WGU D209 TASK 1 REV 0 - MATTINSON

KNN Classification Using Churn Data

Mike Mattinson

Master of Science, Data Analytics, WGU.edu

D209: Data Mining I

Dr. Festus Elleh

October 14, 2021

Abstract. This paper provides the _____.

Keywords. Customer churn prediction. Data Mining. K-Nearest Neighbors (KNN).

Partial Reuse. Portions of this notebook have been reused from previous WGU course work.

Mattinson, M. (2021, September). WGU D208 TASK 1 REV 8 - MATTINSON. Retrieved from: wgu.edu

Mattinson, M. (2021, October). WGU D208 TASK 2 REV 3 - MATTINSON. Retrieved from: wgu.edu

Apply Custom Notebook Styles. Apply custom .css styles to the notebook.

In [1]:
```
# Styling notebook with custom css
s = 'custom.css' # in the root folder
print('custom styles are found in {}'.format(s))
from IPython.core.display import HTML
HTML(open(s, "r").read())
```

```
custom styles are found in custom.css
```

Out[1]:

PART I: RESEARCH QUESTION

A1. RESEARCH QUESTION

Primary Research Question. ______ A typical services company's revenue is maximized based on the total number of customers and how much each of those customers pay for those services. If the company charges too much, then the customer may stop the service, this is known as churn. If the company charges too little, then it will not maximize its revenue. This analysis will attempt to predict the probability of a customer's churn (dependent variable is 'Churn' which is a binary categorical data) using logistic regression with high degree of accuracy based on a minimum set of predictor variables. The final set of predictor variables should include both numeric (e.g., Tenure, Child, and Income, etc.) and categorical data (e.g., Techie, Gender, and Internet Service type, etc.).

A2. OBJECTIVES AND GOALS

Data Preparation. Data Preparation objectives are addressed in Part III below and include the following:

A. Convert categorical data.

B. Mitigate missing data.

C. Select data required for the analysis.

D. Remove data deemed unneccesary.

E. Explore data.

F. Visualize data.

G. Provide copy of final data.

Model Analysis. Model Analysis objectives are addressed in Part IV below and include the following:

A. Eliminate predictor variables with high p-values.
B. Eliminate predictor variables with high degree of multicollinearity.
C. Create initial model using all the data.
D. Refine model using a reduced set of the data.
E. Summarize results.
F. Ensure independent and dependent variables are linear.
G. Ensure independent variables are not highly collinear
H. Ensure final model residuals are normally distributed.

PART II: METHOD JUSTIFICATION

B1. ASSUMPTIONS

Assumptions. According to Massaron and Boschetti (2019), the logistic regression analysis is based on the following assumptions:

A. **Binary Dependent Variable**. Binary logistic regression requires the dependent variable to be binary.
B. **Desired Outcome**. For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
C. **Only Meaningful Variables**. Only the meaningful variables should be included.
D. **Multi-Collinearity**. The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
E. **Independent Variable Linear to Log Odds**. The independent variables are linearly related to the log odds.
F. **Large Sample Size**. Logistic regression requires large sample sizes.

Massaron, L., Boschetti, A. (2016). Regression Analysis with Python. Retrieved from: https://www.packtpub.com/product/regression-analysis-with-python/9781785286315

B2. SUMMARIZE ONE ASSUMPTION OF

KNN

Assumption 1. ______.

B3. REQUIRED PACKAGES

The following packages are required: _____

PART III: DATA PREPARATION

C1. DESCRIBE ONE GOAL

C3. EXPLAIN EACH STEP

Select Data. From the original data, determine which attributes fit the best for the primary research question. Load the data from the provided .csv file as a pandas dataframe.

Mitigate Missing Data. Look through data for missing rows or columns. Also, look for Null or NaN values. If found, decide how best to mitigate the issue.

Remove Data. Once data is determined not to be of value to the analysis, use the pandas .drop() method to remove the data.

Convert Categorical Data. In order to use categorical data in the regression model, each variable must be converted into numeric dummy data. I will use pandas .get_dummies() method. This will generate new numeric variables based on the unique values and this will also remove the original attribute.

Explore Data. Explore customer data by calculating traditional statistics. Look for patterns and relationships between attributes. If possible, create visualizations to add in the exploratory process.

Visualize Data. Continue to explore data and their relationships using histogram, countplots, barplots and scatter plot diagrams. Use matplotlib and sns packages to generate these univariate and bivariate diagrams.

Import Packages. Import and configure required math, plotting and analysis packages.

In [2]:
```
# import standard libraries
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
from IPython.core.display import HTML
from IPython.display import display
```

In [3]:
```
# import and configure matplotlib
import matplotlib.pyplot as plt
plt.rc("font", size=14)
%matplotlib inline
```

In [55]:
```
# import and configure sklearn
from sklearn.metrics import confusion_matrix
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [5]:
```
# import and configure seaborn
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

In [6]:
```
# import and configure pandas
import pandas as pd
pd.set_option('precision',3)
pd.set_option('max_columns',9)
pd.set_option('display.width', None)
```

Configure Scrollbars. Disable scrollbars in notebook. And, Disable automatically scroll to bottom.

In [7]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

In [8]:
```
%%javascript
require("notebook/js/notebook").Notebook.prototype.scroll_to_bottom = function () {}
```

Toggle Warnings. Use the following code to toggle warning messages in the notebook. Another piece of code courtesy of stackoverflow (2021).

In [9]:
```
from IPython.display import HTML
HTML('''<script>
code_show_err=false;
function code_toggle_err() {
 if (code_show_err){
 $('div.output_stderr').hide();
 } else {
 $('div.output_stderr').show();
 }
 code_show_err = !code_show_err
}
$( document ).ready(code_toggle_err);
</script>
To toggle on/off output_stderr, click <a href="javascript:code_toggle_err()">here</a>.''')
```

Out[9]: To toggle on/off output_stderr, click here.

Stackoverflow (2021, October). Hide all warnings in ipython. Retrieved from:

https://stackoverflow.com/questions/9031783/hide-all-warnings-in-ipython

Helper Functions. Here are some helper functions that will be used thoughout the notebook. The coorelation matrix helpers were developed courtesy of stackoverflow (2021).

Constants. Here are a couple of global variables that will be reused thoughout the notebook.

In [10]:
```
# constants
COURSE = 'd209' # name of course to be added to filename of generated figures and tables.
target = 'Churn' # this is the column name of the primary research column
```

Select Data. The customer dataset as a .csv file is loaded into Python as a Pandas dataframe using the .read_csv() method.

After the dataframe is created, I use the df.shape function to show number of rows and columns. To begin the analysis, I have selected to load all of the data from the .csv file.

In [11]:
```
# read csv file
df = pd.read_csv('churn_clean.csv', header=0) # in the root folder
df.shape
```

Out[11]:
```
(10000, 50)
```

There are 10,000 customer records with fifty (50) attributes for each customer.

Mitigate Missing Data. Use .info() and .isna().any() methods to view a summary of possible missing data. I do not expect to find any missing data as the dataset provided has already been cleaned.

In [12]:
```
# explore missing data
missing = df[df.columns[df.isna().any()]].columns
df_missing = df[missing]
print(df_missing.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Empty DataFrameNone
```

Analysis of the raw data shows no missing data, each attribute has 10,000 non-null values.

Duplicate Data. Look for duplicate data in rows and columns. This dataset had been provided to this assignment in a very clean, ready state, so I don't expect to find anything here.

In [13]:
```
# look for duplicate data - looking for zero rows
df[df.duplicated()]
```

Out[13]:

CaseOrder	Customer_id	Interaction	UID	...	Item5	Item6	Item7	Item8

0 rows × 50 columns

In [14]:
```
# check if any cols are duplicated - Looking for False
df.columns.duplicated().any()
```

Out[14]:
```
False
```

In [15]:
```
# check if any rows are duplicated - looing for False
df.duplicated().any()
```

Out[15]:
```
False
```

Remove Data. Identify columns that are not needed for the analysis and then use the .drop() method to remove the data. Looking at the data, I select some of the demographic data, customer identification data and the survey data to be removed.

In [16]:
```
# drop unwanted data
cols_to_be_removed = ['City','County','Zip','Job','TimeZone', 'State',
            'Lat', 'Lng', 'UID', 'Customer_id','Interaction', 'CaseOrder',
            'Item1','Item2','Item3','Item4','Item5','Item6','Item7','Item8']

# print list of dropped data
print('data to be removed: {}'.format(cols_to_be_removed))

# loop through list, if in current df, drop col
for c in cols_to_be_removed:
    if c in df.columns:
        df.drop(columns = c, inplace=True)
        print('Data named [{}] has been removed.'.format(c))
```

```
data to be removed: ['City', 'County', 'Zip', 'Job', 'TimeZone', 'State', 'Lat', 'Lng',
'UID', 'Customer_id', 'Interaction', 'CaseOrder', 'Item1', 'Item2', 'Item3', 'Item4', 'It
em5', 'Item6', 'Item7', 'Item8']
Data named [City] has been removed.
Data named [County] has been removed.
Data named [Zip] has been removed.
Data named [Job] has been removed.
Data named [TimeZone] has been removed.
Data named [State] has been removed.
Data named [Lat] has been removed.
Data named [Lng] has been removed.
Data named [UID] has been removed.
Data named [Customer_id] has been removed.
Data named [Interaction] has been removed.
Data named [CaseOrder] has been removed.
Data named [Item1] has been removed.
Data named [Item2] has been removed.
Data named [Item3] has been removed.
Data named [Item4] has been removed.
Data named [Item5] has been removed.
Data named [Item6] has been removed.
Data named [Item7] has been removed.
Data named [Item8] has been removed.
```

C2. INITIAL VARIABLES

Numerical Data. Excluding target data, here is the final list of input variables that are numerical:

In [17]:

```python
# print out and describe input variables
print('\n{}'.format("Numerical data:"))
num_cols = df.select_dtypes(include="number").columns
for idx, c in enumerate(df.loc[:, df.columns != target]):
    if df.dtypes[c] != "object":
        print('\n{}. {} is numerical.'.format(idx+1, c))
        #print('{}'.format(df[c].describe().round(3)))
        #groups = df.groupby([target, pd.cut(df[c], bins=4)])
        #print(groups.size().unstack().T)
```

```
Numerical data:

1. Population is numerical.

3. Children is numerical.

4. Age is numerical.

5. Income is numerical.

8. Outage_sec_perweek is numerical.

9. Email is numerical.

10. Contacts is numerical.

11. Yearly_equip_failure is numerical.

27. Tenure is numerical.

28. MonthlyCharge is numerical.

29. Bandwidth_GB_Year is numerical.
```

Observations on Numerical Data. ________

Categorical Data. Excluding target data, here is the final list of input variables that are categorical:

In [18]:
```python
# list all categorical variables
print('\n{}'.format("Categorical data:"))
cat_cols = df.select_dtypes(include="object").columns
for idx, c in enumerate(df.loc[:, df.columns != target]):
    if df.dtypes[c] == "object":
        print('\n{}. {} is categorical: {}.'.format(idx+1,c,df[c].unique()))
        #for idx,name in enumerate(df[c].value_counts().index.tolist()):
        #    print('\t{:<20}:{:>6}'.format(name,df[c].value_counts()[idx]))
        #print('{}'.format(df[c].describe()))
```

```
Categorical data:

2. Area is categorical: ['Urban' 'Suburban' 'Rural'].

6. Marital is categorical: ['Widowed' 'Married' 'Separated' 'Never Married' 'Divorced'].

7. Gender is categorical: ['Male' 'Female' 'Nonbinary'].

12. Techie is categorical: ['No' 'Yes'].

13. Contract is categorical: ['One year' 'Month-to-month' 'Two Year'].

14. Port_modem is categorical: ['Yes' 'No'].

15. Tablet is categorical: ['Yes' 'No'].

16. InternetService is categorical: ['Fiber Optic' 'DSL' 'None'].

17. Phone is categorical: ['Yes' 'No'].

18. Multiple is categorical: ['No' 'Yes'].

19. OnlineSecurity is categorical: ['Yes' 'No'].

20. OnlineBackup is categorical: ['Yes' 'No'].

21. DeviceProtection is categorical: ['No' 'Yes'].

22. TechSupport is categorical: ['No' 'Yes'].

23. StreamingTV is categorical: ['No' 'Yes'].

24. StreamingMovies is categorical: ['Yes' 'No'].

25. PaperlessBilling is categorical: ['Yes' 'No'].

26. PaymentMethod is categorical: ['Credit Card (automatic)' 'Bank Transfer(automatic)'
'Mailed Check'
 'Electronic Check'].
```

Observations on Categorical Data. The first thing is there are many string variables that are Yes-No. I will convert each of those to boolean variables. Boolean type is treated as int(1) for True and int(0) for False, this will work for the models I plan on using. I will also include the target variable 'Churn', it is also a Yes-No variable.

In [19]:
```
# convert categorical yes-no data to boolean
for b in ['PaperlessBilling', 'StreamingMovies','StreamingTV',
          'TechSupport','DeviceProtection','OnlineBackup',
           'OnlineSecurity','Multiple','Phone', 'Tablet',
          'Port_modem','Techie','Churn']:
    df[b] = df[b].replace({"No":False, "Yes":True})
    df[b] = df[b].astype('bool')
```

Next, there are a couple of variables that have a lot of unique values, I will consolidate those variables so there are just a couple of unique values.

PaymentMethod. The PaymentMethod column has many categories and we can reduce the number of unique values in order to produce a better model. Let's combine all of the data into two (2) categories, 'Automatic' and 'Check'.

In [20]:
```
# re-cateogize Marital data
df['PaymentMethod']=np.where(df['PaymentMethod'] =='Credit Card (automatic)', 'Auto',df['Pa
df['PaymentMethod']=np.where(df['PaymentMethod'] =='Bank Transfer(automatic)', 'Auto',df['P
df['PaymentMethod']=np.where(df['PaymentMethod'] =='Mailed Check', 'Check',df['PaymentMetho
df['PaymentMethod']=np.where(df['PaymentMethod'] =='Electronic Check', 'Auto',df['PaymentMe
df['PaymentMethod'].unique()
```

Out[20]: array(['Auto', 'Check'], dtype=object)

Marital. The Marital column has many categories and we can reduce the number of unique values in order to produce a better model. Let's combine all of the data into two (2) categories, 'Married' and 'Not_Married'.

In [21]:
```
# re-cateogize Marital data
df['Marital']=np.where(df['Marital'] =='Widowed', 'Not_Married',df['Marital'])
df['Marital']=np.where(df['Marital'] =='Separated', 'Not_Married',df['Marital'])
df['Marital']=np.where(df['Marital'] =='Never Married', 'Not_Married',df['Marital'])
df['Marital']=np.where(df['Marital'] =='Divorced', 'Not_Married',df['Marital'])
df['Marital'].unique()
```

Out[21]: array(['Not_Married', 'Married'], dtype=object)

In [22]:
```
df.head(4).T
```

Out[22]:

	0	1	2	3
Population	38	10446	3735	13863
Area	Urban	Urban	Urban	Suburban
Children	0	1	4	1
Age	68	27	50	48
Income	28561.99	21704.77	9609.57	18925.23
Marital	Not_Married	Married	Not_Married	Married
Gender	Male	Female	Female	Male
Churn	False	True	False	False
Outage_sec_perweek	7.978	11.699	10.753	14.914
Email	10	12	9	15
Contacts	0	0	0	2
Yearly_equip_failure	1	1	1	0
Techie	False	True	True	True
Contract	One year	Month-to-month	Two Year	Two Year
Port_modem	True	False	True	False
Tablet	True	True	False	False
InternetService	Fiber Optic	Fiber Optic	DSL	DSL
Phone	True	True	True	True
Multiple	False	True	True	False
OnlineSecurity	True	True	False	True
OnlineBackup	True	False	False	False
DeviceProtection	False	False	False	False
TechSupport	False	False	False	False
StreamingTV	False	True	False	True
StreamingMovies	True	True	True	False
PaperlessBilling	True	True	True	True
PaymentMethod	Auto	Auto	Auto	Check
Tenure	6.796	1.157	15.754	17.087
MonthlyCharge	172.456	242.633	159.948	119.957
Bandwidth_GB_Year	904.536	800.983	2054.707	2164.579

Observations. `Observations:`

A. The **Income** of churned customers is slightly higher.

B. The **Tenure** of churned customers is significantly lower.

C. The **MonthlyCharge** of churned customers is considerable higher.

D. The **Bandwidth_GB_Year** is significantly lower.

Convert Selected Categorical Data. Now that I have selected some of the categorical data that seem to be a good predictors of the outcome, I will convert these categorical data to dummy, numeric data. Each new variable will have a value of either one (1) or zero (0).

C4. PROVIDE COPY OF DATA

Final Data. Here is the final list of columns after all data cleaning.

In [23]:
```
# Provide copy of the prepared data set.
final_data = 'd209_task1_final_data.csv'
df.to_csv(final_data, index=False, header=True)
print('File saved to: {}'.format(final_data))
print(df.columns.to_series().groupby(df.dtypes).groups)
```

```
File saved to: d209_task1_final_data.csv
{bool: ['Churn', 'Techie', 'Port_modem', 'Tablet', 'Phone', 'Multiple', 'OnlineSecurity',
'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Pap
erlessBilling'], int64: ['Population', 'Children', 'Age', 'Email', 'Contacts', 'Yearly_eq
uip_failure'], float64: ['Income', 'Outage_sec_perweek', 'Tenure', 'MonthlyCharge', 'Band
width_GB_Year'], object: ['Area', 'Marital', 'Gender', 'Contract', 'InternetService', 'Pa
ymentMethod']}
```

```
In [24]: df.head(4).T
```

Out[24]:

	0	1	2	3
Population	38	10446	3735	13863
Area	Urban	Urban	Urban	Suburban
Children	0	1	4	1
Age	68	27	50	48
Income	28561.99	21704.77	9609.57	18925.23
Marital	Not_Married	Married	Not_Married	Married
Gender	Male	Female	Female	Male
Churn	False	True	False	False
Outage_sec_perweek	7.978	11.699	10.753	14.914
Email	10	12	9	15
Contacts	0	0	0	2
Yearly_equip_failure	1	1	1	0
Techie	False	True	True	True
Contract	One year	Month-to-month	Two Year	Two Year
Port_modem	True	False	True	False
Tablet	True	True	False	False
InternetService	Fiber Optic	Fiber Optic	DSL	DSL
Phone	True	True	True	True
Multiple	False	True	True	False
OnlineSecurity	True	True	False	True
OnlineBackup	True	False	False	False
DeviceProtection	False	False	False	False
TechSupport	False	False	False	False
StreamingTV	False	True	False	True
StreamingMovies	True	True	True	False
PaperlessBilling	True	True	True	True
PaymentMethod	Auto	Auto	Auto	Check
Tenure	6.796	1.157	15.754	17.087
MonthlyCharge	172.456	242.633	159.948	119.957
Bandwidth_GB_Year	904.536	800.983	2054.707	2164.579

In [25]:
```
churn_df = df[['Tenure','MonthlyCharge','Churn']]
churn_df['Number'] = churn_df.index + 1
churn_df.head(20)
```

Out[25]:

	Tenure	MonthlyCharge	Churn	Number
0	6.796	172.456	False	1
1	1.157	242.633	True	2
2	15.754	159.948	False	3
3	17.087	119.957	False	4
4	1.671	149.948	True	5
5	7.001	185.008	False	6
6	13.237	200.119	True	7
7	4.264	114.951	True	8
8	8.221	117.469	False	9
9	3.422	162.483	False	10
10	19.267	174.958	False	11
11	10.522	149.962	False	12
12	13.011	137.439	False	13
13	16.879	184.972	False	14
14	10.060	159.966	True	15
15	13.870	177.651	True	16
16	15.782	194.966	True	17
17	2.303	202.683	True	18
18	17.110	152.491	False	19
19	12.806	149.945	True	20

PART IV: ANALYSIS

D1. SPLIT DATA

Split Data. Split data into training and test data.

In [26]:
```
# train test split
train_df, test_df = train_test_split(churn_df, test_size=0.4, random_state=26)
```

Provide Data. Provide copy of both datasets.

In [27]:
```
# Provide copy of train data
train_file = 'd209_task1_train_data.csv'
train_df.to_csv(train_file, index=False, header=True)
print('File saved to: {}'.format(train_file))
print(train_df.shape)

# Provide copy of test data
test_file = 'd209_task1_test_data.csv'
test_df.to_csv(test_file, index=False, header=True)
print('File saved to: {}'.format(test_file))
print(test_df.shape)
```

```
File saved to: d209_task1_train_data.csv
(6000, 4)
File saved to: d209_task1_test_data.csv
(4000, 4)
```

Define New Customer. Setup new customer variable.

In [28]:
```
## new customer -> list[int]
newCustomer = pd.DataFrame([{'MonthlyCharge': 250.6, 'Tenure': 10.4}])
newCustomer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   MonthlyCharge  1 non-null      float64
 1   Tenure         1 non-null      float64
dtypes: float64(2)
memory usage: 144.0 bytes
```

Create Figure. Create scatter plot of data.

In [57]:
```python
## scatter plot
def plotDataset(ax, data, showLabel=True, **kwargs):
    subset = data.loc[data['Churn']==True]
    ax.scatter(subset.MonthlyCharge, subset.Tenure, marker='o',
               label='Churn' if showLabel else None, color='C1', **kwargs)
    subset = data.loc[data['Churn']==False]
    ax.scatter(subset.MonthlyCharge, subset.Tenure, marker='D',
               label='Non Churn' if showLabel else None, color='C0', **kwargs)

fig, ax = plt.subplots()
fig.set_size_inches(18.5, 10.5)
x_std = train_df['MonthlyCharge'].std()
y_std = train_df['Tenure'].std()
x = newCustomer['MonthlyCharge'][0] # float
y = newCustomer['Tenure'][0] # float

plotDataset(ax, train_df)
plotDataset(ax, test_df, showLabel=False, facecolors='none')
ax.scatter(newCustomer.MonthlyCharge, newCustomer.Tenure, marker='*',
          label='New customer', color='black', s=270)
plt.title('MonthlyCharge vs Tenure Scatter Plot')
plt.xlabel('MonthlyCharge')
plt.ylabel('Tenure')
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, -0.05),
          fancybox=True, shadow=True, ncol=5)
fig.savefig('d209_fig_4_1_monthlycharge_vs_tenure_scatter.png', dpi=200)
plt.show()
```

FIGURE 4.1. SCATTER PLOT OF MONTHLYCHARGE VS TENURE FOR TRAINING SET (SOLID MARKERS) AND TEST SET (HOLLOW MARKERS) AND THE NEW CUSTOMER (STAR MARKER) TO BE CLASSIFIED.

In [30]:
```
# initialize normalized training, validation and complete data frames
scaler = preprocessing.StandardScaler()
scaler.fit(train_df[['MonthlyCharge','Tenure']])
```

Out[30]:
```
StandardScaler()
```

In [31]:
```
# transform the full dataset
churn_norm = pd.concat([pd.DataFrame(scaler.transform(churn_df[['MonthlyCharge','Tenure']]
                                    columns=['zMonthlyCharge','zTenure']),
                       churn_df[['Churn','Number']]], axis=1)
train_norm = churn_norm.iloc[train_df.index]
test_norm = churn_norm.iloc[test_df.index]
newCustomer_norm = pd.DataFrame(scaler.transform(newCustomer),
                               columns=['zMonthlyCharge','zTenure'])
```

TABLE 4.X. RUNNING KNN FOR A SINGLE VALUE, K = 4

In [32]:
```
# use NearestNeighbors from scikit-learn to compute knn
from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(n_neighbors=4)
knn.fit(train_norm.iloc[:,0:2])
distances, indices = knn.kneighbors(newCustomer_norm)
train_norm.iloc[indices[0],:]
```

Out[32]:

	zMonthlyCharge	zTenure	Churn	Number
133	1.865	-0.915	True	134
887	1.864	-0.936	True	888
4910	1.749	-0.917	True	4911
2606	1.865	-0.948	True	2607

Accuracy. Calculate accuracy and error rates for many k-values.

In [33]:
```
# code for measuring accuracy of different k values on validation set
train_X = train_norm[['zMonthlyCharge','zTenure']]
train_y = train_norm['Churn']
test_X = test_norm[['zMonthlyCharge','zTenure']]
test_y = test_norm['Churn']
```

In [60]:
```
# Calculating error for K values between 1 and 40
error = []
for i in range(1, 60):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(train_X, train_y)
    pred_i = knn.predict(test_X)
    error.append(np.mean(pred_i != test_y))
```

In [64]:
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 60), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate by k-value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
fig.savefig('d209_fig_4_2_error_rates_by_k_value.png', dpi=200)
plt.show()
```

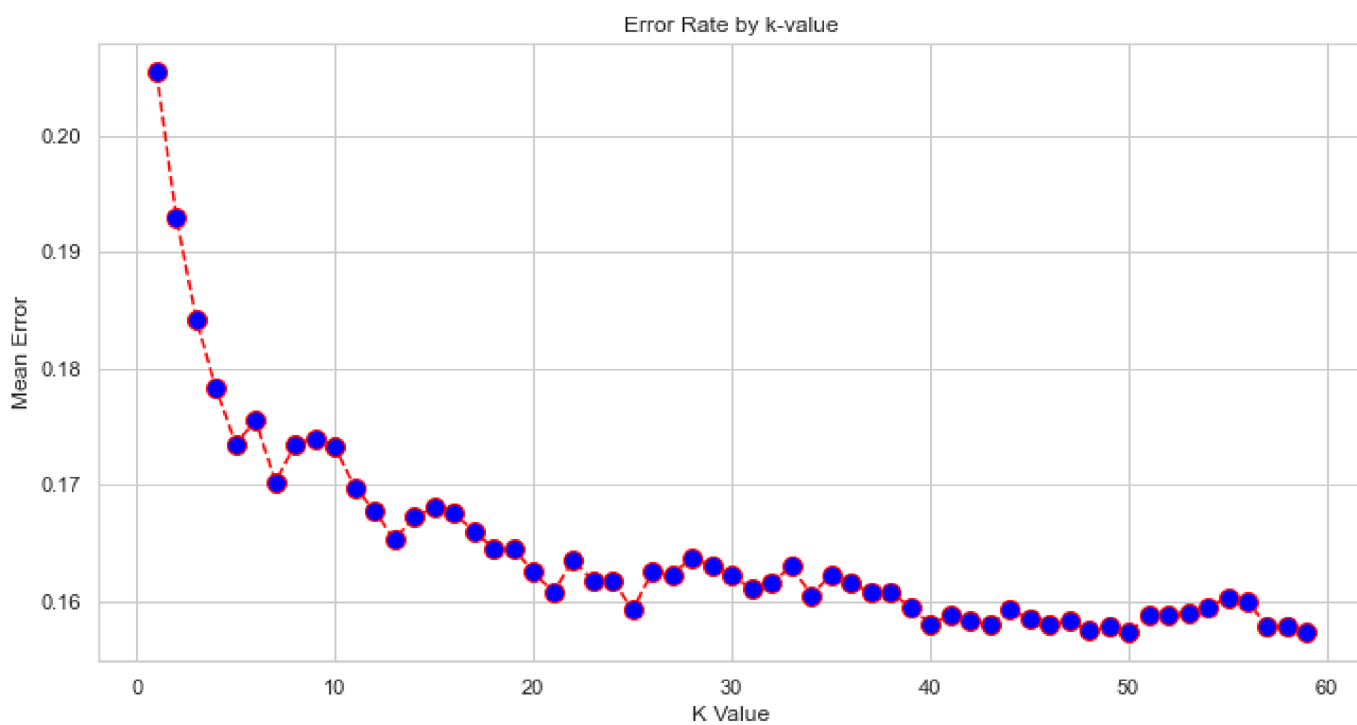

FIGURE 4.2. ERROR RATES BY K-VALUE

TABLE 4.x. ACCURACY BY K-VALUE

In [52]:
```python
# train a classifier for different values of k

results =[]
for k in range(1,20):
    knn = KNeighborsClassifier(n_neighbors=k).fit(train_X,train_y)
    results.append({
        'k': k,
        'accuracy': accuracy_score(test_y, knn.predict(test_X)).round(2),
        'error': 1 - accuracy_score(test_y, knn.predict(test_X)).round(2)
    })
results = pd.DataFrame(results) # convert results to pandas dataframe
print(results)
```

```
     k  accuracy  error
0    1      0.79   0.21
1    2      0.81   0.19
2    3      0.82   0.18
3    4      0.82   0.18
4    5      0.83   0.17
5    6      0.82   0.18
6    7      0.83   0.17
7    8      0.83   0.17
8    9      0.83   0.17
9   10      0.83   0.17
10  11      0.83   0.17
11  12      0.83   0.17
12  13      0.83   0.17
13  14      0.83   0.17
14  15      0.83   0.17
15  16      0.83   0.17
16  17      0.83   0.17
17  18      0.84   0.16
18  19      0.84   0.16
```

TABLE 4.X. CLASSIFYING NEW CUSTOMER USING SPECIFIC K-VALUE

In [50]:
```python
# retrain with full dataset
k = 5 # select value from accuracy table above
churn_X = churn_norm[['zMonthlyCharge','zTenure']]
churn_y = churn_norm['Churn']
knn = KNeighborsClassifier(n_neighbors=k).fit(churn_X,churn_y)
distances, indices = knn.kneighbors(newCustomer_norm)
print(knn.predict(newCustomer_norm))
print('Distances',distances)
print('Indices',indices)
print(churn_norm.iloc[indices[0],:])
```

```
[ True]
Distances [[0.05350408 0.05894091 0.05950446 0.06550454 0.06583702]]
Indices [[ 133  460 2901  887 2077]]
      zMonthlyCharge  zTenure  Churn  Number
133            1.865   -0.915   True     134
460            1.864   -0.926   True     461
2901           1.865   -0.926   True    2902
887            1.864   -0.936   True     888
2077           1.864   -0.844   True    2078
```

PART V: DATA SUMMARY AND IMPLICATIONS

E1. EXPLAIN ACCURACY

Accuracy. For an example, with k=3, we found that the three (3) nearest neighbors to the new customer (with MonthlyCharge = $155.60 and Tenure = 12.4) are customers 1215, 370, and 2498. Since two (2) of them are False and one (1) of them are True, we can estimate for the new customer a probability of 2/3 = 0.667 of not churning (and a probability of 1/3 = 0.333 of churning).

E4. RECOMMENDATIONS

Recommendations. Customers will be less likely to churn if their **MonthlyCharge** is minimized and if the customer has any of the additionaly available services. Focus marketing efforts on which additional services are best for each customer, maybe bundle some of the services at a slightly reduced monthly payment. Increase customers' awareness of the value of the additional services for what they are paying each month. Keeping their monthly payment low and increasing the number of extra services will minimize likelihood of churn, and provide company with increase in the customer lifetime revenue.

PART VI: DEMONSTRATION

F. PROVIDE PANOPTO VIDEO

The Panopto video recording was created and here is the link: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a90e1811-2440-4e27-a081-adbb01855443 (https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a90e1811-2440-4e27-a081-adbb01855443)

G. PROVIDE WEB SOURCES

In [ ]:

```

```