Task 2: Logistic Regression for Predictive Modeling

Sina Bozorgmehr

Western Governors University

Task 2: Logistic Regression for Predictive Modeling

Introduction

The fierce competition in the telecommunications industry makes it difficult to have loyal customers stay in a particular company for a long time. Customers are looking for better services with lower costs and if they are not stuck in a contract can easily switch to another company. This issue is one of the main reasons that the churn rate is normally high in the industry. As data analysts, we are trying to review the customers' data and create a model based on the trend and different metrics to predict which customer are about to churn and then take the action required to prevent this.

Research Question

Based on the dataset, 25% of the customers left the company in the last month. Amongst them, some had been a customer only for a month and some been loyal for more than five years. It would be marvelous if we could predict which customers are about to churn based on the data trends and metrics that we have.

In this data analysis, a logistic regression model is created to predict the probability of churn for a specific customer. Multiple features will be used to generate an initial model and with stepwise feature selection, a reduced and simpler model will be created with fewer features.

Method Justification

The Logistic Regression (LR) is used in this analysis to predict if a customer will churn. LR makes several assumptions. Many of the main assumptions of linear regression and general linear models based on ordinary least squares algorithms, such as linearity, normality, homoscedasticity, and measurement degree, are not made in logistic regression. Other assumptions, however, remain true (Assumptions of Logistic Regression, 2020).

To begin, binary logistic regression necessitates a binary dependent variable, whereas ordinal logistic regression necessitates an ordinal dependent variable. Second, logistic regression presupposes the independence of the observations. To put it another way, the observations should not be based on repeated measurements or matched information. Third, there must be little to no multicollinearity among the independent variables for logistic regression to work. This implies that the independent variables should have a low correlation with one another. Fourth, the independent variables are linear and that log odds are positive. While the dependent and independent variables do not have to be linearly related in this analysis, the independent variables must be linearly related to the log odds. And the last assumption is that the logistic regression generally necessitates a large number of samples. For each independent variable in your model, a rule of thumb is that you need at least 10 instances with the least common outcome (Assumptions of Logistic Regression, 2020).

In this analysis, Python and its famous packages is used for easier and faster model creation. Pandas is a python package that makes the dataset exploring very easy and fast. Also, Sci-kit learn package is the main library to create the predictive model and parameters analyses. The whole script is written and run in Jupyter Notebook for better annotation and visualization.

Since we are predicting a binary dependent variable with multiple continuous and categorical independent variables, LR is a good candidate to create this model.

Data Preparation

The dataset has 10,000 observations with 50 variables that is checked for missing values and the type of each variable. Churn is the dependent binary variable, and the rest of the variables act as independent variables. First, we eliminate the features related to the customers' demographics since they have no value in our analysis (Figure 1). Next, two separate list are

created for continuous and categorical variables for the purpose of easier visualization (Figure 2). Figures 3 and 5 show the univariate visualization of numeric and categorical variables, respectively. A correlation matrix plot is created to show the possible multicollinearity between the features. As figure 4 demonstrates, Tenure and Bandwidth per year have a super strong covariation which leads to removing the Bandwidth per year from our feature set. Figures 6 and 7 demonstrate the bivariate visualization of independent variables versus the dependent variable. All eight questions in the survey demonstrate a very similar distribution in the responses and we can see this in the correlation matrix (Figure 4) as well. Therefore, all the eight questions were converted into a single variable with the code seen in figure 8. In order to make categorical variables with string values ready for LR, Pandas *get_dummies* is used to convert this variables' values into numeric levels (figure 9). One of the levels is dropped out to avoid multicollinearity (Kumar, 2020). After all this preparation the final cleaned dataset was save into a csv file. This dataset contains 10,000 observations and 36 columns.

Figure 1

Removing the demographics from the dataset

```
#Eliminating variables that have no importance in our analysis

column = ['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County',
          'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone','Job'
         ]
df.drop(columns=column, inplace=True)
df.head()
```

Figure 2

Creating two separate lists for continuous and categorical variables

```
#Creating two lists, containing continuous and categorical variables

cont_var = ['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure',
            'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6',
            'Item7', 'Item8'
           ]
cat_var = [i for i in df.columns if i not in cont_var]

print('Continuous variables are:\n\n {} \n\n and Categorical variables are:\n\n {}'.format(cont_var, cat_var))
```

Figure 3

Univariate visualization of continuous variables

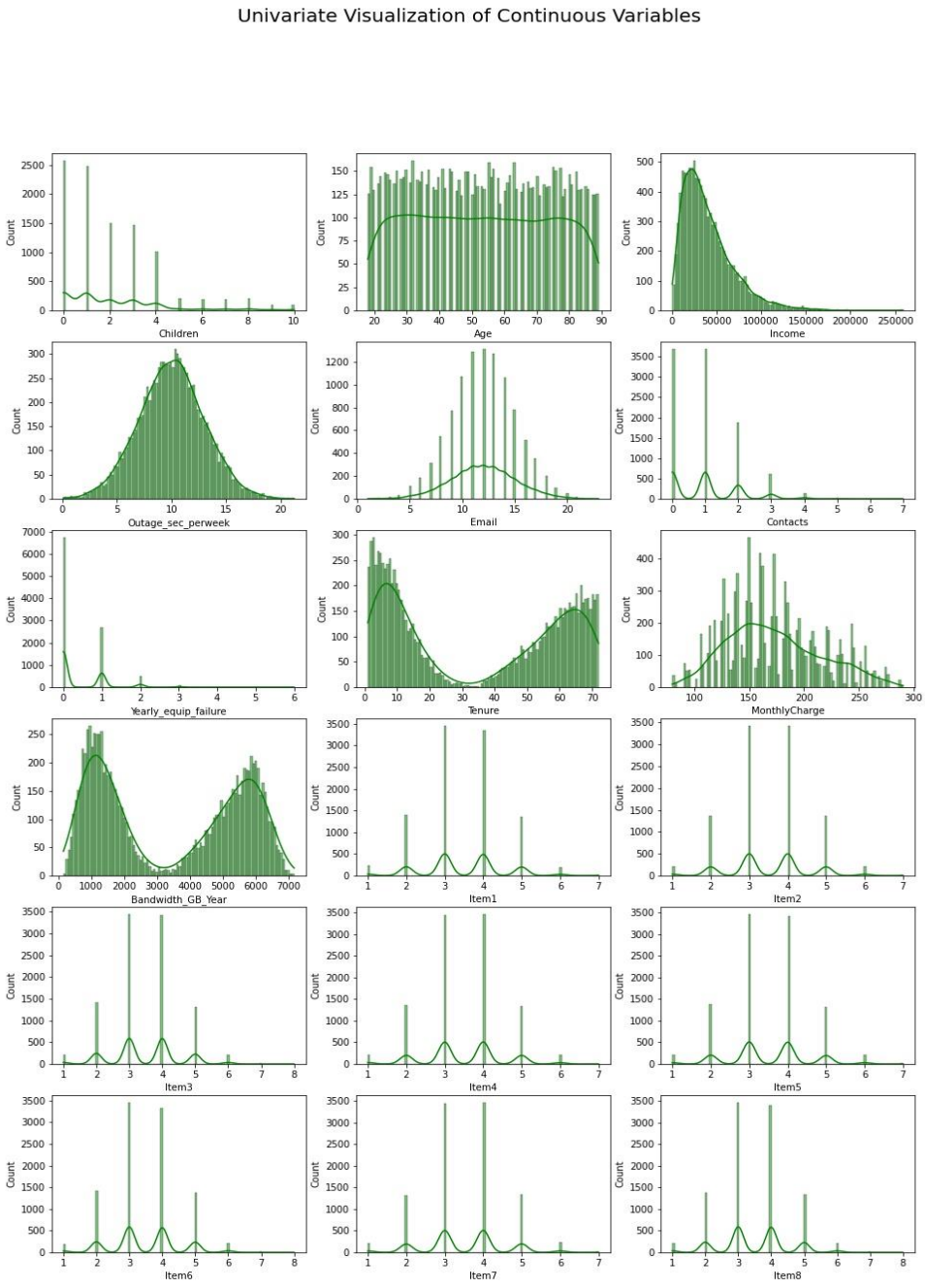


Figure 4

Correlation matrix heatmap for the continuous variables

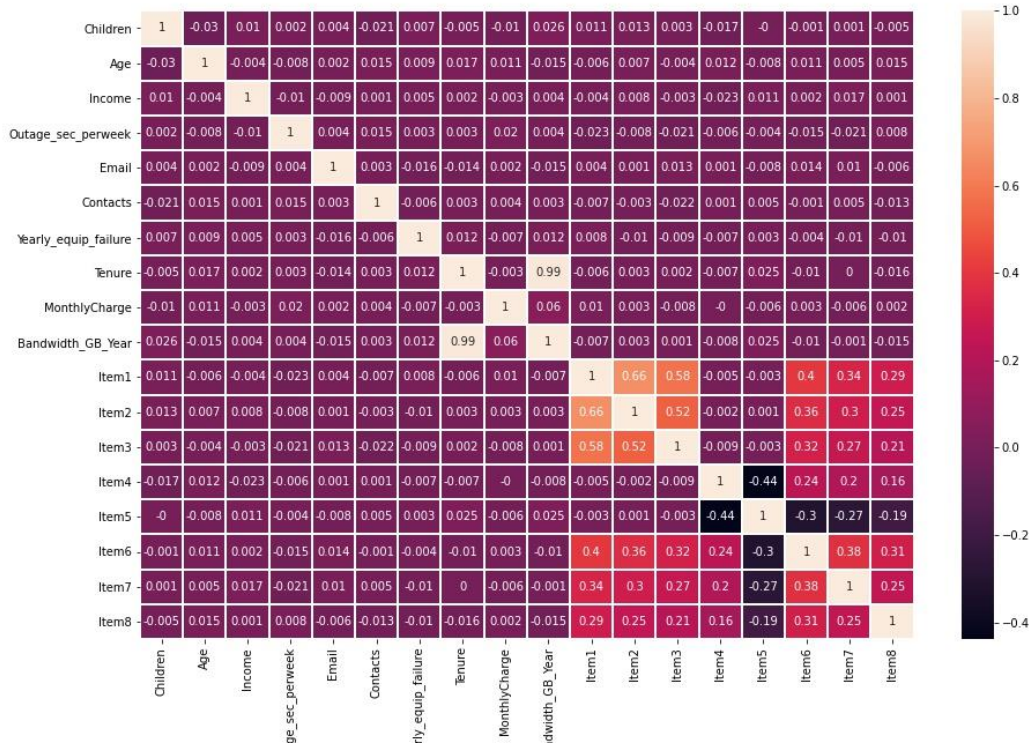


Figure 5

Univariate visualization of categorical variables

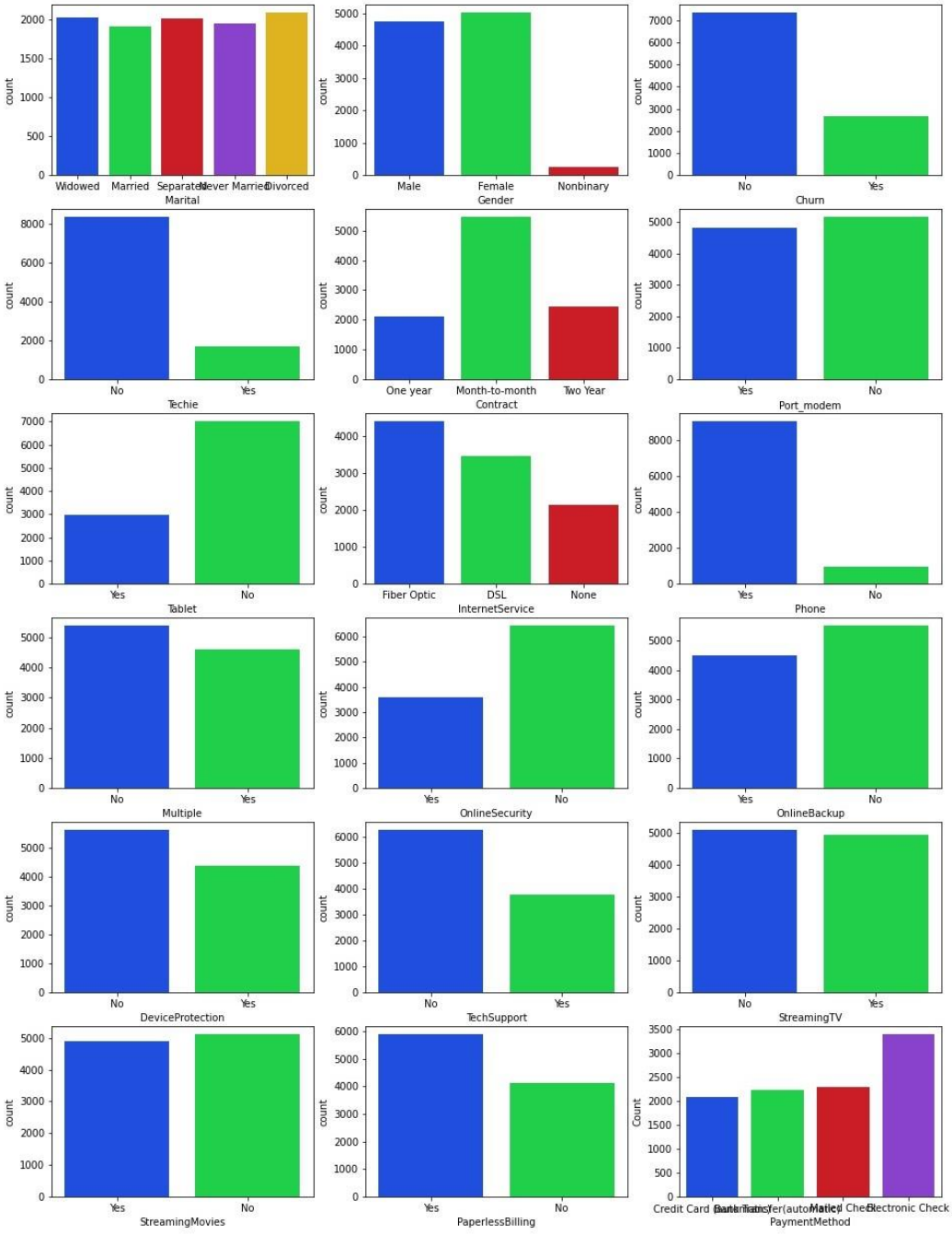


Figure 6

Bivariate visualization of continuous variables vs. churn

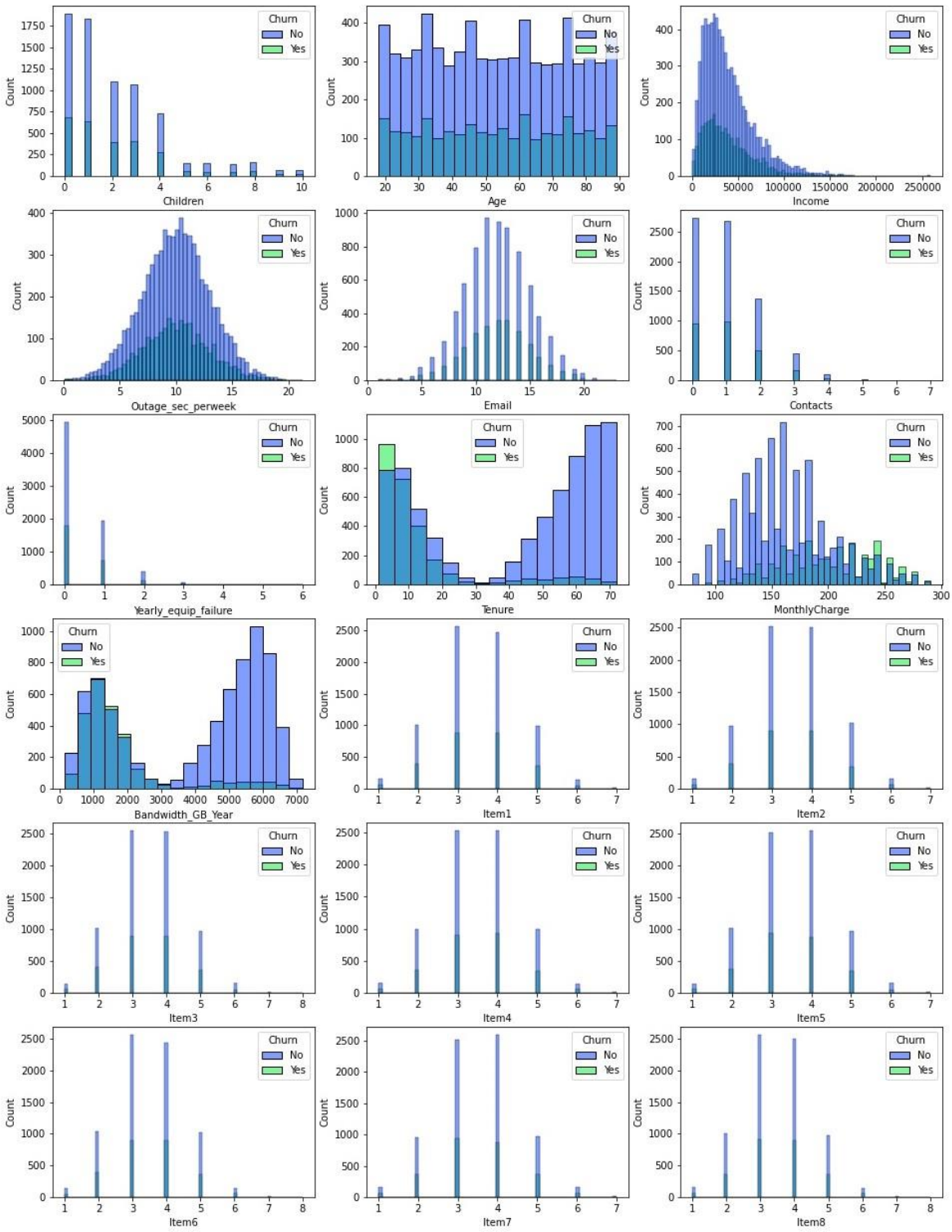


Figure 7

Bivariate visualization of categorical variables vs. churn

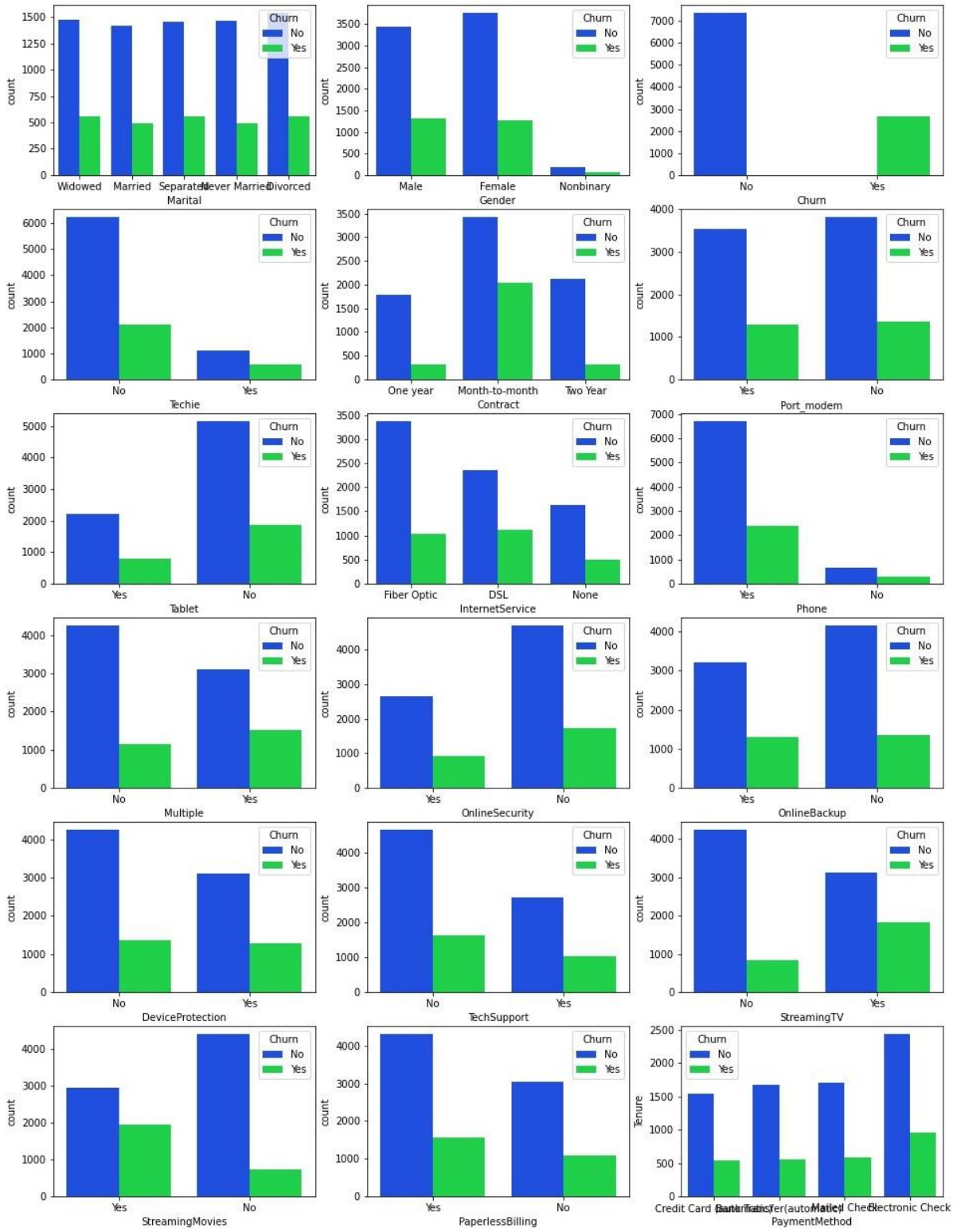


Figure 8

Creating a single variable survey

```
# Creating a new feature by averaging all the 8 questions in survey:

surv = ['Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8']
df['Survey'] = df[surv].astype(float).mean(axis=1).round()
df.drop(columns=surv, inplace=True)

#Graphing the new feature

sns.histplot(data=df, x='Survey', kde=True, color='blue', bins='sqrt')
plt.xticks(range(1,8,1))
plt.show()
```

Figure 9

Creating dummy variables

```
# creating dummy variables for all categorical features

cleaned_df = pd.get_dummies(cleaned_df, drop_first=True)
cleaned_df.head()
```

Model Comparison and Analysis

To be able to evaluate the predictive model on out of samples data, 30% of the observation is left out of training using the Sci-kit Learn method, train_test_split (Figure 10). 7,000 observations are dedicated for training and the remaining 3,000 for evaluation. The feature set includes numeric values ranging widely amongst predictors that requires the scaling before fitting the LR model. Since the categorical variables' values are already either 1 or 0, MinMaxSclaer is applied to normalize all the values (Figure 11).

Figure 10

Splitting dataset into training and test sets

```
#Splitting data into training and test sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=110)

print('Training Set Observations: %.f' % X_train.shape[0])
print('Test Set Observations: %.f' % X_test.shape[0])
```

Figure 11

Normalization of dataset

```
# Normalizing the features

from sklearn.preprocessing import MinMaxScaler

index_train = X_train.index
index_test = X_test.index


n_scaler = MinMaxScaler()
X_train = pd.DataFrame(n_scaler.fit_transform(X_train), index=index_train, columns=X.columns)
X_test = pd.DataFrame(n_scaler.transform(X_test), index=index_test, columns=X.columns)
```

Sci-kit Learn LogistcRegression is used to create the initial model using the training set (Figure 12). The accuracy for the initial model is 0.907 that is a fantastic score. To evaluate the multicollinearity amongst the predictors, Variance Inflation Factor (VIF) is calculated (Figure 13) for all the predictors that shows some of the independent variables have a very high score (Figure 14). Stepwise Feature Elimination is used to reduce the feature set. The forward stepwise regression method employs a series of steps that allow features to join or exit the regression model one by one. This method often uses a p-value threshold to determine entry and exit of the predictors to reduces the number of features (Kuhn & Johnson, 2019). In this analysis, the p-value threshold is set to 0.05 (Figure 15). After using this function, the independent variables are reduced to 8 predictors (Figure 16). Now, the VIF for all the predictors are below 10 which

shows that we were successful in eliminating the multicollinearity (Figure 17). Using the reduced features set, the reduced model is fitted with the training set and the accuracy, recall and precisions scores are printed (Figure 18). All the scores are almost the same as in the initial model, however, the reduced model has fewer predictors with lower VIF. The model is now evaluated with the test set and as the result proves, the reduced model has the same reliability on the out of samples datapoints (Figure 19). The confusion matrix supports the test scores (Figure 20).

Figure 12

Creating the initial model

```
#Creating the initial LR model:

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score

init_mod = LogisticRegression().fit(X_train, y_train)


print('The accuracy score is: %.3f' % accuracy_score(y_train, init_mod.predict(X_train)))
print('The recall score is: %.3f' %recall_score(y_train, init_mod.predict(X_train)))
print('The precision score is: %.3f' %precision_score(y_train, init_mod.predict(X_train)))
```

```
The accuracy score is: 0.907
The recall score is: 0.805
The precision score is: 0.836
```

Figure 13

Calculating the VIF for initial model predictors

```
#Claculating Variance Inflation Factor (VIF):

from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif["features"] = X_train.columns

vif.round(1)
```

Figure 14

Initial model predictors' VIF

	features	VIF Factor
0	Children	1.9
1	Age	3.7
2	Income	2.9
3	Outage_sec_perweek	10.2
4	Email	11.4
5	Contacts	2
6	Yearly_equip_failure	1.4
7	Tenure	2.6
8	MonthlyCharge	135.6
9	Survey	6.6
10	Marital_Married	1.9
11	Marital_Never Married	1.9
12	Marital_Separated	1.9
13	Marital_Widowed	1.9
14	Gender_Male	1.9
15	Gender_Nonbinary	1
16	Techie_Yes	1.2
17	Contract_One year	1.4
18	Contract_Two Year	1.5
19	Port_modem_Yes	1.9
20	Tablet_Yes	1.4
21	InternetService_Fiber Optic	4.5
22	InternetService_None	2
23	Phone_Yes	9
24	Multiple_Yes	8.3
25	OnlineSecurity_Yes	1.6
26	OnlineBackup_Yes	4.8
27	DeviceProtection_Yes	2.7
28	TechSupport_Yes	2.4
29	StreamingTV_Yes	13.5
30	StreamingMovies_Yes	19.5
31	PaperlessBilling_Yes	2.4
32	PaymentMethod_Credit Card (automatic)	1.9
33	PaymentMethod_Electronic Check	2.5
34	PaymentMethod_Mailed Check	2

Figure 15

Function for the stepwise feature selection

```
#Function for Stepwise Feature Selection:

def stepwise_selection(data, target,SL_in=0.05,SL_out = 0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while (len(initial_features)>0):
        remaining_features = list(set(initial_features)-set(best_features))
        new_pval = pd.Series(index=remaining_features, dtype='float64')
        for new_column in remaining_features:
            model = sm.Logit(target, sm.add_constant(data[best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if(min_p_value<SL_in):
            best_features.append(new_pval.idxmin())
            while(len(best_features)>0):
                best_features_with_constant = sm.add_constant(data[best_features])
                p_values = sm.Logit(target, best_features_with_constant).fit().pvalues[1:]
                max_p_value = p_values.max()
                if(max_p_value >= SL_out):
                    excluded_feature = p_values.idxmax()
                    best_features.remove(excluded_feature)
                else:
                    break
        else:
            break
    return best_features
```

Figure 16

Selected features for the reduced model

```
print(features)
```

```
['Tenure', 'MonthlyCharge', 'InternetService_Fiber Optic', 'Contract_Two Year', 'Contract_One year', 'Techie_Yes', 'OnlineBa
ckup_Yes', 'TechSupport_Yes', 'PaymentMethod_Electronic Check', 'InternetService_None', 'Multiple_Yes', 'Gender_Male', 'Devi
ceProtection_Yes', 'OnlineSecurity_Yes', 'Port_modem_Yes']
```

Figure 17

Reduced model predictors' VIF

	VIF Factor	features
0	2.3	Tenure
1	7.4	MonthlyCharge
2	2.3	InternetService_Fiber Optic
3	1.4	Contract_Two Year
4	1.3	Contract_One year
5	1.2	Techie_Yes
6	2.0	OnlineBackup_Yes
7	1.6	TechSupport_Yes
8	1.5	PaymentMethod_Electronic Check
9	1.5	InternetService_None
10	2.3	Multiple_Yes
11	1.8	Gender_Male
12	1.8	DeviceProtection_Yes
13	1.5	OnlineSecurity_Yes
14	1.8	Port_modem_Yes

Figure 18

Creating the reduced model

```
#Creating the reduced model:

re_mod = LogisticRegression().fit(re_X_train, y_train)

print('The accuracy score is: %.3f' % accuracy_score(y_train, re_mod.predict(re_X_train)))
print('The recall score is: %.3f' % recall_score(y_train, re_mod.predict(re_X_train)))
print('The precision score is: %.3f' % precision_score(y_train, re_mod.predict(re_X_train)))
```

```
The accuracy score is: 0.904
The recall score is: 0.795
The precision score is: 0.833
```

Figure 19

Evaluating the reduced model with test dataset

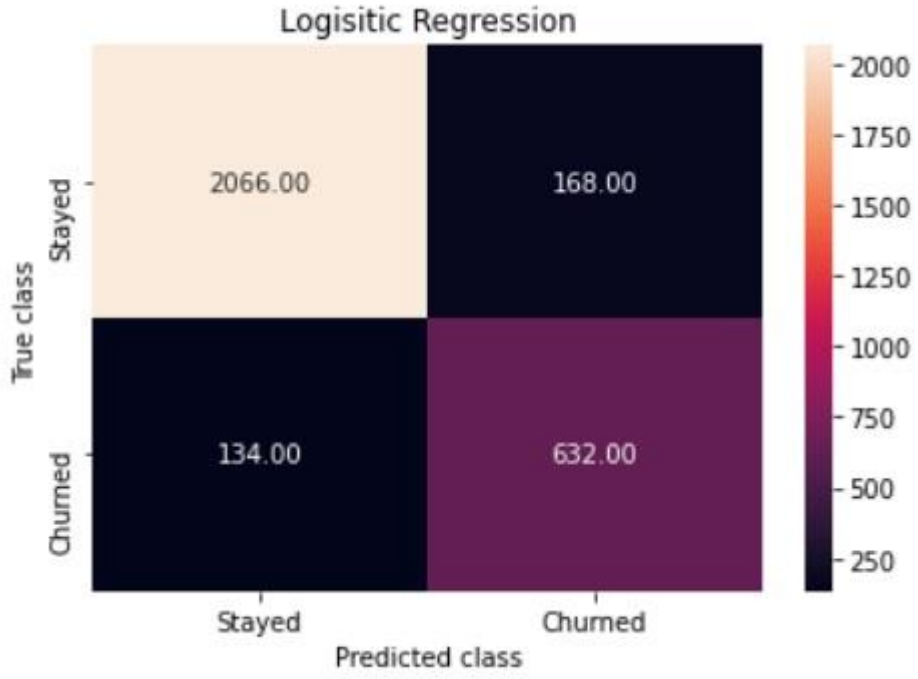
```
# Model scores with the test datapoints:

print('The accuracy score is: %.3f' % accuracy_score(y_test, re_mod.predict(re_X_test)))
print('The recall score is: %.3f' % recall_score(y_test, re_mod.predict(re_X_test)))
print('The precision score is: %.3f' % precision_score(y_test, re_mod.predict(re_X_test)))
```

```
The accuracy score is: 0.899
The recall score is: 0.790
The precision score is: 0.825
```

Figure 20

Confusion matrix for the reduced model



Data Summary and Implications

The predictors coefficients are shown in Figure 21. Based on these coefficients, we can create the logistic equation:

$$P(y) = \frac{1}{1 + e^{-(1.65 - 7.24X1 + 11.27X2 - 2.25X3 - 3.04X4 - 2.85X5 + 0.76X6 - 0.46X7 - 0.45X8 + 0.4X9 - 0.55X10 - .34X11 + .33X12 - 0.26X13 - 0.25X14 + 0.18X15}}$$

While *P(y)* is the probability of churn and *X1* through *X2* are the predictors seen in figure 16. In the reduced model, we have decreased the predictors down to 15 and compared to the initial model, the VIF for these predictors are much lower. The logistic regression used in this analysis might not be the right model. One thing that can be done is model this question with different classification models and compare them to choose the best one.

Figure 21

Reduced model coefficients

```
# New variables' coeficients:

re_mod.coef_
```

```
array([[-7.24207448, 11.27131723, -2.25401211, -3.04082961, -2.85121304,
         0.75707891, -0.46264004, -0.45281683,  0.40169175, -0.55308488,
        -0.33984439,  0.33484034, -0.25803439, -0.25236153,  0.18322409]])
```

This result can help the company to predict what percentage of their customers and who will potentially churn in the next month. Knowing that, marketing team and customer service can apply different approaches to prevent this.

Sources

The dataset used in this analysis was acquired form WGU portal:

https://access.wgu.edu/ASP3/aap/content/d9rkejv84kd9rk30fi2l.zip

The stepwise feature selection function was acquired form:

https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/

References

Assumptions of Logistic Regression. (2020, June 22). Statistics Solutions. https://www.statisticssolutions.com/assumptions-of-logistic-regression/

Kuhn, M., & Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models* (1st ed.). Chapman and Hall/CRC.

Kumar, S. (2020, December 27). How to avoid multicollinearity in Categorical Data - Towards Data Science. Medium. https://towardsdatascience.com/how-to-avoid-multicollinearity-in-categorical-data-46eb39d9cd0d