Matthew Blessing
Student ID: ***000507424***
NVM2 - Task 2, D209 - Data Mining

Performance Assessment Task 2: Predictive Analysis - NVM2

Part I: Research Question

A1.
In this analysis, we will attempt to answer the question of whether it is possible to utilize a ridge regression model to predict customer tenure.

A2.
Our goal will be to build a simple ridge regression model, analyzing its performance in order to determine if the model is accurate enough to accurately predict customer tenure based on our selected predictor variables.

Part II: Method Justification

B1.
In standard regression, a line is fitted to the data using an ordinary least squares (OLS) loss function approach, which seeks to minimize the sum of the squared residuals. This finds a line of best fit between the data points, with the slope being the regression coefficients. Ridge regression is a regularized regression model that applies penalties to large coefficients, be they positive or negative, and is useful when working with a large number of features (*Regression - Regularized Regression*, n.d.).

We will also be utilizing cross-validation folds in our model, which gives us a more accurate R-squared value by splitting the data into a specified number of folds. Each fold holds the fold as a test set, fits the model to the remaining four folds, and then predicts using the test set fold. This occurs the same way for each specified number of folds, giving us several different R-squared values that we can then average to get a more accurate R-squared value for our model (*Regression - Regularized Regression*, n.d.).

We can expect that our ridge regression model will attempt to predict how our response variable is affected by our explanatory variables. Our model will also output varying accuracy scores based on our selected alpha, as well as how many cross-validation folds we utilize (*Regression - Regularized Regression*, n.d.).

B2.
With our model being based on multiple linear regression, one assumption we can make about regression is that there should be homoscedasticity within our data. Homoscedasticity can be defined as the residuals, or error term, being constant in our model, and should not vary significantly as our predictor variables change (Statistics Solutions, 2021).

B3.
For this analysis, we will be utilizing Python within a Jupyter notebook, along with the following libraries:

- *Pandas*
- *NumPy*
- *Matplotlib*

- *Seaborn*
- *SciKit-Learn*

Python is an object-oriented programming language that is extremely popular for data science due to it being a powerful, easy to learn language that is extremely expandable with a large library of data science packages, such as NumPy, SciPy, Pandas, and Matplotlib (*Advantages of Learning Python for Data Science*, 2018). These libraries easily allow users to implement classification, regression, machine learning and more on chosen data sets.

Seaborn and Matplotlib are imported primarily for their powerful visualization tools to show how data within our dataset is distributed, allowing us to easily produce histograms, as well as bar charts, scatterplots, and box plots.

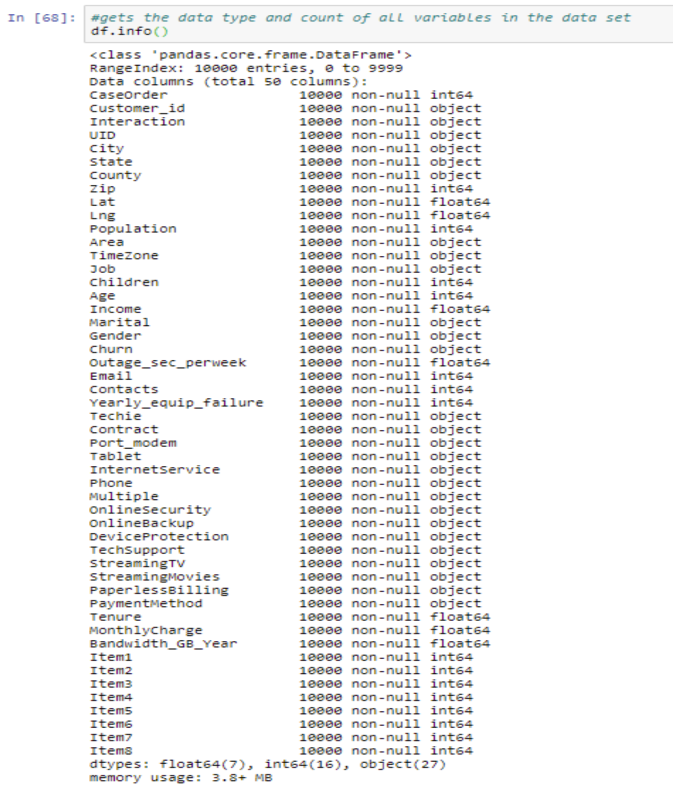
SciKit-Learn is a powerful machine learning library for Python, that offers several classification, regression, and clustering algorithms (*SciKit-Learn: Machine Learning in Python — Scikit-Learn 1.0 Documentation*, n.d.).

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text (*Project Jupyter*, n.d.).

Part III: Data Preparation

C1.
To ensure our model is accurate, we should ensure that all of our chosen variables have the same amount of observations.

We can do this simply by utilizing Pandas .info() function, that will return all of the variables in our data frame, alert us to any variables that contain missing (null) values, and inform us of each variable's data type:



```
In [68]: #gets the data type and count of all variables in the data set
         df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
CaseOrder                   10000 non-null int64
Customer_id                 10000 non-null object
Interaction                 10000 non-null object
UID                         10000 non-null object
City                        10000 non-null object
State                       10000 non-null object
County                      10000 non-null object
Zip                         10000 non-null int64
Lat                         10000 non-null float64
Lng                         10000 non-null float64
Population                  10000 non-null int64
Area                        10000 non-null object
TimeZone                    10000 non-null object
Job                         10000 non-null object
Children                    10000 non-null int64
Age                         10000 non-null int64
Income                      10000 non-null float64
Marital                     10000 non-null object
Gender                      10000 non-null object
Churn                       10000 non-null object
Outage_sec_perweek          10000 non-null float64
Email                       10000 non-null int64
Contacts                    10000 non-null int64
Yearly_equip_failure        10000 non-null int64
Techie                      10000 non-null object
Contract                    10000 non-null object
Port_modem                  10000 non-null object
Tablet                      10000 non-null object
InternetService             10000 non-null object
Phone                       10000 non-null object
Multiple                    10000 non-null object
OnlineSecurity              10000 non-null object
OnlineBackup                10000 non-null object
DeviceProtection            10000 non-null object
TechSupport                 10000 non-null object
StreamingTV                 10000 non-null object
StreamingMovies             10000 non-null object
PaperlessBilling            10000 non-null object
PaymentMethod               10000 non-null object
Tenure                      10000 non-null float64
MonthlyCharge               10000 non-null float64
Bandwidth_GB_Year           10000 non-null float64
Item1                       10000 non-null int64
Item2                       10000 non-null int64
Item3                       10000 non-null int64
Item4                       10000 non-null int64
Item5                       10000 non-null int64
Item6                       10000 non-null int64
Item7                       10000 non-null int64
Item8                       10000 non-null int64
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

Luckily, the dataset appears to be clean and we can see that all variables share the same number of observations (10,000), and none of the variables contain null values.

C2.
To select our variables, we would like to look at independent variables that may be able to impact our dependent variable, tenure. It would be useful for a business to be able to look at these selected variables and be able to predict a customer's tenure for forecasting and revenue purposes.

Our selected variables are as follows:

- Age - continuous variable
- Bandwidth_GB_Year - continuous variable
- Income - continuous variable
- MonthlyCharge - continuous variable
- Outage_sec_perweek - continuous variable
- Tenure - continuous variable
- Yearly_equip_failure - continuous variable

C3.
To prepare our data for our analysis, we first import the libraries we will be utilizing:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import statsmodels.api as sm
sns.set_style('darkgrid')

#sets the jupyter notebook window to take up 90% width of the browser
window
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
```

We then read in the dataset from a .csv to a Pandas dataframe:

```
df = pd.read_csv('churn_clean.csv')
```

We use Pandas .info() function to get an overview of our variables, number of observations, null values, and data types:

```
df.info()
```

We then drop any duplicate rows from the dataset:

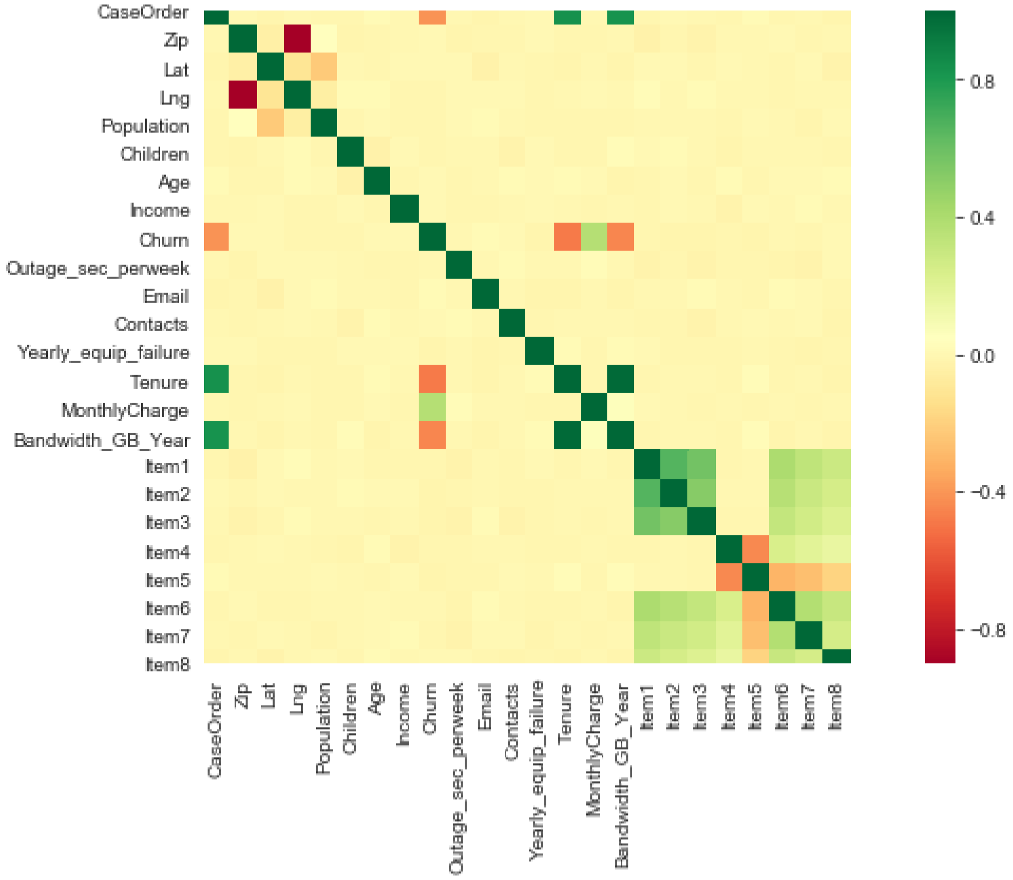
```
df.drop_duplicates()
```

We then double check the dataset for the sum of any null values per variable:

```
df.isnull().sum()
```

We then plot a heatmap to show positive and negative correlations within the dataset:

```
 plt.figure(figsize=(14, 6))
sns.heatmap(df.corr(), square=True, cmap='RdYlGn');
```

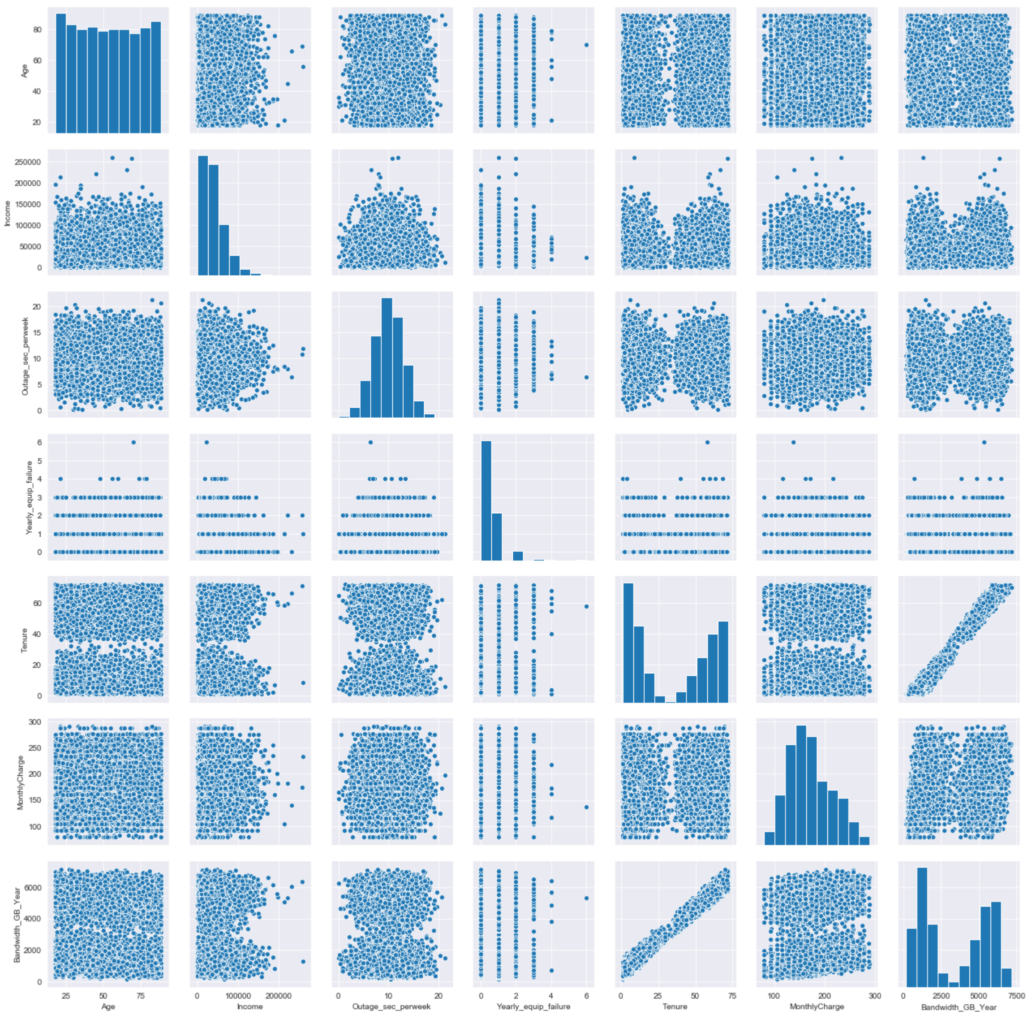


We then create a new data frame with only our selected variables:

```
df2 = df[['Age', 'Income', 'Outage_sec_perweek', 'Yearly_equip_failure',
'Tenure' ,'MonthlyCharge' ,'Bandwidth_GB_Year']]
```

We then create a correlation pairplot using Seaborn's .pairplot() function to quickly visualize relationships between variables:

```
sns.pairplot(df2);
```



We utilize Pandas .info() function to double check that the variables within our new data frame still have the same number of observations and no null values:

```
df2.info()
```

We then export our new data frame to a .csv:

```
df2.to_csv('D209 Task2.csv')
```

We also place our training and testing data into data frames and export those data frames as .csv files:

```
x_train_df = pd.DataFrame(X_train, columns = ['Age','Income',
'Outage_sec_perweek','Yearly_equip_failure','MonthlyCharge','Bandwidth_GB
_Year'])
```

```
x_test_df = pd.DataFrame(X_test, columns = ['Age','Income',
'Outage_sec_perweek','Yearly_equip_failure','MonthlyCharge','Bandwidth_GB
_Year'])
y_train_df = pd.DataFrame(y_train, columns = ['Tenure'])
y_test_df = pd.DataFrame(y_test, columns = ['Tenure'])

x_train_df.to_csv('xtrain.csv')
x_test_df.to_csv('xtest.csv')
y_train_df.to_csv('ytrain.csv')
y_test_df.to_csv('ytest.csv')
```

C4.
Please see the attached 'D209 Task2.csv'.

Part IV: Analysis

D1.
We begin our analysis by first defining our X and y variables for input into our model:

```
X = df2.drop('Tenure', axis=1).values
y = df2['Tenure'].values
```

After defining our X and y variables, we split our data into training and testing sets, defines the test size, and sets a random seed:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Please see the attached training and testing .csv files.

D2-D3.
In splitting the training and testing data, we ensure to specify the size of the test set, which we have set to 0.2, or 20% of our data split to the training set while 80% is split to the testing set. We set a random_state as well, so that we can refine our model, if we so choose, and still maintain the same random split, allowing us to see if we can improve our classification accuracy.

After we have defined our variables and split our data into training and testing sets, we then define a function to display a plot showing our R-squared values vs. different alphas (*Regularization II: Ridge | Python*, n.d.):

```
#defines the alpha vs cv plot
def display_plot(cv_scores, cv_scores_std):
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.plot(alpha_space, cv_scores)

    std_error = cv_scores_std / np.sqrt(10)
    ax.fill_between(alpha_space, cv_scores + std_error, cv_scores - 
std_error, alpha=0.2)
    ax.set_ylabel('CV Score +/- Std Error')
    ax.set_xlabel('Alpha')
    ax.axhline(np.max(cv_scores), linestyle='--', color='.5')
    ax.set_xlim([alpha_space[0], alpha_space[-1]])
    ax.set_xscale('log')
    plt.show()
```

We then create arrays for our alpha values and lists to store our ridge scores:

```
alpha_space = np.logspace(-4, 0, 50)
ridge_scores = []
ridge_scores_std = []
```

We create our ridge regressor:

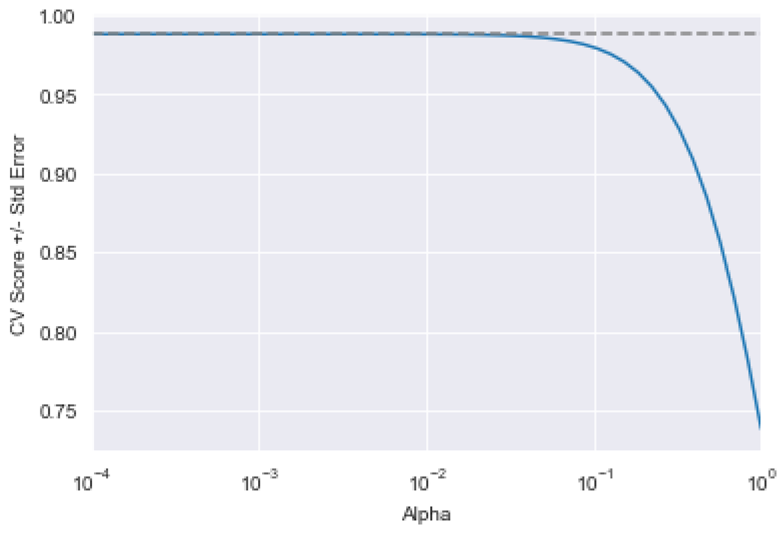
```
ridge = Ridge(normalize=True)
```

We then create a for loop to compute our cross-validation scores over the range of alphas and define the number of cross-validation folds to use, which we have set at 5 (*Regularization II: Ridge | Python*, n.d.):

```
for alpha in alpha_space:
    ridge.alpha = alpha
    ridge_cv_scores = cross_val_score(ridge, X_train, y_train, cv=5)
     ridge_scores.append(np.mean(ridge_cv_scores))
ridge_scores_std.append(np.std(ridge_cv_scores))
```

We then display the plot showing our ridge scores vs our alphas:

```
display_plot(ridge_scores, ridge_scores_std);
```



We then print our ridge scores and the average of our 5 ridge cross-validation scores:

```
print(ridge_cv_scores)
print("Avg 5-Fold CV Score: {}".format(np.mean(ridge_cv_scores)))

[0.73767556 0.73857773 0.74191177 0.73488347 0.7386032 ]
Avg 5-Fold CV Score: 0.7383303455524614
```

Finally, we fit our training data on the ridge regression model, use our ridge regression model to predict on the test data and calculate our MSE:

```
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
mean_squared_error(y_test, y_pred)

182.61555761654517
```

Part V: Data Summary and Implications

E1.
For our model, we decided on five cross-validation folds on our dataset. These five CV scores across a range of alphas returned the following R-squared values:

- 0.73767556
- 0.73857773
- 0.74191177
- 0.73488347
- 0.7386032

The average of these five scores was 0.7383303455524614, telling us that ~74% of the variance in our data can be explained by this model. This suggests a somewhat high level of correlation in the data, which would imply our model could potentially be useful for predicting customer tenure based on our independent variables.

However, when we also look at the mean squared error (MSE) of our model, we see a rather large 182.6 score. Generally speaking, the lower the MSE of a model, the better fit a model is for prediction given the data (Rowe, n.d.).

E2.
If we were to purely look at this model based on the ridge CV scores, we could say that the model is reasonably accurate at being able to predict tenure based on the independent variables we have selected. However, the very large MSE implies that there are large differences (errors) between the observed and the predicted values. This

high MSE also suggests that a ridge regression, or even a multiple linear regression model, simply may not be the best prediction model for our question, or for this dataset.

E3.
A limitation of this analysis is that we have only made a single attempt for a ridge regression model. While this analysis may prove somewhat useful, no model is perfect on a first attempt, and it would be beneficial to iterate on this model to improve the accuracy.

It is possible that we may need to limit our independent variables or change our dependent variable given that our exploration of correlation via a heatmap and via the pairplot show that there are very few linear relationships within the dataset. It is also a possibility that we are simply unable to create an accurate multiple linear regression model with the given dataset.

E4.
As a recommendation to stakeholders in the business for this situation, it would be recommended to pursue multiple iterations of this model in an attempt to increase the R-squared and CV scores, as well as the MSE. However, our MSE has implied that a regression model may not be the best fit to answer the business question. It would be recommended to look at other machine learning models to see if there is another model that can more accurately predict customer tenure.

Part VI: Demonstration
Please see attached Pantopto video.

References

Advantages of Learning Python for Data Science. (2018, March 16). BSD MAG. Retrieved October 14, 2021, from https://bsdmag.org/advantages-of-learning-python-for-data-science/

Multiple Linear Regression (MLR) Definition. (2021, March 7). Investopedia. Retrieved October 18, 2021, from https://www.investopedia.com/terms/m/mlr.asp

Project Jupyter. (n.d.). Project Jupyter. Retrieved October 14, 2021, from https://jupyter.org/

Regression - Regularized regression. (n.d.). DataCamp. Retrieved October 18, 2021, from https://campus.datacamp.com/courses/supervised-learning-with-scikit-learn/regression-2?ex=11

Regularization II: Ridge | Python. (n.d.). DataCamp. Retrieved October 18, 2021, from https://campus.datacamp.com/courses/supervised-learning-with-scikit-learn/regression-2?ex=13

Rowe, W. (n.d.). *Mean Square Error & R2 Score Clearly Explained*. BMC Blogs. Retrieved October 18, 2021, from https://www.bmc.com/blogs/mean-squared-error-r2-and-variance-in-regression-analysis/

SciKit-Learn: machine learning in Python — scikit-learn 1.0 documentation. (n.d.). SciKit-Learn. Retrieved October 14, 2021, from https://scikit-learn.org/stable/

Statistics Solutions. (2021, August 3). *Homoscedasticity*. Retrieved October 18, 2021, from https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/homoscedasticity/