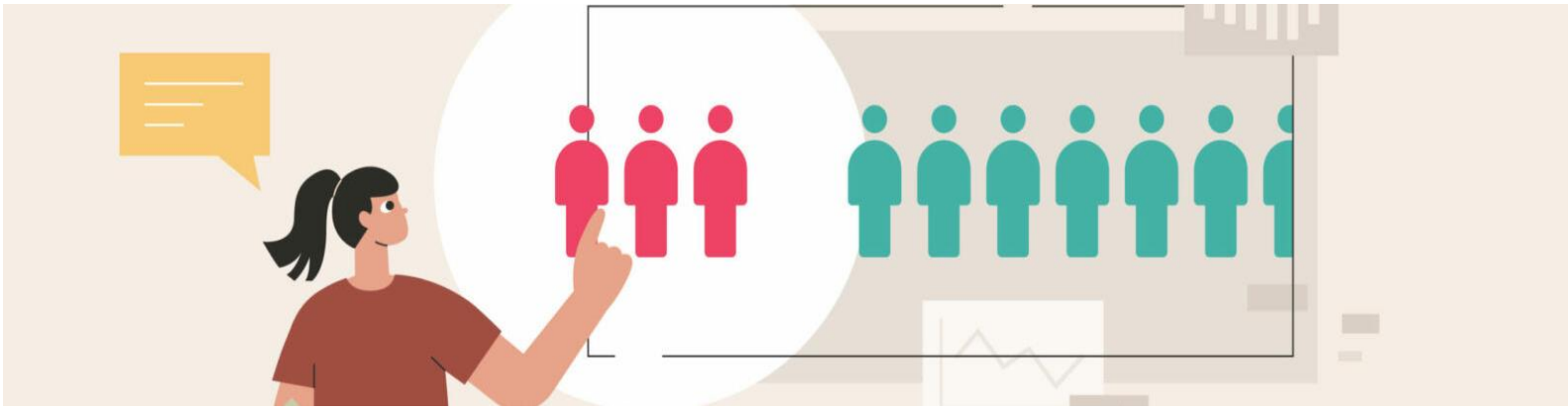


Predicting And Preventing Customer Churn: A Data Science Approach

Author:- Dipesh Ramesh Limaje

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.

1. Problem Definition.

How can we use data science technique to build a model that accurately predicts whether a customer will be retained or not, based on analysis of customer data such as demographics analysis, purchase factors, the service which he/she opt for, tenure, whether he/she uses all the service like mobile device protection, multiline facility, tech support, stream TV, movies, etc.

To solve this problem we have customer data from IBM sample Dataset and we will use a machine learning algorithm or other technique to build and evaluate a predictive model further we will also need to explore different approaches and techniques for improving the accuracy of the model such as feature engineering, model selection or hyperparameter tuning.

2. Data Analysis.

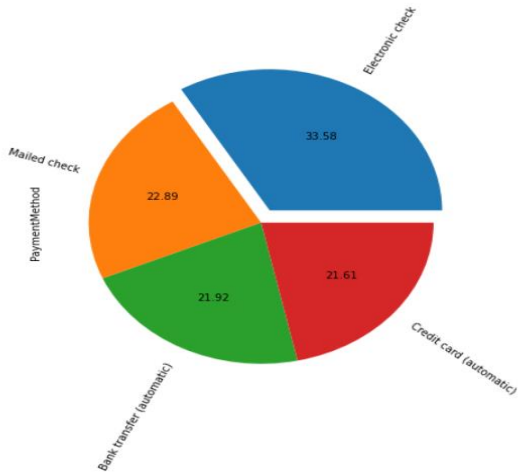
Descriptive analysis: This type of analysis involves summarizing and describing the characteristics of your data, such as the mean, median, and standard deviation of various variables. Descriptive analysis can help you understand the basic properties of your data and identify any unusual or unexpected patterns.

```
n [96]: #describe dataset
        df.describe()
```

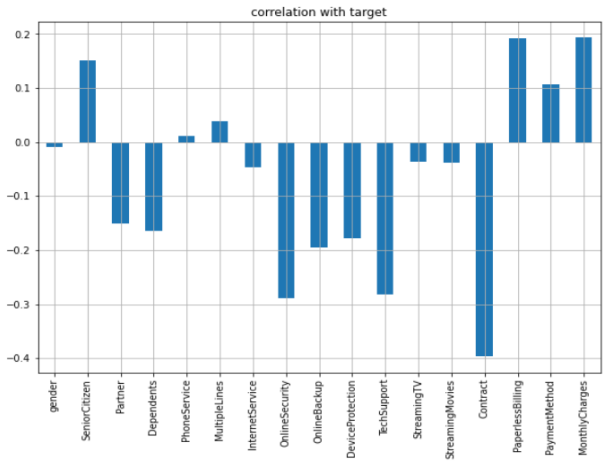
Out[96]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	De
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	
mean	0.504756	0.162147	0.483033	0.299588	32.371149	0.903166	0.940508	0.872923	0.790004	0.906432	
std	0.500013	0.368612	0.499748	0.458110	24.559481	0.295752	0.948554	0.737796	0.859848	0.880162	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	0.000000	0.000000	0.000000	29.000000	1.000000	1.000000	1.000000	1.000000	1.000000	
75%	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	2.000000	1.000000	2.000000	2.000000	
max	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	2.000000	2.000000	2.000000	2.000000	

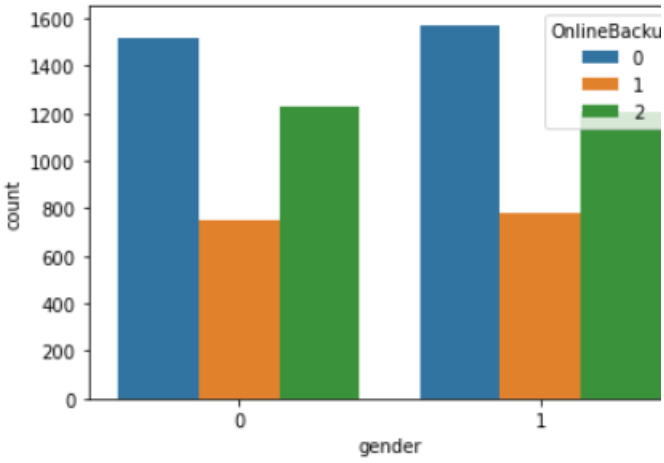
Exploratory analysis: This type of analysis involves visualizing and exploring your data to uncover relationships and patterns that may not be immediately apparent. Exploratory analysis can be particularly useful for identifying potential variables that may be useful for predicting customer churn.



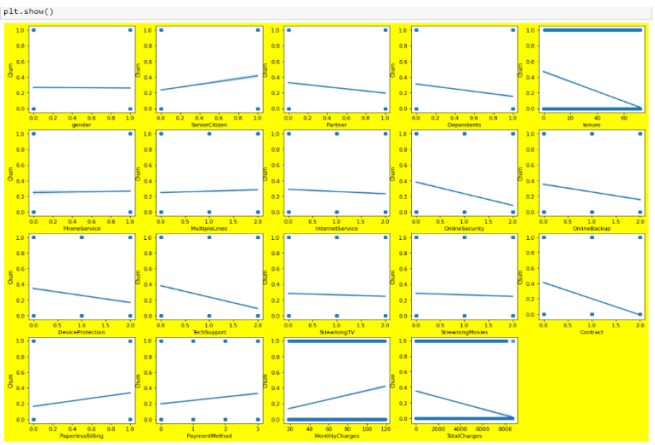
Predictive modeling: This type of analysis involves using machine learning algorithms or other statistical techniques to build a model that can predict the likelihood of customer churn based on certain input variables. Predictive modeling can be a powerful tool for identifying which customers are at risk of churning and for developing strategies to prevent it.



Causal analysis: This type of analysis involves identifying the underlying causes of customer churn and evaluating the impact of different factors on churn. Causal analysis can help you understand why customers are leaving and identify opportunities for intervention.



Network analysis: This type of analysis involves studying the connections and relationships between different customers or groups of customers. Network analysis can help you identify clusters of customers that are more likely to churn and identify opportunities for targeted interventions.

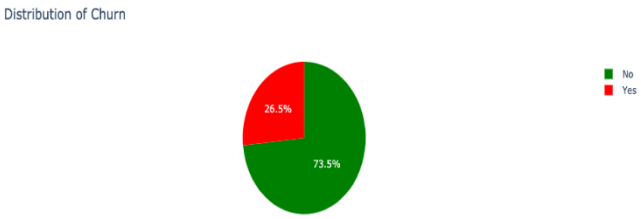


3. EDA Concluding Remark.

Exploratory data analysis (EDA) is a process of exploring and analysing a dataset to better understand its contents and any patterns or relationships that might exist within it. It is an important step in the data science process, as it helps to identify any potential issues or limitations with the data, as well as to uncover any interesting insights that might not have been immediately apparent.

1] EDA on Univariate Analysis

A] Values counts on Target Variable

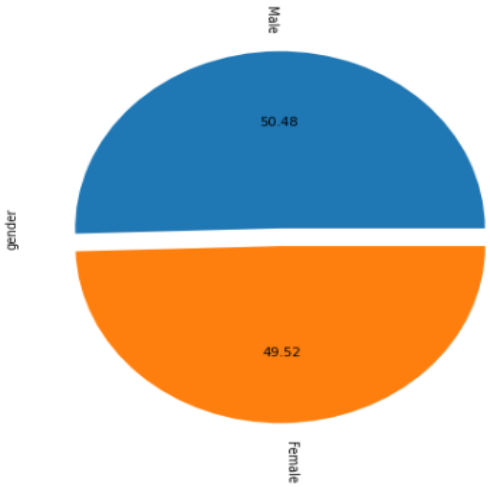


So we are trying to predict the user that left the company so it is a binary classification problem with an un-balance target variable

Churn No:- 73.50%

Churn Yes:- 26.50%

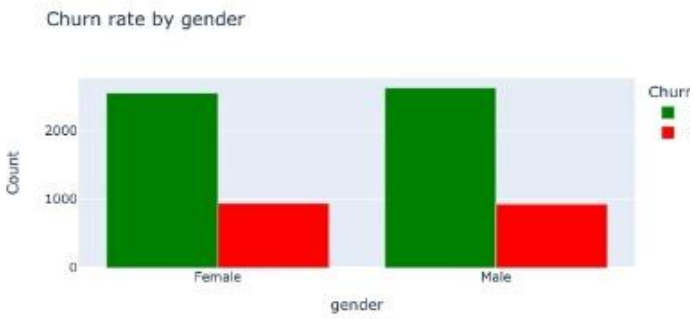
B] Values Counts on Gender



From the Above count plot, we observe the ratio for males and females is almost equal. the males in the dataset are 3555 and the female in the dataset are 3488.

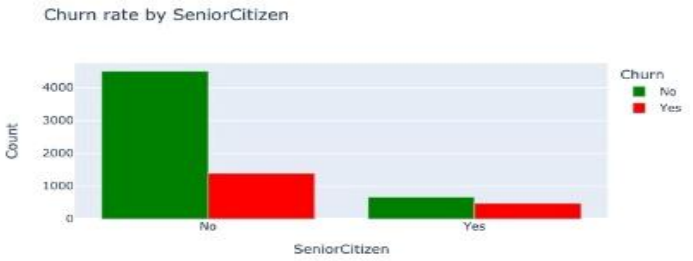
2] EDA on Bivariate Analysis

A] Gender VS Churn



The value counts of the distribution of males and females are 50.48% and 49.52% respectively.

B] Senior citizen VS Churn

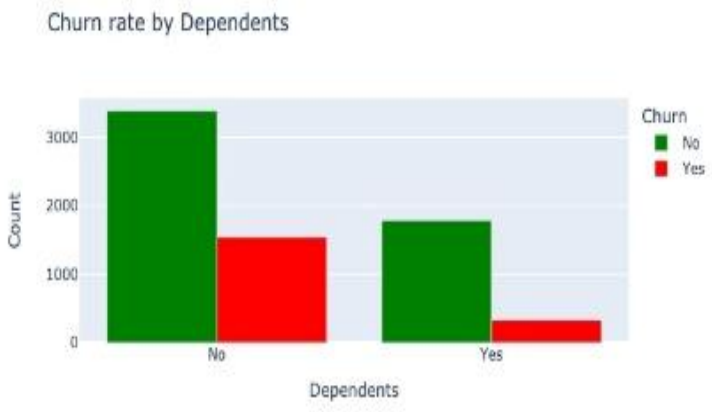


The value counts of the distribution of NO and YES are 83.79% and 16.21% respectively.

C] Partner VS Churn

The value counts of the distribution of NO and YES are 51.70% and 48.30% respectively

D] Dependent VS Churn

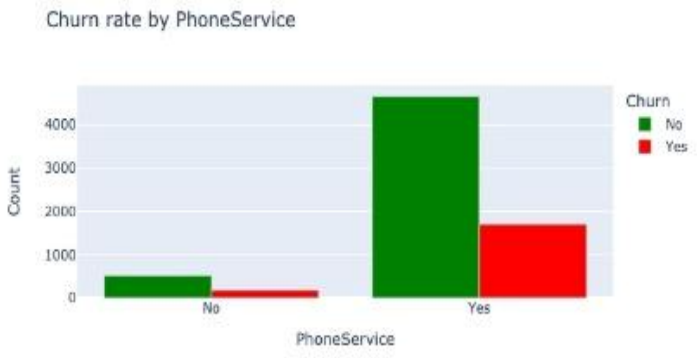


The value counts of the distribution of NO and YES are 70.04% and 29.96% respectively

The Information From The Above 4 Graphs

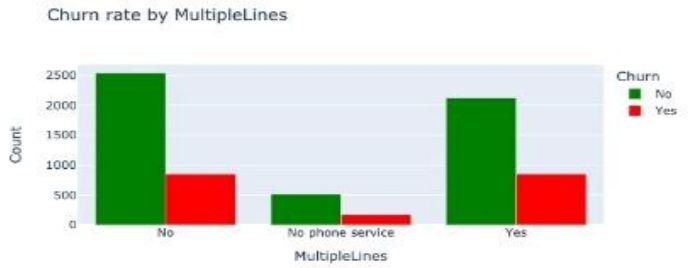
Customer demographics, including gender, age (as indicated by the presence or absence of senior citizenship), and relationship status (whether or not the customer has a partner). The analysis has found that the distribution of these variables is relatively evenly balanced, with no one group being significantly larger or smaller than the others. However, the analysis has also found that there is a slightly higher proportion of customers who churn (i.e., who cancel their service or product) among females, and a higher proportion of churn among younger customers, those with no partners, and those with no dependents. These findings suggest that these particular demographic groups may be more likely to churn than others, although the difference in churn rates is described as being small and potentially ignorable.

E] Phone Service VS Churn



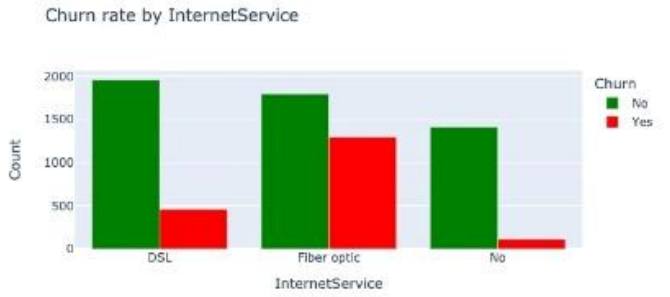
The value counts of the distribution of YES and NO are 90.3% and 9.7% respectively

F] Multiple Lines VS Churn



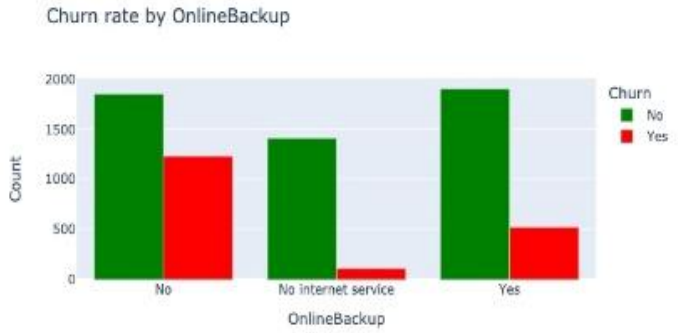
The value counts of the distribution of NO and YES and No Phone Service are 48.1% and 42.2% and 9.7% respectively

G] Internet Service VS Churn



The value counts of the distribution of Fiber optic, DSL & NO are 44.0% and 34.4%, and 21.7% respectively

H] Online Backup VS Churn



The value counts of the distribution of No, Yes & No internet service are 43.8% and 34.5%, and 21.7% respectively

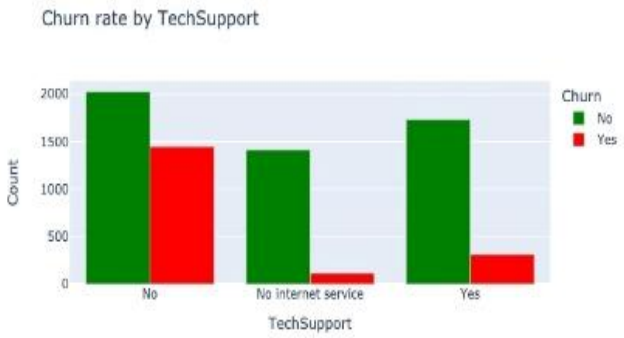
I] Online Security VS Churn

The value counts of the distribution of No, Yes & No internet service are 49.7% and 28.7%, and 21.7% respectively

J] Device Protection VS Churn

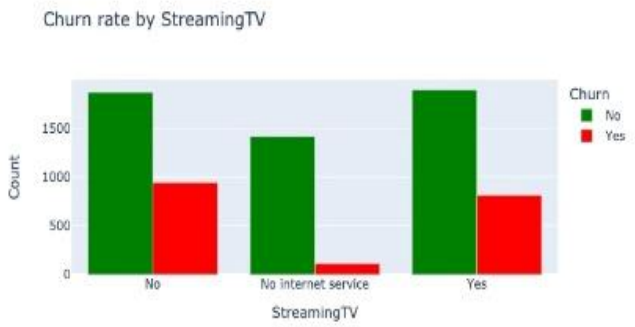
The value counts of the distribution of No, Yes & No internet service are 43.9% and 34.4%, and 21.7% respectively

K] Tech Support VS Churn



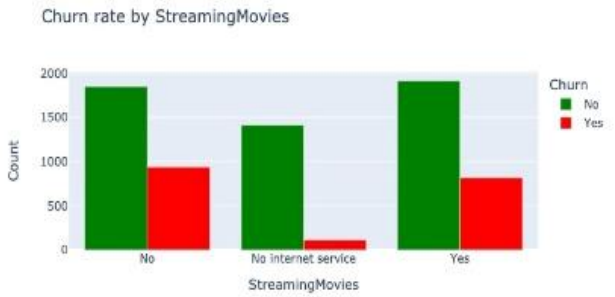
The value counts of the distribution of No, Yes & No internet service are 49.3% and 29.0%, and 21.7% respectively

L] Stream TV VS Churn



The value counts of the distribution of No, Yes & No internet service are 39.9% and 38.4%, and 21.7% respectively

M] Stream Movies VS Churn



The value counts of the distribution of No, Yes & No internet service are 39.5% and 38.8%, and 21.7% respectively

The Information From The Above 9 Graphs

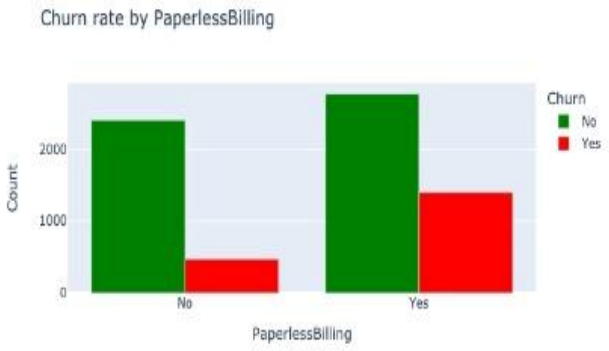
The dataset being analysed includes information about various features or characteristics of the customers, such as their phone and internet service plans, and whether or not they have certain additional services (e.g., online security, online backup, device protection, and tech support). The analysis has found that there are significant variations in these features and that some of them are correlated with a higher or lower likelihood of churn.

the analysis has found that customers who do not have phone service are unable to have multiple lines and that among customers who do have phone service, there is a higher rate of churn. Additionally, the analysis has found that customers who have fiber optic internet service are more likely to churn, which may be due to a variety of reasons such as Moderate prices, competition, or good customer service. In contrast, customers who have certain additional services (e.g., online security, online backup, device protection, and tech support) are likely to churn. Finally, the analysis has found that the presence or absence of streaming services does not seem to be predictive of churn, as it is evenly distributed among customers who do and do not have this service.

N]Contract VS Churn

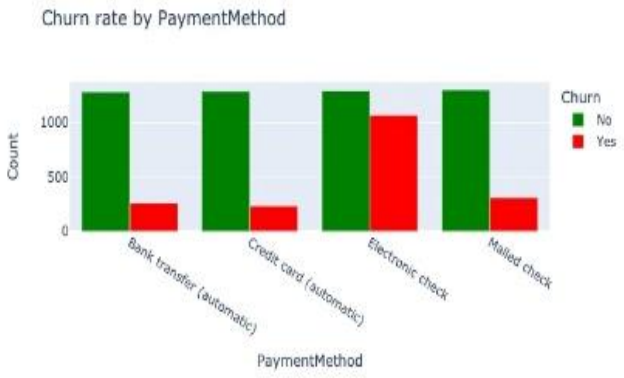
Value count of distribution of month-to-month, Two-year, and one year is 55%,24.1%, and 20.9% respectively.

O] Paperless billing VS Churn



The value counts of the distribution of YES and NO are 59.2% and 40.8% respectively

P] Payment Method VS Churn



Value count of distribution of electronic checks, mailed checks, bank transfers, and credit cards are 33.6%,22.9%, 21.9%, and 21.6% respectively.

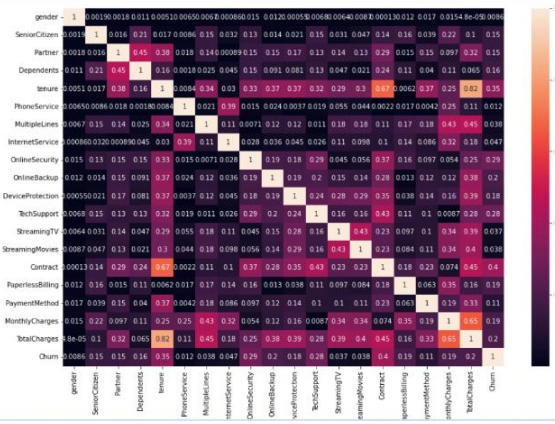
The Information From The Above 3 Graphs

the analysis has found that companies prefer to have long-term relationships with their customers because shorter contracts tend to have a higher churn rate. The churn rate refers to the percentage of customers who cancel or stop using a product or service within a certain time period. We also Analysed that customers who opt for paperless billing are more likely to churn, with about 59.2% of customers using this method. Additionally, customers who pay with electronic checks are more likely to churn, and this form of payment is more common than other payment types. This suggests that companies may want to consider factors such as billing and payment methods when trying to reduce churn and maintain long-term relationships with their customers

3] EDA on Multivariate Analysis

As the name suggests, Multivariate analysis looks at more than two variables. Multivariate analysis is one of the most useful methods to determine relationships and analyze patterns for any dataset. A heat map is widely been used for Multivariate Analysis Heat Map gives the correlation between the variables, whether it has a positive or negative correlation.

```
In [98]: corr=df.corr().abs()
         plt.figure(figsize=(15,10))
         sns.heatmap(corr,annot=True)
         plt.show()
```



4. Pre-Processing Pipeline.

A pre-processing pipeline for customer churn analysis typically involves the following steps:

1] **Data Collection:** The first step is to gather all the relevant data that will be used for the analysis as we have customer data from IBM Sample Dataset that contains all the information which we need to have to create the model like Gender, Tenure, Phone-Services, purchase history, contract, etc.

```
In [1]: #lets import necessary library
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')

In [2]: #import the dataset
        df=pd.read_csv('Telecom_customer_churn.csv')
        df.head()
```

Out[2]:

Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	P
No	1	No	No phone service	DSL	No	...	No	No	No	No	Month-to-month	
No	34	Yes	No	DSL	Yes	...	Yes	No	No	No	One year	
No	2	Yes	No	DSL	Yes	...	No	No	No	No	Month-to-month	
No	45	No	No phone service	DSL	Yes	...	Yes	Yes	No	No	One year	
No	2	Yes	No	Fiber optic	No	...	No	No	No	No	Month-to-month	

2] Data cleaning and preparation: Once the data is collected, it is important to clean and prepare the data for analysis. Like we have done on this project, like checking the null values, checking for the duplicates, checking the datatypes of every column, and ensuring the data is in a consistent format.

```
In [4]: df.columns
Out[4]: Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
               'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
               'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
               'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
               'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
              dtype='object')

In [5]: df.shape
Out[5]: (7043, 21)

In [6]: df.dtypes
Out[6]: customerID          object
        gender              object
        SeniorCitizen        int64
        Partner             object
        Dependents          object
        tenure               int64
        PhoneService        object
        MultipleLines       object
        InternetService     object
        OnlineSecurity      object
        OnlineBackup        object
        DeviceProtection    object
        TechSupport         object
        StreamingTV         object
        StreamingMovies     object
        Contract            object
        PaperlessBilling    object
        PaymentMethod       object
        MonthlyCharges     float64
        TotalCharges        object
        Churn               object
        dtype: object
```

3] Data exploration and visualization: Before building a model to predict churn, it is helpful to explore and visualize the data to understand trends and patterns. This can be done using tools like Matplotlib, and Seaborn (Histogram, Pie chart, Bar chart) like we have done in an earlier phase.

4] **Feature engineering:** The most important phase in Feature engineering is to handle outliers because it ensures that our model is trained on accurate data which leads to accurate models. Feature Engineering also involves selecting the creating the most relevant and predictive features for the model for this method we can use 2-Method first is SelectKbest where we give the number of best features we require and the Second is the SelectPercentile method where we give how much percentage of feature we want, For this particular Problem I have to use SelectKBest Method.

Feature selection with SelectKBest

```
In [103]: # selecting 12 best feature
          from sklearn.feature_selection import SelectKBest , f_classif
```

```
In [104]: #select best feature
          # selecting 12 best feature
          from sklearn.feature_selection import SelectKBest , f_classif

          best_feature = SelectKBest(score_func=f_classif , k=12)

          fit = best_feature.fit(X,y)

          df_scores = pd.DataFrame(fit.scores_)

          df_column = pd.DataFrame(X.columns)

          feature_scores = pd.concat ([df_column , df_scores ] , axis =1)

          feature_scores.columns=['Feature_Name' , 'Scores']

          print (feature_scores.nlargest (12, 'Scores'))
```

```
        Feature_Name       Scores
13          Contract  1315.088872
7     OnlineSecurity   643.162019
10       TechSupport   610.610024
8       OnlineBackup   279.877370
16   MonthlyCharges   273.463704
14  PaperlessBilling   268.985218
9   DeviceProtection   230.744618
3         Dependents   195.149314
1     SeniorCitizen   164.041424
2           Partner   163.060036
15    PaymentMethod    81.641664
6   InternetService    15.782320
```

5] Data partitioning: The next step is to partition the data into training and test sets. But before that, we have to scale the data because it makes it easy for a model to learn and understand by bringing the data into the same scale range so that biasness can be ignored, for this particular model I have Standard Scaler which brings all the data into the same scale (0 to 1). The training set is used to build the model, while the test set is used to evaluate the model's performance.

Scaling the data

```
In [100]: #scale our data using standard sclar
          from sklearn.preprocessing import StandardScaler
          scalar = StandardScaler()
          X_scaled=scalar.fit_transform(X)
```

```
In [ ]:
```

6] Model training and evaluation: Finally, a machine learning model is trained according to the best random state on the training set and evaluated on the test set to predict customer churn.

Training the Model

```
In [114]: # train test split
          X_train_ns,X_test,y_train_ns,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=69)
```

5. *Building Machine Learning Models.*

There is a range of machine learning models that can be used to predict whether a customer will be retained or not, the model I have created is a Decision Tree Classifier, Random Forest Classifier, Support Vector Machine, and KNeighbors Classifier, Each of these models has its own strength and weakness and the best model for a particular situation will depend on the dataset available to solve the specific problem.

1] Decision Tree Classifier Model

The 1st model That I created is a Decision tree Classifier with the help of the feature selection technique SelectKBest and then I scaled the data and then split it into train test split and then created the model Decision tree, So this model gives a score of 98.75% in training and 73.59% in testing.

```
In [116]: # model initilization
          clf_dt = DecisionTreeClassifier()
          clf_dt.fit(X_train_ns,y_train_ns)
```

```
Out[116]: DecisionTreeClassifier()
```

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 [117]: # call the function
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train=False)
```

```
====Training Score====
Accuracy score : 98.750473%
====Testing Score====
Accuracy score : 73.594549%


 Classification report
              precision    recall  f1-score   support

           0       0.82      0.82      0.82      1313
           1       0.48      0.48      0.48       448

    accuracy                           0.74      1761
   macro avg       0.65      0.65      0.65      1761
weighted avg       0.74      0.74      0.74      1761
```

Model Score

```
Training Score = 98.750473%
Testing Score = 73.594549%
```

```
In [ ]:
```

So this is a kind of overfitting of the model I have done hyperparameter tuning by using Grid Search this I got the best parameter on which I have trained the model and the model score after hyperparameter tuning is 78.75% training and 77.22% testing.

```
In [121]: # see best parameters
          best_parameters = grid_search.best_params_
          print(best_parameters)
```

```
{'criterion': 'gini', 'max_depth': 5, 'max_leaf_nodes': 9, 'min_samples_leaf': 1, 'min_samples_split': 5}
```

```
In [122]: #initiate what new parameter we got

          clf_dt=DecisionTreeClassifier(criterion= 'gini', max_depth = 5, min_samples_leaf= 1, min_samples_split= 5)
          clf_dt.fit(X_train_ns,y_train_ns)

          #i tried different combination and i find this is best parameter so using this instead of gridsearch parameters
```

```
Out[122]: DecisionTreeClassifier(max_depth=5, min_samples_split=5)
```

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 [123]: # call the function
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train=False)
```

```
====Training Score====
Accuracy score : 78.758046%
====Testing Score====
Accuracy score : 77.228847%


 Classification report
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      1313
           1       0.55      0.54      0.55       448

    accuracy                           0.77      1761
   macro avg       0.70      0.70      0.70      1761
weighted avg       0.77      0.77      0.77      1761
```

And after that, I have to perform the cross-validation to check if my model is overfitting or not and Cross-validation score is 77.4% which means our model is not overfitted model and at last, I created the confusion Matrix

Cross Validation score for DecisionTree Classifier

```
In [124]: from sklearn.model_selection import cross_val_score
          cross_val_score(clf_dt,X_scalar,y,cv=6)

Out[124]: array([0.76746167, 0.78449744, 0.78534923, 0.75894378, 0.77597956,
                 0.77749361])

In [125]: cross_val_score(clf_dt,X_scalar,y,cv=6).mean()

Out[125]: 0.7749542154466408

In [ ]:
```

Confusion Matrix DecisionTree Classifier

```
In [126]: ### if you want to check confusion matrix

          y_pred=clf_dt.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[126]: array([[1119,  194],
                 [ 207,  241]], dtype=int64)

In [ ]:
```

2] Random Forest Classifier

The 2nd model That I created is a Random Forest Classifier with the help of the feature selection technique SelectKBest and then I scaled the data and then split it into train test split and then created the model Random Forest, So this model gives a score of 98.75% in training and 76.32% in testing.

```
In [133]: # model initilization
          clf_rf = RandomForestClassifier()
          clf_rf.fit(X_train_ns,y_train_ns)

Out[133]: RandomForestClassifier()
          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 [134]: # call the function
          metric_score (clf_rf,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_rf,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 98.750473%
          ====Testing Score====
          Accuracy score : 76.320273%


           Classification report
                        precision    recall  f1-score   support

                     0       0.83      0.86      0.84      1313
                     1       0.54      0.48      0.51       448

              accuracy                           0.76      1761
             macro avg       0.68      0.67      0.68      1761
          weighted avg       0.75      0.76      0.76      1761


          Model Scores

              Training Score = 98.750473%
              Testing Score = 76.320273%
```

So this is a kind of overfitting of the model I have done hyperparameter tuning by using Grid SearchCV this I got the best parameter on which I have trained the model and the model score after hyperparameter tuning is 79.19% training and 78.19% testing

```
In [138]: print ('Best parameters : ' , grd.best_params_) #printing best parameters

          Best parameters :  {'criterion': 'entropy', 'max_depth': 7, 'min_samples_leaf': 8, 'min_samples_split': 6, 'n_estimators': 4}

In [139]: #initiate what new parameter we got
          rf=RandomForestClassifier(criterion= 'entropy', max_depth = 7, min_samples_leaf=8, min_samples_split= 6,n_estimators= 4)
          rf.fit(X_train_ns,y_train_ns)

          #i tried different combination and i find this is best parameter so using this instead of gridsearch parameters

Out[139]: RandomForestClassifier(criterion='entropy', max_depth=7, min_samples_leaf=8,
                                 min_samples_split=6, n_estimators=4)
          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 [140]: # call the function
          metric_score (rf,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (rf,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 79.193487%
          ====Testing Score====
          Accuracy score : 78.194208%


           Classification report
                        precision    recall  f1-score   support

                     0       0.84      0.88      0.86      1313
                     1       0.58      0.51      0.54       448

              accuracy                           0.78      1761
             macro avg       0.71      0.69      0.70      1761
          weighted avg       0.77      0.78      0.78      1761


          Model Scores With Hyperparameter Tuning

              Training Score = 79.193487%
              Testing Score = 78.194208%
```

And after that, I have to perform cross-validation to check if my model is overfitting or not and Cross-validation score is 78.16% which means our model is not overfitted model and at last, I created the confusion Matrix.

Cross Validation score for RandomForestClassifier

```
In [141]: from sklearn.model_selection import cross_val_score
          cross_val_score(rf,X_scalar,y,cv=6)

Out[141]: array([0.76916525, 0.78705281, 0.78705281, 0.7725724 , 0.78194208,
                 0.7885763 ])

In [142]: cross_val_score(rf,X_scalar,y,cv=6).mean()

Out[142]: 0.7816286181657809

In [ ]:
```

Confusion Matrix RandomForestClassifier

```
In [143]: ### if you want to check confusion matrix

          y_pred=clf_rf.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[143]: array([[1130,  183],
                 [ 234,  214]], dtype=int64)
```

3] Support Vector Machine Model

The 3rd model That I created is a Support Vector Machine with the help of the feature selection technique SelectKBest and then I scaled the data and then split it into train test split and then created the model Support Vector Machine, So this model gives a score of 78.75% in training and 79.78% in testing

```
In [149]: clf_svc=SVC()
          clf_svc.fit(X_train_ns,y_train_ns)

Out[149]: SVC()
          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 [150]: # call the function
          metric_score (clf_svc,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_svc,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 78.758046%
          ====Testing Score====
          Accuracy score : 79.784214%


           Classification report
                        precision    recall  f1-score   support

                     0       0.84      0.91      0.87      1315
                     1       0.63      0.48      0.54       446

              accuracy                           0.80      1761
             macro avg       0.74      0.69      0.71      1761
          weighted avg       0.79      0.80      0.79      1761
```

Model Scores

```
Training Score = 78.758046%
Testing Score = 79.784214%
```

So this is a well-fitted model So I tried to improve the score by doing hyperparameter tuning by using Grid SearchCV by this I got the best parameter on which I have trained the model and the model score after hyperparameter tuning is 78.11% training and 80.40% testing.

```
In [155]: #best parameters
          gridsearch.best_params_

Out[155]: {'C': 20, 'gamma': 0.01}

In [158]: # update our model and train again for new score
          svc=SVC(C=20,gamma=0.01)
          svc.fit(X_train_ns,y_train_ns)

Out[158]: SVC(C=20, gamma=0.01)
          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 [159]: # call the function
          metric_score (svc,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (svc,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 78.114351%
          ====Testing Score====
          Accuracy score : 80.408859%


           Classification report
                        precision    recall  f1-score   support

                     0       0.84      0.91      0.87      1315
                     1       0.65      0.49      0.56       446

              accuracy                           0.80      1761
             macro avg       0.75      0.70      0.72      1761
          weighted avg       0.79      0.80      0.79      1761
```

Model Scores With Hyperparameter Tuning

```
Training Score = 78.114351%
Testing Score = 80.408859%
```

And after that, I have to perform cross-validation to check if my model is overfitting or not and Cross-validation score is 78.13% which means our model is not overfitted model and at last, I created the confusion Matrix.

Cross Validation score for SVC

```
In [160]: from sklearn.model_selection import cross_val_score
          cross_val_score(svc,X_scalar,y,cv=6)

Out[160]: array([0.76916525, 0.79216354, 0.79386712, 0.76746167, 0.77683135,
                 0.7885763 ])

In [161]: cross_val_score(svc,X_scalar,y,cv=6).mean()

Out[161]: 0.7813442044719031

In [ ]:
```

Confusion Matrix for SVC

```
In [162]: ### if you want to check confusion matrix

          y_pred=svc.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[162]: array([[1198,  117],
                 [ 228,  218]], dtype=int64)
```

4] KNeighbors Classifier Model

The 4th model That I created is a] KNeighbors Classifier with the help of the feature selection technique SelectKBest and then I scaled the data and then split it into train test split and then created the model] KNeighbors Classifier, So this model gives a score of 82.52% in training and 75.18% in testing.

```
In [171]: # model initilization
          clf_knn = KNeighborsClassifier()
          clf_knn.fit(X_train_ns,y_train_ns)

Out[171]: KNeighborsClassifier()
          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 [172]: # call the function
          metric_score (clf_knn,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_knn,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 82.563423%
          ====Testing Score====
          Accuracy score : 75.184554%


           Classification report
                        precision    recall  f1-score   support

                     0       0.81      0.86      0.84      1290
                     1       0.54      0.46      0.50       471

              accuracy                           0.75      1761
             macro avg       0.68      0.66      0.67      1761
          weighted avg       0.74      0.75      0.74      1761
```

Model Scores

```
Training Score = 82.563423%
Testing Score = 75.184554%
```

So this is a well-fitted model So I tried to improve the score by doing hyperparameter tuning by using Grid SearchCV by this I got the best parameter on which I have trained the model and the model score after hyperparameter tuning is 80.04% training and 77.79% testing.

```
In [176]: #see the best paramater
          gridsearch.best_params_

Out[176]: {'algorithm': 'kd_tree', 'leaf_size': 5, 'n_neighbors': 13}

In [177]: # we will use the best parameter in our knn algorithm and check if accuracy is increase or not
          clf_knn=KNeighborsClassifier(algorithm = 'kd_tree', leaf_size = 5, n_neighbors = 13)

          clf_knn.fit(X_train_ns,y_train_ns)

Out[177]: KNeighborsClassifier(algorithm='kd_tree', leaf_size=5, n_neighbors=13)
          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 [178]: # call the function
          metric_score (clf_knn,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_knn,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 80.045437%
          ====Testing Score====
          Accuracy score : 77.796706%


           Classification report
                        precision    recall  f1-score   support

                     0       0.82      0.89      0.85      1290
                     1       0.61      0.48      0.54       471

              accuracy                           0.78      1761
             macro avg       0.72      0.68      0.70      1761
          weighted avg       0.77      0.78      0.77      1761
```

Model Scores With Hyperparameter Tuning

```
Training Score = 80.045437%
Testing Score = 77.796706%
```

And after that, I have to perform cross-validation to check if my model is overfitting or not and Cross-validation score is 77.20% which means our model is not overfitted model and at last, I created the confusion Matrix.

Cross Validation score for KNN

```
In [179]: from sklearn.model_selection import cross_val_score
          cross_val_score(clf_knn,X_scalar,y,cv=6)

Out[179]: array([0.75894378, 0.79131175, 0.78279387, 0.75809199, 0.77001704
                 0.77237852])

In [180]: cross_val_score(clf_knn,X_scalar,y,cv=6).mean()

Out[180]: 0.7722561582221216

In [ ]:
```

Confusion Matric for KNN

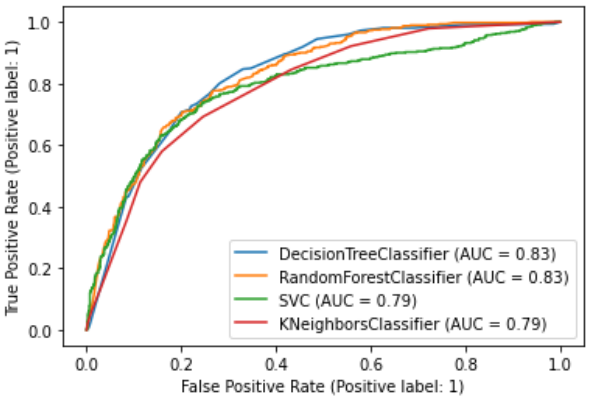
```
In [181]: ### if you want to check confusion matrix

          y_pred=clf_knn.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[181]: array([[1144,  146],
                 [ 245,  226]], dtype=int64)
```

6. *Concluding Remarks*.

In conclusion, a good model for predicting customer churn is crucial for businesses looking to effectively address and reduce churn rates. A well-designed and accurately trained model can help identify at-risk customers, predict the likelihood of churn, and provide insights into the underlying factors contributing to churn. By using a predictive model, businesses can proactively intervene and take targeted actions to improve the customer experience and reduce churn. It is important for businesses to regularly evaluate and optimize their churn prediction models to ensure they are accurately identifying churn risk and providing actionable insights.



So we plotted the AUC-ROC curve for all 4 models and we select the best model according to Cross- the validation score and Area under the curve so we select the Decision tree Classifier as the best performing model for this particular Problem as it covers 83% of Area in AUC-ROC and Cross-validation score is 77.4%.