

# ✓ Case Study: Predict Customer Churn Using Machine Learning

Prepared by Michelle Bonat

## 1: Frame the Problem

### Case Study Overview

Many companies would benefit immensely by investigating their customer churn and taking steps to improve it. Why? Simply stated, it usually takes much less resources to keep a customer than to get a new one. So if you have the information you need on why customers are leaving (churning) you can use this proactively to reduce your churn. Let's look at how we can develop this intelligence using machine learning.

---

### About The Data and The Approach

We're told by our colleagues at the hypothetical company that customer churn is at 50% within 3 months. That means that within 3 months of a set of customers that sign up for the paid product, by the end of 3 months half of them will have cancelled. This is an urgent problem we need to help fix with machine learning!

#### - Dataset:

For this dataset I used an IBM Sample Dataset for customer churn from the TelCo industry which can be found [here](#)

#### - Project Steps:

1. *\*Talk with stakeholders (S/H) to: \**

- **Discuss/determine goals and metrics for the project.** Is reducing churn the right problem to solve? How specifically shall we measure performance improvement? What is our success goal? (Assume here: it is the right problem, we measure performance overall by reducing customer churn, success is reducing customer churn by 10% in next 6 months).
  - **Understand what deliverables are useful for internal stakeholders** (Assume it is churn prediction factors, later a spreadsheet of customer churn predictions, production pipeline and perhaps an internal dashboard).
  - **Discuss an iterative approach with timing and deliverables**, at a high level. First see if the data is predictive. Set a goal for what predictive means (Assume here: 80% +). Come back with findings in 2 weeks.
2. **Work with SME's (Subject Matter Experts) to come up with a hypothesis.** Here we'll assume that tenure, add'l products/features, tech support interactions are key; also possibly how they pay.
  3. **Get the data.** Work with data engineering to get the data, if needed.
  4. **Talk to my team and resource the project.** Include team personal skills development where possible. In this case for example purposes, assume all the team is booked and I'll do the initial analysis which appears here.
  5. **Prototype some approaches.** Is the data usable? Can churn be reliably predicted? Do we need other data? Can it eventually be done at scale?
  6. **When a valid prototype is achieved, begin A/B testing.** If the prototype indicates a workable approach is viable, coordinate with S/H and team on next steps. Put A/B testing in place.
  7. **Iterate data and models learned from A/B testing and internal model testing.** Get to target success metrics on prediction rate and any other target metrics in order to move forward.
  8. *\*Create the production pipeline.* \*Once an initial model prototype has been established that surpasses the desired performance thresholds, it is ready for the production line.
    - Start simple with incremental wins. Expand as it makes sense with priorities and resources.
      - Load the flagged users (potential churners) back to the database as a first step
      - Eventually, move to automated data injection from a data warehouse
      - Refine the model and the data
      - Report performance to stakeholders and team on an ongoing basis. Get feedback and discuss next steps. Implement, learn, improve, repeat.
  9. **Suggest some ways to improve churn in products and processes based on learnings from the activities listed above.**

- These notes are listed at the bottom of this notebook.

## ✓ 2: SetUp: Importing the Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as ms
from sklearn import model_selection, metrics #to include metrics for evaluation # this used to be cross_validation
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

## ✓ 3: Access Data and Clean It Up

We're using the IBM Telco sample dataset for this case study. It provides some of the datapoints from our wish list but not all.

I staged the IBM sample dataset on my own S3 bucket on AWS and I'm accessing it from there:

```
# Read the data and view the top portion to see what we are dealing with.
data=pd.read_csv('Telco-Customer-Churn.csv')
data.head(10)
```



	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No
5	9305-CDSKC	Female	0	No	No	8	Yes	Yes	Fiber optic	No
6	1452-KIOVK	Male	0	No	Yes	22	Yes	Yes	Fiber optic	No
7	6713-OKOMC	Female	0	No	No	10	No	No phone service	DSL	Yes
8	7892-POOKP	Female	0	Yes	No	28	Yes	Yes	Fiber optic	No
9	6388-TABGU	Male	0	No	Yes	62	Yes	No	DSL	Yes

10 rows × 21 columns

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# See if the data is usable.
```

```
data.info()
```

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

```
# Analyze if there is non-numeric data in the 'TotalCharges' column since it's showing as an object instead of float64.
```

```
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors = 'coerce')
```

```
data.loc[data['TotalCharges'].isna()==True]
```



	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecuri
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service	DSL	Y
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No	No	No interr servi
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No	DSL	Y
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes	No	No interr servi
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service	DSL	Y
3331	7644-OMVMY	Male	0	Yes	Yes	0	Yes	No	No	No interr servi
3826	3213-VVOLG	Male	0	Yes	Yes	0	Yes	Yes	No	No interr servi
4380	2520-SGTTA	Female	0	Yes	Yes	0	Yes	No	No	No interr servi
5218	2923-ARZLG	Male	0	Yes	Yes	0	Yes	No	No	No interr servi
6670	4075-WKNIU	Female	0	Yes	Yes	0	Yes	Yes	DSL	I
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes	DSL	Y

11 rows × 21 columns

```
# Above we see that the blank "TotalCharges" happen when customers have 0 months tenure so we will change those values to $0.
data[data['TotalCharges'].isna()==True] = 0
data['OnlineBackup'].unique()
```

```
array(['Yes', 'No', 'No internet service', 0], dtype=object)
```

```
# See how many rows and columns.
data.shape
```

```
(7043, 21)
```

More data cleanup: next we'll convert the categorical values into numeric values.

```
data['gender'].replace(['Male','Female'],[0,1],inplace=True)
data['Partner'].replace(['Yes','No'],[1,0],inplace=True)
data['Dependents'].replace(['Yes','No'],[1,0],inplace=True)
data['PhoneService'].replace(['Yes','No'],[1,0],inplace=True)
data['MultipleLines'].replace(['No phone service','No', 'Yes'],[0,0,1],inplace=True)
data['InternetService'].replace(['No','DSL','Fiber optic'],[0,1,2],inplace=True)
data['OnlineSecurity'].replace(['No','Yes','No internet service'],[0,1,0],inplace=True)
data['OnlineBackup'].replace(['No','Yes','No internet service'],[0,1,0],inplace=True)
data['DeviceProtection'].replace(['No','Yes','No internet service'],[0,1,0],inplace=True)
data['TechSupport'].replace(['No','Yes','No internet service'],[0,1,0],inplace=True)
data['StreamingTV'].replace(['No','Yes','No internet service'],[0,1,0],inplace=True)
data['StreamingMovies'].replace(['No','Yes','No internet service'],[0,1,0],inplace=True)
data['Contract'].replace(['Month-to-month', 'One year', 'Two year'],[0,1,2],inplace=True)
data['PaperlessBilling'].replace(['Yes','No'],[1,0],inplace=True)
data['PaymentMethod'].replace(['Electronic check', 'Mailed check', 'Bank transfer (automatic)','Credit card (automatic)'],[0,1,2,3],
data['Churn'].replace(['Yes','No'],[1,0],inplace=True)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
#   Column                Non-Null Count  Dtype
#   ...
```

```

---  -----
0   customerID      7043 non-null   object
1   gender          7043 non-null   int64
2   SeniorCitizen   7043 non-null   int64
3   Partner         7043 non-null   int64
4   Dependents      7043 non-null   int64
5   tenure          7043 non-null   int64
6   PhoneService    7043 non-null   int64
7   MultipleLines    7043 non-null   int64
8   InternetService  7043 non-null   int64
9   OnlineSecurity   7043 non-null   int64
10  OnlineBackup     7043 non-null   int64
11  DeviceProtection 7043 non-null   int64
12  TechSupport      7043 non-null   int64
13  StreamingTV      7043 non-null   int64
14  StreamingMovies  7043 non-null   int64
15  Contract         7043 non-null   int64
16  PaperlessBilling 7043 non-null   int64
17  PaymentMethod    7043 non-null   int64
18  MonthlyCharges   7043 non-null   float64
19  TotalCharges     7043 non-null   float64
20  Churn            7043 non-null   int64

```

```
dtypes: float64(2), int64(18), object(1)
```

```
memory usage: 1.1+ MB
```

```
<ipython-input-10-c120edf7c7ec>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chain
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are s
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] =

```

data['gender'].replace(['Male','Female'],[0,1],inplace=True)
<ipython-input-10-c120edf7c7ec>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fu
data['gender'].replace(['Male','Female'],[0,1],inplace=True)
<ipython-input-10-c120edf7c7ec>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chain
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are s

```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] =

```

data['Partner'].replace(['Yes','No'],[1,0],inplace=True)
<ipython-input-10-c120edf7c7ec>:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fu

```



```
data['Partner'].replace(['Yes', 'No'], [1, 0], inplace=True)
<ipython-input-10-c120edf7c7ec>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainable assignment. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting the values is a copy.
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] =
```

```
data['Dependents'].replace(['Yes', 'No'], [1, 0], inplace=True)
<ipython-input-10-c120edf7c7ec>:3: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version of pandas.
data['Dependents'].replace(['Yes', 'No'], [1, 0], inplace=True)
<ipython-input-10-c120edf7c7ec>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainable assignment. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting the values is a copy.
```

Our goal is to avoid multicollinearity by dropping features that are closely correlated with each other. For example here it is TotalCharges and MonthlyCharges. So we will drop TotalCharges.

```
data.pop('TotalCharges')
```



### TotalCharges

0	29.85
1	1889.50
2	108.15
3	1840.75
4	151.65
...	...
7038	1990.50
7039	7362.90
7040	346.45
7041	306.60
7042	6844.50

7043 rows × 1 columns

**dtype:** float64

```
# Run info again to make sure TotalCharges has been dropped (popped off).
data.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
#   Column          Non-Null Count  Dtype
---  -
0   customerID      7043 non-null   object
1   gender          7043 non-null   int64
2   SeniorCitizen   7043 non-null   int64
3   Partner         7043 non-null   int64
4   Dependents      7043 non-null   int64
```

```

5 tenure 7043 non-null int64
6 PhoneService 7043 non-null int64
7 MultipleLines 7043 non-null int64
8 InternetService 7043 non-null int64
9 OnlineSecurity 7043 non-null int64
10 OnlineBackup 7043 non-null int64
11 DeviceProtection 7043 non-null int64
12 TechSupport 7043 non-null int64
13 StreamingTV 7043 non-null int64
14 StreamingMovies 7043 non-null int64
15 Contract 7043 non-null int64
16 PaperlessBilling 7043 non-null int64
17 PaymentMethod 7043 non-null int64
18 MonthlyCharges 7043 non-null float64
19 Churn 7043 non-null int64
dtypes: float64(1), int64(18), object(1)
memory usage: 1.1+ MB

```

Rerun corr chart after cleanup. TotalCharges should not appear in the corr chart.

## ✓ 4: Explore The Data

```

# Explore how many churn data points we have.
print(len(data['Churn']))

```

↔ 7043

```

# Explore how many customers in this dataset have churned. Is this dataset 50% as the team suggests is the overall customer churn rate?
data['Churn'].value_counts()
# We see this dataset actually has less than the overall 50% churn rate of the entire company reported data (it's actually 26.54% of the

```



count

Churn

0 5174

1 1869

**dtype:** int64

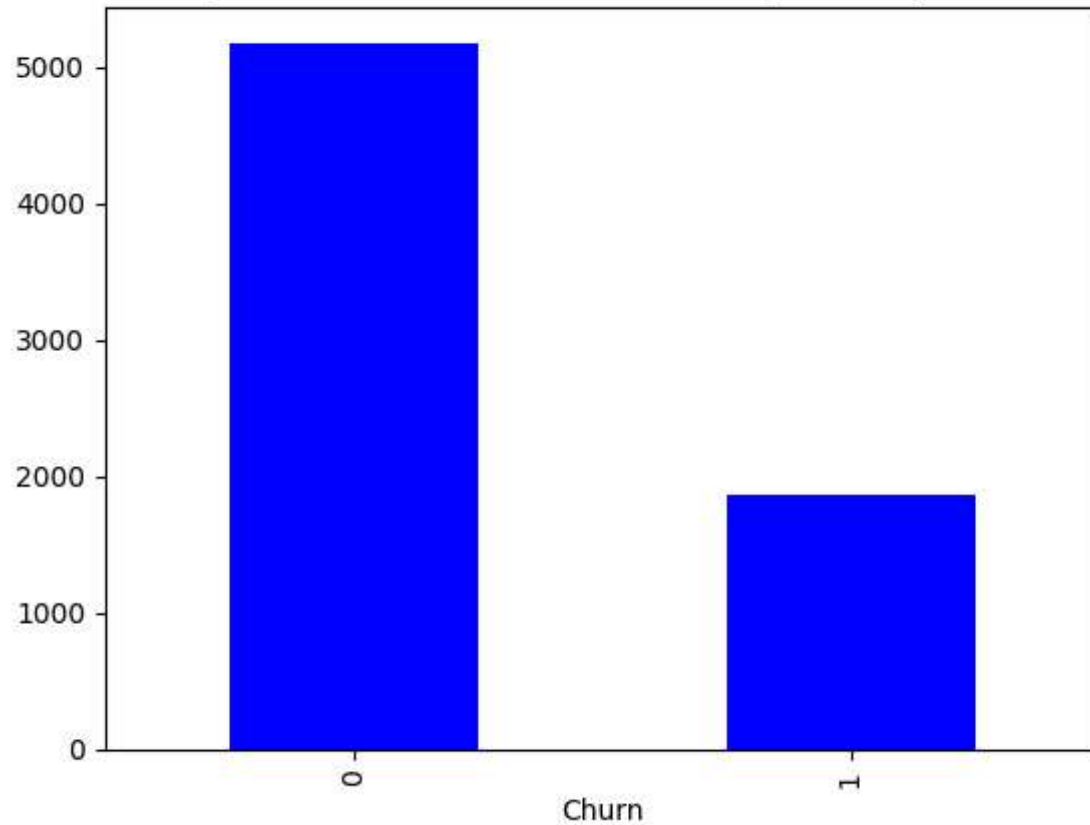
```
# This creates a bar graph of churn (Yes vs. No) so we can check how the data is balanced.
```

```
data['Churn'].value_counts().plot(kind = 'bar', title = 'Bar Graph of Non-Churners vs Churners by Count (Churn is a 1)', color = 'b')  
plt.show()
```

```
# The dataset does not have a huge imbalance which is good news! But also we clearly see it does not have the 50% as we would have t
```




Bar Graph of Non-Churners vs Churners by Count (Churn is a 1)



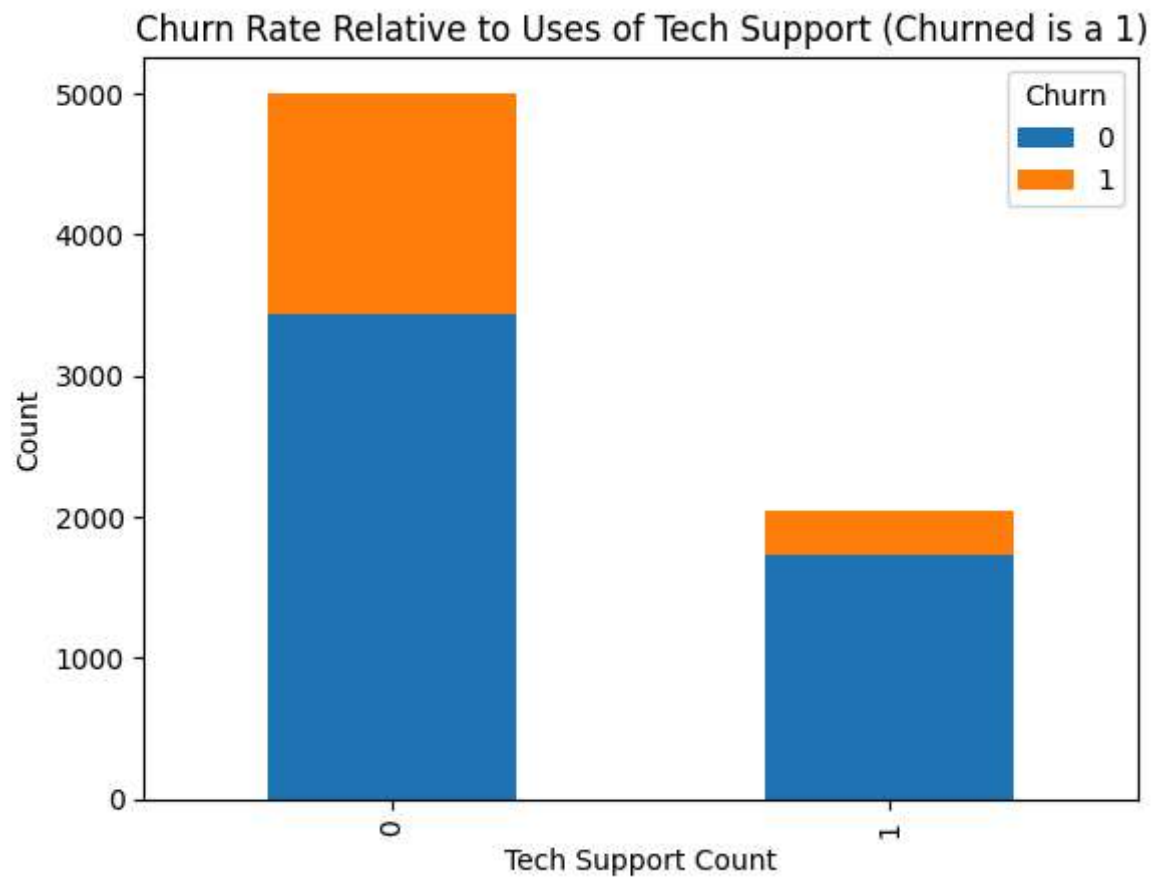
Explore some contingencies on how some features relate to churn.

```
# Creates initial contingency table between Churn and gender. Male is 0, Female is 1.  
gender_churn_contingency = pd.crosstab(data["gender"], data["Churn"])  
display(gender_churn_contingency)  
# Male and females churn at about the same rate, so not much to see here. Let's keep moving.
```

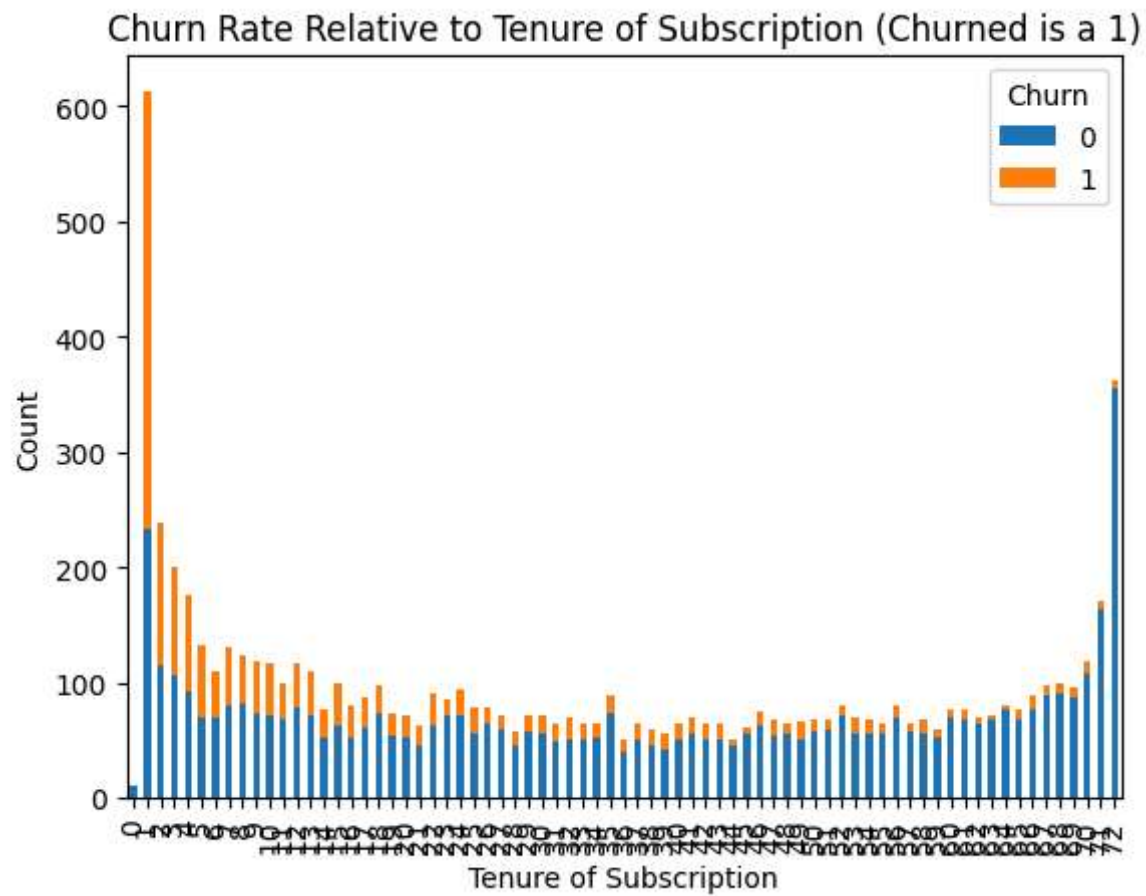


	Churn	
	0	1
gender		
0	2630	930
1	2544	939

```
# Explore the relationship between instances of Tech Support and Churn.
# Stacked Bar of Tech Support and Churn.
tech_support_churn = pd.crosstab(data['TechSupport'], data['Churn'])
tech_support_churn.plot(kind = 'bar', stacked = True)
plt.ylabel('Count')
plt.xlabel('Tech Support Count')
plt.title('Churn Rate Relative to Uses of Tech Support (Churned is a 1)')
plt.show()
# We can see that non-churners use tech support more often than customers that end up churning.
# So let's explore some ways to get people to use Tech Support more often so they cancel (churn) less. You can see notes for this at
```



```
# Churn rate relative to tenure.  
# Stacked bar of tenure and churn.  
tenure_churn = pd.crosstab(data['tenure'], data['Churn'])  
tenure_churn.plot(kind = 'bar', stacked = True)  
plt.ylabel('Count')  
plt.xlabel('Tenure of Subscription')  
plt.title('Churn Rate Relative to Tenure of Subscription (Churned is a 1)')  
plt.show()  
# We can clearly see the longer a customer stays as a subscriber, the less they are likely to churn!
```



```
# Distribution of features.
```

```
features = ['gender', 'SeniorCitizen','Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'Online  
data[features].describe()
```





	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecu
<b>count</b>	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.00
<b>mean</b>	0.494534	0.162147	0.481755	0.298026	32.371149	0.901888	0.421269	1.222206	0.28
<b>std</b>	0.500006	0.368612	0.499702	0.457424	24.559481	0.297487	0.493798	0.779535	0.45
<b>min</b>	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
<b>25%</b>	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	1.000000	0.00
<b>50%</b>	0.000000	0.000000	0.000000	0.000000	29.000000	1.000000	0.000000	1.000000	0.00
<b>75%</b>	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	1.000000	2.000000	1.00
<b>max</b>	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	1.000000	2.000000	1.00

```
# Does how a customer pays have to do with their churn?
```

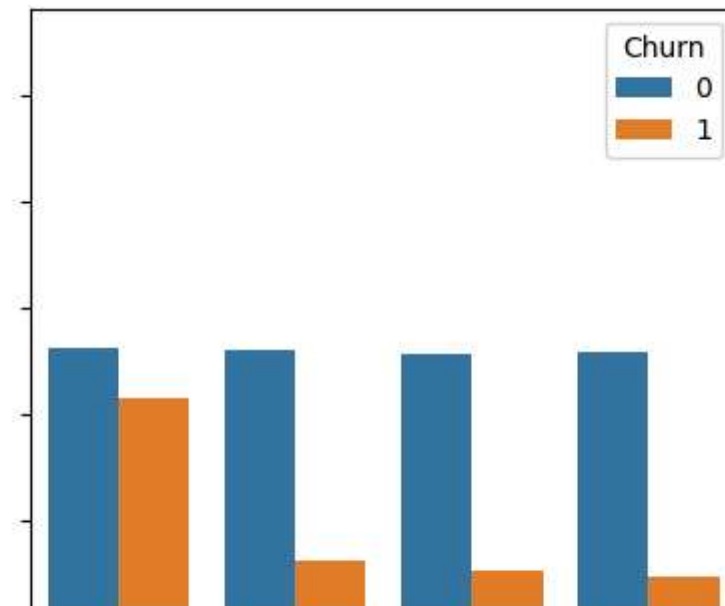
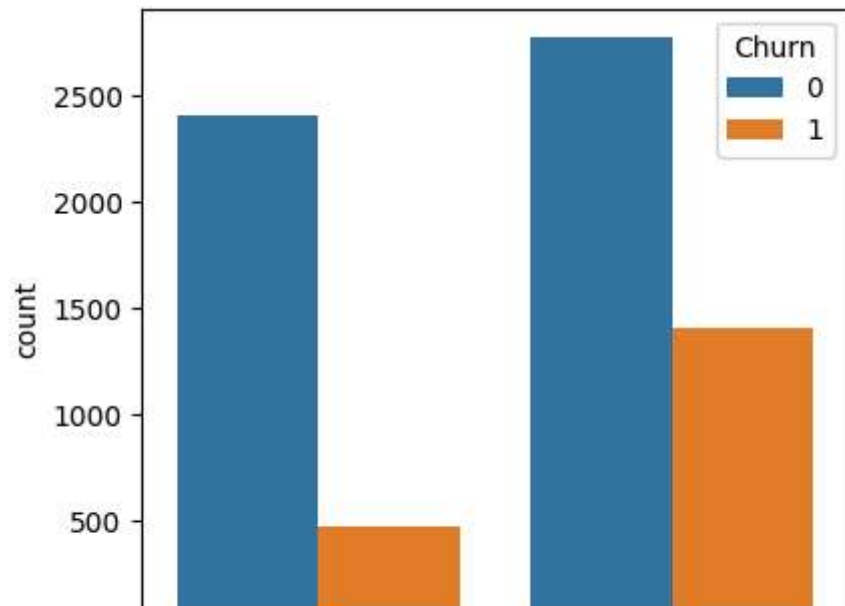
```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
```

```
sns.countplot(x='PaperlessBilling', hue='Churn',  
              data=data, ax=axes[0]);
```

```
sns.countplot(x='PaymentMethod', hue='Churn',  
              data=data, ax=axes[1]);
```

```
# We can see that customers that use paperless billing are much more likely to churn (0 = don't have paperless billing). That seems
```

```
# We can see that customers that have the 0 payment method (electronic check) are much more likely to churn. Let's discourage that c
```



# See if the other products they have from this company has to do with their churn.

```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
```

```
sns.countplot(x='PhoneService', hue='Churn',  
              data=data, ax=axes[0]);
```

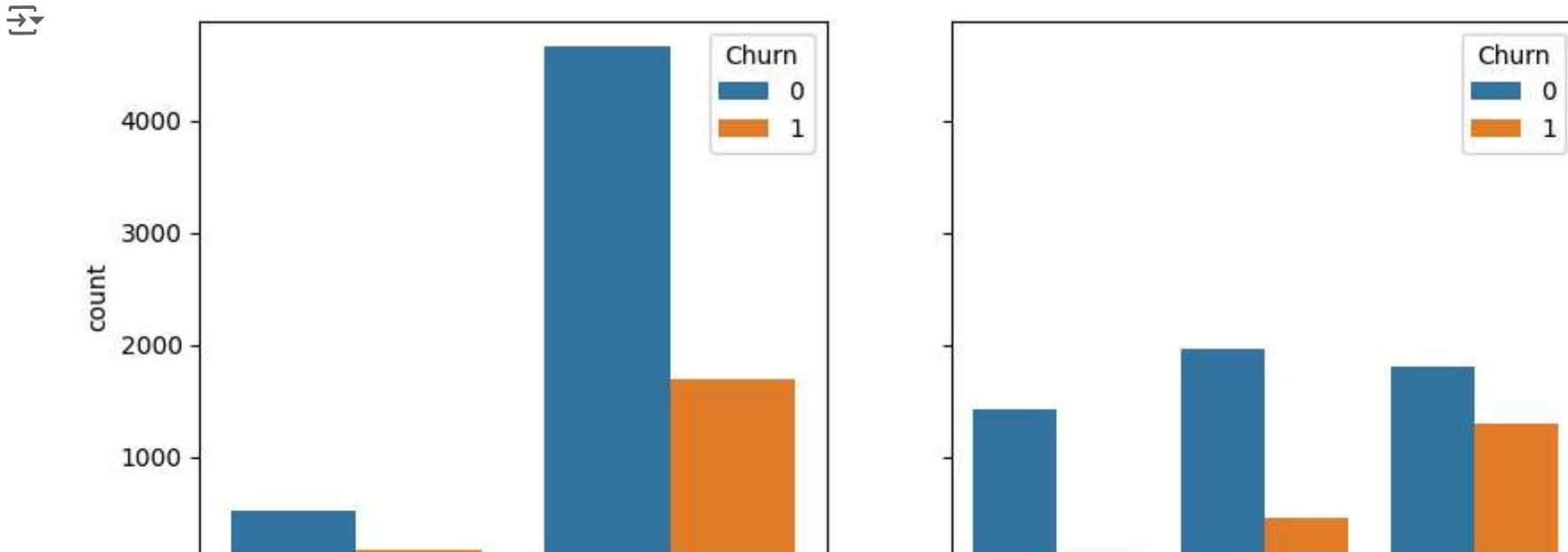
```
sns.countplot(x='InternetService', hue='Churn',  
              data=data, ax=axes[1]);
```

# If they don't have Phone Service, they are more likely to churn.

# If they don't have Internet Service, they are more likely to churn. Those customers with the highest Internet Service are least li

# Conclusion: This makes sense. Customers with other products from the company, and premium products, churn less.

# Offer customers these additional products, perhaps even at a deep discount, so they take them and are less likely to churn.



## ✓ 5: Prepare the Data

# Splitting the data for testing and training.

Double-click (or enter) to edit

```
X_train, X_test, y_train, y_test = train_test_split(data.drop('Churn',axis=1),
                                                    data['Churn'], test_size=0.30,
                                                    random_state=101)
```

```
train=pd.concat([X_train,y_train],axis=1)
```

# Function to estimate the best value of n\_estimators and fit the model with the given data.

```

def modelfit(alg, dtrain, predictors, useTrainCV=True, cv_folds=5, early_stopping_rounds=50):

    if useTrainCV:
        #to get the parameters of xgboost
        xgb_param = alg.get_xgb_params()

        #to convert into a datastructure internally used by xgboost for training efficiency
        # and speed
        xgtrain = xgb.DMatrix(dtrain[predictors].values, label=dtrain[target].values)

        #xgb.cv is used to find the number of estimators required for the parameters
        # which are set
        cvresult = xgb.cv(xgb_param, xgtrain,
                           num_boost_round=alg.get_params()['n_estimators'], nfold=cv_folds,
                           metrics='auc', early_stopping_rounds=early_stopping_rounds)

        #setting the n_estimators parameter using set_params
        alg.set_params(n_estimators=cvresult.shape[0])

        print(alg.get_xgb_params())

    #Fit the algorithm on the data
    alg.fit(dtrain[predictors], dtrain['Churn'], eval_metric='auc')

    return alg

# Function to get the accuracy of the model on the test data given the features considered.

```

```
def get_accuracy(alg, predictors):
    dtrain_predictions = alg.predict(X_test[predictors])
    dtrain_predprob = alg.predict_proba(X_test[predictors])[:,1]
    print ("\nModel Report")
    print ("Accuracy : %.4g" % metrics.accuracy_score(y_test.values,
                                                    dtrain_predictions))

    print ("AUC Score (Train): %f" % metrics.roc_auc_score(y_test.values,
                                                            dtrain_predprob))

# Function to get the feature importances based on the model fit.

def get_feature_importances(alg):
    #to get the feature importances based on xgboost we use fscore
    feat_imp = pd.Series(alg._Booster.get_fscore()).sort_values(ascending=False)
    print(feat_imp)

    #this shows the feature importances on a bar chart
    feat_imp.plot(kind='bar', title='Feature Importances')
    plt.ylabel('Feature Importance Score')

target = 'Churn'
IDcol = 'customerID'
```

## ✓ 6: Model Selection, Predictions, and Metrics

```
!pip install xgboost
# XGBoost converts weak learners to strong learners through an ensemble method.
# Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous model
```

```
➡ Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
def XgbClass(learning_rate =0.1,n_estimators=1000,max_depth=5,min_child_weight=1,
             gamma=0,subsample=0.8,colsample_bytree=0.8):
    xgb1 = XGBClassifier(learning_rate=learning_rate,
                        n_estimators=n_estimators,
                        max_depth=max_depth,
                        min_child_weight=min_child_weight,
                        gamma=gamma,
                        subsample=subsample,
                        colsample_bytree=colsample_bytree)

    return xgb1
```

# Function to return the list of predictors.

# These are the initial parameters before tuning.

```
def drop_features(l):
    return [x for x in train.columns if x not in l]
```

## ✓ First Prediction: Use of initial parameters and without feature engineering

```
from xgboost import XGBClassifier
import xgboost as xgb
```

## ✓ Second Prediction: Using initial Parameters and removing features of least importances

# Model after removing the features of least importance.

```
drop1=['DeviceProtection','Dependents','Dependents','gender','StreamingMovies','MultipleLines']
```

```
drop1_first=drop1+[target,IDcol]
```

### ✓ Third Prediction: Again removing the features of least importance

```
drop11=drop1+['Partner','PhoneService','OnlineBackup','TechSupport','OnlineSecurity']
```

```
drop1_second=drop1_first+['Partner','PhoneService','OnlineBackup','TechSupport','OnlineSecurity']
```

```
predictors=drop_features(drop1_second)
```

```
get_accuracy(third_model,predictors)
```

```
# Tune max_depth and min_child_weight.
```