

**ANL252**

**Python for Data Analytics**

# **End-of-Course Assessment**

**July 2020 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Tong Yew Tiong Alex** | **H0914367** |

**Tutorial Group: ­­­­­­­­­ T 09**

**Instructor’s Name: Dr. Munish Kumar**

**Submission Date: 11/09/2021**

|  |
| --- |
| 1ai) |
| # Import numpy and pandas to facilitate on the programming task.  # Create DataFrame "ship" and read "ship.csv", indicate the 6 observations where MS and Y are "." as missing values with na\_values.  # Create DataFrame "nan\_value" to see the replacement of "." with “na\_value”. "nan\_value" will be used later in (1aiv) as reference to replace NaN value.  import numpy as np  import pandas as pd  ship = pd.read\_csv("ship.csv", na\_values = ".")  ship\_nan = pd.DataFrame(ship, columns=["T", "A", "P", "MS", "Y"])  nan\_values = ship\_nan[ship\_nan.isna().any(axis=1)]  display(nan\_values) |
| Output 1ai: |
|  |
|  |
| 1aii) |
| # Rename the variable names to better describe the nature of the variables: “T” to “types”, “A” to "c\_years", “P” to "o\_periods", “MS” to "s\_months", “Y” to "incidents".  names = ["T", "A", "P", "MS", "Y"]  ship\_rename = ship  ship\_rename.rename({"T":"types", "A":"c\_years", "P":"o\_periods", "MS":"s\_months", "Y":"incidents"}, axis=1, inplace=True)  ship\_rename.head() |

|  |
| --- |
| Output 1aii: |
|  |
|  |
| 1aiii) |
| # Finding the average service months and incidents, by types and operation periods, and rounded to the nearest integers. Store the resulting table to an object named "shipgroup".  import pickle  shipgroup = ship\_rename.groupby(['types', 'o\_periods'])[['s\_months', 'incidents']].mean().round()  with open('shipgroup', 'wb') as file:  pickle.dump(shipgroup, file)  shipgroup = pickle.load(open("shipgroup", "rb"))  print(shipgroup) |
| Output 1aiii: |
|  |

|  |
| --- |
| 1aiv) |
| # Using "nan\_values" from (1ai) as index to replace NaN values for "ship\_rename". The corresponding values to replace missing values are chosen from (1aiii) with reference from "shipgroup" based on “types” and “o\_period” using “.loc”.  indices = [6, 14, 22, 30, 33, 38]  ship\_rename.loc[indices,'s\_months'] = (911, 26852, 914, 296, 1047, 664)  ship\_rename.loc[indices,'incidents'] = (3, 36, 2, 1, 7, 4)  print(ship\_rename) |
| Output 1aiv: |
|  |

|  |
| --- |
| Output 1aiv: |
|  |
|  |
| 1av) |
| # From “ship\_rename” DataFrame will drop columns "c\_years", "o\_periods", "s\_months", keeping "incidents" as “Y” DataFrame.  names = ["c\_years", "o\_periods", "s\_months","incidents"]  Y = pd.DataFrame(ship\_rename, columns=names).drop(["c\_years", "o\_periods", "s\_months"], axis = 1)  Y.head() |
| Ouput 1av: |
|  |

|  |
| --- |
| 1bi) |
| # Checking the data type before converting  ship\_rename.info() |
| Output1bi: |
|  |
| 1bi) |
| # Converting to categorical variables for "types", "c\_years", and "o\_periods" and checking datatype after conversion  ship\_convert = ship\_rename.astype({"types":"category", "c\_years":"category", "o\_periods":"category"})  ship\_convert.info() |
| Output1bi: |
|  |

|  |
| --- |
| 1bii) |
| # Convert all categorical variables to dummy variable  X = pd.get\_dummies(ship\_convert, columns=["types", "c\_years", "o\_periods"])  X.head() |
| Output1bii: |
|  |
|  |
| 1biii) |
| # Performing a log-transformation for “s\_months” and name the transformed variable "log\_s\_months", and that performed transformation be updated to both DataFrames "X" and "ship"  X["log\_s\_months"] = np.log(X["s\_months"])  ship = X  ship.head() |

|  |
| --- |
| Output 1biii: |
|  |
|  |
| 1c) |
| Normally to evaluate the predictive power of the model, the DataFrame is split into training and testing datasets. From how the question was set, with the progression from (1a) to (1b). We are going to conduct a generalised linear equation with the dependent variable "incidents" as "Y", and independent variable "log\_s\_months", "types", "c\_years" and "o\_periods" as "X".  When the DataFrame is split, the test data is "removed" from the original dataset, leaving lesser data to train. For example, a dataset of 40, a train test\_size = 0.2 will mean 32 training and 8 testing sets. In our example, it seems like ships from "type\_2" had both higher "log\_s\_months" and "higher incidents". In total there are 12 "X" sub-categories after splitting the original 4 “X” variables into dummy subgroups: "type\_1 to type\_5", "c\_years\_1 to c\_years\_4" and "o\_period\_1 and o\_period\_2". It creates a challenge when the test data is selected either at randon or when it is systematic. Removing just 1 set of data from each sub-category will leave only 28 dataset to train. It may leave too few data in each subgroup or categories in "X" to form an accuracy analysis on the model. Spliting in this case will make the result less than ideal, as the data left for training may not be reflective of the model and the outcome of may be skewed(dispersion problem). |

|  |
| --- |
| 1d) |
| # Save the prepare DataFrame "ship" as a new csv text fill called "ship\_prepared.csv"  ship.to\_csv("ship\_prepared.csv", index = False) |
| # Create database called "ship.db" and export the DataFrame to the database as tables.  # I am able to create "ship.db" as .sqlite file, however, an error occurs when calling out the file.  # When database name changed to "ship\_db", the table can be retrieved.  %reload\_ext sql  %sql sqlite://  from sqlalchemy import create\_engine  cnx = create\_engine(r"sqlite:////Users/striv/Desktop/Python/EOC/ship\_db.sqlite", echo = False)  ship.to\_sql("ship\_db", con=cnx, index = False)  %sql sqlite:////Users/striv/Desktop/Python/EOC/ship\_db.sqlite  %sql SELECT \* from ship\_db limit 5; |
| Output 1d: |
|  |

|  |
| --- |
| 2a) |
| Using the scikit-learn official website:  <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PoissonRegressor.html>  <https://scikit-learn.org/stable/modules/linear_model.html#generalized-linear-regression>  <https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>  The corresponding module, estimator, fit and predict functions, as well as their parameters. Will be as follows.  Module - It is the specific file or source we will be drawing to be used. Given that Poisson is a special case of the generalised linear models (Chapman & Hall, 1983) the corresponding module will be: "from sklearn import linear\_model"  Estimator - According to the website the estimator chosen will be a “regression” and the corresponding estimator used will be “linear\_model.PoissonRegressor()”. The Poisson estimator is a method used in statistics on probabilities.  Fit - Describes how well scikit uses the given estimator to find a best "fit" or training for the function. To match the Poisson regression with variables from "ship\_prepared". The corresponding ".fit(X,Y)" will be used  Predicted functions - Given the formula: log 𝔼(𝑌) = 𝛽0 + 𝛽1𝑋1 + 𝛽2𝑋2 + ⋯ + 𝛽𝑘𝑋𝑘. The predicted function to be used will be ".predit(X)", it is the coefficient of determination. Which decribes to relationship of the dependent and independent variables. For example 0.5“X1”, will mean a 10"Y" increment for every 5"X" of increment while holding other variables constant.  Parameters - From the official website scikit-learn. There are 3 paramenters: “X” which is the training data used to calculate or predict “Y” the target data, and “Sample weight”, which will be “0”. |

|  |
| --- |
| 2b) |
| # Cleaning the dataset for “X” using “.drop” for “s\_months” and “incidents”, and to analyse the DataFrame "X" and "Y".  X= X.drop(columns=["s\_months", "incidents"])  print(X.head(3))  print(Y.head(3))  print(X.shape, Y.shape) |
| Output 2b: |
|  |
| 2b) |
| # .ravel() was added after initial "fit" generates an error and python advise the use of it.  X = X.to\_numpy()  Y = Y.to\_numpy()  Y = Y.ravel()  Y = np.array(Y).astype(float) |

|  |
| --- |
| 2b) |
| # Create a program to fit a Poisson regression by following the instructions from the official website of scikit-learn.  from sklearn import linear\_model  eqn = linear\_model.PoissonRegressor()  module = eqn.fit(X, Y, sample\_weight = None)  accuracy = eqn.score(X, Y, sample\_weight = None)  coef\_determination = eqn.coef\_.round(3)  intercept = eqn.intercept\_.round(3)  print(module)  print(accuracy)  print(coef\_determination)  print(intercept) |
| Output 2b: |
|  |
| 2b) |
| # Generate a table to present the coefficients with the corresponding labels.  from astropy.table import Table  data\_rows = [(-3.495, 0.049, 0.087, -0.245, -0.067, 0.175, -0.246, 0.155, 0.177, -0.086, -0.145,  0.145, 0.709)]  t = Table(rows=data\_rows, names=('𝛽0','𝛽1', '𝛽2', '𝛽3', '𝛽4','𝛽5', '𝛽6', '𝛽7', '𝛽8', '𝛽9', '𝛽10', '𝛽11', '𝛽12'))  print(t) |
| Output2b: |
|  |

|  |
| --- |
| 2c) |
| # From (2a), use “.predict(X)”, find the predicted "Y" as “E\_Y” and compute "D".  # I've made an assumption here due to the change in the formula: {Ylog[Y/E(Y)] = 0, if "Y" = 0  E\_Y = eqn.predict(X)  D = []  for y in Y:  if (y==0):  D = 2\*(sum(-(y - E\_Y))).round(2)  else:  D = 2\*(sum(y\*np.log(y/E\_Y) - (y - E\_Y))).round(2)  print(D) |
| Output 2c: |
|  |