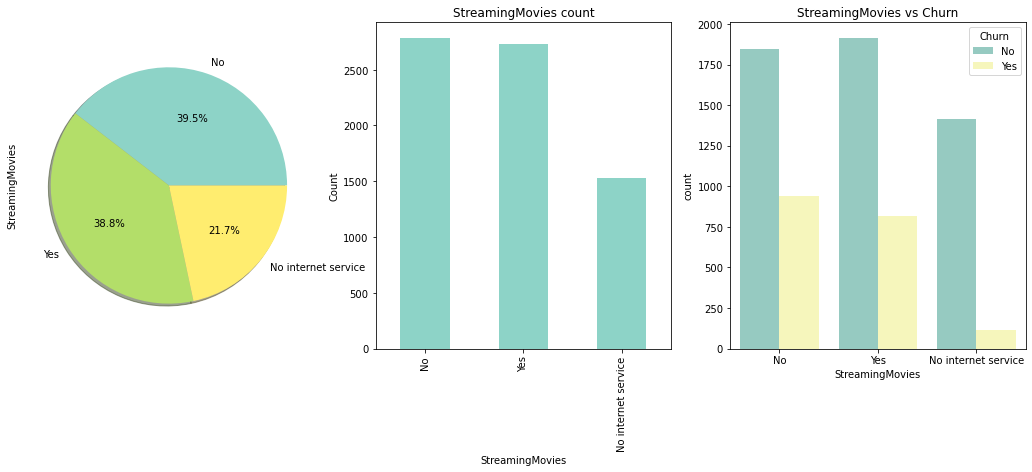


The percentage of Churn in customers with no TechSupport is higher when compared to the customers with TechSupport.

**--🡪 Checking Churn in StramingTV**

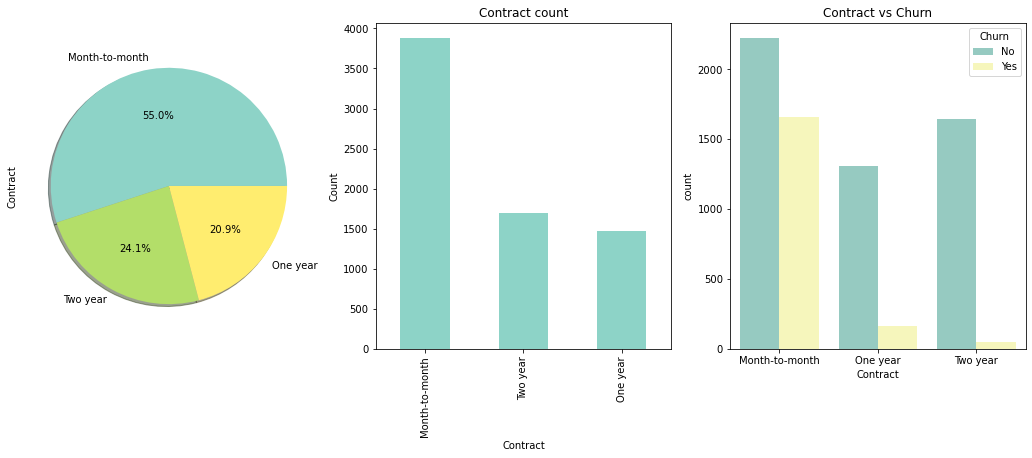


-🡪 **Checking Churn by StreamingMovies**



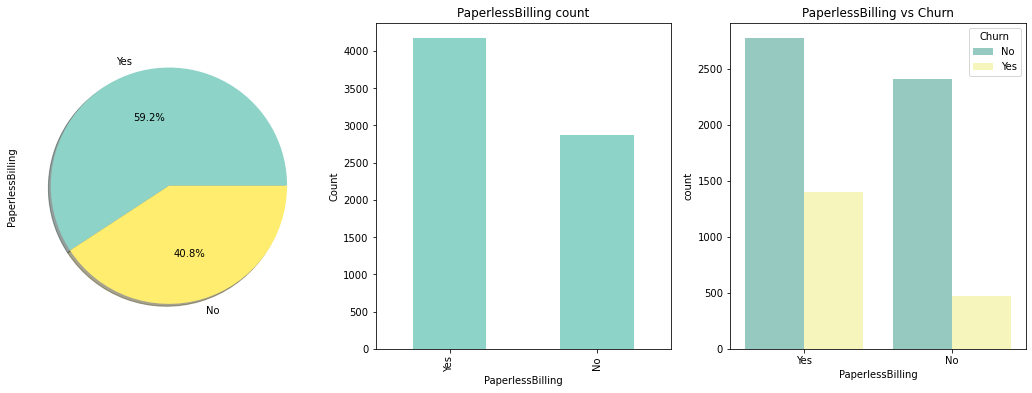
The churn rate is almost the same in terms of StreamingMovies.

**-🡪 Checking Churn by Contract**

****

We can see that the churn is lesser in longer contract periods when compared to Month-to-Month contract.

**-🡪** **Checking Churn by PaperlessBilling**

****

The churn rate is lesser in no PaperlessBilling when compared with the customers who has used PaperlessBilling.

**We have analysed the data in the various columns against the target column.**

**DATA PRE-PROCESSING**

Let us process thedata now. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

We can see that there is categorical data many columns of the dataset. We need to convert it to numerical data. I am applying the following code to replace the categorical data with numerical data.

df['Churn'].replace({"No": 0, "Yes": 1}, inplace = **True**)

df['gender'].replace({"Female": 0, "Male": 1}, inplace = **True**)

df['Partner'].replace({"No": 0, "Yes": 1}, inplace = **True**)

df['Dependents'].replace({"No": 0, "Yes": 1}, inplace = **True**)

df['PhoneService'].replace({"No": 0, "Yes": 1}, inplace = **True**)

df['MultipleLines'].replace({"No": 0, "Yes": 1, "No phone service":2}, inplace = **True**)

df['InternetService'].replace({"No": 0, "DSL": 1, "Fiber optic":2}, inplace = **True**)

df['OnlineSecurity'].replace({"No": 0, "Yes": 1, "No internet service":2}, inplace = **True**)

df['OnlineBackup'].replace({"No": 0, "Yes": 1, "No internet service":2}, inplace = **True**)

df['DeviceProtection'].replace({"No": 0, "Yes": 1, "No internet service":2}, inplace = **True**)

df['TechSupport'].replace({"No": 0, "Yes": 1, "No internet service":2}, inplace = **True**)

df['StreamingTV'].replace({"No": 0, "Yes": 1, "No internet service":2}, inplace = **True**)

df['StreamingMovies'].replace({"No": 0, "Yes": 1, "No internet service":2}, inplace = **True**)

df['Contract'].replace({"Month-to-month": 0, "One year": 1, "Two year":2}, inplace = **True**)

df['PaperlessBilling'].replace({"No": 0, "Yes": 1}, inplace = **True**)

df['PaymentMethod'].replace({"Electronic check": 0, "Mailed check": 1, "Bank transfer (automatic)":2, "Credit card (automatic)":3}, inplace = **True**)

**-🡪Dropping the columns that are not useful in prediction.**

We can see that the percentage senior citizens is less when compared to non-SeniorCitizen in the data. This is not going to affect the data much. Hence dropping this column.

*# dropping SeniorCitizen column*

df=df.drop('SeniorCitizen', axis=1)

We know that the total charges are the sum of the charges for the services taken multiplied by number of months(tenure) for which the service is availed. As we have all the other information, dropping this column.

*# dropping TotalCharges column*

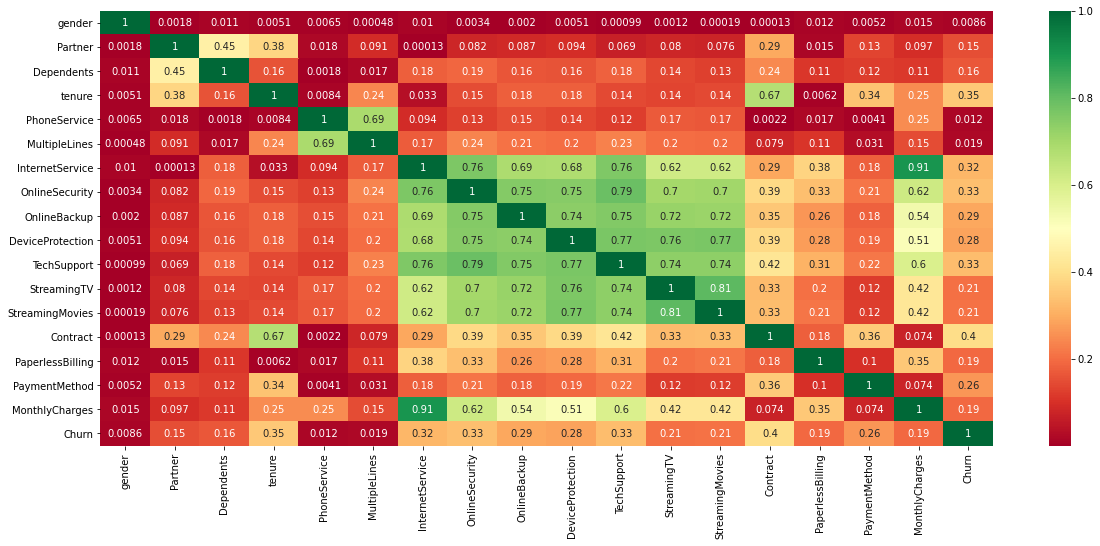
df=df.drop('TotalCharges', axis=1)

Also, we know that the customerID do not affect the prediction of the model. So dropping this column.

*# dropping customerID column*

df=df.drop('customerID', axis=1)

**-🡪 Checking CORRELATION**

****

The above **heat map** is a two-dimensional representation of information with the help of colors. We can see the correlation between the columns with each other here.

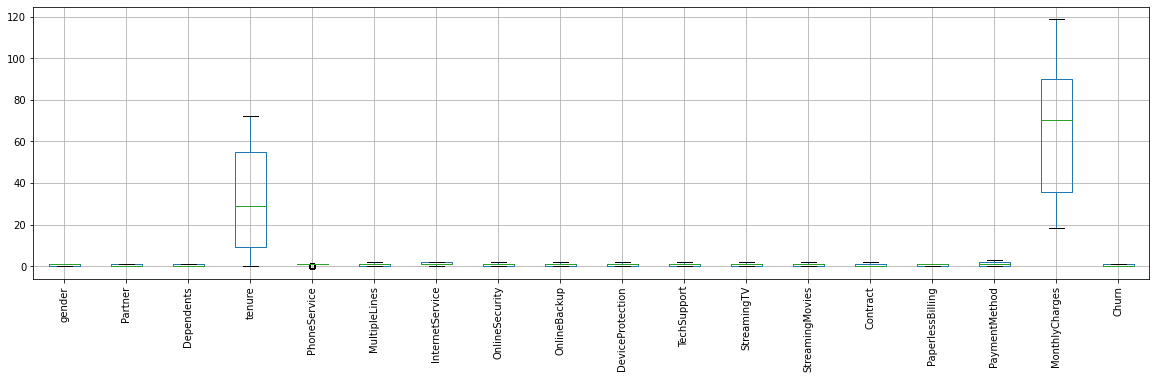
SKEWNESS

df.skew()

We can see that PhoneService has highest skewness followed by Churn while the skewness is least in gender followed by 0.067922. Most of the data of this dataset varies between no skewness to moderate skewness.

Let us check for **outliers** now.

df.boxplot( rot = 90, figsize=(20,5))



We can see that there are no considerable outliers in the data.

As we have analysed the data and processed the necessary information, lets move to creating the model.

PREDICTIVE MODELLING

Importing the required libraries.

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn** **import** svm

**from** **sklearn.ensemble** **import** RandomForestClassifier

**from** **sklearn.neighbors** **import** KNeighborsClassifier

**from** **sklearn.naive\_bayes** **import** GaussianNB

**from** **sklearn.tree** **import** DecisionTreeClassifier

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** metrics

**from** **sklearn.metrics** **import** confusion\_matrix

**from** **sklearn.model\_selection** **import** GridSearchCV

**Splitting data for validation**

X=df.drop(['Churn'], axis=1)

y=df['Churn']

Here, I am splitting the data in 70% training and 30% for testing.

X\_train, X\_test,y\_train,y\_test = train\_test\_split(X,y, test\_size = 0.3, random\_state=42)

Checking the training data shape.

print(X\_train.shape, y\_train.shape)

Output: (4930, 17) (4930,)

We can see that there are 4930 rows and 17 columns in the X\_train and 4930 rows in y\_train.

**-----> Checking Accuracies**

model\_log = LogisticRegression(solver='liblinear')

model\_log.fit(X\_train, y\_train)

prediction\_log = model\_log.predict(X\_test)

model\_tree = DecisionTreeClassifier()

model\_tree.fit(X\_train, y\_train)

prediction\_tree = model\_tree.predict(X\_test)

model\_knn = KNeighborsClassifier()

model\_knn.fit(X\_train, y\_train)

prediction\_knn = model\_knn.predict(X\_test)

model\_svm\_l = svm.SVC(kernel='linear', C=0.1, gamma=0.1)

model\_svm\_l.fit(X\_train, y\_train)

prediction\_svm\_l = model\_svm\_l.predict(X\_test)

model\_random = RandomForestClassifier(n\_estimators=300)

model\_random.fit(X\_train, y\_train)

predict\_random = model\_random.predict(X\_test)

model\_gaus = GaussianNB()

model\_gaus.fit(X\_train, y\_train)

prediction\_gaus = model\_gaus.predict(X\_test)

print('The accuracy of the Logistic Regression is',metrics.accuracy\_score(prediction\_log, y\_test))

print('The accuracy of the Decision Tree is ', metrics.accuracy\_score(prediction\_tree, y\_test))

print('The accuracy of the K-Nearest Neighbours is ', metrics.accuracy\_score(prediction\_knn, y\_test))

print('The accuracy of the Linear Support Vector Machine is ', metrics.accuracy\_score(prediction\_svm\_l, y\_test))

print('The accuracy of the Random Forest is ', metrics.accuracy\_score(predict\_random, y\_test))

print('The accuracy of the Gaussian Naive Bayes is ', metrics.accuracy\_score(prediction\_gaus, y\_test))

The accuracy of the Logistic Regression is 0.8106956933270232

The accuracy of the Decision Tree is 0.7269285376242309

The accuracy of the K-Nearest Neighbours is 0.7605300520586843

The accuracy of the Linear Support Vector Machine is 0.8092759110269758

The accuracy of the Random Forest is 0.7875059157595835

The accuracy of the Gaussian Naive Bayes is 0.7079981069569332

**-----> Checking Cross Validation Scores**

**from** **sklearn.model\_selection** **import** KFold

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.model\_selection** **import** cross\_val\_predict

**from** **sklearn.model\_selection** **import** cross\_val\_score

scr1=cross\_val\_score(model\_log,X,y,cv=5)

scr2=cross\_val\_score(model\_tree,X,y,cv=5)

scr3=cross\_val\_score(model\_knn,X,y,cv=5)

scr4=cross\_val\_score(model\_svm\_l,X,y,cv=5)

scr5=cross\_val\_score(model\_random,X,y,cv=5)

scr6=cross\_val\_score(model\_gaus,X,y,cv=5)

print("Cross Validation Score of Logistic Regression Model:", scr1.mean())

print("Cross Validation Score of Decision Tree Model:", scr2.mean())

print("Cross Validation Score of K-Nearest Neighbours Model:", scr3.mean())

print("Cross Validation Score of Linear Support Vector Machine Model:", scr4.mean())

print("Cross Validation Score of Random Forest Model:", scr5.mean())

print("Cross Validation Score of Gaussian Naive Bayes Model:", scr6.mean())

Output:

Cross Validation Score of Logistic Regression Model: 0.8027837481450417

Cross Validation Score of Decision Tree Model: 0.7232702513065359

Cross Validation Score of K-Nearest Neighbours Model: 0.7728230450351635

Cross Validation Score of Linear Support Vector Machine Model: 0.8006529655139042

Cross Validation Score of Random Forest Model: 0.785035728111491

Cross Validation Score of Gaussian Naive Bayes Model: 0.7004112160461966

---> We can see that the difference between Accuracy and Cross Validation Score is almost the same for all the models.

**Confusion Matrix for the above models**

f, ax =plt.subplots(2,3, figsize=(12,10))

y\_pred = cross\_val\_predict(svm.SVC(kernel='linear'),X,y,cv=10)

sns.heatmap(confusion\_matrix(y,y\_pred), ax=ax[0,0], annot=**True**,fmt='2.0f')

ax[0,0].set\_title('Linear SVM')

y\_pred = cross\_val\_predict(KNeighborsClassifier(n\_neighbors=9) ,X,y,cv=10)

sns.heatmap(confusion\_matrix(y,y\_pred), ax=ax[0,1], annot=**True**,fmt='2.0f')

ax[0,1].set\_title('KNN')

y\_pred = cross\_val\_predict(LogisticRegression(solver='liblinear') ,X,y,cv=10)

sns.heatmap(confusion\_matrix(y,y\_pred), ax=ax[0,2], annot=**True**,fmt='2.0f')

ax[0,2].set\_title('Logistic Regression')

y\_pred = cross\_val\_predict(RandomForestClassifier(n\_estimators=100) ,X,y,cv=10)

sns.heatmap(confusion\_matrix(y,y\_pred), ax=ax[1,0], annot=**True**,fmt='2.0f')

ax[1,0].set\_title('Random Forest')

y\_pred = cross\_val\_predict(DecisionTreeClassifier() ,X,y,cv=10)

sns.heatmap(confusion\_matrix(y,y\_pred), ax=ax[1,1], annot=**True**,fmt='2.0f')

ax[1,1].set\_title('Decision Tree')

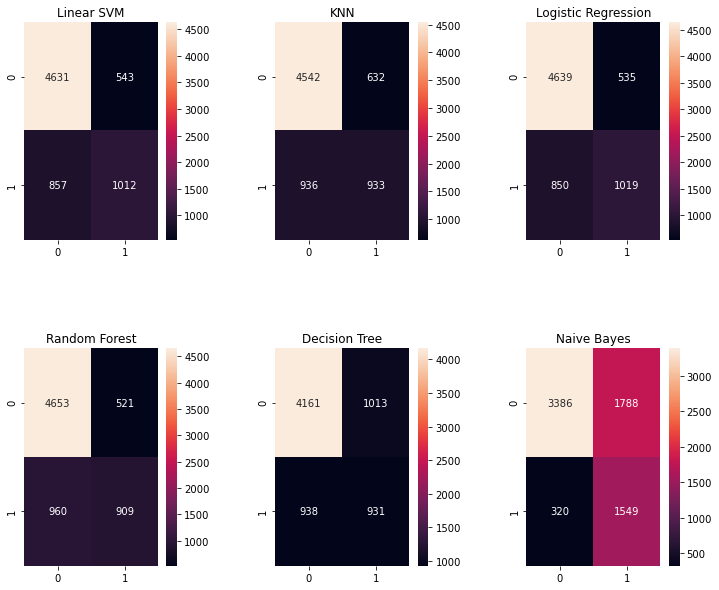
y\_pred = cross\_val\_predict(GaussianNB() ,X,y,cv=10)

sns.heatmap(confusion\_matrix(y,y\_pred), ax=ax[1,2], annot=**True**,fmt='2.0f')

ax[1,2].set\_title('Naive Bayes')

plt.subplots\_adjust(hspace=0.5, wspace=0.5)

plt.show()



We can see that Random Forest Model has a higher chance in correctly predicting the churn.  
---> 4647 for non Churn and 917 for Churn predicted correctly.

ROC CURVE

y\_pred\_proba\_RF = model\_random.predict\_proba(X\_test)[::,1]

fpr1, tpr1, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba\_RF)

auc1 = metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_RF)

y\_pred\_proba\_DT = model\_tree.predict\_proba(X\_test)[::,1]

fpr2, tpr2, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba\_DT)

auc2 = metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_DT)

y\_pred\_proba\_NB = model\_gaus.predict\_proba(X\_test)[::,1]

fpr3, tpr3, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba\_NB)

auc3 = metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_NB)

y\_pred\_proba\_LR = model\_log.predict\_proba(X\_test)[::,1]

fpr4, tpr4, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba\_LR)

auc4 = metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_LR)

y\_pred\_proba\_KNN = model\_knn.predict\_proba(X\_test)[::,1]

fpr5, tpr5, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba\_KNN)

auc5 = metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_KNN)

plt.figure(figsize=(10,7))

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr1,tpr1,label="Random Forest, auc="+str(round(auc1,2)))

plt.plot(fpr2,tpr2,label="Decision Tree, auc="+str(round(auc2,2)))

plt.plot(fpr3,tpr3,label="Naive Bayes, auc="+str(round(auc3,2)))

plt.plot(fpr4,tpr4,label="Logistic Regression), auc="+str(round(auc4,2)))

plt.plot(fpr5,tpr5,label="KNN), auc="+str(round(auc5,2)))

plt.legend(loc=3, title='Models', facecolor='white')

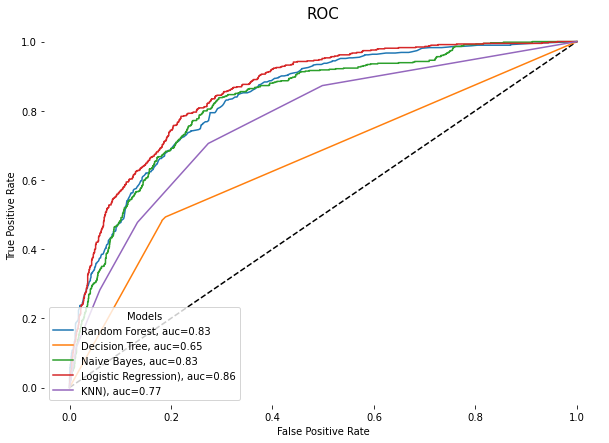
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC', size=15)

plt.box(**False**)

plt.savefig('ImageName', format='png', dpi=200, transparent=**True**);



### HYPER PARAMETER TUNING

**---> Random Forest Model**

n\_estimator =range(50, 1000, 50)

hyper = {'n\_estimators': n\_estimator}

gd = GridSearchCV(estimator=RandomForestClassifier(random\_state=0), param\_grid=hyper, verbose=**True**)

gd.fit(X,y)

print(gd.best\_score\_)

print(gd.best\_estimator\_)

Fitting 5 folds for each of 19 candidates, totalling 95 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 95 out of 95 | elapsed: 11.7min finished

0.7892946722369185

RandomForestClassifier(n\_estimators=750, random\_state=0)

gd.best\_params\_

{'n\_estimators': 750}

**from** **sklearn.metrics** **import** r2\_score, mean\_squared\_error

best\_reg = gd.best\_estimator\_

y\_pred = best\_reg.predict(X\_test) final\_mse = mean\_squared\_error(y\_test, y\_pred)

final\_rmse = np.sqrt(final\_mse) final\_rmse

0.06526367213191407

**--->Logistic Regression**

*# Necessary imports*

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.model\_selection** **import** GridSearchCV

*# Creating the hyperparameter grid*

c\_space = np.logspace(-5, 8, 15)

param\_grid = {'C': c\_space}

*# Instantiating logistic regression classifier*

logreg = LogisticRegression()

*# Instantiating the GridSearchCV object*

logreg\_cv = GridSearchCV(logreg, param\_grid, cv = 5)

logreg\_cv.fit(X, y)

*# Print the tuned parameters and score*

print("Tuned Logistic Regression Parameters: **{}**".format(logreg\_cv.best\_params\_))

print("Best score is **{}**".format(logreg\_cv.best\_score\_))

Tuned Logistic Regression Parameters: {'C': 0.006105402296585327}

Best score is 0.8023576117814052

**--->Decision Tree Regression Model**

parameters={"splitter":["best","random"],

"max\_depth" : [1,3,5,7,9,11,12],

"min\_samples\_leaf":[1,2,3,4,5,6,7,8,9,10],

"min\_weight\_fraction\_leaf":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],

"max\_features":["auto","log2","sqrt",**None**],

"max\_leaf\_nodes":[**None**,10,20,30,40,50,60,70,80,90] }

tuning\_model=GridSearchCV(model\_tree,param\_grid=parameters,scoring='neg\_mean\_squared\_error',cv=3,verbose=3)

*# function for calculating how much time take for hyperparameter tuning*

**def** timer(start\_time=**None**):

**if** **not** start\_time:

start\_time=datetime.now()

**return** start\_time

**elif** start\_time:

thour,temp\_sec=divmod((datetime.now()-start\_time).total\_seconds(),3600)

tmin,tsec=divmod(temp\_sec,60)

*#print(thour,":",tmin,':',round(tsec,2))*

X=df.iloc[:,:-1]

y=df.iloc[:,-1]

**from** **sklearn.tree** **import** DecisionTreeRegressor

tuned\_hyper\_model= DecisionTreeRegressor(max\_depth=5,max\_features='auto',max\_leaf\_nodes=50,min\_samples\_leaf=2,min\_weight\_fraction\_leaf=0.1,splitter='random')

*# fitting model*

tuned\_hyper\_model.fit(X\_train,y\_train)

*# prediction*

tuned\_pred=tuned\_hyper\_model.predict(X\_test)

We get, DecisionTreeRegressor(max\_depth=5, max\_features='auto', max\_leaf\_nodes=50,

min\_samples\_leaf=2, min\_weight\_fraction\_leaf=0.1,

splitter='random')

plt.scatter(y\_test,tuned\_pred)



*# With hyperparameter tuned Decision Tree Model*

**from** **sklearn** **import** metrics

print('MAE:', metrics.mean\_absolute\_error(y\_test,tuned\_pred))

print('MSE:', metrics.mean\_squared\_error(y\_test, tuned\_pred))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, tuned\_pred)))

MAE: 0.30245717870384736

MSE: 0.14873035485725403

RMSE: 0.3856557465632452

We can see that the Random Forest Model got the least RMSE score in these three hypertuned models.

### PREDICTIONS

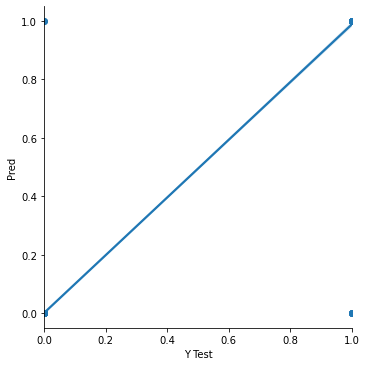
OutputDF=pd.DataFrame({'Actual Data':y\_test,'Predicted Data':y\_pred})

OutputDF.head()

|  |  |  |
| --- | --- | --- |
|  | Actual data | Predicted data |
| 185 | 1 | 1 |
| 2715 | 0 | 0 |
| 3825 | 0 | 0 |
| 1807 | 1 | 1 |
| 132 | 0 | 0 |

OutputDF = pd.DataFrame({'Y Test':y\_test , 'Pred':y\_pred},columns=['Y Test','Pred'])

sns.lmplot(x='Y Test',y='Pred',data=OutputDF,palette='rainbow')



The model is created now. Let us save it using **joblib**

SAVING THE MODEL

**import** **joblib**

joblib.dump(model\_random, 'customer\_churn\_analysis.pkl')

Output: ['customer\_churn\_analysis.pkl']

*# Load the model from the file*

model\_random\_from\_joblib = joblib.load('customer\_churn\_analysis.pkl')

*# Use the loaded model to make predictions*

model\_random\_from\_joblib.predict(X\_test)

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

CONCLUSION

The ability to identify customers that aren’t happy with provided solutions allows businesses to learn about product or pricing plan weak points, operation issues, as well as customer preferences and expectations to proactively reduce reasons for churn. Churn rate is a health indicator for subscription-based companies.

It’s important to define data sources and observation period to have a full picture of the history of customer interaction. If the data is qualitative, predictions would be precise So selecting the significant features for a model would influence its predictive performance.

If the companies has a large customer base and numerous offerings, they would benefit from customer segmentation. The number and choice of ML models may also depend on segmentation results. The frequency with which a model performance is tested depends on how fast data becomes outdated in an organization. We may also need to monitor deployed models, and revise and adapt features to maintain the desired level of prediction accuracy.