Telecom Industry Customer Churn Prediction

By : Varsha Waingankar

Overview

- Data source
- Data cleaning and preprocessing
- Code
- Visualizations
- Handling imbalanced data
- Feature engineering
- Predictive modeling
- Accuracy and best model

Data source and Problem Statement

- **Data source**
  https://www.ibm.com/communities/analytics/watson-analytics-blog/guide-to-sample-datasets/

- Predict behavior to retain customers. To analyze all relevant customer data and develop focused customer retention programs.

Data Frame

```
#Output of how the dataframe looks
telecom_cust.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No

5 rows × 21 columns

```
#The columns of the data frame
telecom_cust.columns.values
```

```
array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn'], dtype=object)
```

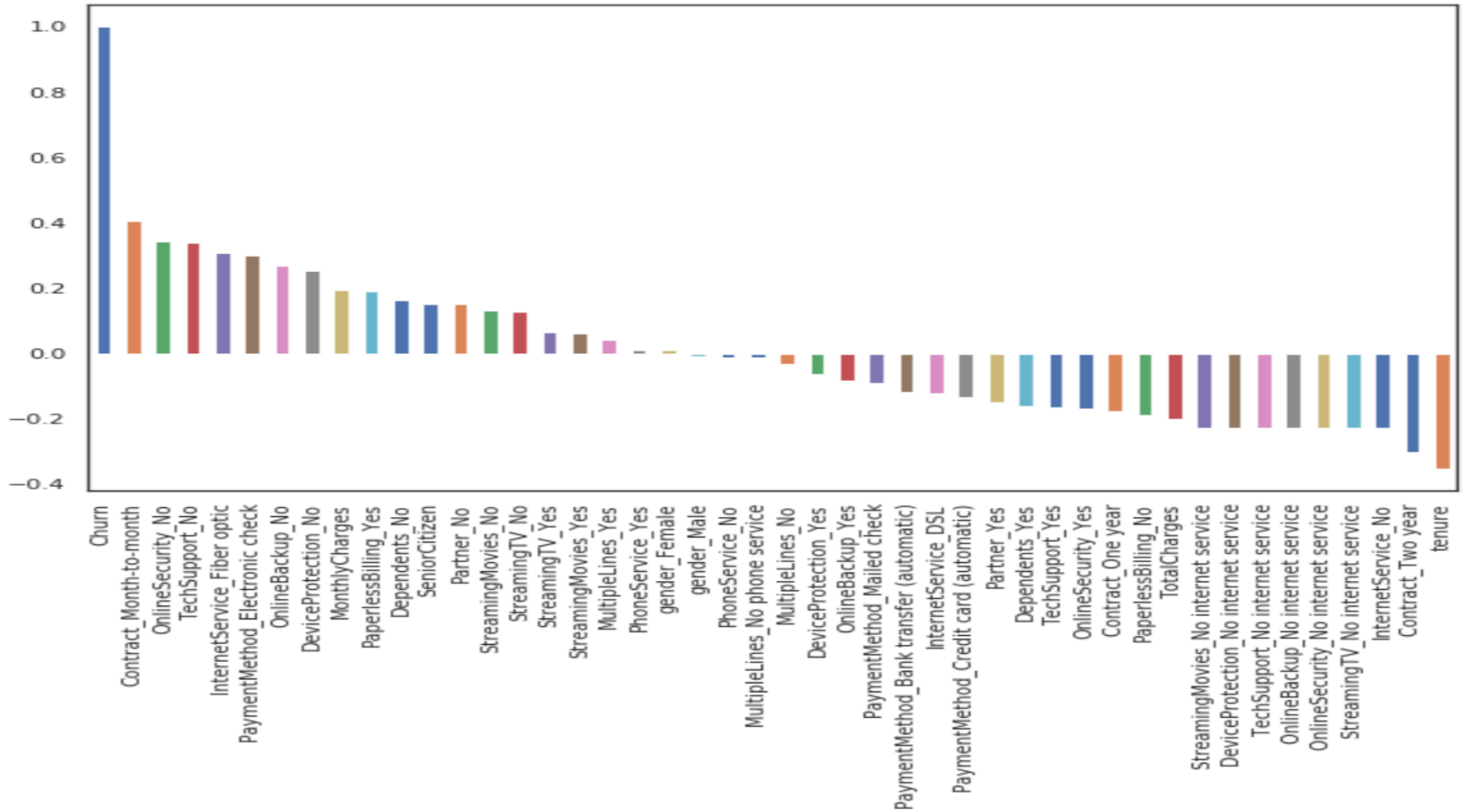
Data Manipulation

```
#Removing missing values  ---All nulls are dropped
telecom_cust.dropna(inplace = True)
#Remove customer IDs from the data set - customer id doesnt hold much information in predictive analysis
df2 = telecom_cust.iloc[:,1:]

#Converting the predictor variable in a binary numeric variable(Transforming from categorical to bindary)
df2['Churn'].replace(to_replace='Yes', value=1, inplace=True)
df2['Churn'].replace(to_replace='No',  value=0, inplace=True)

#Let's convert all the categorical variables into dummy variables
#All the categorical variables are converted into binary
#Can aslo be done using Label Encoder
df_dummies = pd.get_dummies(df2)
df_dummies.head()
```

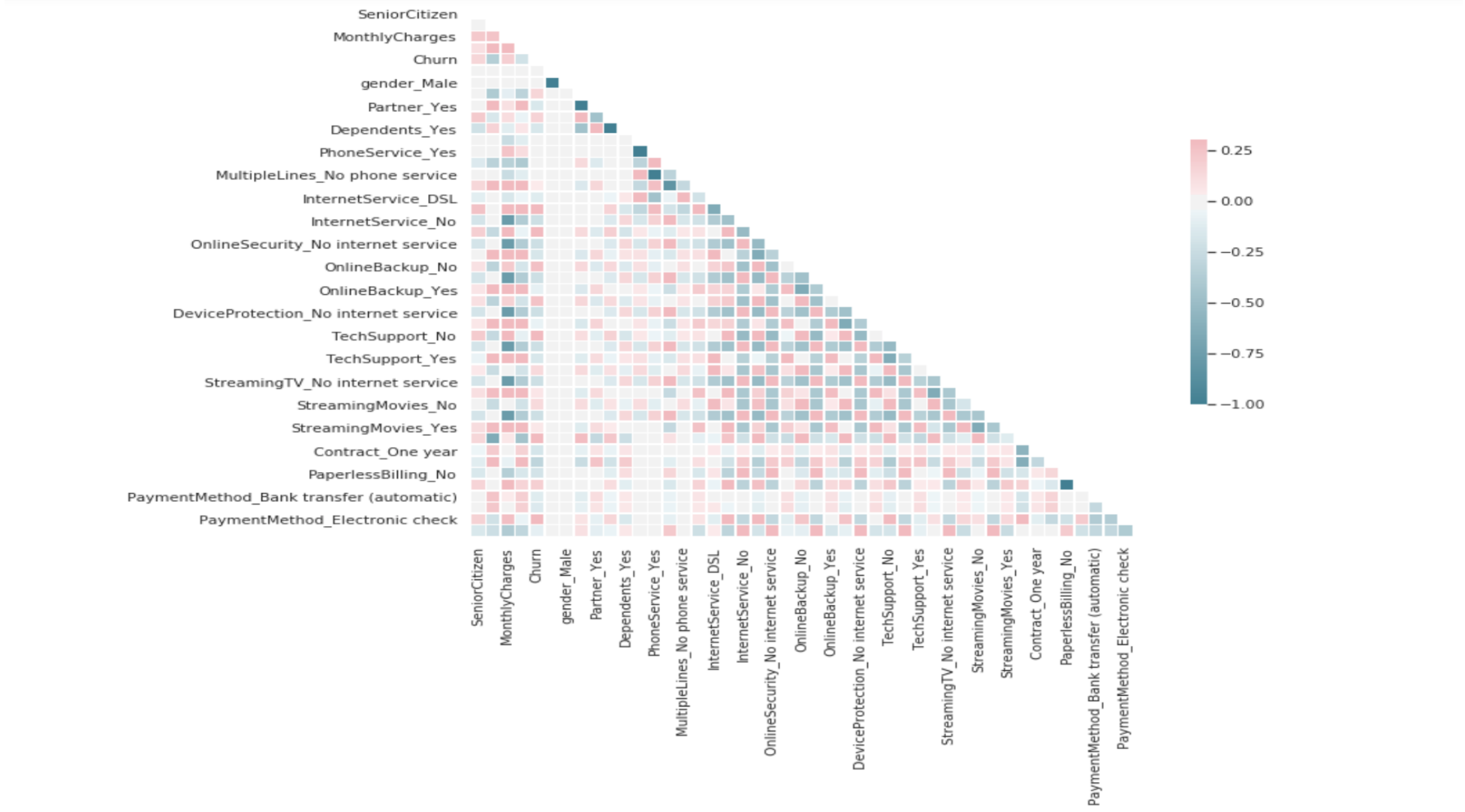
	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	...	StreamingMovies_Yes
0	0	1	29.85	29.85	0	1	0	0	1	1	...	0
1	0	34	56.95	1889.50	0	0	1	1	0	1	...	0
2	0	2	53.85	108.15	1	0	1	1	0	1	...	0
3	0	45	42.30	1840.75	0	0	1	1	0	1	...	0
4	0	2	70.70	151.65	1	1	0	1	0	1	...	0

5 rows × 46 columns

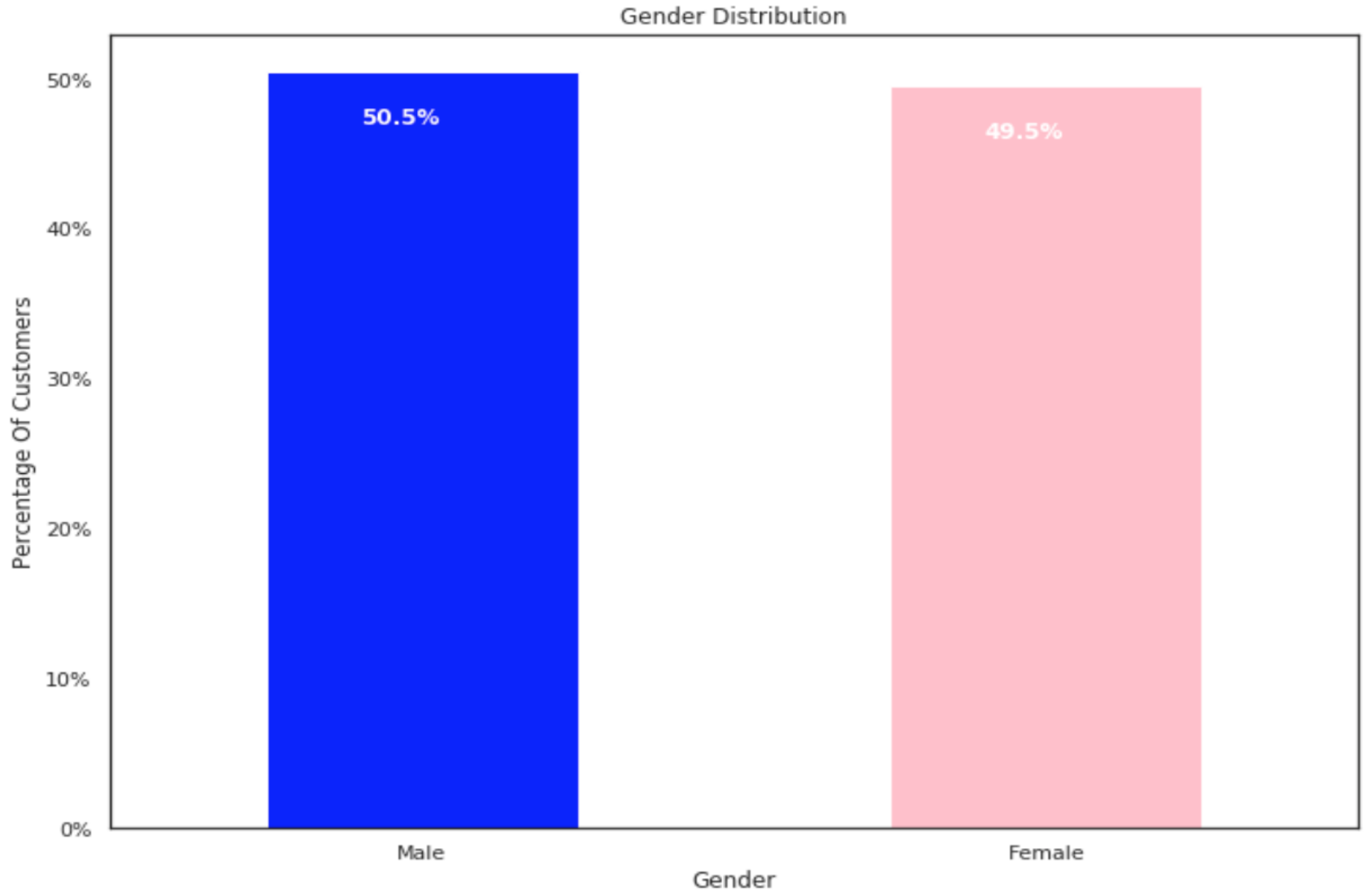
Correlation Of Variables



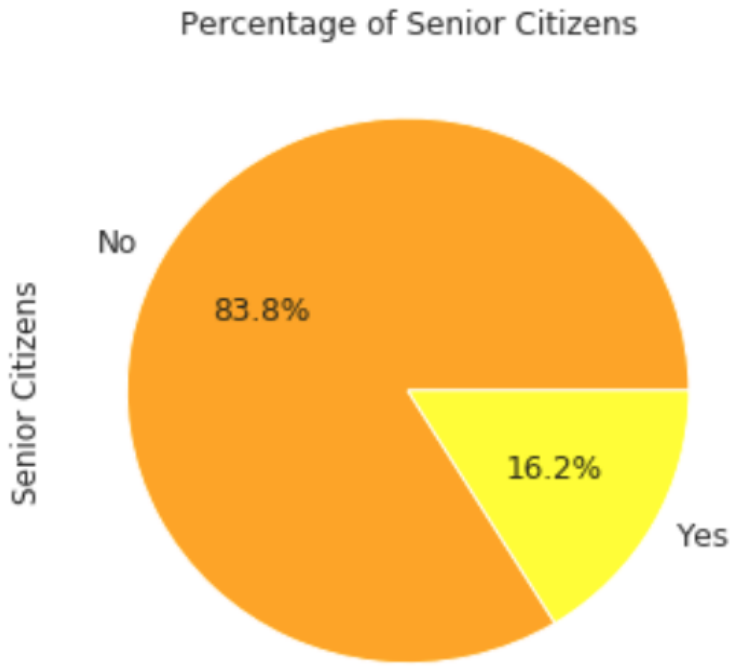
Correlation Matrix



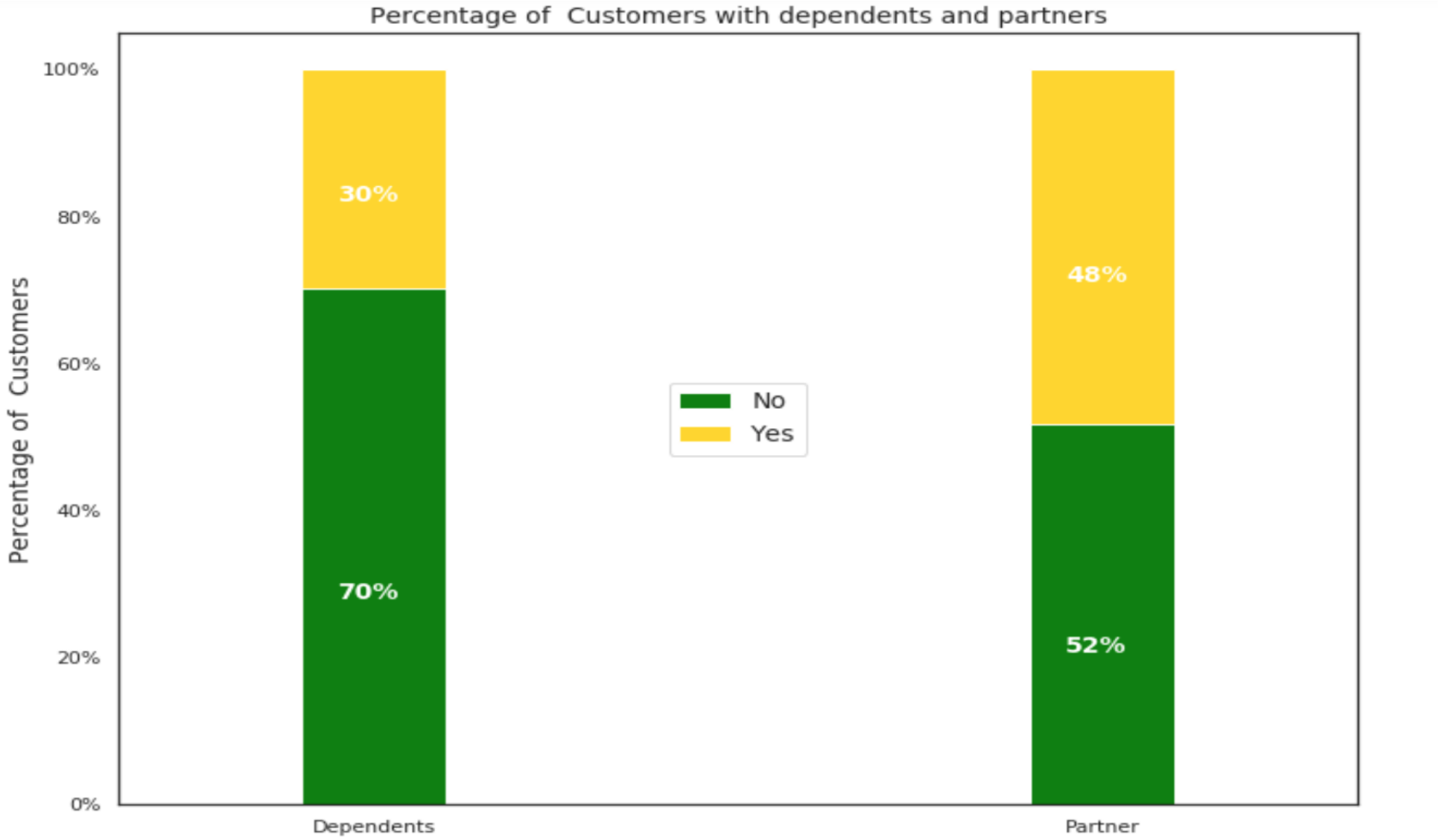
Exploratory Data Analysis - Gender



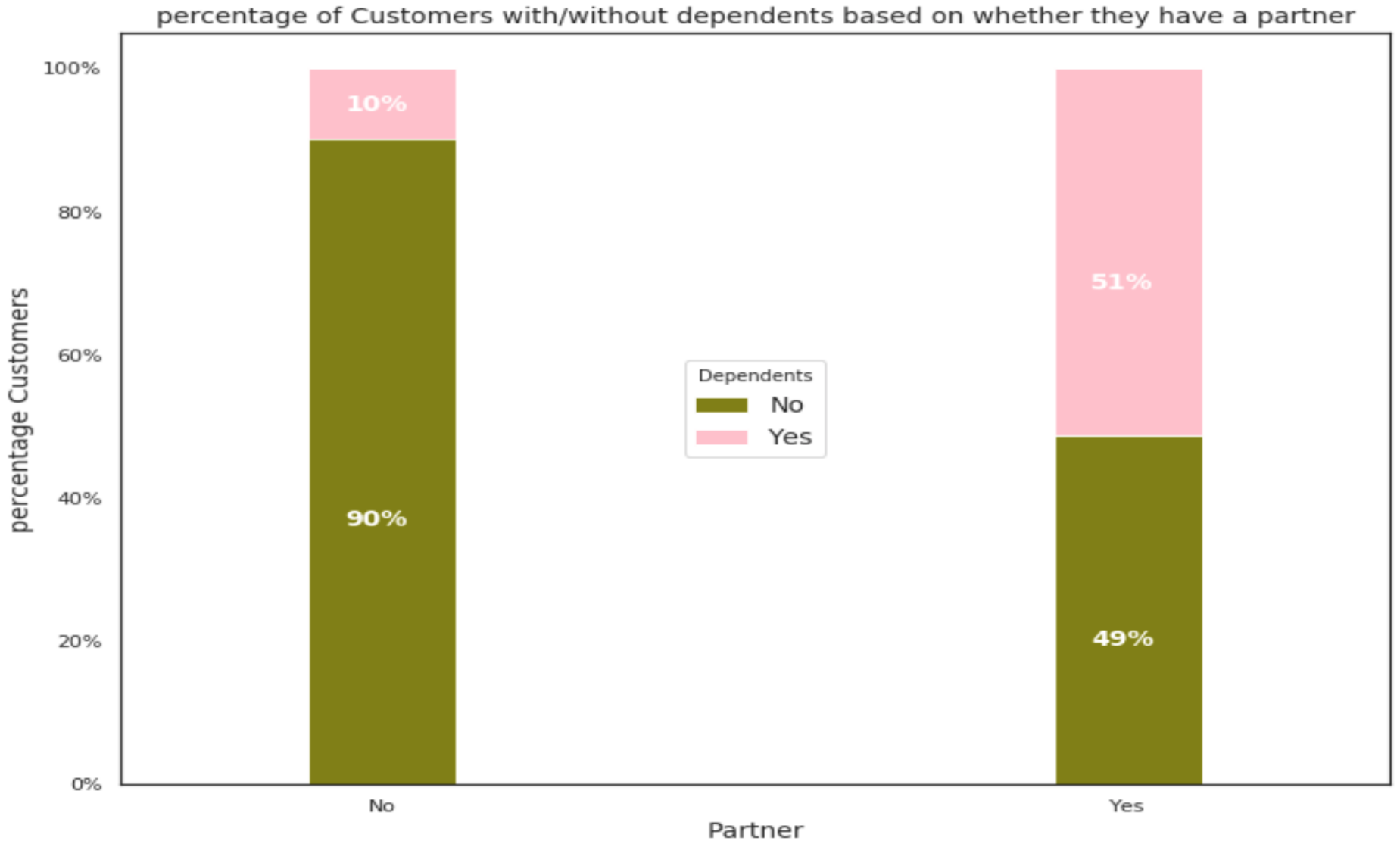
EDA – Senior Citizens



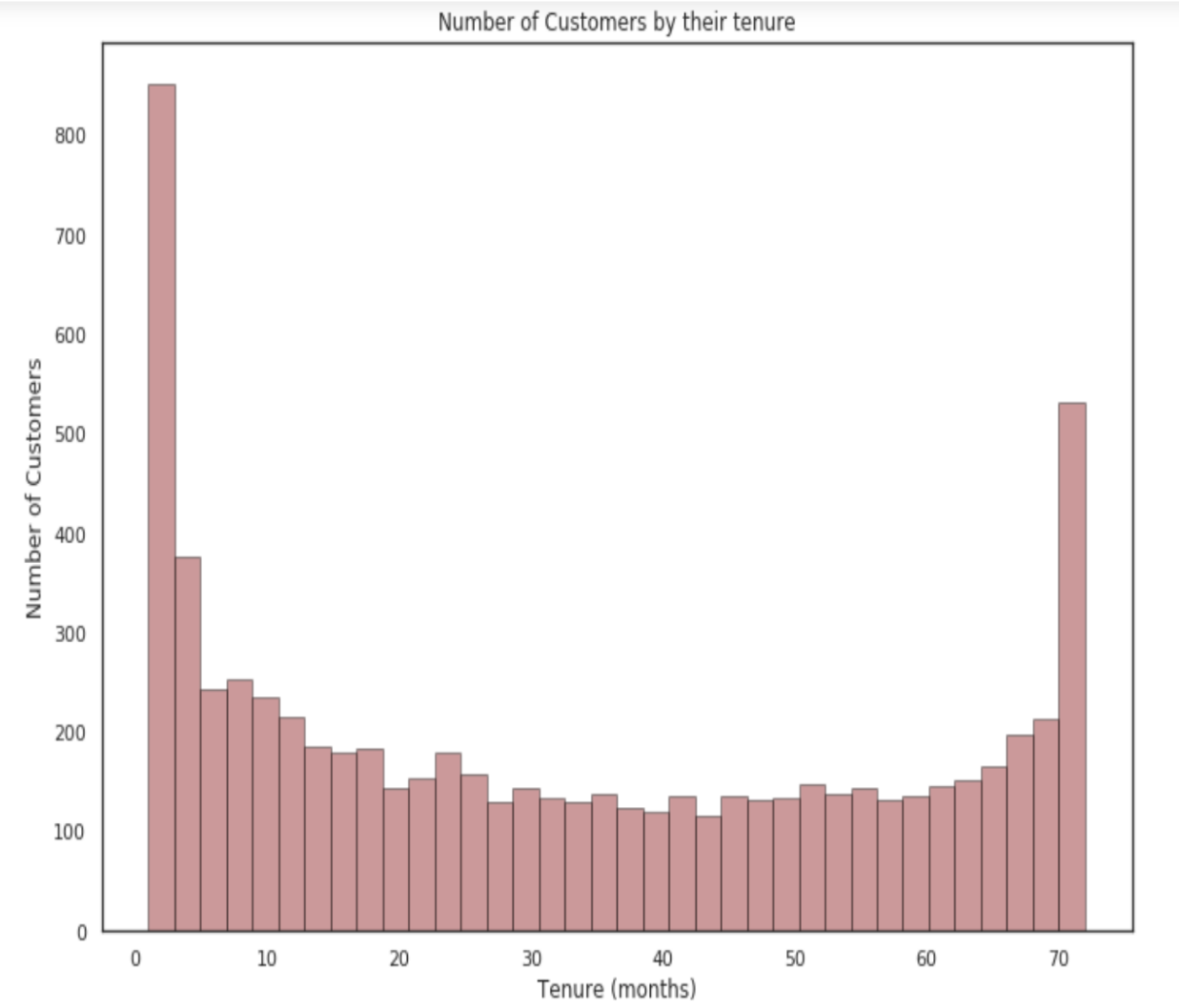
Customers with dependents vs partners

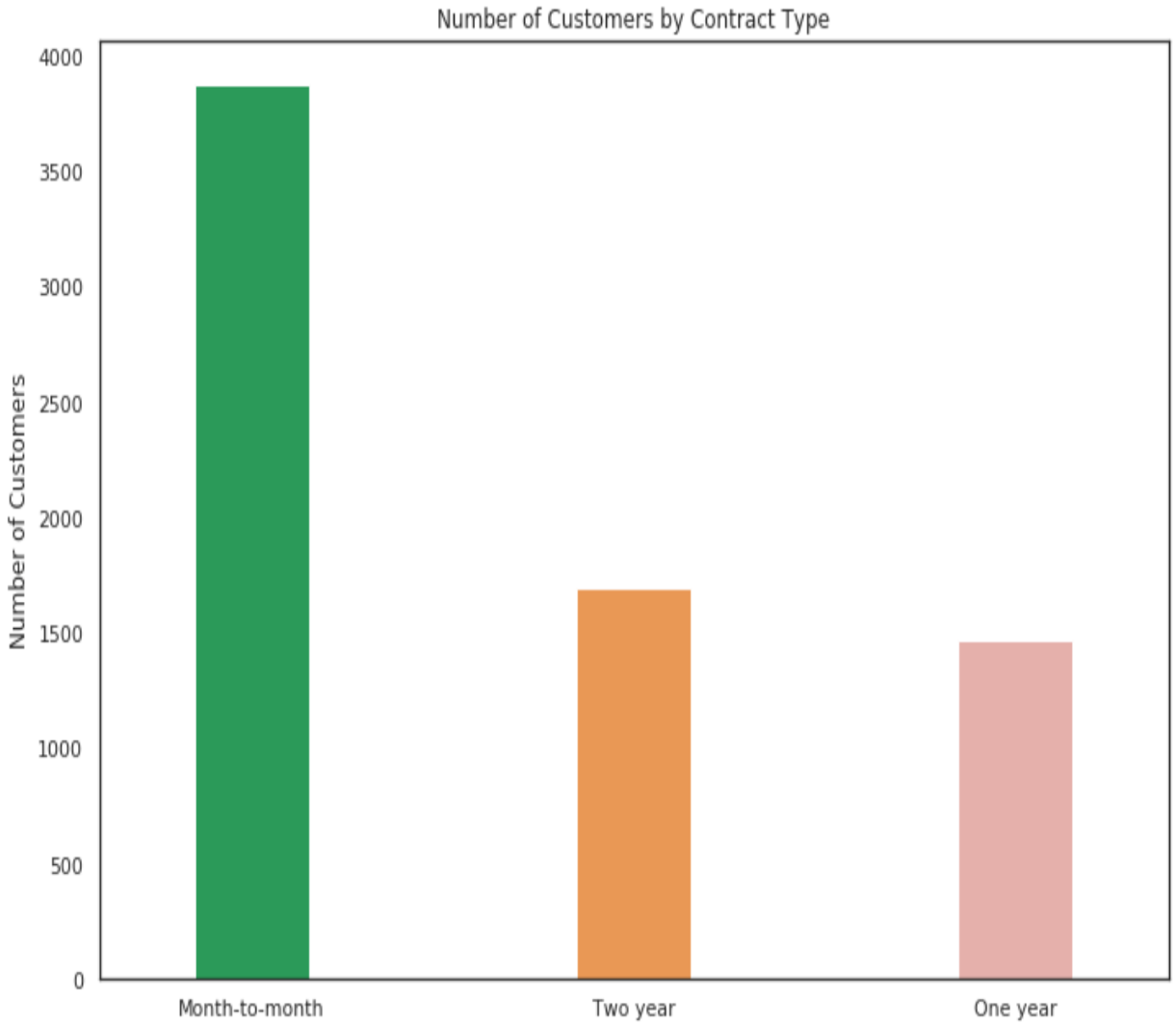


Customers with/without dependents based on whether they have a partner

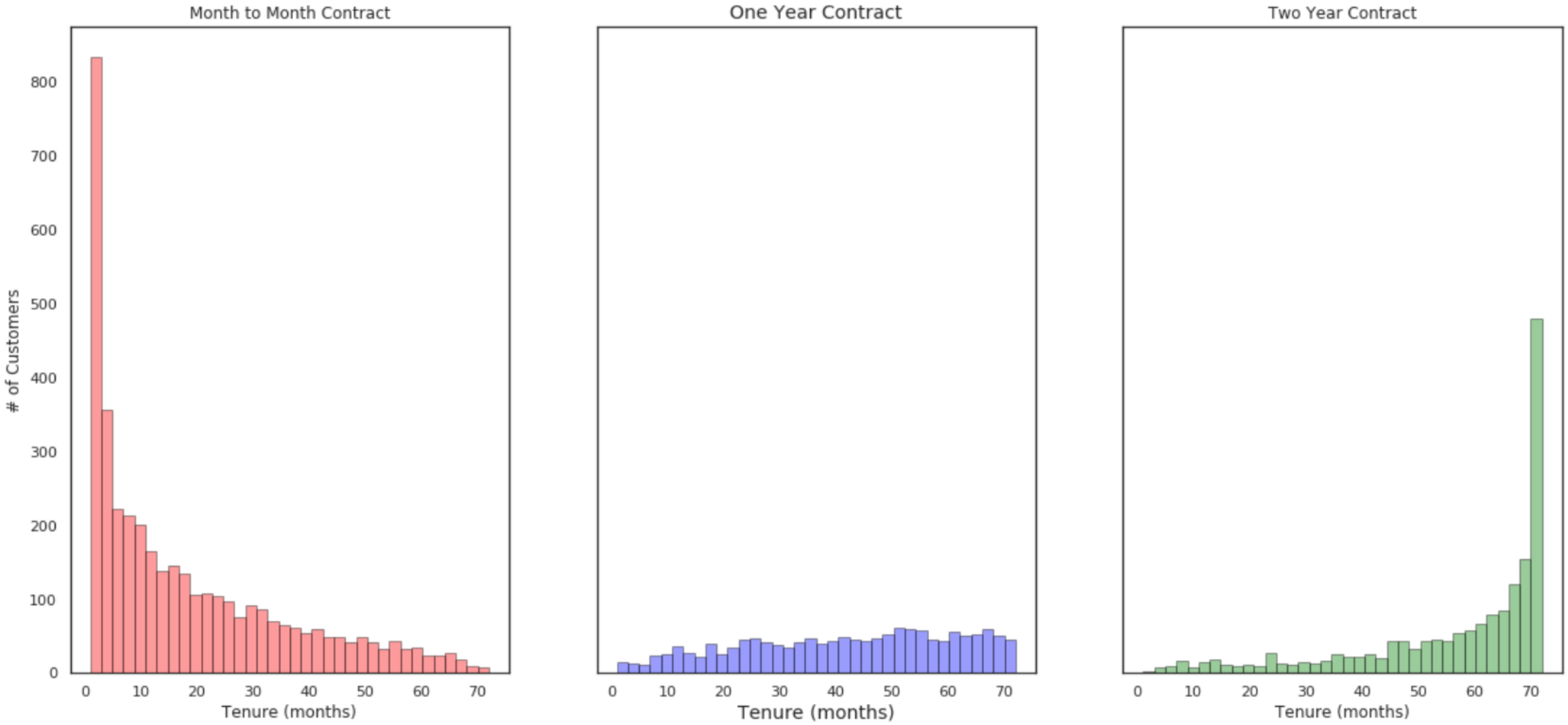


Customer by Tenure and Contract Type

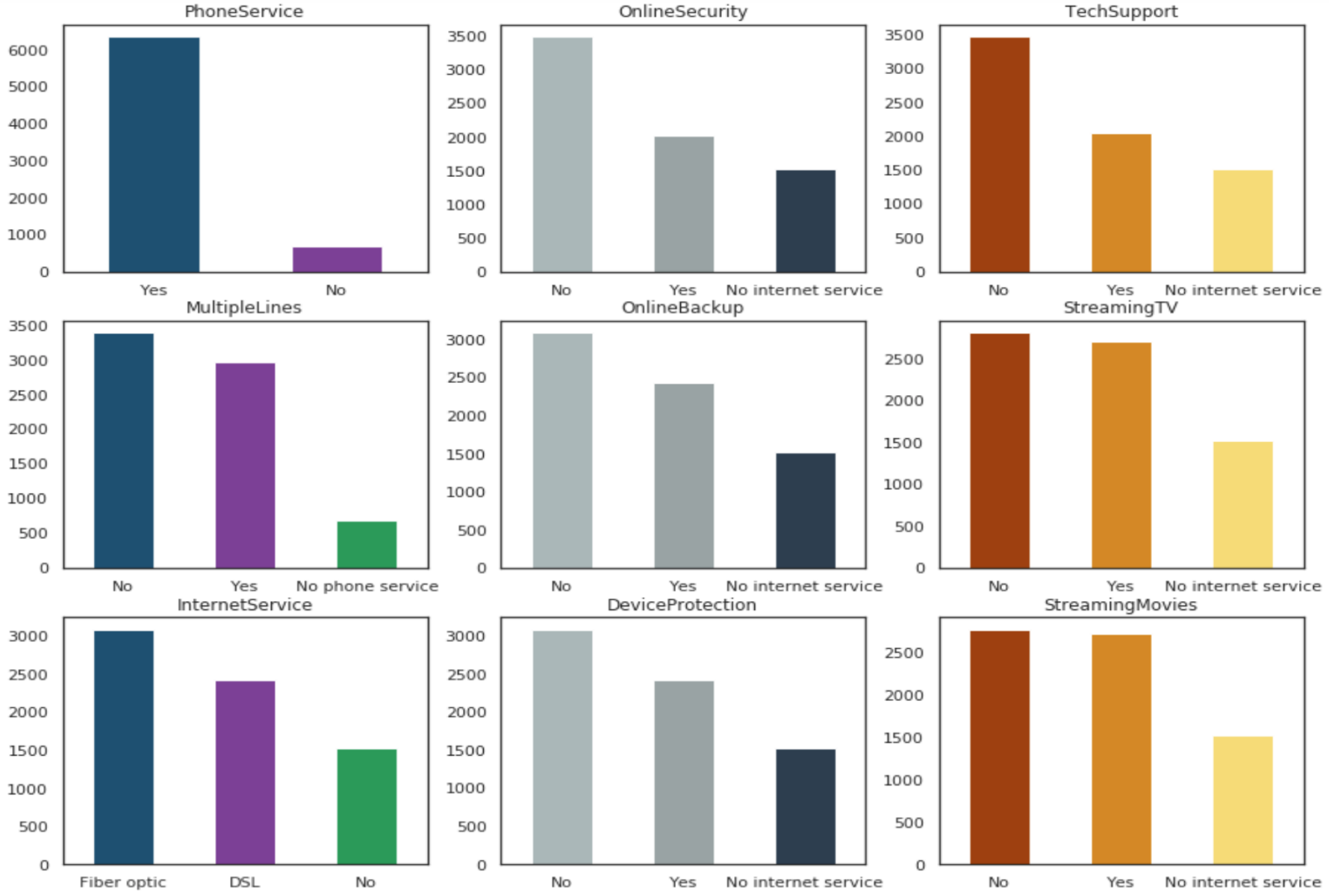




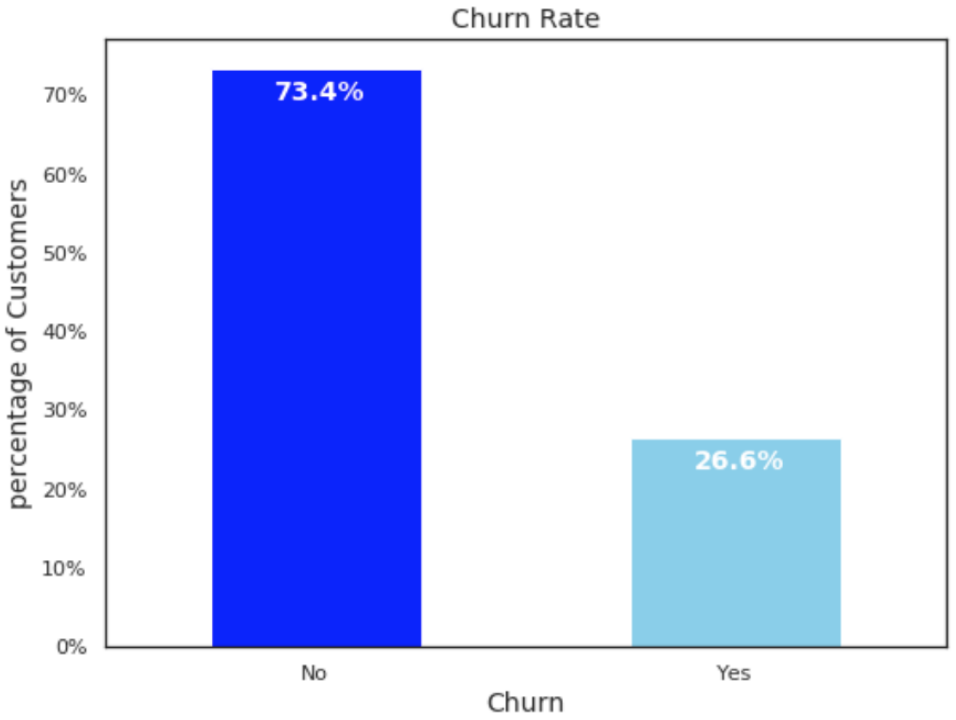
Distribution of customers for month and years

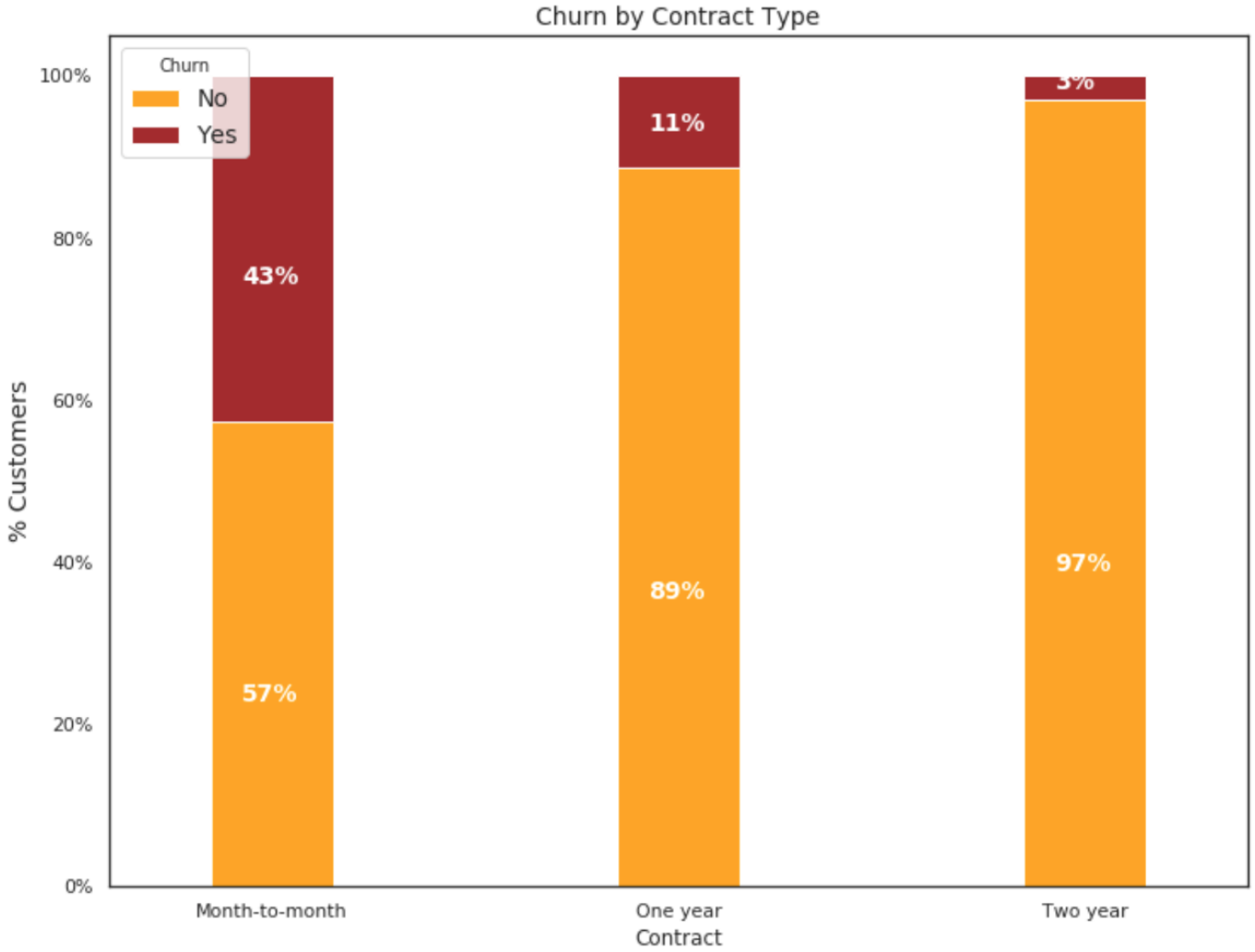


Distribution of services used by customers

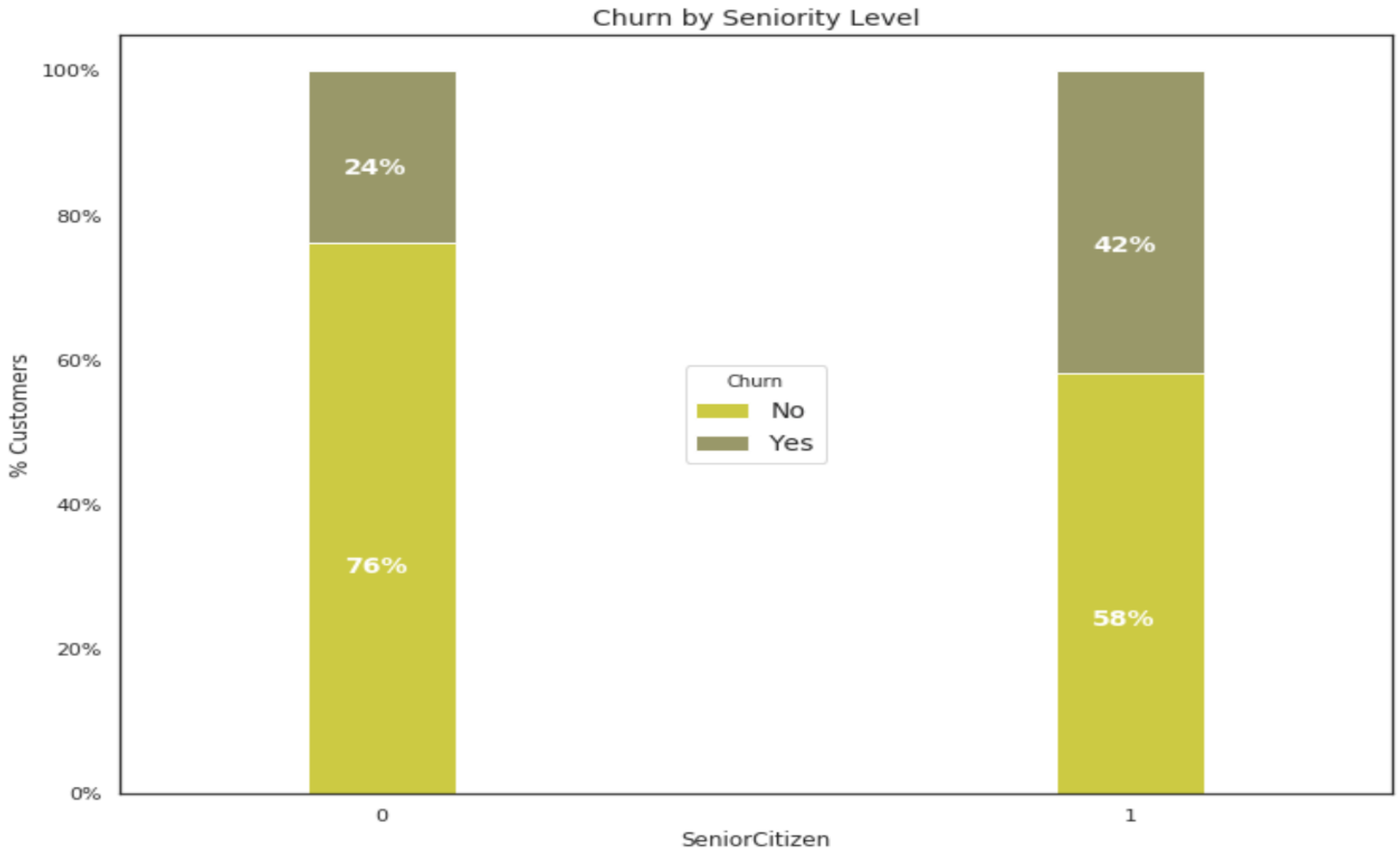


Churn Rate

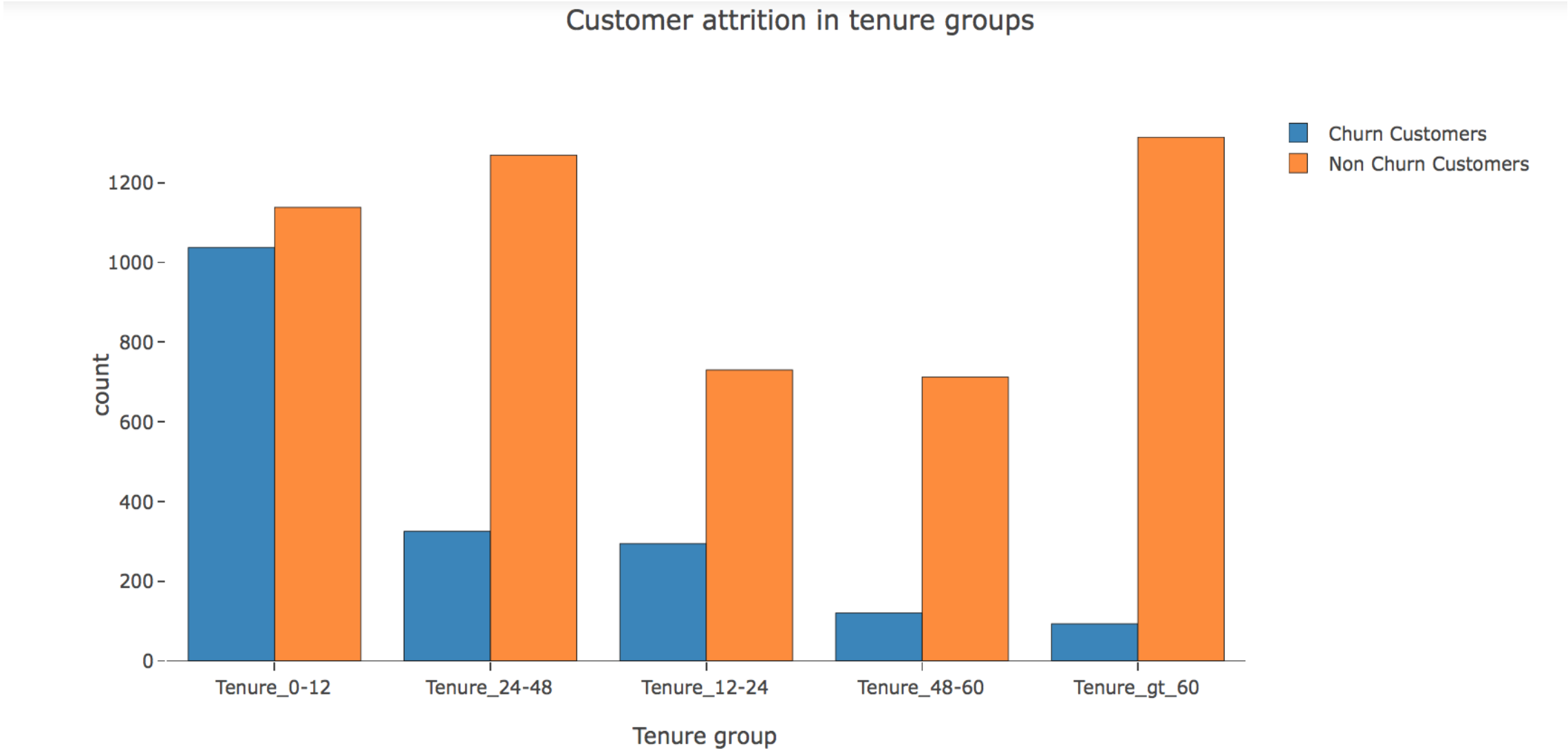

Churn Rate
73.4%
26.6%
percentage of Customers
70%
60%
50%
40%
30%
20%
10%
0%
No
Yes
Churn


Churn by Contract Type
Churn
No
Yes
% Customers
100%
80%
60%
40%
20%
0%
43%
57%
11%
89%
3%
97%
Month-to-month
One year
Two year
Contract

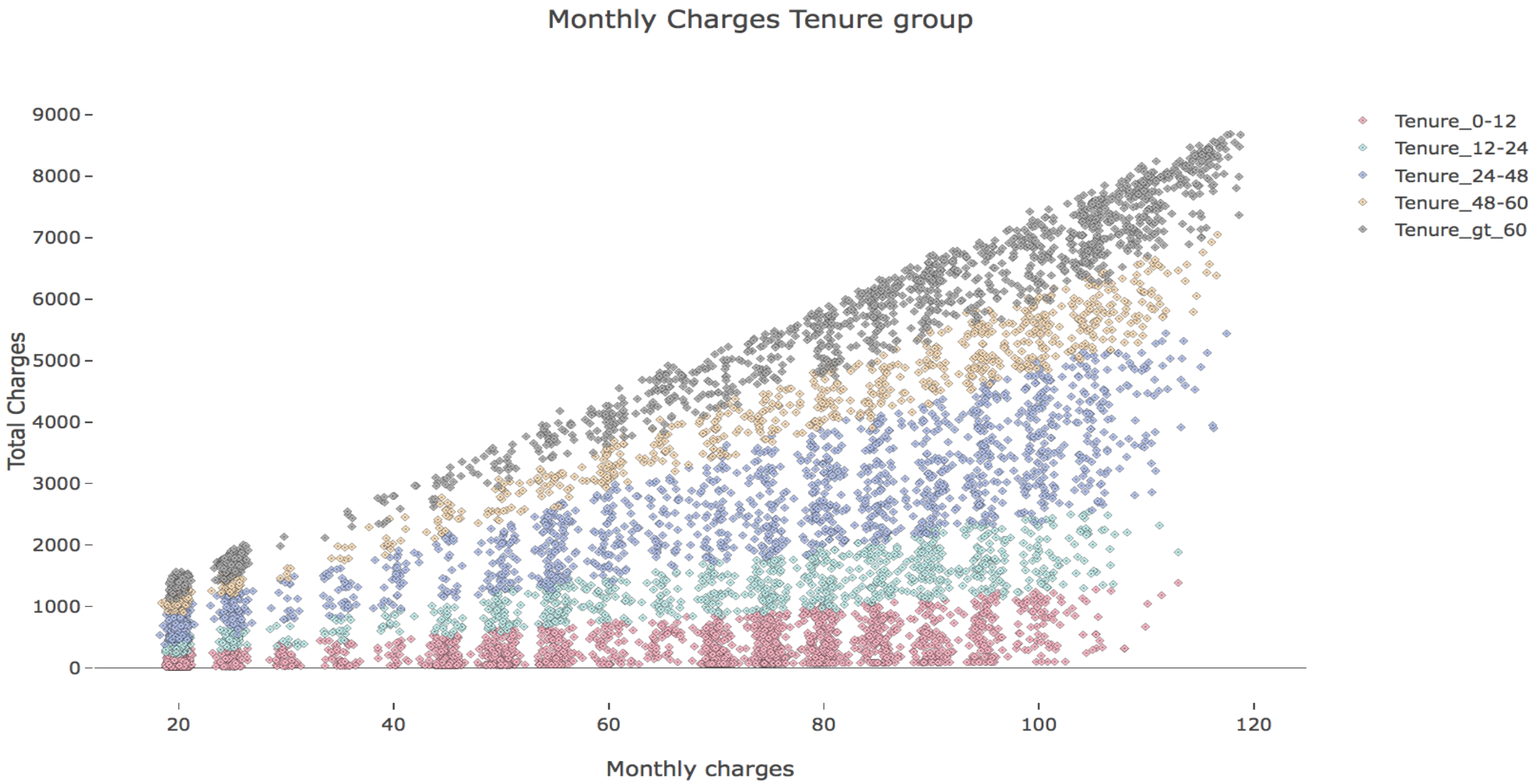
Senior Citizen and Churn Rate



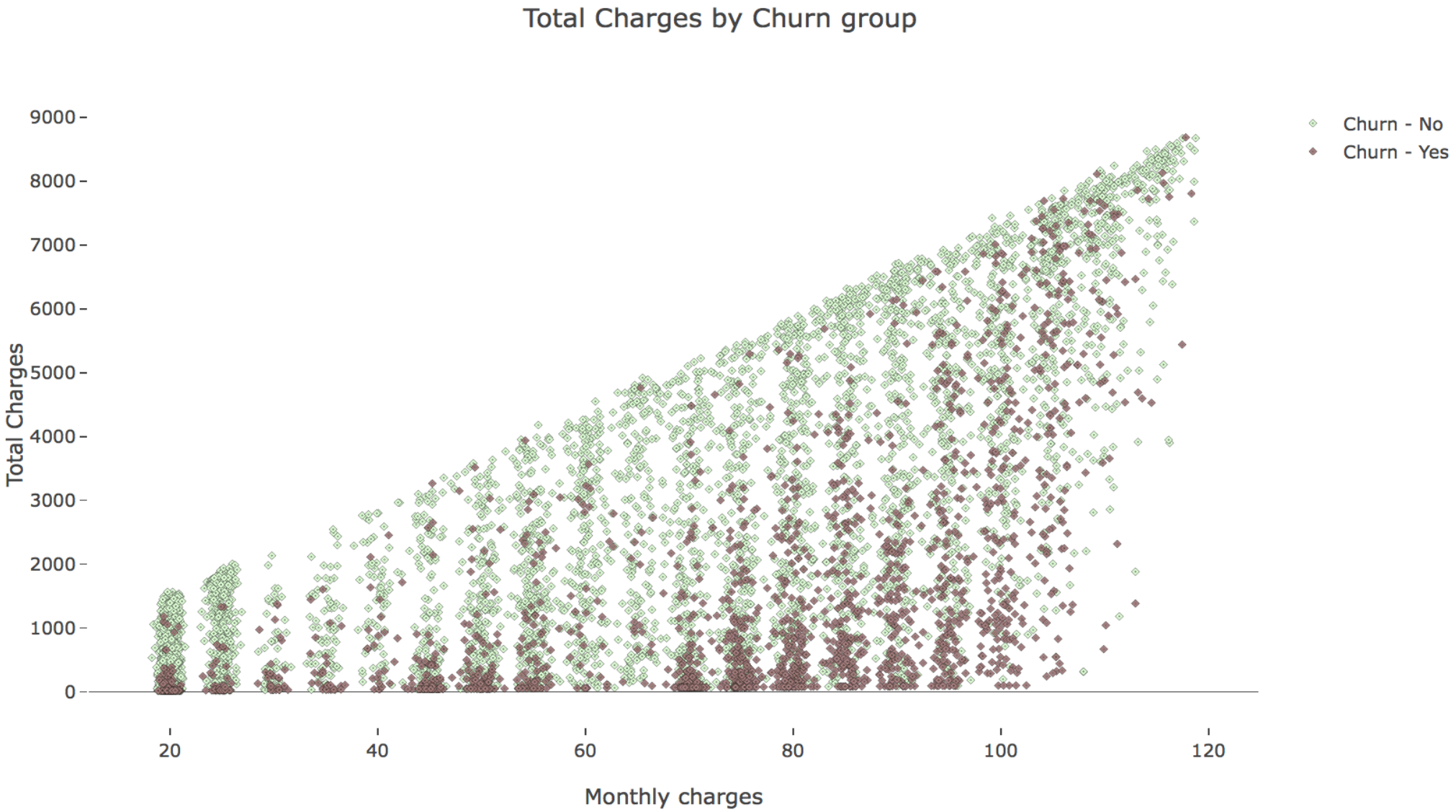
Churn Count Based on Tenure Group



Monthly Charges based on Tenure Group



Total Charges based on Churn Group



Count for Churn Categories

```
# Separate majority and minority classes
df_majority = df_dummies[df_dummies['Churn']==0]
df_minority = df_dummies[df_dummies['Churn']==1]
```

Distribution of the churn values

```
df_dummies['Churn'].value_counts()
```

```
0    5163
1    1869
Name: Churn, dtype: int64
```

Upscaling the minority points for even distribution

```
from sklearn.utils import resample

# Upsample minority class
df_minority_oversampling = resample(df_minority,
                                    replace=True,     # sample with replacement
                                    n_samples=5077,    # to match majority class
                                    random_state=587) # reproducible results
# Combine majority class with upsampled minority class
df_oversample = pd.concat([df_majority, df_minority_oversampling])
# Display new class counts
print("Now the distribution of non default and default are almost close")
df_oversample['Churn'].value_counts()
```

```
Now the distribution of non default and default are almost close

0    5163
1    5077
Name: Churn, dtype: int64
```

Using MinMaxScaler/ Train-Test Data

```
# We will use the data frame where we had created dummy variables
y = df_oversample['Churn'].values
X = df_oversample.drop(columns = ['Churn'])

# Scaling all the variables to a range of 0 to 1
from sklearn.preprocessing import MinMaxScaler
features = X.columns.values
scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(X)
X = pd.DataFrame(scaler.transform(X))
X.columns = features
```

```
# Create Train & Test Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=100)
```

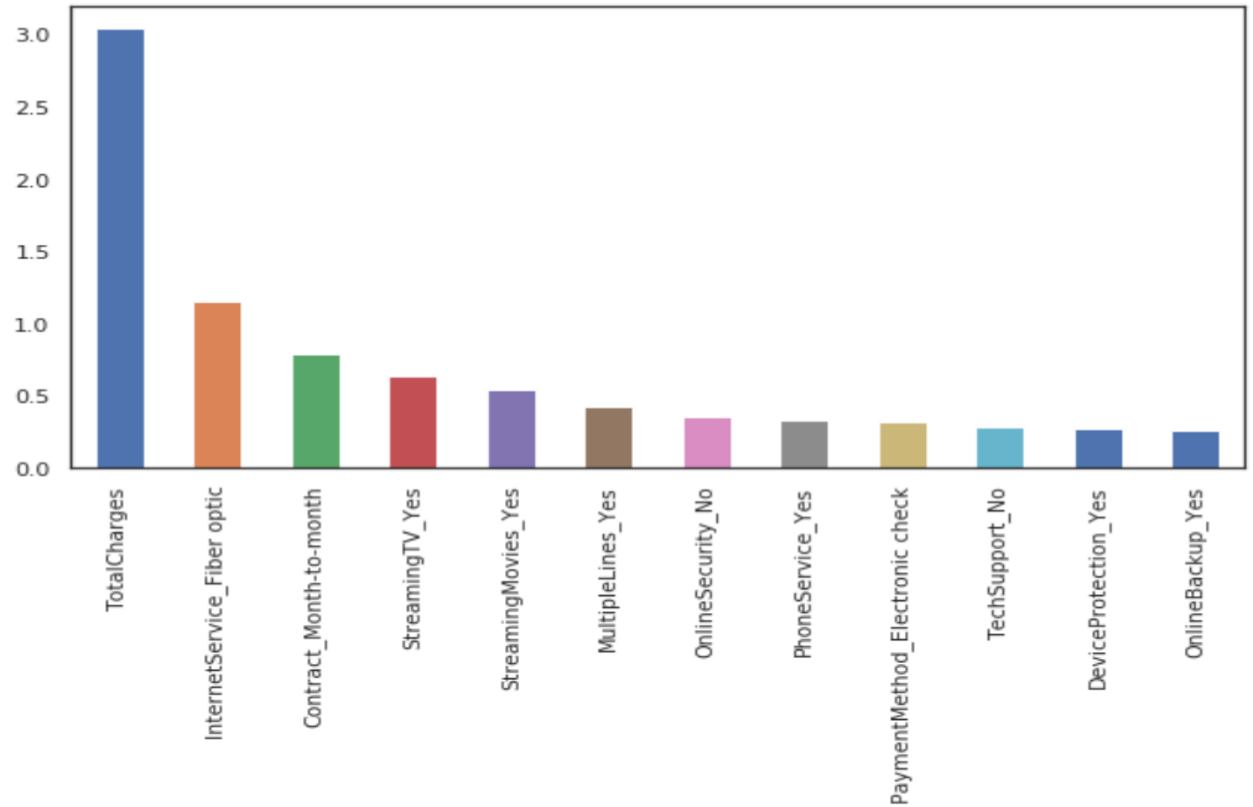
Logistic Regression Model

```
# Create dictionary for storing values of all models

prediction = dict()

#Run the logistic Regression model
#import the linear_model class from sklearn package
from sklearn import linear_model

#create an object of the class, logreg is the object of class LogisticRegression
logreg = linear_model.LogisticRegression(C=1e5)

#call object.fir on (X_train----Set of predictors, Y_train ------target variable. 80 percent is used for training)
logreg.fit(X_train, y_train)
#Model learns from training process
#After training the model -- predict the the class for rest of 20 percent of data
prediction['Logistic'] = logreg.predict(X_test)

#after predicting we check for the accuracy
#Accuracy is defined as comparison between the actual class of target variable from the test data vs predicted
print("accuaracy of model")
a= accuracy_score(y_test, prediction['Logistic'])
a=a*100
print(a)
```

Weights of the variables

```
# To get the weights of all the variables
weights = pd.Series(logreg.coef_[0],
                    index=X.columns.values)
print (weights.sort_values(ascending = False)[:12].plot(kind='bar',figsize=(10,5)))
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



K Nearest Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
prediction['KNN']= classifier.predict(X_test)
print("accuaracy of model")
a= accuracy_score(y_test, prediction['KNN'])
a=a*100
print(a)


from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,prediction['KNN'])

#print(confusion_matrix)
import scikitplot as skplt
skplt.metrics.plot_confusion_matrix(y_test, prediction['KNN'])
plt.show()

skplt.metrics.plot_confusion_matrix(y_test, prediction['KNN'],normalize=True)
plt.show()


average_precision = average_precision_score(y_test,prediction['KNN'])

print('Average precision-recall score: {0:0.2f}'.format(
      average_precision))
```

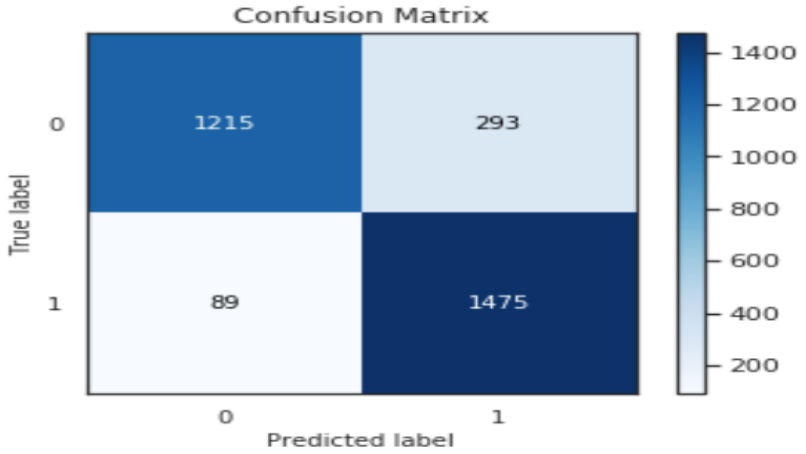
Decision Tree Classifier

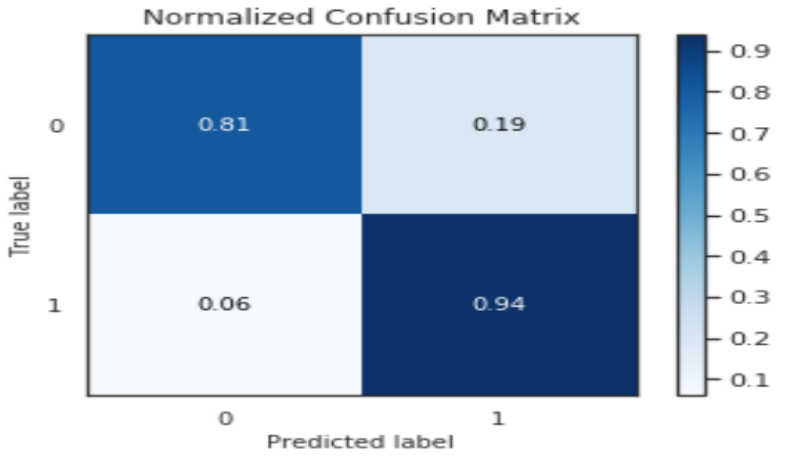
```
#Calling the Decision TRee Classifier class
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,
                                  max_depth=3, min_samples_leaf=5)


clf_gini.fit(X_train, y_train)

clf_gini.fit(X_train,y_train)
prediction['DecisionTree'] = clf_gini.predict(X_test)
print("accuracy of the model")
a=accuracy_score(y_test, prediction['DecisionTree'])
a=a*100
print(a)



from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,prediction['DecisionTree'])

#print(confusion_matrix)


import scikitplot as skplt
skplt.metrics.plot_confusion_matrix(y_test, prediction['DecisionTree'])
plt.show()


skplt.metrics.plot_confusion_matrix(y_test, prediction['DecisionTree'],normalize=True)
plt.show()
```

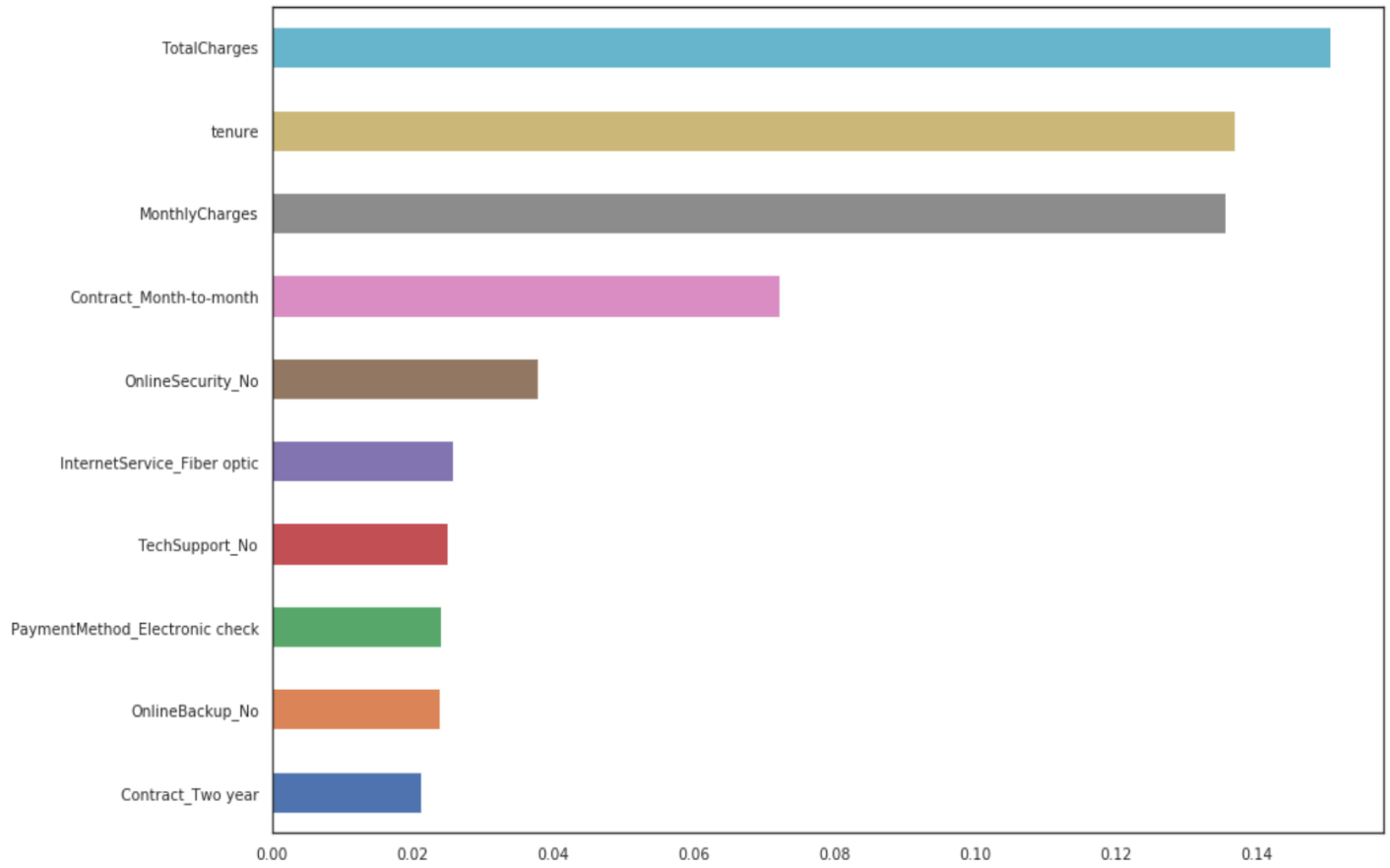
Random Forest Classifier

```
#calling the RandomForest Classifier


clf = RandomForestClassifier(n_jobs=1000,
                             random_state=9,
                             #criterion=RFC_METRIC,
                             n_estimators=11,
                             verbose=False)
clf.fit(X_train,y_train)
prediction['RandomForest'] = clf.predict(X_test)
a= accuracy_score(prediction['RandomForest'], y_test)
a= a*100
print(a)


from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,prediction['RandomForest'])

#print(confusion_matrix)
```

Confusion Matrix /Features Random Forest

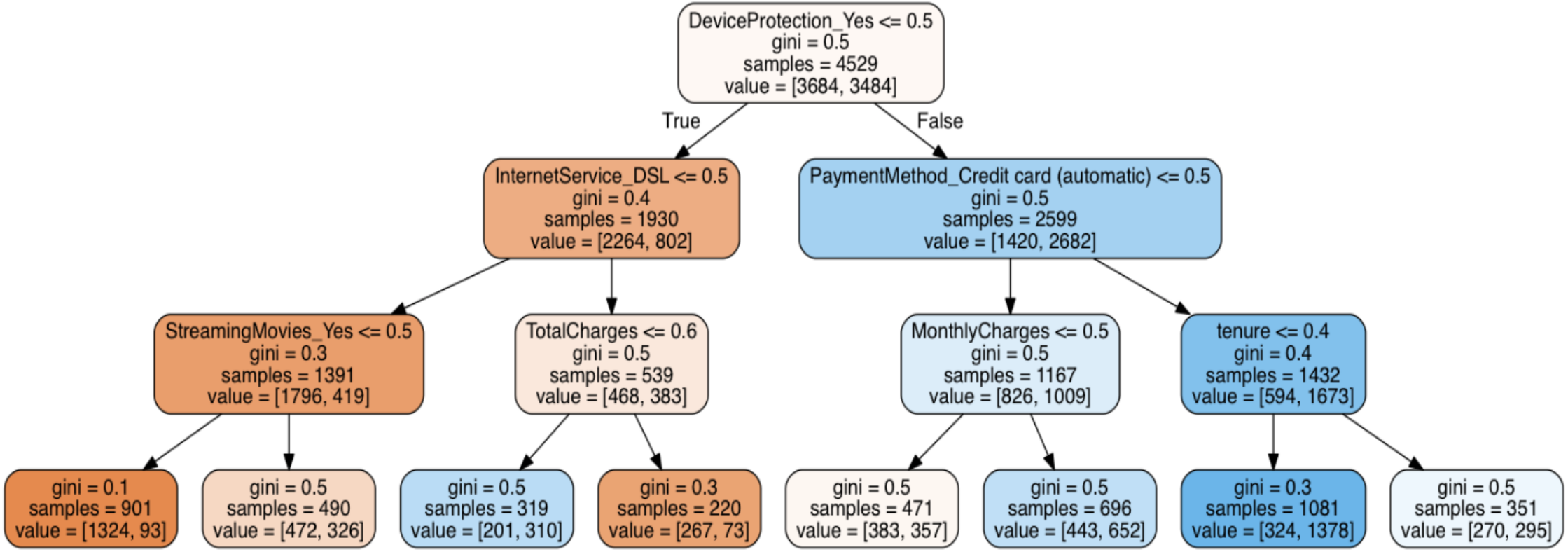







```
Average precision-recall score: 0.82
```

Tree Picture of Random Forest with 3 Levels

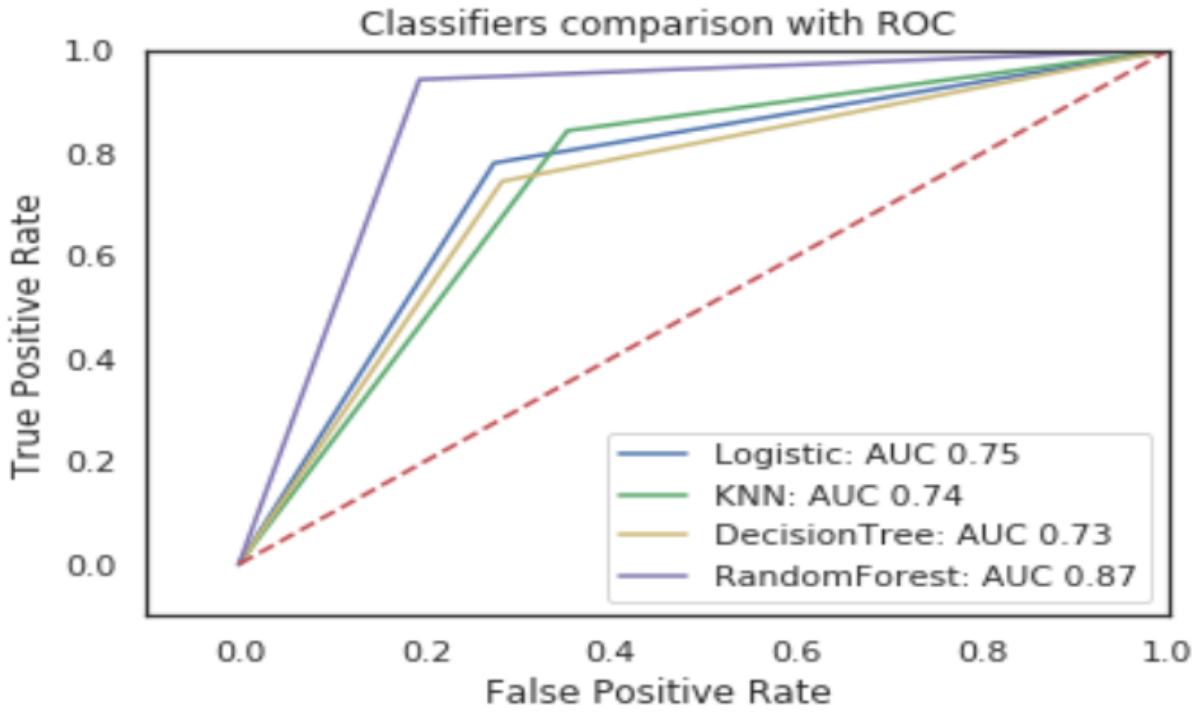


Accuracy

```
from sklearn.metrics import accuracy_score

cmp = 0
for model, predicted in prediction.items():
    accuracy = accuracy_score(y_test, predicted)
    accuracy
    print(model, accuracy*100)
    cmp += 1
```

```
Logistic 75.35807291666666
KNN 74.67447916666666
DecisionTree 73.11197916666666
RandomForest 87.56510416666666
```

Area Under the Curve Metrics(ROC)



Learning Process

- Exploratory Data Analysis
- Visualization of various features, using plotly
- Data Manipulation
- Scaling the Data
- Feature importance
- Running various models, comparing their accuracy
- Interpreting the results of confusion matrix and Area under the curve

Conclusion

- Random forest classifier is the best model, for customer attrition prediction, with a accuracy of 88 percent, with precision recall score of .82 and area under the curve of 0.87.
- The best set of predictors are Total charges, tenure, monthly charges, contract month to month. Cost is the deciding factor if customers with churn the service.
- Monthly charges are lowest for the first one year, keeps increasing based on tenure.
- Maximum customer attrition happens in the first 12 months.
- Senior citizens are more likely to churn the service.

Links (Github)

- https://varsha1288.github.io/Customer-Churn-Prediction/