Churn prediction

Customer attrition or churn, is when customers stop doing business with a company. It can have a significant impact on a company's revenue and it's crucial for businesses to find out the reasons why customers are leaving and take steps to reduce the number of customers leaving. One way to do this is by identifying customer segments that are at risk of leaving, and implementing retention strategies to keep them. Also, by using data and machine learning techniques, companies can predict which customers are likely to leave in the future and take actions to keep them before they decide to leave.

We are going to build a basic model for predicting customer churn using Telco Customer Churn dataset . We are using some classification algorithm to model customers who have left, using Python tools such as pandas for data manipulation and matplotlib for visualizations.

Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data split and preprocessing

In [2]:
```
from sklearn.preprocessing import OneHotEncoder , LabelEncoder
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.preprocessing import StandardScaler, Normalizer
from sklearn.model_selection import train_test_split, cross_validate, Stra
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, au
```

ML algorithem

In [3]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import ExtraTreesClassifier, GradientBoostingClassi
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
```

In [4]:
```
# set seed
SEED = 123
# filtering warnings
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
df = pd.read_excel(r"C:\Users\hp\Downloads\Telecom Churn Rate Dataset.xls>
```

In [6]:
```
df.head()
```

Out[6]:

	Gender	Senior_Citizen	Partner	Dependents	Tenure	Phone_Service	Multiple_Lines	Int
0	Male	Yes	No	No	1	No	No phone service	
1	Female	Yes	Yes	No	71	Yes	Yes	
2	Male	Yes	Yes	No	2	Yes	No	
3	Male	Yes	No	No	1	Yes	No	
4	Female	Yes	No	No	43	Yes	Yes	

5 rows × 22 columns

Data Explanation

gender=============> Customer gender

SeniorCitizen======> Does the customer is a SeniorCitizen or not? (0: No , 1: Yes)

Partner============> Does the customer have partner or not? (0: No , 1: Yes)

Dependents=========> Does the customer have partner or not? (0: No , 1: Yes)

tenure=============> number of mounths the costomer has stayed with the company

PhoneService=======> Does the customer have PhoneService or not? (No , Yes)

MultipleLines======> Does the customer have MultipleLines or not? (Yes, No or No phone service)

InternetService====> Does the customer have InternetService or not?(Fiber optic, DSL or No)

OnlineSecurity====> Does the customer have OnlineSecurity or not? (Yes, No or No internet service)

OnlineBackup======> Does the customer have OnlineBackup or not? (Yes, No or No internet service)

DeviceProtection==> Does the customer have DeviceProtection or not? (Yes, No or No internet service)

TechSupport=======> Does the customer have TechSupport or not? (Yes, No or No internet service)

StreamingTV=======> Does the customer have StreamingTV or not? (Yes, No or No internet service)

StreamingMovies===> Does the customer have StreamingMovies or not? (Yes, No or No internet service)

Contract==========> Types of contract (Month-to-month, Two year, One year)

PaperlessBilling==> Does the customer have Paperless Billing or not? (Yes, No)

PaymentMethod=====> Types of Payment Method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))

Monthly_Charges====> The amount charged to the customer in monthly

Yearly_Charges======> The total amount charged to the customer

Admin_Tickets ======>

Tech_Tickets =======>

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1142 entries, 0 to 1141
Data columns (total 22 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             1142 non-null   object 
 1   Senior_Citizen     1142 non-null   object 
 2   Partner            1142 non-null   object 
 3   Dependents         1142 non-null   object 
 4   Tenure             1142 non-null   int64  
 5   Phone_Service      1142 non-null   object 
 6   Multiple_Lines     1142 non-null   object 
 7   Internet_Service   1142 non-null   object 
 8   Online_Security    1142 non-null   object 
 9   Online_Backup      1142 non-null   object 
 10  Device_Protection  1142 non-null   object 
 11  Tech_Support       1142 non-null   object 
 12  Streaming_TV       1142 non-null   object 
 13  Streaming_Movies   1142 non-null   object 
 14  Contract           1142 non-null   object 
 15  Paper_less_Billing 1142 non-null   object 
 16  Payment_Method     1142 non-null   object 
 17  Monthly_Charges    1142 non-null   float64
 18  Yearly_Charge      1142 non-null   int64  
 19  Admin_Tickets      1142 non-null   int64  
 20  Tech_Tickets       1142 non-null   int64  
 21  Churn              1142 non-null   object 
dtypes: float64(1), int64(4), object(17)
memory usage: 196.4+ KB
```

In [9]:
```
df.describe()
```

Out[9]:

	Tenure	Monthly_Charges	Yearly_Charge	Admin_Tickets	Tech_Tickets
count	1142.000000	1142.000000	1142.000000	1142.000000	1142.000000
mean	33.295972	798.203590	9578.443082	0.513135	0.684764
std	24.188530	237.640267	2851.683204	1.296967	1.550357
min	1.000000	189.500000	2274.000000	0.000000	0.000000
25%	10.000000	701.500000	8418.000000	0.000000	0.000000
50%	31.000000	848.500000	10182.000000	0.000000	0.000000
75%	56.000000	980.750000	11769.000000	0.000000	0.000000
max	72.000000	1174.500000	14094.000000	5.000000	9.000000

In [11]:
```
df.describe(include="O")
```

Out[11]:

	Gender	Senior_Citizen	Partner	Dependents	Phone_Service	Multiple_Lines	Interr
count	1142	1142	1142	1142	1142	1142	
unique	2	1	2	2	2	3	
top	Male	Yes	Yes	No	Yes	Yes	
freq	574	1142	573	1051	1038	665	

In [13]:
```
df.dtypes
```

Out[13]:
```
Gender                 object
Senior_Citizen         object
Partner                object
Dependents             object
Tenure                  int64
Phone_Service          object
Multiple_Lines         object
Internet_Service       object
Online_Security        object
Online_Backup          object
Device_Protection      object
Tech_Support           object
Streaming_TV           object
Streaming_Movies       object
Contract               object
Paper_less_Billing     object
Payment_Method         object
Monthly_Charges       float64
Yearly_Charge           int64
Admin_Tickets           int64
Tech_Tickets            int64
Churn                  object
dtype: object
```

In [15]:
```
df.select_dtypes("O").columns
```

Out[15]:
```
Index(['Gender', 'Senior_Citizen', 'Partner', 'Dependents', 'Phone_Servi
ce',
       'Multiple_Lines', 'Internet_Service', 'Online_Security',
       'Online_Backup', 'Device_Protection', 'Tech_Support', 'Streaming_
TV',
       'Streaming_Movies', 'Contract', 'Paper_less_Billing', 'Payment_Me
thod',
       'Churn'],
      dtype='object')
```

In [16]:
```
df.groupby("Churn").size()
```

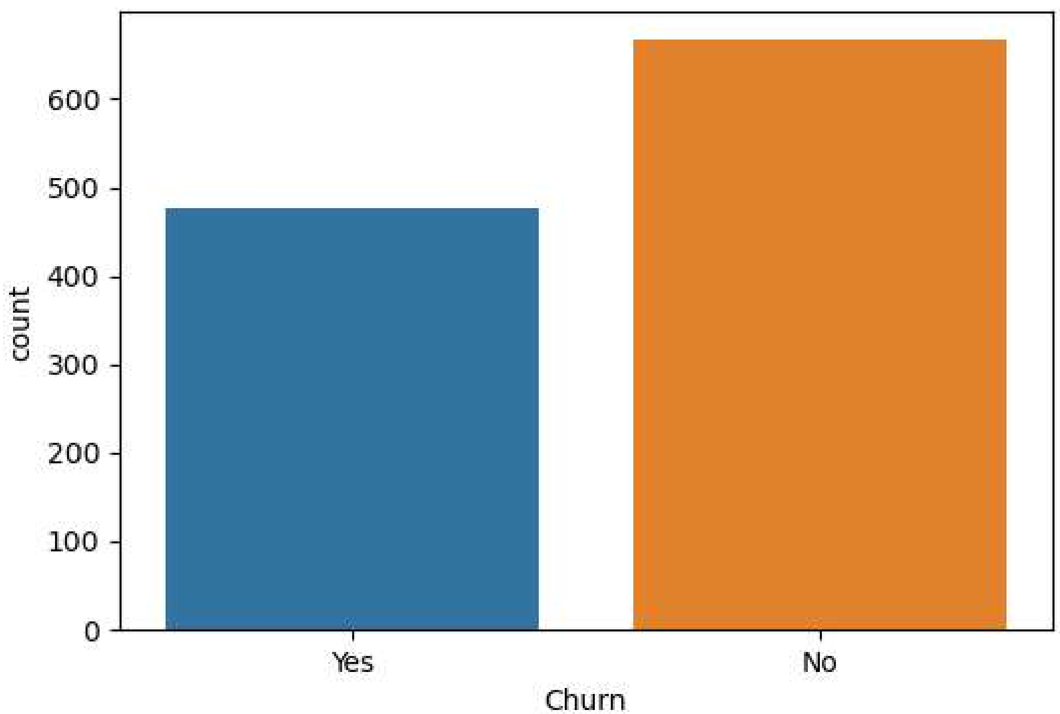
Out[16]:
```
Churn
No     666
Yes    476
dtype: int64
```

In [17]:
```
# percentage of coutomer that are leaving
df.Churn.value_counts(normalize=True)
```

Out[17]:
```
No     0.583187
Yes    0.416813
Name: Churn, dtype: float64
```

In [18]:
```
plt.figure(figsize=(6,4))
sns.countplot(data=df,x= "Churn")
```

Out[18]:
```
<Axes: xlabel='Churn', ylabel='count'>
```



You can see that the percentage of leaving for this company is 26%. Without doing anything special, 74% of customers stay with the company, so we need to build a model that scores higher than 74%

In [19]:
```
# gender of Customer

df.groupby("Gender").size()
```

Out[19]:
```
Gender
Female    568
Male      574
dtype: int64
```

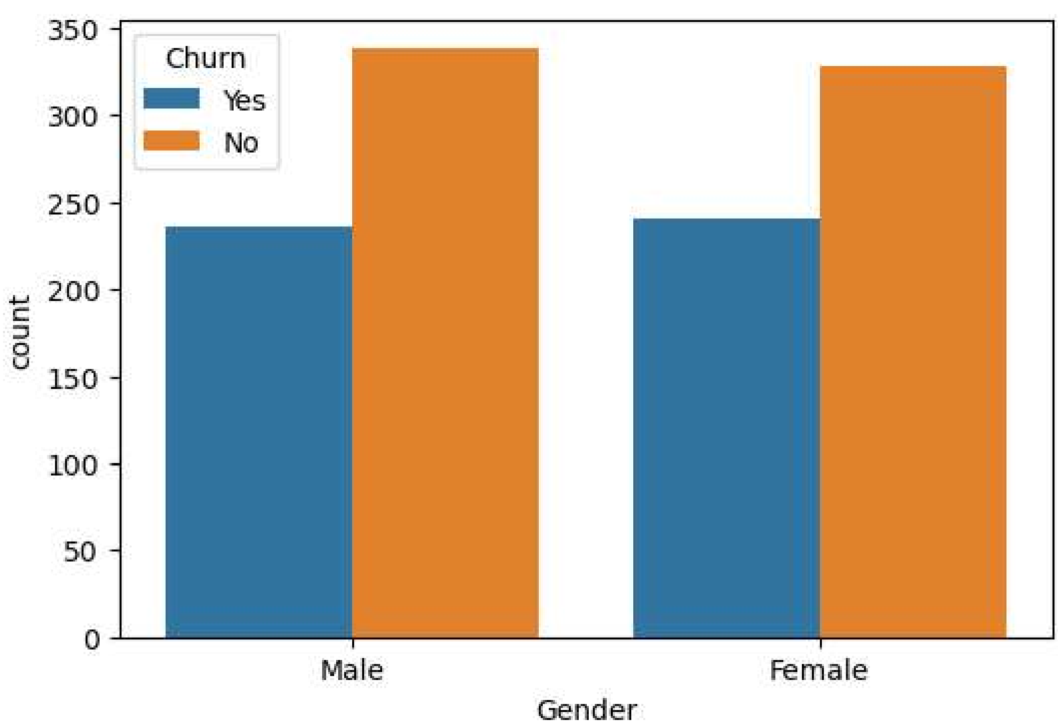
In [20]:
```
# Customer churn by gender

df.groupby(["Gender","Churn"]).size().reset_index()
```

Out[20]:

	Gender	Churn	0
0	Female	No	328
1	Female	Yes	240
2	Male	No	338
3	Male	Yes	236

In [21]:
```
plt.figure(figsize=(6,4))
sns.countplot(data = df, x="Gender" , hue = "Churn")
```

Out[21]: <Axes: xlabel='Gender', ylabel='count'>

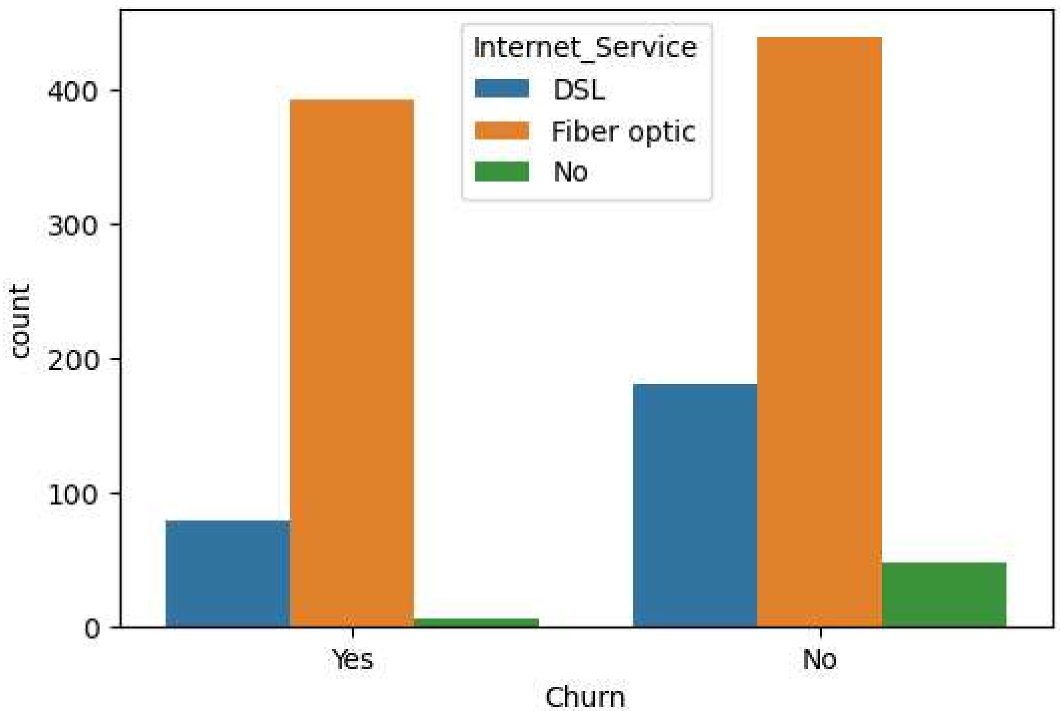


In [22]:
```
# churn count for intenet service
df.groupby(["Churn", "Internet_Service"]).size()
```

Out[22]:
```
Churn  Internet_Service
No     DSL                 181
       Fiber optic         438
       No                   47
Yes    DSL                  78
       Fiber optic         393
       No                    5
dtype: int64
```

In [23]:
```
plt.figure(figsize=(6,4))
sns.countplot(data=df, x="Churn", hue="Internet_Service")
```

Out[23]:
```
<Axes: xlabel='Churn', ylabel='count'>
```

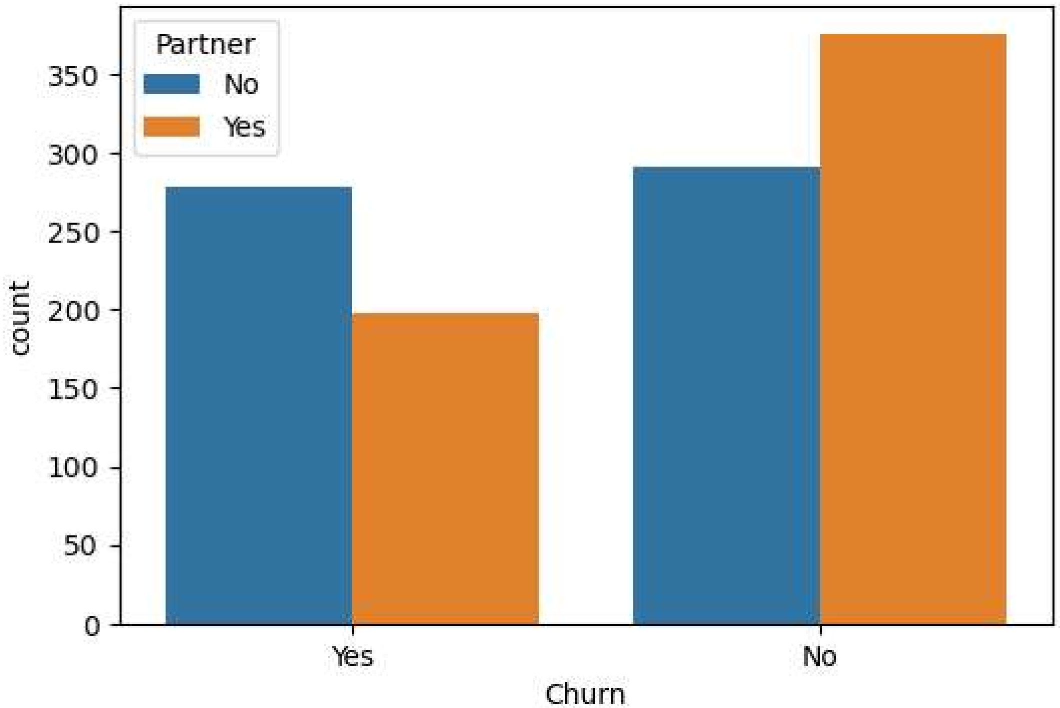


Oboservations

customer who stay with company almost use DSL. customer who churn mostly used Fiber Optic.

In [24]:
```
plt.figure(figsize=(6,4))
sns.countplot(data = df, x = "Churn", hue = "Partner")
```

Out[24]: <Axes: xlabel='Churn', ylabel='count'>

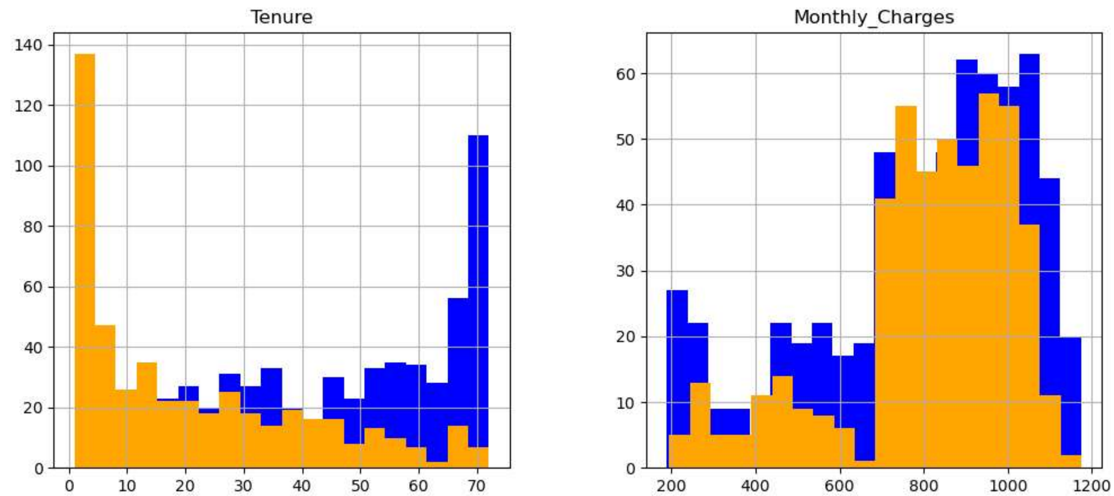


Observation The Cuatomer who often left the company almost did not have a partner.

In [25]:
```
# Distribution of numerical columns for churn
```

In [26]:
```
num_features = ["Tenure", "Monthly_Charges"]
fig, ax = plt.subplots(1 , 2, figsize= (12,5))
df[df["Churn"]=="No"][num_features].hist(bins= 20, color = "blue", ax= ax)
df[df["Churn"]=="Yes"][num_features].hist(bins= 20, color = "orange", ax=
```

Out[26]:
```
array([<Axes: title={'center': 'Tenure'}>,
       <Axes: title={'center': 'Monthly_Charges'}>], dtype=object)
```

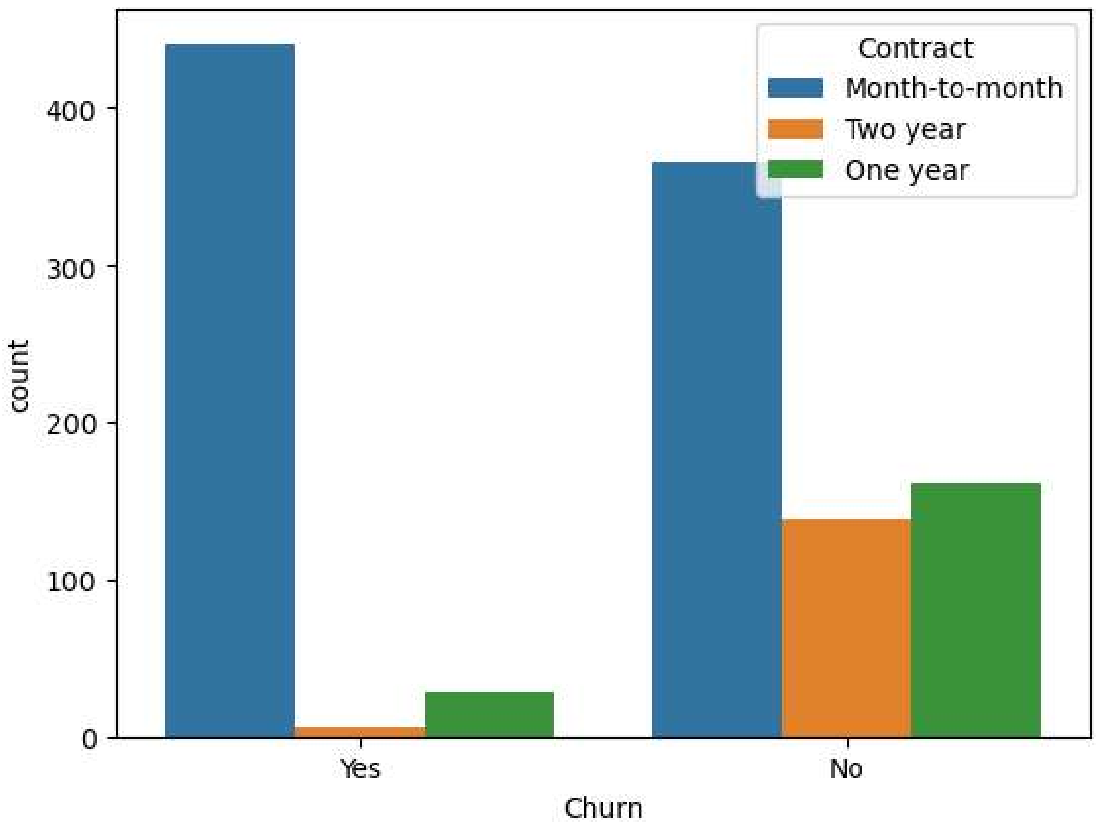


Observation

The result shows that churn customers end up leaving the company up to 10 months later, and they often pay less than 100 dollar a month.

In [27]:
```
#churn count for contract types
sns.countplot(data=df, x="Churn", hue = "Contract")
```

Out[27]: <Axes: xlabel='Churn', ylabel='count'>



Data Preprocessing

In [28]:
```
# checking null values
df.isnull().sum()
```

Out[28]:
```
Gender               0
Senior_Citizen       0
Partner              0
Dependents           0
Tenure               0
Phone_Service        0
Multiple_Lines       0
Internet_Service     0
Online_Security      0
Online_Backup        0
Device_Protection    0
Tech_Support         0
Streaming_TV         0
Streaming_Movies     0
Contract             0
Paper_less_Billing   0
Payment_Method       0
Monthly_Charges      0
Yearly_Charge        0
Admin_Tickets        0
Tech_Tickets         0
Churn                0
dtype: int64
```

In [29]:
```
df["Yearly_Charge"] = df["Yearly_Charge"].astype("float64")
```

In [30]:
```
# Convert all of the non-numeric to numeric
for column in df.columns:
    if df[column].dtypes == np.number:
        continue
    df[column] = LabelEncoder().fit_transform(df[column])
```

In [32]:
```
df.dtypes
```

Out[32]:
```
Gender                int32
Senior_Citizen        int32
Partner               int32
Dependents            int32
Tenure                int64
Phone_Service         int32
Multiple_Lines        int32
Internet_Service      int32
Online_Security       int32
Online_Backup         int32
Device_Protection     int32
Tech_Support          int32
Streaming_TV          int32
Streaming_Movies      int32
Contract              int32
Paper_less_Billing    int32
Payment_Method        int32
Monthly_Charges     float64
Yearly_Charge       float64
Admin_Tickets         int64
Tech_Tickets          int64
Churn                 int32
dtype: object
```

In [33]:
```
df.head()
```

Out[33]:

	Gender	Senior_Citizen	Partner	Dependents	Tenure	Phone_Service	Multiple_Lines	Int
0	1	0	0	0	0	0	1	
1	0	0	1	0	70	1	2	
2	1	0	1	0	1	1	0	
3	1	0	0	0	0	1	0	
4	0	0	0	0	42	1	2	

5 rows × 22 columns

In [35]:
```
x = df.drop("Churn", axis=1)
y = df["Churn"]
```

In [37]:
```
# Scaling the data set
st = StandardScaler()
df_st = st.fit_transform(x)
```

In [38]:
```
# scaling the data set
st = StandardScaler()
df_st = st.fit_transform(x)
```

In [39]:
```
# Scaling the data set
scale_col = ["Tenure", "Monthly_Charges", "Yearly_Charge"]
norm = Normalizer()
df[scale_col] = norm.fit_transform(df[scale_col])
```

In [40]:
```
x = pd.DataFrame(df_st)
x
```

Out[40]:

	0	1	2	3	4	5	6	7	
0	0.994760	0.0	-1.003509	-0.294252	-1.335762	-3.159236	-0.278394	-1.673080	-0.626
1	-1.005268	0.0	0.996503	-0.294252	1.559439	0.316532	0.810395	0.370404	1.695
2	0.994760	0.0	0.996503	-0.294252	-1.294402	0.316532	-1.367183	0.370404	-0.626
3	0.994760	0.0	-1.003509	-0.294252	-1.335762	0.316532	-1.367183	-1.673080	-0.626
4	-1.005268	0.0	-1.003509	-0.294252	0.401359	0.316532	0.810395	0.370404	-0.626
...	...	...	...	...	...	...	...	...	
1137	-1.005268	0.0	0.996503	-0.294252	1.228559	0.316532	0.810395	0.370404	-0.626
1138	-1.005268	0.0	-1.003509	-0.294252	-1.128962	-3.159236	-0.278394	-1.673080	-0.626
1139	0.994760	0.0	0.996503	-0.294252	0.897679	0.316532	0.810395	-1.673080	1.695
1140	0.994760	0.0	-1.003509	-0.294252	-1.335762	0.316532	0.810395	0.370404	-0.626
1141	0.994760	0.0	0.996503	-0.294252	-1.211682	0.316532	0.810395	0.370404	-0.626

1142 rows × 21 columns

In [41]:
```
df = df_st.copy()
```

In [42]:
```
# Data split
```

In [45]:
```
x_train, x_val , y_train , y_val = train_test_split(x, y , train_size=0.8
skf = StratifiedKFold(n_splits=7, shuffle=True, random_state=SEED)
```

Logistic Regression

In [46]:

```
logr = LogisticRegression(random_state=SEED)
cv = cross_validate(logr , x_train, y_train, cv = skf , scoring="accuracy'
model_logr = cv["estimator"]
```

In [47]:

```
y_hats = [model.predict(x_val) for model in model_logr]
y_hats
acc_log = np.mean([accuracy_score(y_hat, y_val) for y_hat in y_hats])
print( "accuracy of Logestic regression is :",f'{acc_log:0.2%}' )
```

```
accuracy of Logestic regression is : 81.66%
```

Light Gradiant Boosting Model

In [49]:

```
lgbm = LGBMClassifier(random_state=SEED)
cv = cross_validate(lgbm , X_train, y_train, cv =skf , scoring="accuracy",
model_lgbm = cv["estimator"]
```

```
[LightGBM] [Info] Number of positive: 326, number of negative: 456
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the over
head of testing was 0.002262 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 634
[LightGBM] [Info] Number of data points in the train set: 782, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.416880 -> initscore
=-0.335595
[LightGBM] [Info] Start training from score -0.335595
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

In [50]:

```
y_hats = [model.predict(X_val) for model in model_lgbm]
acc_lgbm = np.mean([accuracy_score(y_hat, y_val) for y_hat in y_hats])
print( "accuracy of LGBM is :",f'{acc_lgbm:0.2%}' )
```

```
accuracy of LGBM is : 81.29%
```

Exterme Gradiant Boosting

In [51]:
```
xgb  = XGBClassifier(random_state=SEED)
cv = cross_validate(xgb, x_train, y_train, cv =skf , scoring="accuracy", r
model_xgb = cv["estimator"]
```

In [52]:
```
y_hat = [model.predict(x_val) for model in model_xgb]
acc_xgb = np.mean([accuracy_score(y_hat, y_val)for y_hat in y_hats])
print("accuracy of XGB is :",f'{acc_xgb:0.2%}' )
```

```
accuracy of XGB is : 81.29%
```

CatBoost

In [53]:
```
cat = CatBoostClassifier(random_state=SEED)
cv = cross_validate(cat, x_train, y_train, cv =skf , scoring="accuracy", r
model_cat = cv["estimator"]
clear_output(wait=False)
```

```
654:	learn: 0.2127936	total: 1.57s	remaining: 830ms
655:	learn: 0.2125579	total: 1.58s	remaining: 827ms
656:	learn: 0.2124282	total: 1.58s	remaining: 825ms
657:	learn: 0.2123154	total: 1.58s	remaining: 822ms
658:	learn: 0.2122047	total: 1.58s	remaining: 820ms
659:	learn: 0.2120904	total: 1.58s	remaining: 817ms
660:	learn: 0.2119964	total: 1.59s	remaining: 814ms
661:	learn: 0.2118732	total: 1.59s	remaining: 812ms
662:	learn: 0.2116585	total: 1.59s	remaining: 809ms
663:	learn: 0.2115702	total: 1.59s	remaining: 807ms
664:	learn: 0.2113733	total: 1.6s	remaining: 804ms
665:	learn: 0.2113184	total: 1.6s	remaining: 802ms
666:	learn: 0.2112126	total: 1.6s	remaining: 799ms
667:	learn: 0.2111319	total: 1.6s	remaining: 797ms
668:	learn: 0.2111012	total: 1.6s	remaining: 794ms
669:	learn: 0.2109490	total: 1.61s	remaining: 791ms
670:	learn: 0.2109329	total: 1.61s	remaining: 788ms
671:	learn: 0.2108546	total: 1.61s	remaining: 786ms
672:	learn: 0.2106830	total: 1.61s	remaining: 783ms
673:	learn: 0.2105460	total: 1.61s	remaining: 780ms
```

In [54]:
```
y_hats = [model.predict(x_val) for model in model_cat]
acc_cat = np.mean([accuracy_score(y_hat, y_val) for y_hat in y_hats])
print( "accuracy of CatBoost is :",f'{acc_cat:0.2%}' )
```

```
accuracy of CatBoost is : 81.60%
```

ExtraTree

In [56]:
```
extree = ExtraTreesClassifier(random_state=SEED)
cv = cross_validate(extree, X_train, y_train, cv =skf , scoring="accuracy'
model_extree = cv["estimator"]
```

In [57]:
```
y_hats = [model.predict(x_val) for model in model_extree]
acc_ext = np.mean([accuracy_score(y_hat, y_val) for y_hat in y_hats])
print( "accuracy of extree is :",f'{acc_ext:0.2%}' )
```

```
accuracy of extree is : 70.62%
```

KNN

In [58]:
```
knn = KNeighborsClassifier()
cv = cross_validate(knn, x_train, y_train, cv =skf , scoring="accuracy", r
model_knn = cv["estimator"]
```

In [59]:
```
y_hats = [model.predict(x_val) for model in model_knn]
acc_knn = np.mean([accuracy_score(y_hat, y_val) for y_hat in y_hats])
print( "accuracy of KNN is :",f'{acc_knn:0.2%}' )
```

```
accuracy of KNN is : 76.36%
```

Gradiant Boosting

In [60]:
```
gbm = GradientBoostingClassifier(random_state=SEED)
cv = cross_validate(gbm, x_train, y_train, cv =skf , scoring="accuracy", r
model_gbm = cv["estimator"]
```

In [61]:
```
y_hats = [model.predict(x_val) for model in model_gbm]
acc_gra_bos = np.mean([accuracy_score(y_hat, y_val) for y_hat in y_hats])
print( "accuracy of Gradiant Boosting is :",f'{acc_gra_bos:0.2%}' )
```

```
accuracy of Gradiant Boosting is : 80.35%
```

Table of Results

In [62]:
```
print( "accuracy of Logestic regression is :",f'{acc_log:0.2%}' )
print( "accuracy of LGBM is :",f'{acc_lgbm:0.2%}' )
print( "accuracy of XGB is :",f'{acc_xgb:0.2%}' )
print( "accuracy of CatBoost is :",f'{acc_cat:0.2%}' )
print( "accuracy of extree is :",f'{acc_ext:0.2%}' )
print( "accuracy of KNN is :",f'{acc_knn:0.2%}' )
print( "accuracy of Gradiant Boosting is :",f'{acc_gra_bos:0.2%}' )
```

```
accuracy of Logestic regression is : 81.66%
accuracy of LGBM is : 81.29%
accuracy of XGB is : 81.29%
accuracy of CatBoost is : 81.60%
accuracy of extree is : 70.62%
accuracy of KNN is : 76.36%
accuracy of Gradiant Boosting is : 80.35%
```

As we can see, The best Accuracy is for Logestic Regression Model