# 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.neighbors import KNeighborsClassifier

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

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

from sklearn.model\_selection import GridSearchCV

from sklearn import metrics

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import roc\_auc\_score

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

# 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','DummyContract','DummyGender','DummyInternetService']].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()

# Setting 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\_log.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\_log.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)

# Export X\_train dataset

X\_train\_df = pd.DataFrame(X\_train)

X\_train\_df.to\_csv(r'C:\Users\Hydraconix\Desktop\X\_train.csv')

# Export X\_test dataset

X\_test\_df = pd.DataFrame(X\_test)

X\_test\_df.to\_csv(r'C:\Users\Hydraconix\Desktop\X\_test.csv')

# Export y\_train dataset

y\_train\_df = pd.DataFrame(y\_train)

y\_train\_df.to\_csv(r'C:\Users\Hydraconix\Desktop\y\_train.csv')

# Export y\_test dataset

y\_test\_df = pd.DataFrame(X\_test)

y\_test\_df.to\_csv(r'C:\Users\Hydraconix\Desktop\y\_test.csv')

# Initialize KNN model

knn = KNeighborsClassifier(n\_neighbors = 7)

# Fit data to KNN model

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

# Predict outcomes from test set

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

# Export y\_pred dataset

y\_pred\_df = pd.DataFrame(y\_pred)

y\_pred\_df.to\_csv(r'C:\Users\Hydraconix\Desktop\y\_pred.csv')

# Print initial accuracy score of KNN model

print('Initial accuracy score KNN model: ', accuracy\_score(y\_test, y\_pred))

# Compute classification metrics

print(classification\_report(y\_test, y\_pred))

# Set steps for pipeline object

steps = [('scaler', StandardScaler()),

('knn', KNeighborsClassifier())]

# Initiate pipeline

pipeline = Pipeline(steps)

# Split dataframe

X\_train\_scaled, X\_test\_scaled, y\_train\_scaled, y\_test\_scaled = train\_test\_split(X, y, test\_size = 0.2, random\_state = 1)

# Scale dateframe with pipeline object

knn\_scaled = pipeline.fit(X\_train\_scaled, y\_train\_scaled)

# Predict from scaled dataframe

y\_pred\_scaled = pipeline.predict(X\_test\_scaled)

# Print new accuracy score of scaled KNN model

print('New accuracy score of scaled KNN model: {:0.3f}'.format(accuracy\_score(y\_test\_scaled, y\_pred\_scaled)))

# Compute classification metrics after scaling

print(classification\_report(y\_test\_scaled, y\_pred\_scaled))

#Confusion\_matrix & generate results

cf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(cf\_matrix)

# Visual confusion matrix

group\_names = ['True Neg', 'False Pos', 'False Neg', 'True Pos']

group\_counts = ["{0:0.0f}".format(value) for value in cf\_matrix.flatten()]

group\_percentages = ["{0:.2%}".format(value) for value in cf\_matrix.flatten()/np.sum(cf\_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group\_names,group\_counts,group\_percentages)]

labels = np.asarray(labels).reshape(2,2)

sns.heatmap(cf\_matrix, annot=labels, fmt='', cmap='Blues')

# Set up parameters grid

param\_grid = {'n\_neighbors': np.arange(1, 50)}

# Re-initializing KNN for cross validation

knn = KNeighborsClassifier()

# Initializing GridSearch cross validation

knn\_cv = GridSearchCV(knn , param\_grid, cv=5)

# Fit model to

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

# Print best parameters

print('Best parameters for this KNN model: {}'.format(knn\_cv.best\_params\_))

# Generate model best score

print('Best score for this KNN model: {:.3f}'.format(knn\_cv.best\_score\_))

# Fit it to the data

knn\_cv.fit(X, y)

# Compute predicted probabilities: y\_pred\_prob

y\_pred\_prob = knn\_cv.predict\_proba(X\_test)[:,1]

# Compute and print AUC score

print("The Area under curve (AUC) on validation dataset is: {:.4f}".format(roc\_auc\_score(y\_test, y\_pred\_prob)))

# Compute cross-validated AUC scores: cv\_auc

cv\_auc = cross\_val\_score(knn\_cv, X, y, cv=5, scoring='roc\_auc')

# Print list of AUC scores

print("AUC scores computed using 5-fold cross-validation: {}".format(cv\_auc))