# Standard Data Science Imports

import numpy as np

import pandas as pd

from pandas import DataFrame

# Visualization libraries

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

# Scikit-learn

import sklearn

from sklearn import datasets

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn import preprocessing

from sklearn.metrics import mean\_absolute\_error as MAE

from sklearn.metrics import mean\_squared\_error as MSE

# Import model, splitting method & metrics from sklearn

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

# Loading the data set into Pandas dataframe

churn\_df = pd.read\_csv(r'C:\Users\Hydraconix\Desktop\DATA\churn\_clean.csv')

# Examining fist five records of dataset

churn\_df.head()

# Viewing DataFrame descriptive information

churn\_df.info

# Getting an overview of descriptive stats

churn\_df.describe()

# Getting data types of features

churn\_df.dtypes

# Checking for null values

churn\_df.isnull()

# Renaming the last 8 Survey Columns for better description of variables

churn\_df.rename(columns = {'Item1' : 'TimelyResponse',

'Item2' : 'Fixes' ,

'Item3' : 'Replacements' ,

'Item4' : 'Reliability' ,

'Item5' : 'Options' ,

'Item6' : 'Respectfulness' ,

'Item7' : 'Courteous' ,

'Item8' : 'Listening'},

inplace=True)

# Converting ordinal categorical data into numeric variables

churn\_df['DummyInternetService'] = churn\_df.InternetService.map({'None' : 0, 'DSL' : 1, 'Fiber Optic' : 2})

churn\_df['DummyContract'] = churn\_df.Contract.map({'Month-to-month' : 0, 'One year' : 1, 'Two Year' : 2})

churn\_df['DummyGender'] = churn\_df.Gender.map({'Nonbinary' : 0, 'Male' : 1, 'Female' : 2})

# Histograms of continuous variables

churn\_df[['Age', 'Bandwidth\_GB\_Year', 'Children', 'Contacts', 'Email', 'Income', 'MonthlyCharge',

'Outage\_sec\_perweek', 'Tenure', 'Yearly\_equip\_failure','DummyInternetService', 'DummyContract','DummyGender']].hist()

plt.savefig('churn\_pyplot.jpg')

plt.tight\_layout()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=churn\_df['MonthlyCharge'], y=churn\_df['Churn'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=churn\_df['Outage\_sec\_perweek'], y=churn\_df['Churn'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=churn\_df['Tenure'], y=churn\_df['Churn'], color='green')

plt.show()

# I will now set the plot style to ggplot

plt.style.use('ggplot')

# Countplots of categorical variables

plt.figure()

sns.countplot(x='DeviceProtection', hue='Churn', data=churn\_df, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

plt.figure()

sns.countplot(x='Multiple', hue='Churn', data=churn\_df, palette ='RdBu')

plt.xticks([0,1],['No','Yes'])

plt.show()

plt.figure()

sns.countplot(x='Techie', hue='Churn', data=churn\_df, palette ='RdBu')

plt.xticks([0,1],['No','Yes'])

plt.show()

plt.figure()

sns.countplot(x='TechSupport', hue='Churn', data=churn\_df, palette ='RdBu')

plt.xticks([0,1],['No','Yes'])

plt.show()

# A scatter matrix of the discrete variables for high level overview of potential relationships & distributions

churn\_discrete = churn\_df[['Churn','TimelyResponse', 'Fixes', 'Replacements', 'Reliability', 'Options',

'Respectfulness', 'Courteous', 'Listening']]

pd.plotting.scatter\_matrix(churn\_discrete, figsize = [30, 30])

# An individual scatterplot for viewing relationship of key financial feature against target variable

sns.scatterplot(x = churn\_df['TimelyResponse'], y = churn\_df['Churn'], color='red')

plt.show()

sns.scatterplot(x = churn\_df['Fixes'], y = churn\_df['Churn'], color='red')

plt.show()

sns.scatterplot(x = churn\_df['Replacements'], y = churn\_df['Churn'], color='red')

plt.show()

# Converting binary categorical variables to numeric variables

churn\_df['DummyChurn'] = [1 if v == 'Yes' else 0 for v in churn\_df['Churn']]

churn\_df['DummyTechie'] = [1 if v == 'Yes' else 0 for v in churn\_df['Techie']]

churn\_df['DummyPort\_modem'] = [1 if v == 'Yes' else 0 for v in churn\_df['Port\_modem']]

churn\_df['DummyTablet'] = [1 if v == 'Yes' else 0 for v in churn\_df['Tablet']]

churn\_df['DummyPhone'] = [1 if v == 'Yes' else 0 for v in churn\_df['Phone']]

churn\_df['DummyMultiple'] = [1 if v == 'Yes' else 0 for v in churn\_df['Multiple']]

churn\_df['DummyOnlineSecurity'] = [1 if v == 'Yes' else 0 for v in churn\_df['OnlineSecurity']]

churn\_df['DummyOnlineBackup'] = [1 if v == 'Yes' else 0 for v in churn\_df['OnlineBackup']]

churn\_df['DummyDeviceProtection'] = [1 if v == 'Yes' else 0 for v in churn\_df['DeviceProtection']]

churn\_df['DummyTechSupport'] = [1 if v == 'Yes' else 0 for v in churn\_df['TechSupport']]

churn\_df['DummyStreamingTV'] = [1 if v == 'Yes' else 0 for v in churn\_df['StreamingTV']]

churn\_df['DummyStreamingMovies'] = [1 if v == 'Yes' else 0 for v in churn\_df['StreamingMovies']]

churn\_df['DummyPaperlessBilling'] = [1 if v == 'Yes' else 0 for v in churn\_df['PaperlessBilling']]

# Drop original categorical features from dataframe for further analysis

churn\_df = churn\_df.drop(columns=['Churn', 'Contract', 'DeviceProtection', 'Gender', 'InternetService',

'Multiple' , 'OnlineBackup', 'OnlineSecurity', 'PaperlessBilling',

'Phone', 'Port\_modem', 'StreamingMovies', 'StreamingTV', 'Tablet',

'Techie', 'TechSupport'])

# Remove the other less meaningful categorical variables from dataset to provide fully numerical dataframe for further analysis

churn\_df = churn\_df.drop(columns=['CaseOrder','Customer\_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population',

'Area', 'TimeZone', 'Job', 'Marital', 'PaymentMethod'])

# Provide a copy of the prepared data set

churn\_df.to\_csv(r'C:\Users\Hydraconix\Desktop\'churn\_prepared\_dt.csv')

# List features for analysis

features = (list(churn\_df.columns[:-1]))

print('Features for analysis include: \n', features)

# Re-read fully numerical prepared dataset

churn\_df = pd.read\_csv(r'C:\Users\Hydraconix\Desktop\'churn\_prepared\_dt.csv')

# Set predictor features & target variable

X = churn\_df.drop('DummyChurn', axis=1).values

y = churn\_df['DummyChurn'].values

# Create training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state =1)

# Instantiate Decision Tree Regressor model

dt = DecisionTreeRegressor(max\_depth = 8, min\_samples\_leaf = 0.1, random\_state = 1)

# Fit dataframe to Decision Tree Regressor model

dt.fit(X\_train, y\_train)

# Predict Outcomes from test set

y\_pred = dt.predict(X\_test)

# Compute test set MSE

mse\_dt = MSE(y\_test, y\_pred)

# Compute test set RMSE

rmse\_dt = mse\_dt\*\*(1/2)

# Print initial RMSE

print('Initial RMSE score Decision Tree Regressor model: {:.3f}'.format(rmse\_dt))

# Compute the coefficient of determination (R-squared)

scores = cross\_val\_score(dt, X, y, scoring='r2')

# Print R-squared value

print('Cross validation R-squared values: ', scores)

# Print Mean Squared Error

print('With a manual calculation, the Mean Squared Error: {:.3f} '.format(sum(abs(y\_test - y\_pred)\*\*2)/len(y\_pred)))

# Or

print('Using scikit-lean, the Mean Squared Error: {:.3f}'.format(MSE(y\_test, y\_pred)))

# Calculate & print the Root Mean Squared Error

RMSE = MSE(y\_test, y\_pred)\*\*(1/2)

# Print the Root Mean Squared Error

print('Root Mean Squared Error: {:.3f} '.format(RMSE))

# Get parameters of Decision Tree Regression model for cross validation

dt.get\_params()

# Define grid of hyperparameters

params\_dt = {'max\_depth': [4, 6, 8],

'min\_samples\_leaf': [0.1, 0.2],

'max\_features': ['log2', 'sqrt']}

# Re-intantiate Decision Tree Regressor for cross validation

dt = DecisionTreeRegressor()

# Instantiate GridSearch cross validation

dt\_cv = GridSearchCV(estimator=dt,

param\_grid=params\_dt,

scoring='neg\_mean\_squared\_error',

cv=5,

verbose=1,

n\_jobs=-1)

# Fit model to

dt\_cv.fit(X\_train, y\_train)

# Print best parameters

print('Best parameters for this Decision Tree Regressor model: {}'.format(dt\_cv.best\_params\_))

# # Generate model best score

print('Best score for this Decision Tree Regressor model: {:.3f}'.format(dt\_cv.best\_score\_))