WGU D209 TASK 1 REV 3 - MATTINSON

KNN Classification Using Churn Data

Mike Mattinson

Master of Science, Data Analytics, WGU.edu

D209: Data Mining I

Task 1 - 1st Submission

Dr. Festus Elleh

October 28, 2021

Part I: Research Question

A. Describe the purpose of this data mining report by doing the following:

A1. Propose one question relevant to a real-world organizational situation that you will answer using one of the following classification methods: (a) k-nearest neighbor (KNN) or (b) Naive Bayes.

Primary Goal. The question has come up for a telecommunications company regarding churn. **Churn** is defined when a customer chooses to stop services. If the company has data on customers that have and have not churned in the past, is it possible to classify a new (or existing) customer based on their similarity to other customers with similar attributes that have and have not churned in the past. This analysis will consider two (2) attributes, **MonthlyCharge** and **Tenure** within the company's customer data of 10,000 customers. In addition, if the prediction is made, the analysis will also attempt to quantify the accuracy of the prediction.

A2. Define one goal of the data analysis. Ensure that your goal is reasonable within the scope of the scenario and is represented in the available data.

Primary Goal. The analysis will attempt to predict **Churn** for a new customer with values of **MonthlyCharge** = $170.00 and **Tenure** = 1.0. This goal is within the scope of the company's customer data, both attributes are contained with the data for 10,000 customers and should provide adequate data for the prediction. The analysis will use K-nearest neighbors (KNN) to classify the new customer based on the k-nearest other customers with similar attributes.

In [1]:
```
# define new customer
import pandas as pd
newCustomer = pd.DataFrame([{'Tenure': 1.0,
                             'MonthlyCharge': 170.0,
                            'zTenure': 0.0,
                            'zMonthlyCharge': 0.0}])
```

Part II: Method Justification

B. Explain the reasons for your chosen classification method from part A1 by doing the following:

B1. Explain how the classification method you chose analyzes the selected data set. Include expected outcomes.

Explain Method. KNN classification will look for similar attributes in the closest k-neighbors, that are in close proximity to the target value to be classified. It will decide which classification value occurs most frequently in those k-neighbors and then output a classification prediction based on those values. I would expect the results to show the target variable as it relates to the k-neighbors and accuracy summaries for the model.

B2. Summarize one assumption of the chosen classification method.

One Assumption. One key assumption for KNN modeling is that similar things are close to each other. To classify the new customer, it will look for similar customer records and then make a classification based on which class occurs most frequently in those close neighbors.

B3. List the packages or libraries you have chosen for Python or R, and justify how each item on the list supports the analysis.

In [2]:
```
# 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)
```

Pandas is a data workhorse. It will allow data to be read in from company data .CSV file and then manipulated in many ways to clean and present the data.

In [3]:
```
# import and configure scientific computing
import numpy as np
import scipy.stats as stats
#import statsmodels.api as sm
#import statsmodels.formula.api as smf
```

Numpy and Scipy allow math and scientific computations.

In [4]:
```
# 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
from sklearn.neighbors import NearestNeighbors
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

For this analysis, sklearn provides all of the modeling and measurements.

In [5]:
```
# import and configure matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
plt.rc("font", size=14)
%matplotlib inline
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

Matplotlib and Seaborn provide all of the plotting functions. All of the figures created in this notebook were created using matplotlib and seaborn packages.

In [6]:
```
# helper function to plot a given dataset with the target data
def plotDataset(ax, data, xFeature, yFeature, target, neighbors, showLabel=True, **kwargs)

    # Churn == True
    subset = data.loc[data[target]==True]
    ax.scatter(subset[xFeature], subset[yFeature], marker='o',
              label=str(target)+'=True' if showLabel else None, color='C1', **kwargs)

    # Churn == False
    subset = data.loc[data[target]==False]
    ax.scatter(subset[xFeature], subset[yFeature], marker='D',
              label=str(target)+'=False' if showLabel else None, color='C0', **kwargs)

    # labels
    if len(neighbors) > 0:
        for idx, row in data.iterrows():
            ax.annotate(str(idx), (row[xFeature], row[yFeature]))
```

In [7]:
```
# helper function to standardize the format
# a filename for figures and tables
COURSE = 'D209' # global
TASK = 'Task1' # global
FTYPE = 'PNG' # global

def getFilename(title: str, sect: str,
            caption: str, ftype = FTYPE,
            course = COURSE, task = TASK,
               subfolder='figures') -> str:
    """
    Prepare filename for a figure or table
    """
    temp = subfolder + '/'  # subfolder for tables and figures, default is 'fig'
    temp += COURSE + '_'
    temp += TASK + '_'
    temp += sect + '_'
    temp += subfolder[0:3] + " " +caption + '_' #
    temp += title
    temp += '.' + ftype

    return temp.replace(' ','_').upper()
```

In [8]:
```
f = getFilename('hello', sect='d2',
    subfolder='tables', caption='4 1', ftype='CSV' )
print(f)
```

```
TABLES/D209_TASK1_D2_TAB_4_1_HELLO.CSV
```

Part III: Data Preparation

C. Perform data preparation for the chosen data set by doing the following:

C1. Describe one data preprocessing goal relevant to the classification method

from part A1.

One Data Preprocessing Goal. In order to apply the KNN classification analysis to this problem, the company data must be imported into the Python environment and then the raw numerical data must be normalized. In addition, the company data will be broken up into two (2) subsets, 70% in a training dataset, and the remain 30% in a testing or validation dataset. The KNN will then use the training set to build the model, and it will use the test set to validate the model. The main goal for data preparation will be to define these subsets of data is a manner that is as simple and intuitive as possible, to allow anyone to follow the analysis throughout the notebook. The following is a list of the planned data variables for this analysis:

Raw Data.

- **y** = target data (i.e. **Churn** (categorical))
- **X** = feature data (i.e. **MonthlyCharge**, and **Tenure**)
- **rawData** = y.merge(X)

Clean Data.

- **y** = target data (i.e. **Churn** (bool))
- **X** = feature data (i.e. **MonthlyCharge**, **Tenure**, **zMonthlyCharge**, and **zTenure**)
- **cleanData** = y.merge(X)

Training Data. 70% of the cleaned data.

- **X_train** = created using train-test-split (i.e. **zMonthlyCharge**, and **zTenure**)
- **y_train** = created using train-test-split
- **trainData** = y_train.merge(X_train)

Testing Data. The remaining 30% of the cleaned data.

- **X_test** = created using train-test-split (i.e. **zMonthlyCharge**, and **zTenure**)
- **y_test** = created using train-test-split
- **testData** = y_test.merge(X_test)

C2. Identify the initial data set variables that you will use to perform the analysis for the classification question from part A1, and classify each variable as continuous or categorical.

In [9]:
```
# read subset of company's customer data from csv file
import pandas as pd
y = pd.read_csv('data/churn_clean.csv',
        usecols=['Churn'])
X = pd.read_csv('data/churn_clean.csv',
        usecols=['Tenure','MonthlyCharge'])
```

Identify Initial Variables. For this analysis, I will consider two (2) features, **MonthlyCharge** and **Tenure**, and one (1) target, **Churn**. Pandas is used to read the .CSV raw data file, the USECOLS option retrieves only selected data from the file.

- **MonthlyCharge** (FEATURE) the amount charged to the customer monthly, it reflects an average per customer
- **Tenure** (FEATURE) the number of months the customer has stayed with the provider
- **Churn** (TARGET) is whether the customer has discontinued service within the last month (yes, no).

TABLE 3-1. SELECTED RAW CUSTOMER DATA. THIS IS THE PRIMARY DATASET IDENTIFIED AS 'RAWDATA'. NOTICE CHURN VALUES (YES AND NO)

Ref. (1) https://stackoverflow.com/questions/15017072/pandas-read-csv-and-filter-columns-with-usecols (https://stackoverflow.com/questions/15017072/pandas-read-csv-and-filter-columns-with-usecols)

In [10]:
```
# describe raw data
rawData = y.merge(X, left_index=True, right_index=True)
d = rawData
f = getFilename('RAW', sect='C2',
        subfolder='tables', caption='3 1', ftype='CSV')
d.to_csv(f, index=True, header=True)
display(d.head())
print('Table saved to: {}'.format(f))
```

	Churn	Tenure	MonthlyCharge
0	No	6.796	172.456
1	Yes	1.157	242.633
2	No	15.754	159.948
3	No	17.087	119.957
4	Yes	1.671	149.948

```
Table saved to: TABLES/D209_TASK1_C2_TAB_3_1_RAW.CSV
```

Target Data (y). Convert categorical Churn to numeric boolean. Ref: (1) https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

In [11]:
```
# convert churn from object [Yes No] to bool [True False]
target = 'Churn'
y[target] = y[target].replace({"No":False, "Yes":True})
y[target] = y[target].astype('bool')
print('\n{}. {} is boolean (BINARY): {}.'.format(0,target,y[target].unique()))
```

```
0. Churn is boolean (BINARY): [False  True].
```

Input Data Cleaning (X). Transform each variable using standard transformation.

In [12]:
```
# scale variables using standard transformation
for c in X.columns:
    X['z'+c] = (X[c] - X[c].mean()) / X[c].std()
```

In [13]:
```
# identify the initial set of variables
for idx, c in enumerate(X.columns):
    if X.dtypes[c] in ('float', 'int', 'int64'):
        print('\n{}. {} is numerical (CONTINUOUS).'.format(idx+1, c))
    elif X.dtypes[c] == bool:
        print('\n{}. {} is boolean (BINARY): {}.'.format(idx+1,c,df[c].unique()))
    else:
        print('\n{}. {} is categorical (CATEGORICAL): {}.'.format(idx+1,c,df[c].unique()))
```

```
1. Tenure is numerical (CONTINUOUS).

2. MonthlyCharge is numerical (CONTINUOUS).

3. zTenure is numerical (CONTINUOUS).

4. zMonthlyCharge is numerical (CONTINUOUS).
```

TABLE 3-2. DESCRIBE NUMERIC FEATURE DATA. THESE ARE THE TRADITIONAL STATISTICS FOR THE NUMERIC DATA

In [14]:
```
# describe numeric feature data
d =X.describe().round(3)
f = getFilename('DF STATS', sect='c2',
       subfolder='tables', caption='3 2', ftype='CSV')
d.to_csv(f, index=True, header=True)
display(d)
print('Table saved to: {}'.format(f))
```

	Tenure	MonthlyCharge	zTenure	zMonthlyCharge
count	10000.000	10000.000	10000.000	10000.000
mean	34.526	172.625	0.000	-0.000
std	26.443	42.943	1.000	1.000
min	1.000	79.979	-1.268	-2.157
25%	7.918	139.979	-1.006	-0.760
50%	35.431	167.485	0.034	-0.120
75%	61.480	200.735	1.019	0.655
max	71.999	290.160	1.417	2.737

```
Table saved to: TABLES/D209_TASK1_C2_TAB_3_2_DF_STATS.CSV
```

Summary. The company's customer raw data has been read into the df variable and consists of 10,000 customer records with three (3) variables each. Two (2) of the variables will be used as features and are continuous (numerical) data, and the the third variable is our target, binary variable. In addition to the raw data, a Z-scored column was also included for each variable which is the standard transformation.

C3. Explain each of the steps used to prepare the data for the analysis. Identify the code segment for each step.

Step 1.

Read in selected company data. Applicable customer data (**Churn**, **MonthlyCharge** and **Tenure**) from the company data was read into Python environment using pandas .read_cs() function using the usecols=[] option. This was completed in section C2 [9] above.

Step 2.

Convert cateogrical dataInitially, the **Churn** variable was categorical, each row was Yes or No values, so this step converted the categorical data to boolean data using pandas .replace() function. In Python, boolean data is considered as numerical data, 1 or 0, or type(int). This was completed in section C2 [9] above.

Step 3.

Describe initial set of variablesFor each variable of data, describe the data whether numerical or categorical. I used a function I created to loop through and list each one and a short description. Also, use pandas .describe() method to show descriptive statistics for numerical data. This was completed in section C2 [10] and C2[11] above.

Step 4.

Quick check for null valuesThe company data was previously cleaned and prepared, so I do not expect to find null values, but using the pandas .info() I can observe quickly that there are not any null values for any of the 10,000 customer records. This was completed in section C2 [12] above.

C4. Provide Clean Data

Provide a copy of the cleaned data set.

TABLE 3-3.CLEAN DATA

In [15]:
```
# provide copy of cleaned data
cleanData = y.merge(X, left_index=True, right_index=True) # refreshed after cleaning
d = cleanData
f = getFilename('CLEAN', sect='c4',
        subfolder='tables', caption='3 3', ftype='CSV')
d.to_csv(f, index=True, header=True)
display(d.head())
print(d.info())
print('Shape (rows, cols): {}'.format(d.shape))
print('Table saved to: {}'.format(f))
```

	Churn	Tenure	MonthlyCharge	zTenure	zMonthlyCharge
0	False	6.796	172.456	-1.049	-0.004
1	True	1.157	242.633	-1.262	1.630
2	False	15.754	159.948	-0.710	-0.295
3	False	17.087	119.957	-0.659	-1.226
4	True	1.671	149.948	-1.242	-0.528

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Churn           10000 non-null  bool
 1   Tenure          10000 non-null  float64
 2   MonthlyCharge   10000 non-null  float64
 3   zTenure         10000 non-null  float64
 4   zMonthlyCharge  10000 non-null  float64
dtypes: bool(1), float64(4)
memory usage: 322.4 KB
None
Shape (rows, cols): (10000, 5)
Table saved to: TABLES/D209_TASK1_C4_TAB_3_3_CLEAN.CSV
```

Part IV Analysis

D. Perform the data analysis and report on the results by doing the following:

D1. Split the data into training and test data sets and provide the file(s).

In [16]:
```
# train test split raw data
#trainData, validData = train_test_split(df, test_size=0.3, random_state=13)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=13)
```

In [17]:
```
X.head()
```

Out[17]:

	Tenure	MonthlyCharge	zTenure	zMonthlyCharge
0	6.796	172.456	-1.049	-0.004
1	1.157	242.633	-1.262	1.630
2	15.754	159.948	-0.710	-0.295
3	17.087	119.957	-0.659	-1.226
4	1.671	149.948	-1.242	-0.528

In [18]:
```
y.head()
```

Out[18]:

	Churn
0	False
1	True
2	False
3	False
4	True

In [19]:
```
X.shape
```

Out[19]: (10000, 4)

In [20]:
```
y.shape
```

Out[20]: (10000, 1)

In [21]:
```
X_train.shape
```

Out[21]: (7000, 4)

In [22]:
```
y_train.shape
```

Out[22]: (7000, 1)

In [23]:
```
X_test.shape
```

Out[23]:
```
(3000, 4)
```

In [24]:
```
y_test.shape
```

Out[24]:
```
(3000, 1)
```

TABLE 4-4.TRAINING DATA

In [25]:
```
# provide TRAIN data
trainData = y_train.merge(X_train, left_index=True, right_index=True)
d = trainData
f = getFilename('TRAIN DATA', sect='D1',
        subfolder='tables',caption='4 4', ftype='CSV')
d.to_csv(f, index=True, header=True)
display(d.head())
print(d.index)
print(d.info())
print('Shape (rows, cols): {}'.format(d.shape))
print('Table saved to: {}'.format(f))
```

	Churn	Tenure	MonthlyCharge	zTenure	zMonthlyCharge
4847	False	9.525	92.488	-0.945	-1.866
9992	False	56.472	137.439	0.830	-0.819
4621	False	2.612	124.964	-1.207	-1.110
5774	False	58.787	139.983	0.917	-0.760
9294	False	64.116	255.120	1.119	1.921

```
Int64Index([4847, 9992, 4621, 5774, 9294, 1085, 1073,  950, 9512, 3773,
            ...
            6782, 9114, 4026, 8940,  153, 5876,  866, 7696,   74,  338],
           dtype='int64', length=7000)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7000 entries, 4847 to 338
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Churn           7000 non-null   bool
 1   Tenure          7000 non-null   float64
 2   MonthlyCharge   7000 non-null   float64
 3   zTenure         7000 non-null   float64
 4   zMonthlyCharge  7000 non-null   float64
dtypes: bool(1), float64(4)
memory usage: 538.3 KB
None
Shape (rows, cols): (7000, 5)
Table saved to: TABLES/D209_TASK1_D1_TAB_4_4_TRAIN_DATA.CSV
```

TABLE 4-5.TEST DATA

In [26]:
```
# provide TEST data
testData = y_test.merge(X_test, left_index=True, right_index=True)
d = testData
f = getFilename('TEST DATA', sect='D1',
        subfolder='tables', caption='4 5', ftype='CSV')
d.to_csv(f, index=True, header=True)
display(d.head())
print(d.index)
print(d.info())
print('Shape (rows, cols): {}'.format(d.shape))
print('Table saved to: {}'.format(f))
```

	Churn	Tenure	MonthlyCharge	zTenure	zMonthlyCharge
5952	False	56.633	114.984	0.836	-1.342
1783	False	2.851	117.483	-1.198	-1.284
4811	True	5.664	230.105	-1.091	1.339
145	True	2.733	217.473	-1.202	1.044
7146	True	56.275	200.132	0.822	0.641

```
Int64Index([5952, 1783, 4811,  145, 7146, 2452, 4051, 4311, 9715,  303,
            ...
            1442, 5091, 6525, 1241, 1161, 8654, 9777, 3727, 7848, 4977],
           dtype='int64', length=3000)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3000 entries, 5952 to 4977
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Churn           3000 non-null   bool
 1   Tenure          3000 non-null   float64
 2   MonthlyCharge   3000 non-null   float64
 3   zTenure         3000 non-null   float64
 4   zMonthlyCharge  3000 non-null   float64
dtypes: bool(1), float64(4)
memory usage: 184.7 KB
None
Shape (rows, cols): (3000, 5)
Table saved to: TABLES/D209_TASK1_D1_TAB_4_5_TEST_DATA.CSV
```

D2. Describe the analysis technique you used to appropriately analyze the data. Include screenshots of the intermediate calculations you performed.

Data Exploratory Analysis. I will create a scatter plot of the two (2) features showing differences between Churn=True and Churn=False customers. I will plot the new customer in the same plot to see where the new and existing customers are similar. We will then see what we expect the classification results will yield in the end.

FIGURE 4-1.TRAINING DATA 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.

Ref: (1) Shmueli, G.+ (2020). Data Mining for Business Analytics: Concepts, Techniques and Applications in Python, Chapter 7.

In [27]:
```
# scatter plot using the plotDataset() helper function
xFeature = 'MonthlyCharge'
yFeature = 'Tenure'
target = 'Churn'
neighbors = []

fig, ax = plt.subplots()
fig.set_size_inches(8, 5)
plotDataset(ax, trainData, xFeature, yFeature, target, neighbors)
plotDataset(ax, testData, xFeature, yFeature, target, neighbors, showLabel=False, facecolo

# plot new customer as a Star
ax.scatter(newCustomer.MonthlyCharge, newCustomer.Tenure, marker='*',
           label='New customer', color='red', s=270)

title = 'Scatter Plot'
plt.title(title)
plt.xlabel(xFeature)
plt.ylabel(yFeature)

# configure legend
handles, labels = ax.get_legend_handles_labels()
patch = mpatches.Patch(color='grey', label='Manual Label')
handles.append(patch)
plt.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, -0.15),
           fancybox=True, shadow=True, ncol=5)

# add customer data text
plt.gcf().text(0, -.4, newCustomer.to_string(), fontsize=14)

# add filename
f = getFilename(title, sect='d2',
                subfolder='figures', caption='4 1') # getFilename using helper
plt.gcf().text(0, -.2, f, fontsize=14)

# save file
fig.savefig(f, dpi=150, bbox_inches='tight')
```

FIGURES/D209_TASK1_D2_FIG_4_1_SCATTER_PLOT.PNG

	Tenure	MonthlyCharge	zTenure	zMonthlyCharge
0	1.0	170.0	0.0	0.0

Create and Train the Model. Use KNeighborsClassifier object's fit method, which loads the sample training set (X_train) and target training set (y_train) into the estimator:

In [28]:
```
# find k-nearest neighbors
knn = KNeighborsClassifier()
knn.fit(X=X_train, y=y_train['Churn'])
```

Out[28]:
```
KNeighborsClassifier()
```

In [29]:
```
predicted = knn.predict(X=X_test)
```

In [30]:
```
observed = y_test['Churn']
```

In [31]:
```
wrong = [(p,e) for (p,e) in zip(predicted, observed) if p!=e]
```

In [32]:
```
len(wrong)
```

Out[32]:
```
523
```

523 out of 3,000 were incorrectly predicted, that is 0.1743 or 17.4%.

In [33]:
```
knn.score(X_test, y_test)
```

Out[33]:
```
0.8256666666666667
```

In [34]:
```
# hyperparamter tuning (Ref: Deitel (2020))
for k in range(1,40,2):
    kfold = KFold(n_splits=10, random_state=11, shuffle=True)
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(estimator=knn,
            X=X_train, y=y_train['Churn'], cv=kfold)
    print(f'k={k:<2}; mean accuracy={scores.mean():.2%};')
```

```
k=1 ; mean accuracy=78.53%;
k=3 ; mean accuracy=81.49%;
k=5 ; mean accuracy=81.53%;
k=7 ; mean accuracy=82.30%;
k=9 ; mean accuracy=82.39%;
k=11; mean accuracy=82.64%;
k=13; mean accuracy=82.69%;
k=15; mean accuracy=82.97%;
k=17; mean accuracy=83.06%;
k=19; mean accuracy=82.96%;
k=21; mean accuracy=83.26%;
k=23; mean accuracy=83.21%;
k=25; mean accuracy=83.24%;
k=27; mean accuracy=83.19%;
k=29; mean accuracy=83.33%;
k=31; mean accuracy=83.29%;
k=33; mean accuracy=83.27%;
k=35; mean accuracy=83.30%;
k=37; mean accuracy=83.57%;
k=39; mean accuracy=83.66%;
```

Hyperparameter Tuning. From the tuning data, it appears that anything more than k=7 will give 82% or better accuracy, so I will use k=7 for the prediction.

In [35]:
```
# scale new Customer data
newCustomer['zMonthlyCharge'] = (newCustomer['MonthlyCharge'] - cleanData['MonthlyCharge']
newCustomer['zTenure'] =  (newCustomer['Tenure'] - cleanData['Tenure'].mean() ) / cleanDat
newCustomerNorm = newCustomer[['zTenure','zMonthlyCharge']]
newCustomerNorm
```

Out[35]:

	zTenure	zMonthlyCharge
0	-1.268	-0.061

TABLE 4-7. K-NEAREST "TRAINING" NEIGHBORS

In [36]:
```
# use NearestNeighbors from skikit-learn to compute knn
k=7 # determined from the hyperparameter tuning above
knn = NearestNeighbors(n_neighbors=k)
knn.fit(trainData.iloc[:,3:5])
distances, indices = knn.kneighbors(newCustomerNorm)
training_neighbors = trainData.iloc[indices[0],:]

# provide training neighbors
d = training_neighbors
f = getFilename('TRAINING NEIGHBORS', sect='D2',
        subfolder='tables', caption='4 7', ftype='CSV')
d.to_csv(f, index=True, header=True)
display(d)
print('Table saved to: {}'.format(f))
```

	Churn	Tenure	MonthlyCharge	zTenure	zMonthlyCharge
3009	False	1.125	169.993	-1.263	-0.061
2241	True	1.198	169.938	-1.260	-0.063
557	True	1.411	169.938	-1.252	-0.063
2855	True	1.462	169.945	-1.250	-0.062
53	True	1.553	169.945	-1.247	-0.062
449	False	1.715	169.993	-1.241	-0.061
3503	False	1.740	169.993	-1.240	-0.061

```
Table saved to: TABLES/D209_TASK1_D2_TAB_4_7_TRAINING_NEIGHBORS.CSV
```

Final Prediction. Calculate final prediction using the complete set of scaled data. Select a value for k from the figure above, let's select k=7 which looks like it should have about 82% accurary. Create a list of the neighbors in order to include highlighted neighbors on the next plot.

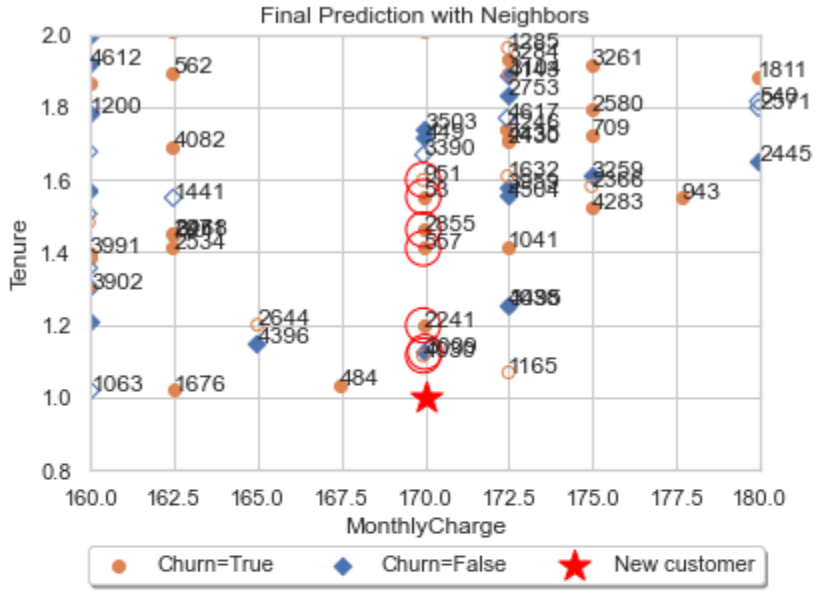
In [37]:
```
# retrain with full data.
k=7
knn = KNeighborsClassifier(n_neighbors=k).fit(X[['zTenure','zMonthlyCharge']], y['Churn'])
distances, indices = knn.kneighbors(newCustomerNorm)
print('Churn prediction (k={}) for \n{} is \n{}'.format(k,newCustomer,knn.predict(newCustor
df_neighbors = cleanData.iloc[indices[0],:]
neighbors = df_neighbors.index
neighbors = neighbors.to_list()
print(df_neighbors)
```

```
Churn prediction (k=7) for
   Tenure  MonthlyCharge  zTenure  zMonthlyCharge
0     1.0          170.0   -1.268          -0.061 is
[ True]
      Churn  Tenure  MonthlyCharge  zTenure  zMonthlyCharge
4030   True   1.115        169.938   -1.264          -0.063
3009  False   1.125        169.993   -1.263          -0.061
2241   True   1.198        169.938   -1.260          -0.063
557    True   1.411        169.938   -1.252          -0.063
2855   True   1.462        169.945   -1.250          -0.062
53     True   1.553        169.945   -1.247          -0.062
951    True   1.600        169.938   -1.245          -0.063
```

FIGURE 4-3. FINAL CLASSIFICATION OF NEW CUSTOMER WITH NEIGHBORS (RED CIRCLES) USED TO CLASSIFY WITH THE NEIGHBOR DATA SORTED BY DISTANCE FROM NEW CUSTOMER

Include data table using .to_string() method. Adjust plot so that text does not get cut off at bottom. Ref: (1) https://stackabuse.com/how-to-iterate-over-rows-in-a-pandas-dataframe/, (2) https://www.youtube.com/watch?v=C8MT-A7Mvk4&ab_channel=KimberlyFessel

In [38]:
```
# scatter plot using the plotDataset() helper function
xFeature = 'MonthlyCharge'
yFeature = 'Tenure'
fig, ax = plt.subplots()
plotDataset(ax, trainData, xFeature, yFeature, target, neighbors)
plotDataset(ax, testData, xFeature, yFeature, target, neighbors, showLabel=False, facecolor

# plot new customer as a Star
ax.scatter(newCustomer.MonthlyCharge, newCustomer.Tenure, marker='*',
           label='New customer', color='red', s=270)

# highlight neighbors with red circles
if len(neighbors) > 0:
    for n in neighbors:
        point = cleanData.iloc[n]
        ax.scatter(point.MonthlyCharge, point.Tenure, marker='o',
                   color='red', s=300, facecolors='none')

title = 'Final Prediction with Neighbors'
plt.title(title)
plt.xlabel(xFeature)
plt.ylabel(yFeature)

# set axis limits centered around the new customer
left = float(newCustomer.MonthlyCharge) - 4
right = float(newCustomer.MonthlyCharge) + 4
top = float(newCustomer.Tenure) - 4
bottom = float(newCustomer.Tenure) + 3
ax.set_xlim(160,180)
ax.set_ylim(.8,2)

handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, -0.15),
          fancybox=True, shadow=True, ncol=5)
#plt.legend(loc="lower center", bbox_to_anchor=(0.5, -0.15), ncol= 2)
f = getFilename(title, sect='d2',
        subfolder='figures', caption='4 3') # getFilename using helper
plt.gcf().text(0, -.2, f, fontsize=14)

# loop through neighbors and include neighbor as table data
#for idx,n in enumerate(df_neighbors.iloc[:, 0:3]):
#    plt.gcf().text(0, -.5+(.05*idx), n, fontsize=10)
plt.gcf().text(0, -.7, df_neighbors.iloc[:, 0:3].to_string(), fontsize=14)

fig.savefig(f, dpi=150, bbox_inches='tight')
plt.show()
```



FIGURES/D209_TASK1_D2_FIG_4_3_FINAL_PREDICTION_WITH_NEIGHBORS.PNG

	Churn	Tenure	MonthlyCharge
4030	True	1.115	169.938
3009	False	1.125	169.993
2241	True	1.198	169.938
557	True	1.411	169.938
2855	True	1.462	169.945
53	True	1.553	169.945
951	True	1.600	169.938

Summary. The KNN model calculated the new customer as Churn=True, with 6 of 7 neighbors with Churn=True.

D3. Provide the code used to perform the classification analysis from part D2.

Code. All code and output is contained within this Jupyter notebook. The notebook file is called **D209_1_x.ipynb** and the associated PDF version is called **D209_1_x - Jupyter Notebook.pdf**.

Part V: Data Summary and Implications

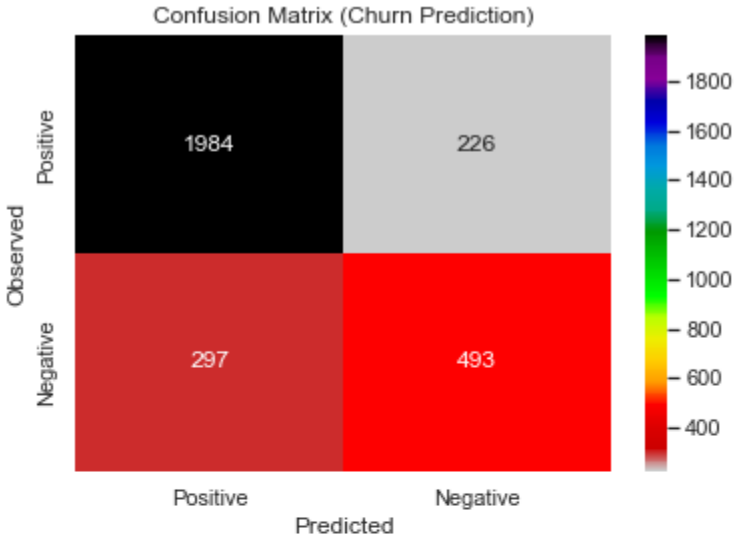
E1. Explain the accuracy and the area under the curve (AUC) of your classification model.

Confusion and Classification Report. Look at confusion and classification report to determine overall accuracy of the KNN model.

FIGURE 5-1. CONFUSION MATRIX AND METRICS

Ref: (1) https://classeval.wordpress.com/introduction/basic-evaluation-measures/, (2) https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

In [39]:
```
# confusion matrix
confusion = confusion_matrix(y_true=observed, y_pred=predicted)

# create plot
fig, ax = plt.subplots()
ax = sns.heatmap(confusion, annot=True,
            cmap='nipy_spectral_r', fmt='d')
title = 'Confusion Matrix (Churn Prediction)'
plt.title(title)
ax.set_xlabel('Predicted');
ax.set_ylabel('Observed');
ax.xaxis.set_ticklabels(['Positive', 'Negative']);
ax.yaxis.set_ticklabels(['Positive', 'Negative']);

# add filename
f = getFilename(title, sect='E1',
                subfolder='figures', caption='5 1') # getFilename using helper
plt.gcf().text(0, -.1, f, fontsize=14)

# add measurements
TN, FP, FN, TP = confusion_matrix(y_true=observed, y_pred=predicted).ravel()
P = TP + FP
N = TN + FN
ERR = (FP + FN) / (TP + TN + FN + FP) # Error rate
ACC = (TP + TN) / (TP + TN + FN + FP) # Accuracy
SN = TP / (TP + FN) # Sensitivity
SP = TN / (TN + FP) # Specificity
PREC = TP / (TP + FP) # Precision
FPR = FP / (TN + FP) # False Positive Rate
COR = TP + TN

plt.gcf().text(0, -.3, 'Error rate (ERR): ' + str(ERR.round(3)), fontsize=14)
plt.gcf().text(0, -.4, 'Accuracy (ACC): ' + str(ACC.round(3)), fontsize=14)
plt.gcf().text(0, -.5, 'Sensitivity (SN): ' + str(SN.round(3)), fontsize=14)
plt.gcf().text(0, -.6, 'Specificity (SP): ' + str(SP.round(3)), fontsize=14)
plt.gcf().text(0, -.7, 'Precision (PREC): ' + str(PREC.round(3)), fontsize=14)
plt.gcf().text(0, -.8, 'False Positive Rate (FPR): ' + str(FPR.round(3)), fontsize=14)
plt.gcf().text(0, -.9, 'Correct Predictions (COR): ' + str(COR.round(3)), fontsize=14)


# save file
fig.savefig(f, dpi=150, bbox_inches='tight')
```



FIGURES/D209_TASK1_E1_FIG_5_1_CONFUSION_MATRIX_(CHURN_PREDICTION).PNG

Error rate (ERR): 0.174

Accuracy (ACC): 0.826

Sensitivity (SN): 0.624

Specificity (SP): 0.898

Precision (PREC): 0.686

False Positive Rate (FPR): 0.102

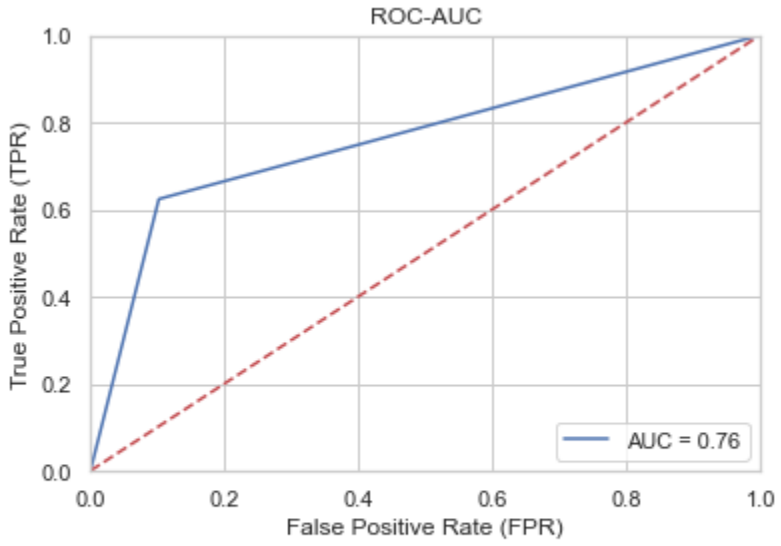
Correct Predictions (COR): 2477

Receiver Operation Characteristic (ROC) ad Area Under Curve (AUC). `Calculate and plot ROC and AOUC. Add custom text annotation to the plot.`

FIGURE 5-2.RECEIVER OPERATION CHARACTERISTIC (ROC)

Ref: (1) https://stackoverflow.com/questions/42435446/how-to-put-text-outside-python-plots (https://stackoverflow.com/questions/42435446/how-to-put-text-outside-python-plots), (2) https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html (https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html), (3) https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python (https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python) and (4) https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5)

In [40]:
```
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(observed, predicted)
auc = metrics.auc(fpr, tpr)

# method I: plt
fig, ax = plt.subplots()
title = 'ROC-AUC'
plt.title(title)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
f = getFilename(title, sect='e1',
    subfolder='figures', caption='5 2') # getFilename using helper
#plt.gcf().text(0, -.1, 'Area Under Curve (AUC): {:.2f}'.format(auc), fontsize=14)
plt.gcf().text(0, -.2, f, fontsize=14)
fig.savefig(f, dpi=150, bbox_inches='tight')
plt.show()
```



FIGURES/D209_TASK1_E1_FIG_5_2_ROC-AUC.PNG

E2. Discuss the results and implications of your classification analysis.

Summary. Looks like $1984 + 493 = 2477$ predictions on the diagonal were correct for an accuracy of about 82.56%. Analysis predicts 83% that the new customer is Churn=True, so, therefore, there is also the 17% chance that the new customer is actually Churn=False.

E3. Discuss one limitation of your data analysis.

Limitations. It occus to me that a new customer is new, that is, their **Tenure** will always be low compared to other existing customers. The KNN analysis will never make it to the higher **Tenure** numbers. Future study may look at other features instead such as **Income**, **Bandwidth_GB_Year**, or **Outage_sec_perweek** which may provide better insight.

E4. Recommend a course of action for the real-world organizational situation from part A1 based on your results and implications discussed in part E2.

Recommendations. Recommend additional data analysis to determine if there are other factors besides Tenure that might be used to better predict and classify customers.

Part VI: Demonstration

F. Provide a Panopto video recording that includes a demonstration of the functionality of the code used for the analysis and a summary of the programming environment.

Video. Panapto video was created and is located at: https://wgu.edu (https://wgu.edu)

G. Record the web sources used to acquire data or segments of third-party code to support the analysis. Ensure the web sources are reliable.

Configure Scrollbars. Disable scrollbars in notebook.

In [41]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

Disable Auto Scroll. Disable automatically scroll to bottom.

In [42]:
```
%%javascript
require("notebook/js/notebook").Notebook.prototype.scroll_to_bottom = function () {}
```

Toggle Notebook Warnings. Use the following code to toggle warning messages in the notebook. Another piece of code courtesy of stackoverflow (2021). https://stackoverflow.com/questions/9031783/hide-all-warnings-in-ipython (https://stackoverflow.com/questions/9031783/hide-all-warnings-in-ipython)

In [43]:
```
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[43]: To toggle on/off output_stderr, click here.

Terminal List Files. List all of the files from the current working directory. Ref: (1) Fessel, K. (2021). How to save a matplotlib figure and fix text cutting off || Matplotlib Tips (https://www.youtube.com/watch?v=C8MT-A7Mvk4&ab_channel=KimberlyFessel) Retrieved from https://www.youtube.com/watch?v=C8MT-A7Mvk4&ab_channel=KimberlyFessel

In [44]:
```
!ls
```

```
'ls' is not recognized as an internal or external command,
operable program or batch file.
```

In [45]:
```
!du -h *.*
```

```
'du' is not recognized as an internal or external command,
operable program or batch file.
```

List Installed Packages. `List of all installed PIP packages and the versions.`

Ref: (1) https://pip.pypa.io/en/stable/cli/pip_list/

In [46]:
```
!pip list
```

```
Package                           Version
--------------------------------- ---------
anyio                             3.3.4
argon2-cffi                       21.1.0
attrs                             21.2.0
Babel                             2.9.1
backcall                          0.2.0
bleach                            4.1.0
certifi                           2021.10.8
cffi                              1.15.0
charset-normalizer                2.0.7
colorama                          0.4.4
cycler                            0.10.0
debugpy                           1.5.1
decorator                         5.1.0
defusedxml                        0.7.1
entrypoints                       0.3
enum34                            1.1.10
idna                              3.3
ipykernel                         6.4.2
ipython                           7.28.0
ipython-genutils                  0.2.0
jedi                              0.18.0
Jinja2                            3.0.2
joblib                            1.1.0
json5                             0.9.6
jsonschema                        4.1.2
jupyter-client                    7.0.6
jupyter-contrib-core              0.3.3
jupyter-contrib-nbextensions      0.5.1
jupyter-core                      4.8.1
jupyter-highlight-selected-word   0.2.0
jupyter-latex-envs                1.4.6
jupyter-nbextensions-configurator 0.4.1
jupyter-server                    1.11.1
jupyterlab                        3.2.1
jupyterlab-pygments               0.1.2
jupyterlab-server                 2.8.2
kiwisolver                        1.3.2
lxml                              4.6.3
MarkupSafe                        2.0.1
matplotlib                        3.4.3
matplotlib-inline                 0.1.3
mistune                           0.8.4
nbclassic                         0.3.3
nbclient                          0.5.4
nbconvert                         6.2.0
nbformat                          5.1.3
nest-asyncio                      1.5.1
notebook                          6.4.5
numpy                             1.21.3
packaging                         21.0
pandas                            1.3.4
pandas-ml                         0.6.1
pandocfilters                     1.5.0
parso                             0.8.2
```

```
pickleshare                   0.7.5
Pillow                        8.4.0
pip                           21.3.1
prometheus-client             0.11.0
prompt-toolkit                3.0.21
pycparser                     2.20
Pygments                      2.10.0
pyparsing                     3.0.1
pyrsistent                    0.18.0
python-dateutil               2.8.2
pytz                          2021.3
pywin32                       302
pywinpty                      1.1.4
PyYAML                        6.0
pyzmq                         22.3.0
requests                      2.26.0
requests-unixsocket           0.2.0
scikit-learn                  1.0.1
scipy                         1.7.1
seaborn                       0.11.2
Send2Trash                    1.8.0
setuptools                    57.4.0
six                           1.16.0
sklearn                       0.0
sniffio                       1.2.0
terminado                     0.12.1
testpath                      0.5.0
threadpoolctl                 3.0.0
tornado                       6.1
traitlets                     5.1.1
urllib3                       1.26.7
wcwidth                       0.2.5
webencodings                  0.5.1
websocket-client              1.2.1
```

Update a specific package within notebook.
Ref: (1) https://stackoverflow.com/questions/54453219/why-can-i-see-pip-list-sklearn-but-not-in-jupyter-when-i-run-a-code

In [47]:
```
!python -m pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in p:\code\venv\lib\site-packages (1.0.1)
Requirement already satisfied: numpy>=1.14.6 in p:\code\venv\lib\site-packages (from scik
it-learn) (1.21.3)
Requirement already satisfied: joblib>=0.11 in p:\code\venv\lib\site-packages (from sciki
t-learn) (1.1.0)
Requirement already satisfied: scipy>=1.1.0 in p:\code\venv\lib\site-packages (from sciki
t-learn) (1.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in p:\code\venv\lib\site-packages (fr
om scikit-learn) (3.0.0)
```

Merget Two Dataframes. Code to merge two dataframes. Ref: (1) https://stackoverflow.com/questions/26265819/how-to-merge-a-series-and-dataframe

In [48]:
```
# merge X and y back together, for example
d = X.merge(y, left_index=True, right_index=True)
display(d.head())
```

	Tenure	MonthlyCharge	zTenure	zMonthlyCharge	Churn
0	6.796	172.456	-1.049	-0.004	False
1	1.157	242.633	-1.262	1.630	True
2	15.754	159.948	-0.710	-0.295	False
3	17.087	119.957	-0.659	-1.226	False
4	1.671	149.948	-1.242	-0.528	True

List .index() Function. The .index() method returns the index of the specified element in the list. Ref: (1) https://www.programiz.com/python-programming/methods/list/index

In [49]:
```
animals = ['cat', 'dog', 'rabbit', 'horse']
# get the index of 'dog'
index = animals.index('dog')
print(index)
```

```
1
```

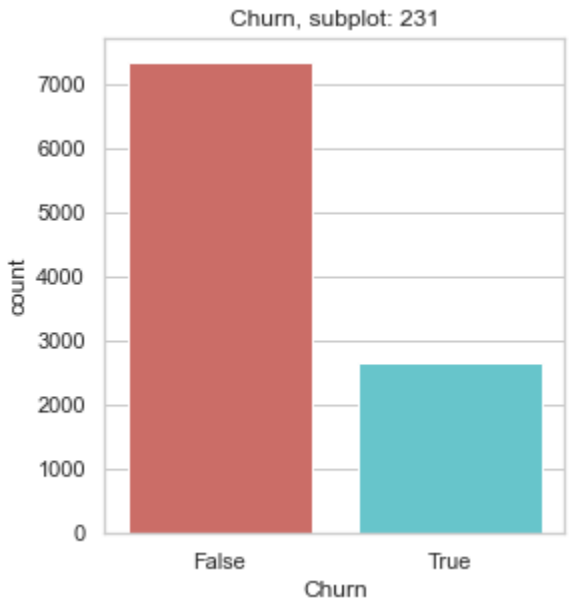
Row Index Names in Pandas. Code to get rows/index names in a Pandas dataframe. Ref: (1) https://www.geeksforgeeks.org/how-to-get-rows-index-names-in-pandas-dataframe/

In [50]:
```
# making data frame
data = cleanData

# calling head() method
# storing in new variable
data_top = data.head()

# iterating the columns
for row in data_top.index:
    print(row, end = " ")
```

```
0 1 2 3 4
```

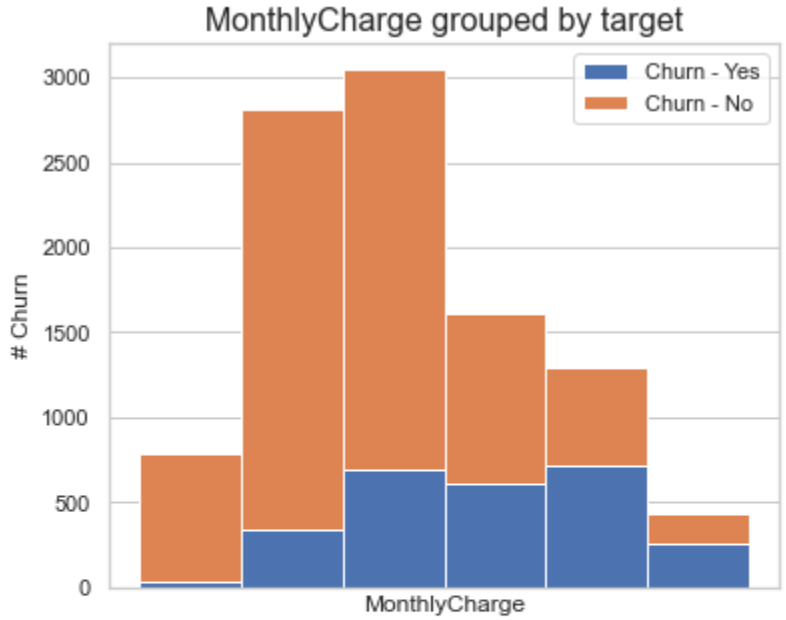
Tutorial Python Subplots. Tutorial: Python Subplots Ref: (1) https://www.kaggle.com/asimislam/tutorial-python-subplots

In [51]:
```
#  Categorical Data
heart_CAT = ['Churn']

#  Categorical Data
a = 2  # number of rows
b = 3  # number of columns
c = 1  # initialize plot counter

fig = plt.figure(figsize=(14,10))

for i in heart_CAT:
    plt.subplot(a, b, c)
    plt.title('{}, subplot: {}{}{}'.format(i, a, b, c))
    plt.xlabel(i)
    sns.countplot(x=i, data=cleanData, palette='hls')
    c = c + 1

plt.show()
```



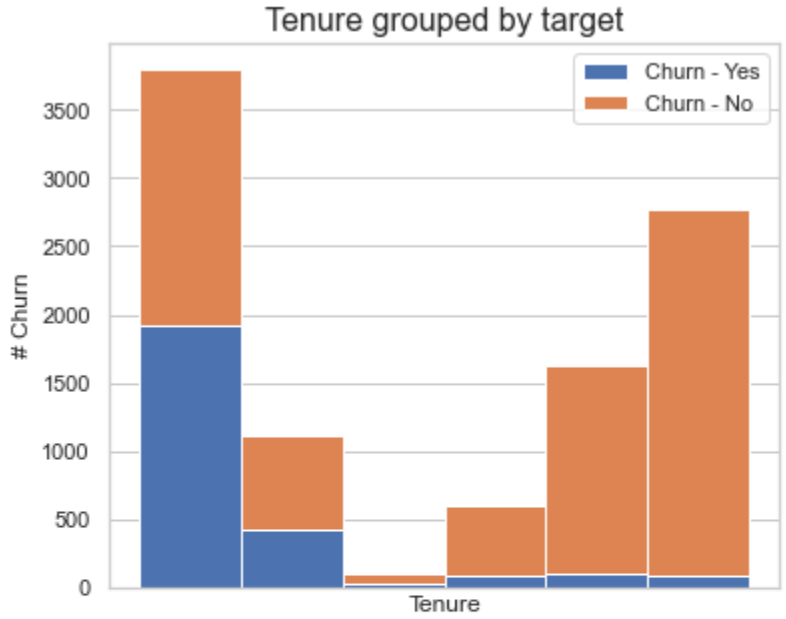
PASS FIG TO CUSTOM PLOT FUNCTION. A great way to do this is to pass a figure object to your code and have your function add an axis then return the updated figure. Here is an example: Ref: (1) https://stackoverflow.com/questions/43925337/matplotlib-returning-a-plot-object

In [52]:
```
def plot_hist_overlay(feature, fig, p, bins=8):

    # data
    df_yes = cleanData[cleanData.Churn==True][feature]
    df_no = cleanData[cleanData.Churn==False][feature]

    # plot stacked hist
    ax = f.add_subplot() # here is where you add the subplot to f
    plt.hist([df_yes,df_no], bins=bins, stacked=True)

    # add title
    plt.title(feature + ' grouped by target', size=16)

    # tick marks
    ax.set_xticks([])
    #ax.set_yticks([]) # use default

    # add axis labels
    plt.xlabel(feature)
    plt.ylabel('# Churn')

    # add legend
    ax.legend(['Churn - Yes','Churn - No'])

    return(f)

target = 'Churn'
features = ['MonthlyCharge','Tenure']
bins = 6
for idx,fea in enumerate(features):
    fig_size = (6,5)
    f = plt.figure(figsize=fig_size)
    f = plot_hist_overlay(fea, fig=f, p=idx+1, bins=bins)
    file = getFilename(fea, 'z1','fig 9 ' + str(idx+1)) # getFilename using helper
    plt.gcf().text(0.1, 0, file, fontsize=14)

    # data table
    b = pd.cut(cleanData[fea], bins=bins) # create bins (b) of numeric feature
    dt = pd.crosstab(cleanData[target], b)
    plt.gcf().text(0.1, -.4, dt.T.to_string(), fontsize=14)
    #print(dt.T)

    f.savefig(file, dpi=150, bbox_inches='tight')

#f = plot_hist_overlay('MonthlyCharge', fig=f, p=3)
#f = plot_hist_overlay('Tenure', fig=f, p=2)
```



FIGURES/D209_TASK1_Z1_FIG_FIG_9_1_MONTHLYCHARGE.PNG

Churn	False	True
MonthlyCharge		
(79.769, 115.009]	755	35
(115.009, 150.039]	2477	338
(150.039, 185.07]	2356	693
(185.07, 220.1]	1009	608
(220.1, 255.13]	579	715
(255.13, 290.16]	174	261



FIGURES/D209_TASK1_Z1_FIG_FIG_9_2_TENURE.PNG

Churn	False	True
Tenure		
(0.929, 12.833]	1876	1924
(12.833, 24.667]	692	418
(24.667, 36.5]	70	28
(36.5, 48.333]	511	83
(48.333, 60.166]	1526	105
(60.166, 71.999]	2675	92

Enabling Jupyter Notebook extensions. Ref: (1) https://tljh.jupyter.org/en/latest/howto/admin/enable-extensions.html

```
pip install jupyter_contrib_nbextensions
jupyter contrib nbextension install --sys-prefix
jupyter nbextension enable scratchpad/main --sys-prefix
jupyter nbextension list
```

How to Use HTML to Open a Link in a New Tab. Ref: (1)https://www.freecodecamp.org/news/how-to-use-html-to-open-link-in-new-tab/

```
<p>Check out <a href="https://www.freecodecamp.org/" target="_blank" rel="noopener
noreferrer">freeCodeCamp</a>.</p>
```

CSS Tutorial. This is a great resource for CSS code with many examples. Ref: (1)https://www.w3schools.com/css/default.asp

HTML Tutorial. This is a great resource for HTML code with many examples. Ref: (1)https://www.w3schools.com/html/default.asp

Inline Styles in HTML. Usually, CSS is written in a separate CSS file (with file extension .css) or in a 'style' tag inside of the 'head' tag, but there is a third place which is also valid. The third place you can write CSS is inside of an HTML tag, using the style attribute. When CSS is written using the style attribute, it’s called an “inline style”. In general, this is not considered a best practice. However, there are times when inline styles are the right (or only) choice. Ref: (1) https://www.codecademy.com/articles/html-inline-styles

H. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

Deitel, P. + (2020). Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud

Geron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools and Techniques to Build Intelligent Systems

Rite, S. (2018). Demystifying 'Confusion Matrix' Confusion
https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd

Robinson, S. (2021). K-Nearest Neighbors Algorithm in Python and Scikit-Learn
https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/

Sharma, A. (2021). K-Nearest Neighbors (KNN) on Customer Churn Data
https://medium.com/data-science-on-customer-churn-data/k-nearest-neighbors-knn-on-customer-churn-data-40e9b2bb9266

Shmueli, G. + (2020). Data Mining for Business Analytics: Concepts, Techniques, and Application in Python

In [ ]: