```
#import 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.pylab as plt
%matplotlib inline
```

```
# Quiet warnings since this is a demo (it quiets future and deprecation warnings).
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

```
# Read the data and view the top portion to see what we are dealing with.
data=pd.read_csv('/content/telecom chrn.csv')
data.head()
```

	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 rows × 21 columns

```
# 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	OnlineSecurity
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No	No	No internet service
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No	DSL	Yes
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes
3331	7644-OMVMY	Male	0	Yes	Yes	0	Yes	No	No	No internet service
3826	3213-VVOLG	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service
4380	2520-SGTTA	Female	0	Yes	Yes	0	Yes	No	No	No internet service
5218	2923-ARZLG	Male	0	Yes	Yes	0	Yes	No	No	No internet service
6670	4075-WKNIU	Female	0	Yes	Yes	0	Yes	Yes	DSL	No
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes	DSL	Yes

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],inp
data['Churn'].replace(['Yes','No'],[1,0],inplace=True)
```

```
data.info()
```

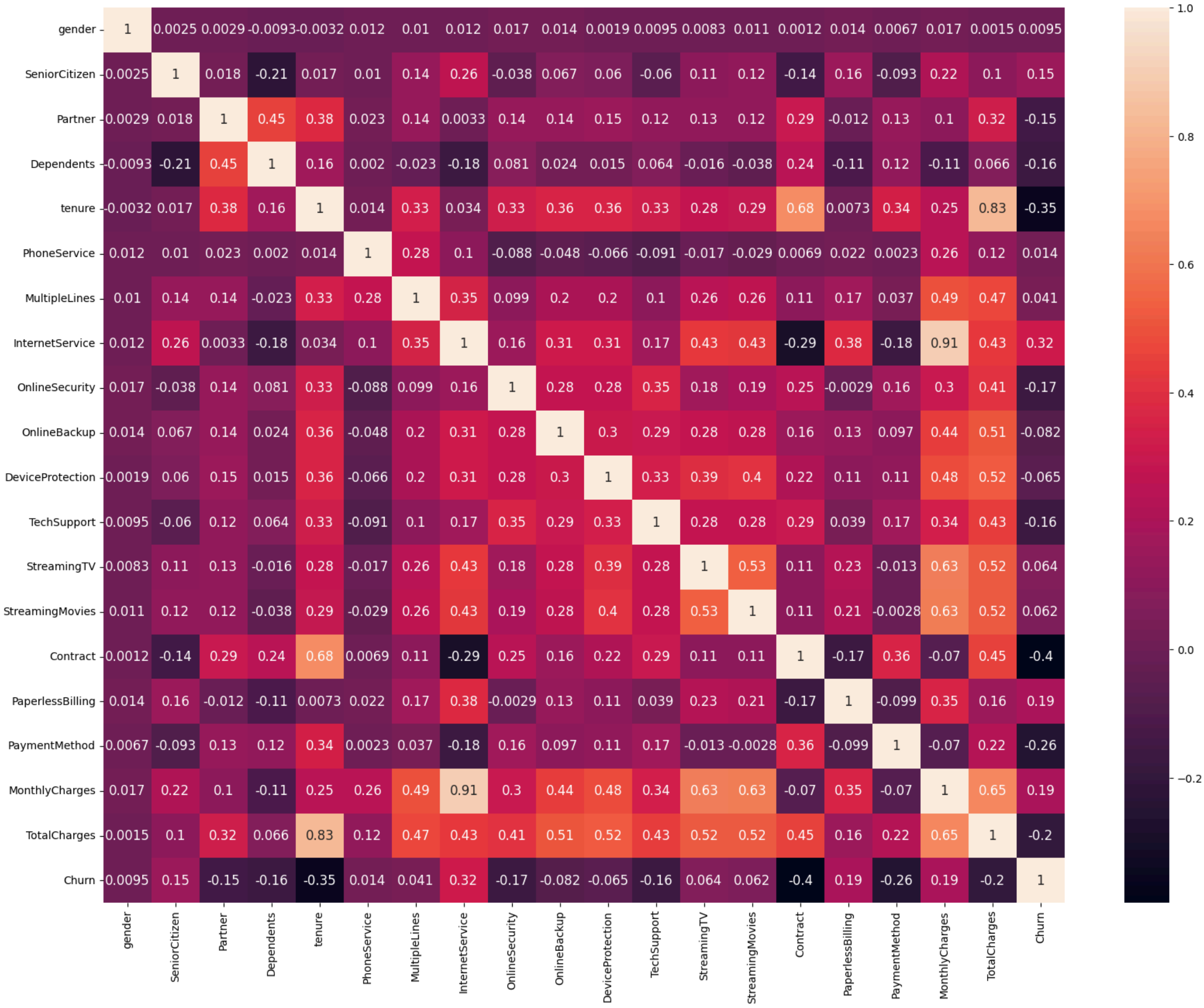
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   int64
 2   SeniorCitizen     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
```

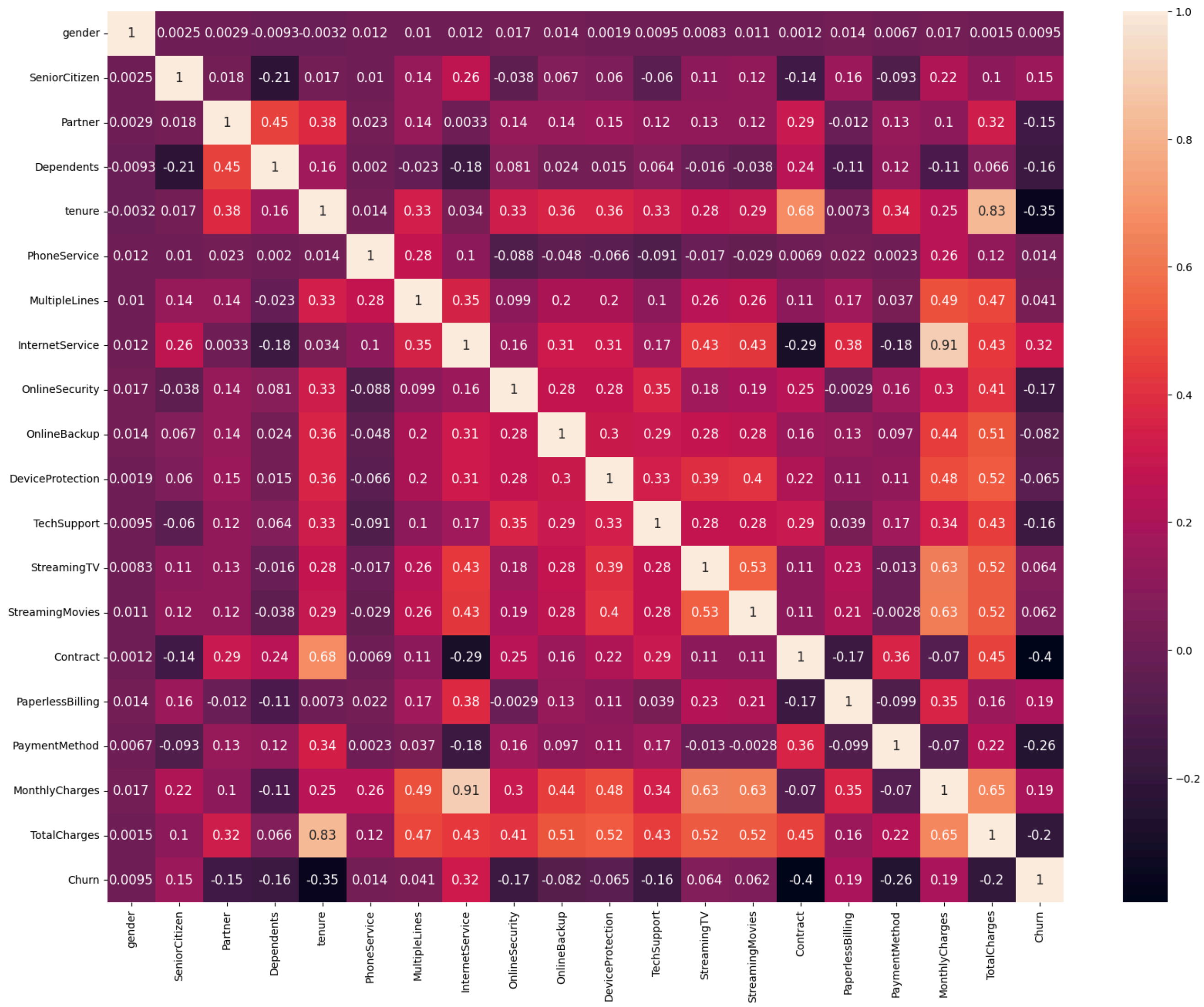
Let's look at relationships between customer data and churn using correlation.

```
# Drop the customerID column as it's not needed for analysis or modeling
data.drop('customerID', axis=1, inplace=True)

# Let's look at relationships between customer data and churn using correlation.
corr = data.corr()
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws={'size':12})
heat_map=plt.gcf()
heat_map.set_size_inches(20,15)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```




```
corr = data.corr()
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws={'size':12})
heat_map=plt.gcf()
heat_map.set_size_inches(20,15)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



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

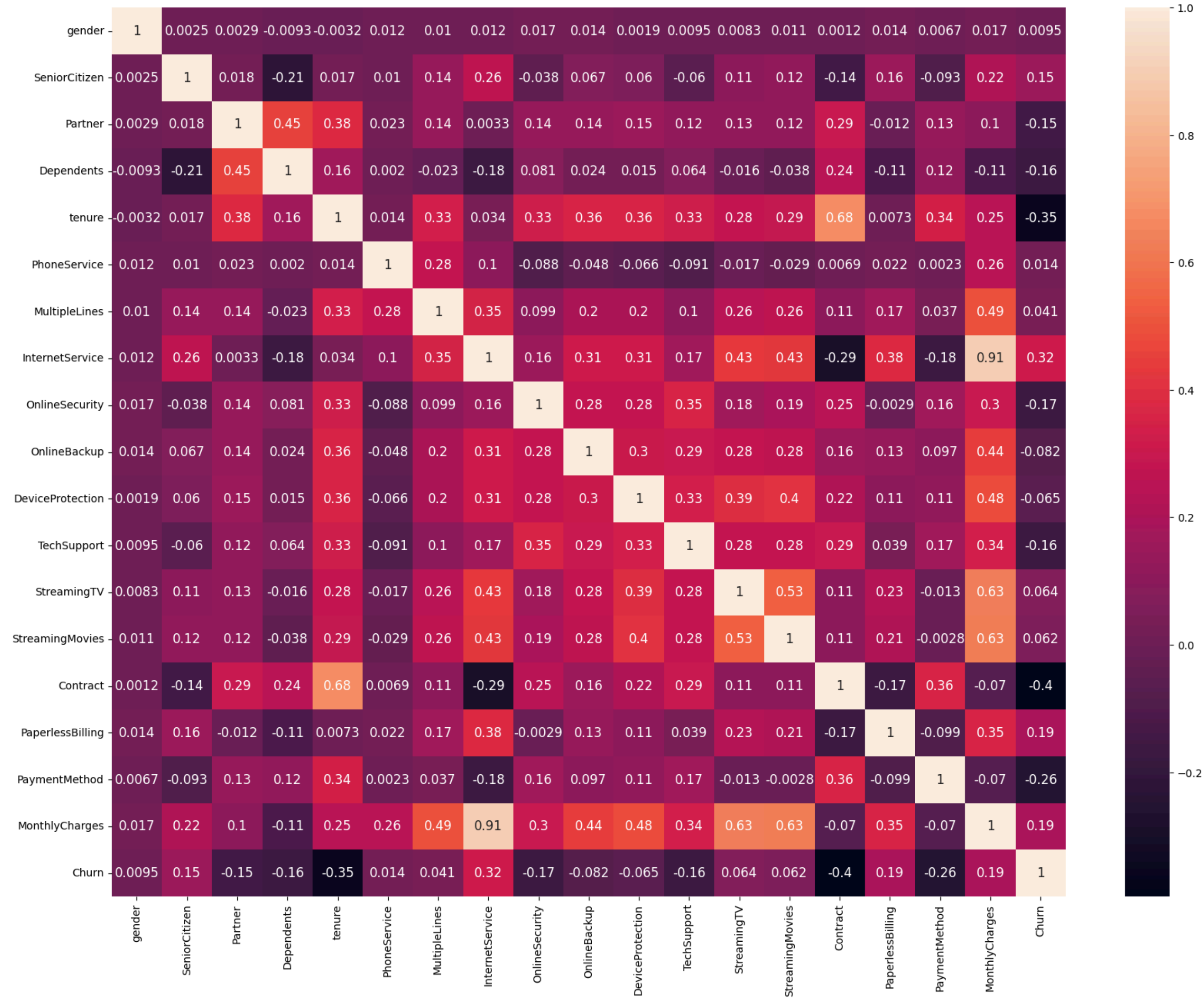
dtype: float64

```
# 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 19 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            7043 non-null   int64
 1   SeniorCitizen     7043 non-null   int64
 2   Partner           7043 non-null   int64
 3   Dependents        7043 non-null   int64
 4   tenure            7043 non-null   int64
 5   PhoneService      7043 non-null   int64
 6   MultipleLines     7043 non-null   int64
 7   InternetService   7043 non-null   int64
 8   OnlineSecurity    7043 non-null   int64
 9   OnlineBackup      7043 non-null   int64
 10  DeviceProtection  7043 non-null   int64
 11  TechSupport       7043 non-null   int64
 12  StreamingTV       7043 non-null   int64
 13  StreamingMovies   7043 non-null   int64
 14  Contract          7043 non-null   int64
 15  PaperlessBilling  7043 non-null   int64
 16  PaymentMethod     7043 non-null   int64
 17  MonthlyCharges    7043 non-null   float64
 18  Churn             7043 non-null   int64
dtypes: float64(1), int64(18)
memory usage: 1.0 MB
```

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

```
corr = data.corr()
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws={'size':12})
heat_map=plt.gcf()
heat_map.set_size_inches(20,15)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



5: Explore The Data

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

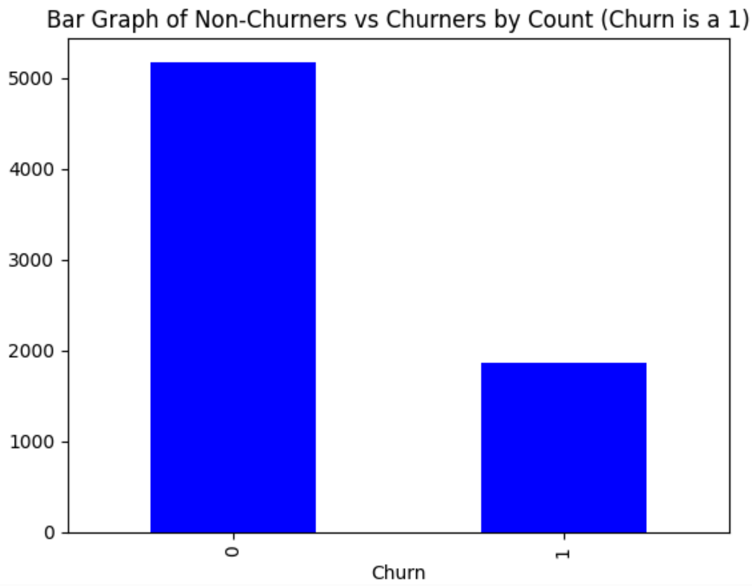
```
7043
```

```
# 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% that h
```

Churn	count
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 = 'blue',
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 though
```



Explore some contingencies on how some features relate to churn.

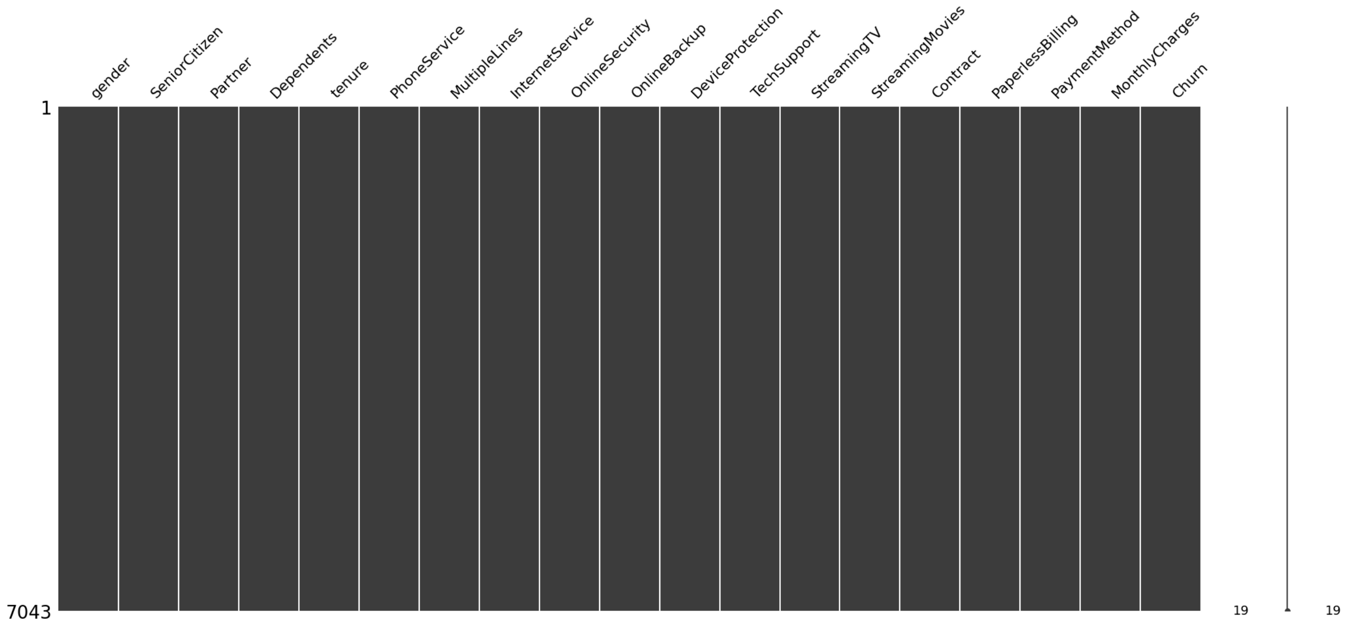
```
# Creates initial contingency table between Churn and gender. Male is 0, Female is 1.
gender_churn_contingency = pd.crosstab(data["gender"], data["Churn"])
display(gender_churn_contingency)
# Male and females churn at about the same rate, so not much to see here. Let's keep moving.
```

gender \ Churn	0	1
0	2630	930
1	2544	939

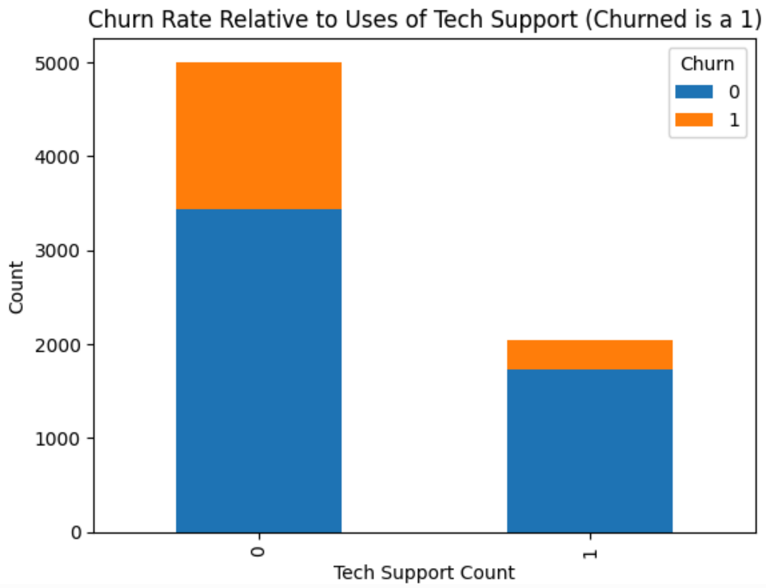
Next steps: Generate code with gender_churn_contingency View recommended plots New interactive sheet

```
# Check the data health. The sections should all be completely black indicating the data is complete.
ms.matrix(data)
# It looks good.
```

<Axes: >

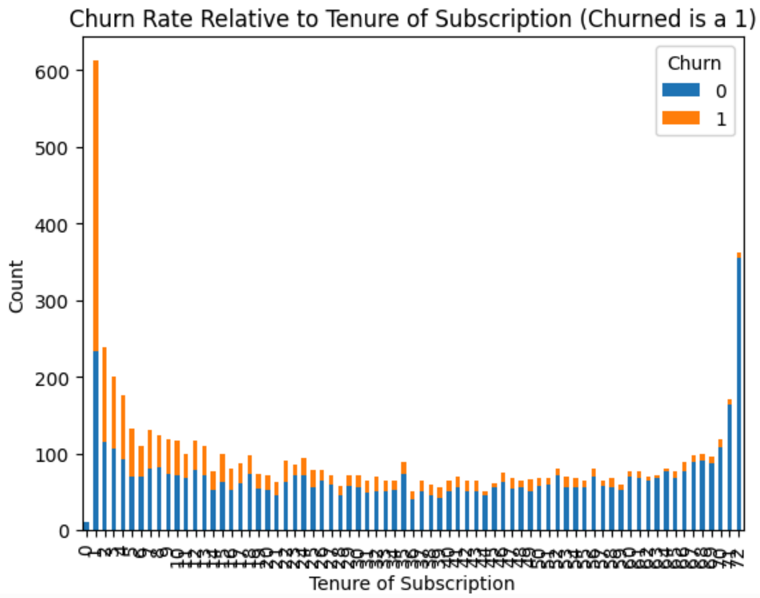



```
# Explore the relationship between instances of Tech Support and Churn.
# Stacked Bar of Tech Support and Churn.
tech_support_churn = pd.crosstab(data['TechSupport'], data['Churn'])
tech_support_churn.plot(kind = 'bar', stacked = True)
plt.ylabel('Count')
plt.xlabel('Tech Support Count')
plt.title('Churn 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 the
# Also, tech support in this data is just a Y/N. It would be useful in future to include how many tech support calls by customer so we co
```




```
# 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()
```

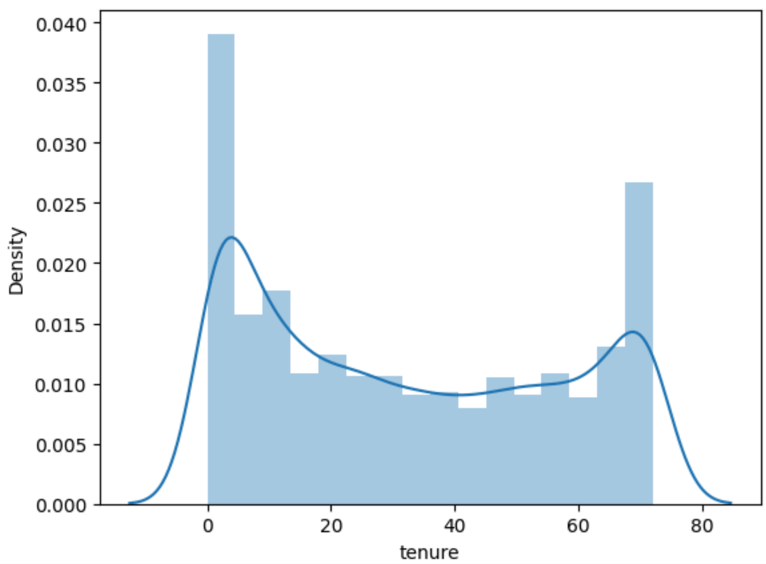
```
plt.show()
# We can clearly see the longer a customer stays as a subscriber, the less they are likely to churn!
```




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

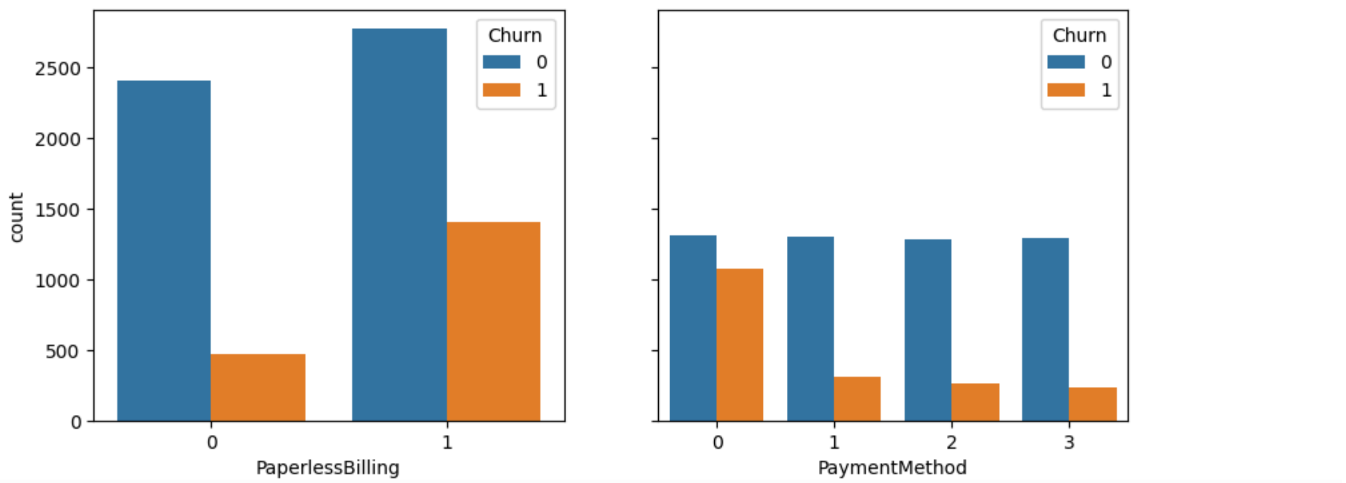
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.00000
mean	0.494534	0.162147	0.481755	0.298026	32.371149	0.901888	0.421269	1.222206	0.28610
std	0.500006	0.368612	0.499702	0.457424	24.559481	0.297487	0.493798	0.779535	0.45196
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	1.000000	0.00000
50%	0.000000	0.000000	0.000000	0.000000	29.000000	1.000000	0.000000	1.000000	0.00000
75%	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	1.000000	2.000000	1.00000
max	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	1.000000	2.000000	1.00000

```
# Plot the distribution of observations for tenure.
sns.distplot(data['tenure']);
# It shows the max tenure is 70. This must be when the data history ends. We'll account for this in our analysis.
```

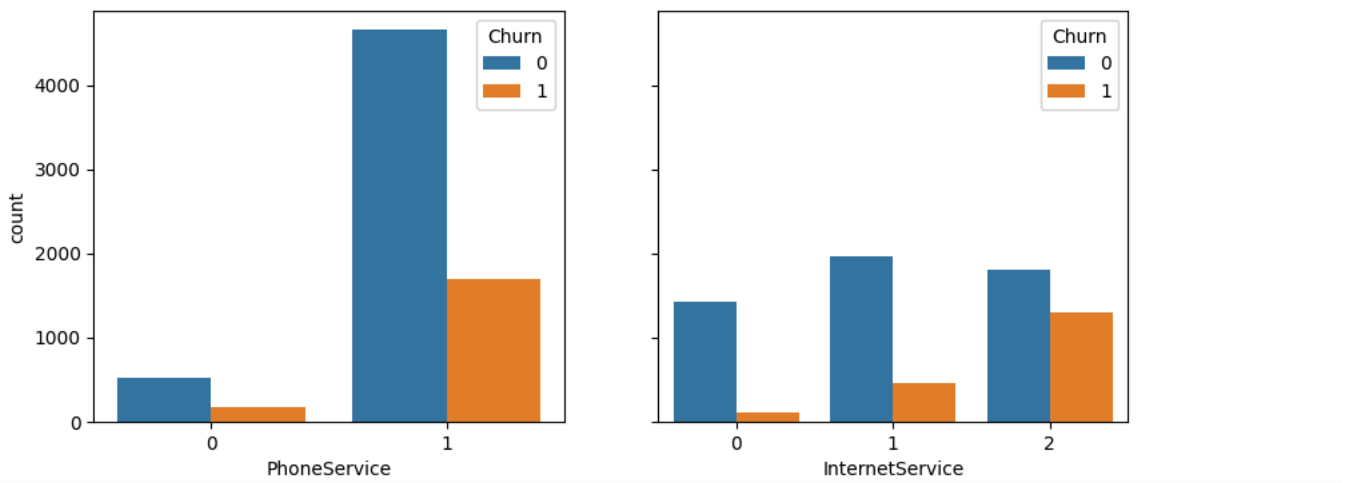



```
# Does how a customer pays have to do with their churn?
_, axes = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
sns.countplot(x='PaperlessBilling', hue='Churn',
              data=data, ax=axes[0]);
sns.countplot(x='PaymentMethod', hue='Churn',
```

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




```
# 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 likely
# 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.
```



6: Prepare the Data

```
# Splitting the data for testing and training.
```

```
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'
```

7: Model Selection, Predictions, and Metrics

```
# To return the XGBClassifier object based on the values of the features.
```

```
!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 models.
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.15.3)
```

```
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


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
    # Remove the eval_metric argument from here as it's not a direct parameter of fit()
    alg.fit(dtrain[predictors], dtrain['Churn'])

    return alg


predictors = drop_features([target, IDcol])
xgb1=XgbClass()
first_model=modelfit(xgb1, train, predictors)
xgb1.fit(train[predictors],train['Churn'])
```

```
{'objective': 'binary:logistic', 'base_score': None, 'booster': None, 'colsample_bylevel': None, 'colsample_bynode': None, 'colsampl
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=33, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
get_accuracy(first_model,predictors)
```

```
Model Report
Accuracy : 0.8055
AUC Score (Train): 0.848949
```

Accuracy is the proportion of true positives and negatives in the whole data set. It determines if a value is accurate compare it to the accepted value; the nearness of a calculation to the true value.

AUC (area under the ROC receiver operating characteristic curve) measures how true positive rate (recall) and false positive rate trade off, so in that sense it is already measuring something else. More importantly, AUC is not a function of threshold. It is an evaluation of the classifier as threshold varies over all possible values. It is in a sense a broader metric, testing the quality of the internal value that the classifier generates and then compares to a threshold.

Accuracy vs AUC: The accuracy depends on the threshold chosen, whereas the AUC considers all possible thresholds. Because of this it is often preferred as it provides a “broader” view of the performance of the classifier, but they still measure different things and as such using one or the other is problem-dependent.

```
get_feature_importances(first_model)
```

```
MonthlyCharges      240.0
tenure              184.0
PaymentMethod        71.0
Contract             46.0
InternetService      44.0
SeniorCitizen        34.0
PhoneService         28.0
PaperlessBilling     26.0
Partner              26.0
TechSupport          25.0
OnlineSecurity       25.0
OnlineBackup         25.0
StreamingTV          22.0
MultipleLines        22.0
DeviceProtection     20.0
StreamingMovies      20.0
gender               19.0
Dependents           19.0
dtype: float64
```