##Negative Binomial Regression model and testing its predictions

##Author : Dr UPASNA SRIVASTAVA

import pandas as pd

from patsy import dmatrices

import numpy as np

import statsmodels.api as sm

import statsmodels.formula.api as smf

import matplotlib.pyplot as plt

#create a pandas DataFrame for the counts data set

df = pd.read\_csv('nyc\_bb\_bicyclist\_counts.csv', header=0, infer\_datetime\_format=True, parse\_dates=[0], index\_col=[0])

#add a few derived regression variables to the X matrix

ds = df.index.to\_series()

df['MONTH'] = ds.dt.month

df['DAY\_OF\_WEEK'] = ds.dt.dayofweek

df['DAY'] = ds.dt.day

#create the training and testing data sets

mask = np.random.rand(len(df)) < 0.8

df\_train = df[mask]

df\_test = df[~mask]

print('Training data set length='+str(len(df\_train)))

print('Testing data set length='+str(len(df\_test)))

#Setup the regression expression in patsy notation. We are telling patsy that BB\_COUNT is our dependent variable and it depends on the regression variables: DAY, DAY\_OF\_WEEK, MONTH, HIGH\_T, LOW\_T and PRECIP

expr = """BB\_COUNT ~ DAY + DAY\_OF\_WEEK + MONTH + HIGH\_T + LOW\_T + PRECIP"""

#Set up the X and y matrices for the training and testing data sets

y\_train, X\_train = dmatrices(expr, df\_train, return\_type='dataframe')

y\_test, X\_test = dmatrices(expr, df\_test, return\_type='dataframe')

#Using the statsmodels GLM class, train the Poisson regression model on the training data set

poisson\_training\_results = sm.GLM(y\_train, X\_train, family=sm.families.Poisson()).fit()

#print out the training summary

print(poisson\_training\_results.summary())

#print out the fitted rate vector

print(poisson\_training\_results.mu)

#Add the λ vector as a new column called 'BB\_LAMBDA' to the Data Frame of the training data set

df\_train['BB\_LAMBDA'] = poisson\_training\_results.mu

#add a derived column called 'AUX\_OLS\_DEP' to the pandas Data Frame. This new column will store the values of the dependent variable of the OLS regression

df\_train['AUX\_OLS\_DEP'] = df\_train.apply(lambda x: ((x['BB\_COUNT'] – x['BB\_LAMBDA'])\*\*2 – x['BB\_LAMBDA']) / x['BB\_LAMBDA'], axis=1)

#use patsy to form the model specification for the OLSR

ols\_expr = """AUX\_OLS\_DEP ~ BB\_LAMBDA – 1"""

#Configure and fit the OLSR model

aux\_olsr\_results = smf.ols(ols\_expr, df\_train).fit()

#Print the regression params

print(aux\_olsr\_results.params)

#train the NB2 model on the training data set

nb2\_training\_results = sm.GLM(y\_train, X\_train,family=sm.families.NegativeBinomial(alpha=aux\_olsr\_results.params[0])).fit()

#print the training summary

print(nb2\_training\_results.summary())

#make some predictions using our trained NB2 model

nb2\_predictions = nb2\_training\_results.get\_prediction(X\_test)

#print out the predictions

predictions\_summary\_frame = nb2\_predictions.summary\_frame()

print(predictions\_summary\_frame)

#plot the predicted counts versus the actual counts for the test data

predicted\_counts=predictions\_summary\_frame['mean']

actual\_counts = y\_test['BB\_COUNT']

fig = plt.figure()

fig.suptitle('Predicted versus actual bicyclist counts on the Brooklyn bridge')

predicted, = plt.plot(X\_test.index, predicted\_counts, 'go-', label='Predicted counts')

actual, = plt.plot(X\_test.index, actual\_counts, 'ro-', label='Actual counts')

plt.legend(handles=[predicted, actual])

plt.show()