

**ANL252**

**PYTHON FOR DATA ANALYTICS**

# **End-of-Course Assessment**

**July 2021 Presentation**

**Submitted by: Glen Ong**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **ONG YEOW HWEE GLEN** | **W2110804** |

**Tutorial Group: ­­­­­­­­­­­­ T09**

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

**Submission Date: 13/09/2021**

**Question 1**

**Prerequisites for question 1 and question 2:**

**Script:**

# Importing prerequisite packages

import numpy as np

import pandas as pd

print("numpy package and pandas package imported!")

# Disabling warning from chained\_assignment to prevent redundant warning

pd.options.mode.chained\_assignment = None

print("Chained assignment has been set to disabled!")

1. (i)

**Script and the corresponding output:**

# Reading in raw ship dataset and display the entire dataset

pd.read\_csv("ship.csv")



# Reading in ship as a .csv with "." as missing values and storing it into a new dataframe "ship"

# Displaying the last 6 records of "ship" to ensure null values have been converted

ship = pd.read\_csv("ship.csv", na\_values = ".")

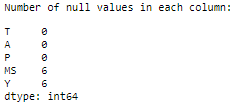
ship.tail(11)



# Displaying the number of null values in each column (should only be observed in column “MS” and “Y”

print("Number of null values in each column:")

ship.isnull().sum(axis = 0)



# Summary of the number of null values in ship (should have 6 null values in total)

print(f"There are a total of {ship.isnull().any(axis = 1).sum()} rows with null values.")



1. (ii)

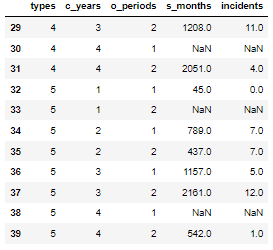
**Script and the corresponding output:**

# Renaming the respective column names to the required names and applying it to ship dataset

# Displaying last 11 records to ensure null values column names have been renamed accordingly

ship = ship.rename({"T": "types", "A": "c\_years", "P": "o\_periods", "MS": "s\_months", "Y": "incidents"}, axis=1)

ship.tail(11)



1. (iii)

**Script and the corresponding output:**

# Grouping the ship dataset by the types and operation period then calculating the mean of service months and mean of the number of incidents

# Storing the respective group mean values into a new dataframe "shipgroup"

# The mean values are rounded to the nearest integer with ".round(0)" and data type is converted into integer with ".astype(int)"

# Displaying "shipgroup" to ensure the group are grouped accordingly and the mean values for each group are produced

print("Mean of service months and mean of incidents by the ship type and operation period:")

shipgroup = ship.groupby(["types", "o\_periods"])[["s\_months", "incidents"]].mean().round(0).astype(int)

display(shipgroup)

Graphical user interface, table

Description automatically generated with medium confidence

1. (iv)

**Script and the corresponding output:**

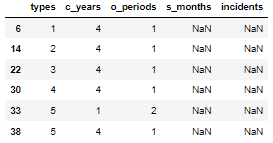
# Replacing missing values of service months and the number of incidents by their respective mean under the same type of group and operation period

# Storing the sliced output into a new dataframe "ship\_null\_sliced" (To be leveraged on later)

# Displaying the records with null values in service months and number of incidents ("ship\_null\_sliced")

ship\_null\_sliced = ship[ship[["s\_months", "incidents"]].isnull().any(axis = 1)]

display(ship\_null\_sliced)



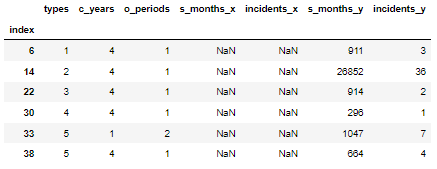
# Merging sliced data by leveraging grouped values of service months and incidents from ship group dataframe as a mapping table

# Storing the merged outcome into a new dataframe "ship\_merged" (To be leveraged on later)

# Diplaying the merged columns from "ship\_merged" to ensure the join was executed correctly

ship\_merged = ship\_null\_sliced.reset\_index().merge(shipgroup, on = ["types", "o\_periods"], how = "left").set\_index("index")

display(ship\_merged)



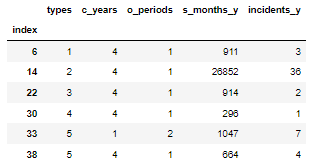
# Removing the previous null containing columns “s\_month\_x” and “incidents\_x” as it will be replaced with the new “s\_month\_y” and “incidents\_y” from ship group dataframe

# Storing the tranformed dataframe into a new dataframe "ship\_null\_dropped" (To be leveraged on later)

# Display the final merged dataset from "ship\_null\_dropped" to ensure the removal of null columns are executed correctly

ship\_null\_dropped = ship\_merged.drop(["s\_months\_x", "incidents\_x"], axis = 1)

display(ship\_null\_dropped)

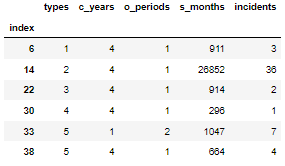


# Renaming columns containing the merged new values to standardize with the requirements and stored into a new dataframe "ship\_mapped" (To be leveraged on later)

# Displaying "ship\_mapped" to ensure respective field names have been renamed accordingly and correctly

ship\_mapped = ship\_null\_dropped.rename({"s\_months\_y": "s\_months", "incidents\_y": "incidents"}, axis=1)

display(ship\_mapped)



# Getting the original dataset with records containing null values in service months and incidents sliced away

# This is stored into a new dataframe "ship\_nonull\_sliced" (To be leveraged of later)

# Displaying the remaining dataset ()"ship\_nonull\_sliced") to ensure the records containing null in column s\_months and incidents are no longer in the new dataframe ("ship\_nonull\_sliced")

ship\_nonull\_sliced = ship[ship[["s\_months", "incidents"]].notnull().any(axis = 1)]

display(ship\_nonull\_sliced)



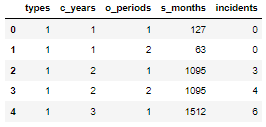
# Union the ship dataframe that does not contain null “ship\_nonull\_sliced” and the transformed records with null values replaced with the respective mean values “ship\_mapped” together

# Storing the unioned dataframe into a new dataframe "ship\_cleanedup" (To be leveraged on later)

# Displaying top 5 the records of "ship\_cleanedup" to dataset remains visibly untouched

ship\_cleanedup = pd.concat([ship\_nonull\_sliced, ship\_mapped], axis = 0).astype(int)

ship\_cleanedup.head()



# Checking if there are nulls contained in the dataframe after union

ship\_cleanedup.isnull().sum()

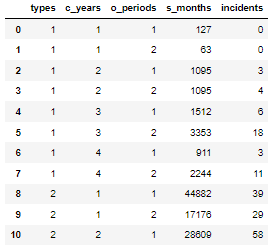


# Sorting the cleaned-up data so that it follows the original position (Ascending index number) and storing the cleaned-up values into the previous dataframe "ship" (To be leveraged on later)

# Displaying top 11 records of "ship" to ensure sorting is Executed correctly

ship = ship\_cleanedup.sort\_index(axis = 0)

ship.head(11)



# Displaying index as a list to check the index sorting order

ship.index



# Displaying the row count and column count to ensure all values are recorded

ship.shape



1. (v)

**Script and the corresponding output:**

# Storing the incident column from the cleaned-up and sorted dataframe into a new dataframe "Y"

# Displaying the top 5 records of dataframe "Y" to ensure the column has been selected correctly

Y = ship[["incidents"]]

Y.head()

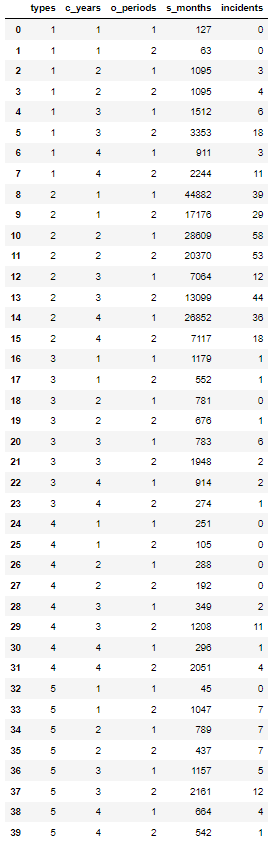


1. (i)

**Script and the corresponding output:**

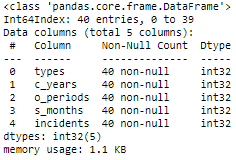
# Displaying the cleaned-up dataframe produced from part (a) for part (b) reference

display(ship)



# Displaying the information of the dataframe to check the current datatype of the fields

ship.info()

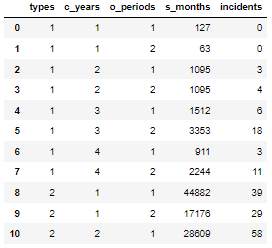


# Selecting the columns for data type transformation and transforming the data type into categorical with ".astype("category")"

# Displaying the transformed dataframe to ensure the values and columns are untouched

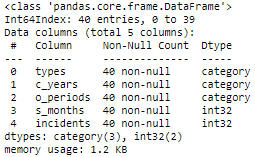
ship[["types", "c\_years", "o\_periods"]] = ship[["types", "c\_years", "o\_periods"]].astype("category")

ship.head(11)



# Displaying the information of the dataframe to check the new datatype of the dataframe

ship.info()



1. (ii)

**Script and the corresponding output:**

# Leveraging “get\_dummies” function from pandas to create dummy variables for all the values in the fields with a categorical data type

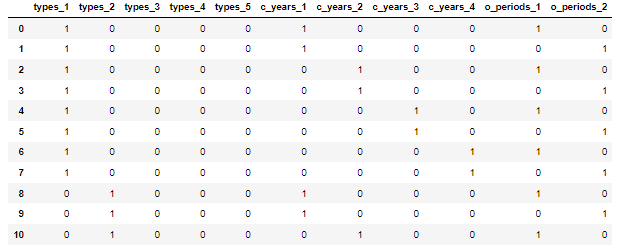
# Dropping the fields that are not categorical variables and storing it into a dataframe “X”

# Diplaying the top 11 records of the "X" to ensure only the dummy variables are stored

X = pd.get\_dummies(ship, columns = ship.loc[:,ship.dtypes == "category"].columns)

X = X.drop(["s\_months", "incidents"], axis = 1)

X.head(11)



1. (iii)

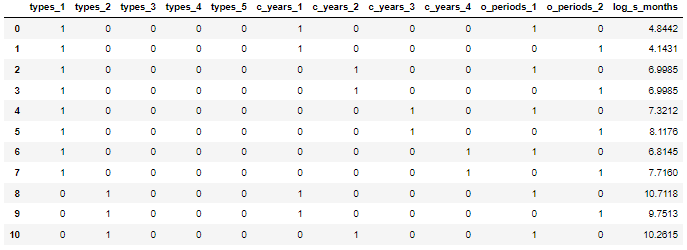
**Script and the corresponding output:**

# Performing a log transformation on the service months column from the original ship data and storing it as a new column "log\_s\_months" in dataframe "X"

# Displaying the top 11 record of "X" to ensure the new column "log\_s\_months" has been created with the respective values contained with the field

X["log\_s\_months"] = round(np.log(ship["s\_months"]),4)

X.head(11)

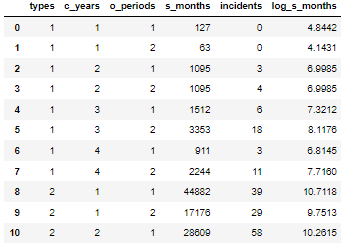


# Performing a log tranformation on the service months column from the original ship data and storing it as a new column "log\_s\_months" in the same dataframe "ship"

# Displaying the top 11 record of "ship" to ensure the new column "log\_s\_months" has been created with the respective values contained with the field

ship["log\_s\_months"] = round(np.log(ship["s\_months"]),4)

ship.head(11)



**Script and the corresponding output:**

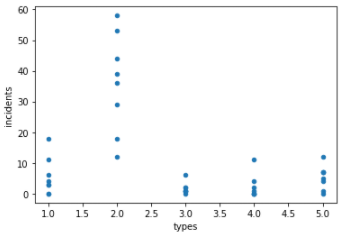
# Importing prerequisite modules

from matplotlib import pyplot as plt

# Plotting a scatter plot with the ship types and incidents columns as independent and dependent variable respectively

ship.plot(kind = "scatter", x = "types", y = "incidents")

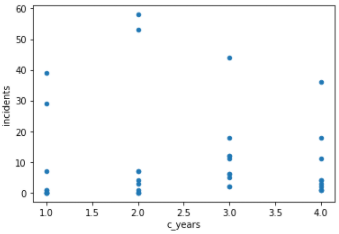
plt.show()



# Plotting a scatter plot with the c\_years and incidents columns as independent and dependent variable respectively

ship.plot(kind = "scatter", x = "c\_years", y = "incidents")

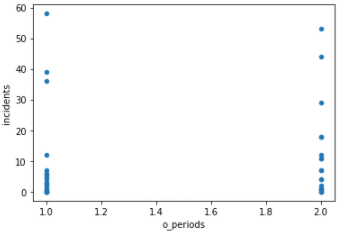
plt.show()



# Plotting a scatter plot with the o\_periods and incidents columns as independent and dependent variable respectively

ship.plot(kind = "scatter", x = "o\_periods", y = "incidents")

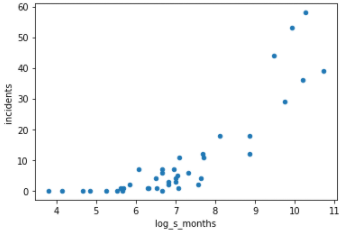
plt.show()



# Plotting a scatter plot with the log\_s\_months and incidents columns as independent and dependent variable respectively

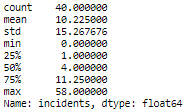
ship.plot(kind = "scatter", x = "log\_s\_months", y = "incidents")

plt.show()



# Summary Statistics of the new column "log\_s\_months"

ship["incidents"].describe()



# Variance of the new column "log\_s\_months"

ship["incidents"].var()



**Description:**

Train-test split refers to the splitting of dataset into two subsets to train and estimate the performance of the model. However, this function would only be appropriate for sufficiently large sample size to ensure all possible cases are covered for training the model.

Looking further into the ship dataset by leveraging on the matplotlib package to visualize the dataset, describe and variance functions from pandas for descriptive statistics. We can observe that within each scatter plot diagram, there are some outliers in each independent variable. This is especially evident in the relationship between the year of construction and the number of incidents. For example, we can observe that there is a cluster between 0 to 10 incidents at the construction year of 2(1965 to 1969). However, there are two outliers within the same year bin with between 50 to 60 incidents. We can also observe a sample size of 40, this sample size may not sufficient and would not be sensible to apply the train-test split function. This may potentially cause under-fitting in the training data and it would not be effective in evaluating the performance of the model.

Additionally, we can observe that the number of incidents has a high range with a minimum number of incidents of 0 and a maximum number of incidents 58. Furthermore, we may also observe a high variance of approximately 233 in the number of incidents. Therefore, the wide dispersion of the values in the target variable “incidents” and the small dataset may suggest that each specific set of independent variables is significant in building the model and the goodness-of-fit would vary significantly based on the specific sets of data used to train the model. Hence, to increase the effectiveness of the model in predicting the number of incidents, the entire dataset should be used to train the model.

**Script and the corresponding output:**

# Importing prerequisite module

import sqlite3

print("sqlite3 package imported!")

# Saving the ship dataframe into a .csv file "ship\_prepared"

# Letting user know if the file save has been executed

ship.to\_csv("ship\_prepared.csv", index = False)

print("ship dataset has been exported successfully")

# Creating a connection to a database "ship.db" that will be used to store the ship table

# Creating a cursor to enable SQL commands

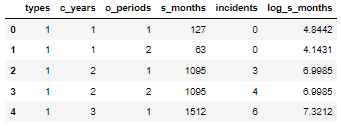
connection = sqlite3.connect("ship.db")

cursor = connection.cursor()

# Storing the prepared .csv file to "ship\_prepared" dataframe and displaying the top 5 records of the dataframe to ensure the correct table is used

ship\_prepared = pd.read\_csv("ship\_prepared.csv",)

ship\_prepared.head()



# Creating SQL table "ship\_table with the data from ship\_prepared dataframe"

ship\_prepared.to\_sql("ship\_table", connection, if\_exists="replace", index = False)

# Reading the table data with a dataframe to ensure data has been stored in correctly

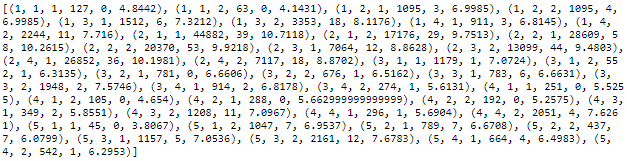
pd.read\_sql("SELECT \* FROM ship\_table", connection)



# Selecting all the data from the ship table and calling all the data in the table out

cursor.execute("SELECT \* FROM ship\_table;")

print(cursor.fetchall())



**Question 2**

With reference to the scikit-learn official website, the “linear models” module is selected as the generalized linear model with Poisson distribution can be accessed from there. Firstly, the Poisson regression estimator of the module is “clf = linear\_model.PoissonRegressor()” where the parameters are the alpha, the intercept, the maximum number of iterations, the tolerance for the stopping criteria, the option to reuse the values from the previous fit() to train the new model and the option to produce a logging output for the processes of the estimator. By default, the estimator would have a 1 multiplier in penalizing the generalized linear model, leverages on the intercept values, have a maximum number of iterations of 100, a tolerance of 1e-4, never reuses previous fit values and disabled the logging of outputs.

The package provides build in functions such to train and prediction the model with “fit(X,y)” and “predict(X)”. The fitting of the model requires 2 important variables, the independent variable as the “X” and the dependent variable as the ”y”. Additionally, we may determine the weights of the sample if required. Lastly, after the model is trained, the “predict(X)” function can be leveraged to predict the corresponding target variable values by stating the independent variable “X”

**Script and the corresponding output:**

# Importing prerequisite modules

from sklearn import linear\_model

print("linear\_model from sklearn imported!")

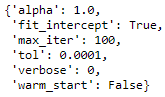
# Creating a linear model with poisson distribution and training the model

clf = linear\_model.PoissonRegressor()

clf = clf.fit(X, Y.values.ravel())

# Displaying the parameters used for estimating the model

clf.get\_params()



# Calculating the coefficiency of each independent variable to the target variable

# Storing the coefficiencies into a column of a dataframe and following the same header name as dataframe X

coef\_list = list([round(value, 4) for value in clf.coef\_])

ship\_coef = pd.DataFrame([coef\_list])

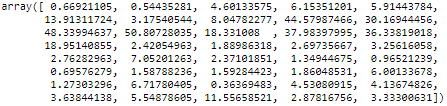
ship\_coef.columns = ["types\_1","types\_2","types\_3","types\_4","types\_5","c\_years\_1","c\_years\_2","c\_years\_3","c\_years\_4","o\_periods\_1","o\_periods\_2","log\_s\_months"]

display(ship\_coef)



# Predicting the number incident based on each record of x

clf.predict(X)



# Calculating the percentage of deviance

clf.score(X,Y.values.ravel())



**Script and the corresponding output:**

# Coverting Y values into a list and storing it into "Y\_list" for calculation purpose

Y\_list = list(Y.values.ravel())

print(Y\_list)



# Creating a list to store each record deviance

# Leveraging on a for-loop on a record level to ensure values in "Y\_list" is mapped with values in "clf.predict(X)" at a positional level

# Conditional setup with the for-loop to ensure all "0" values in "Y\_list" are calculated differently from the other values in "Y\_list"

# Loop will automatically stop once all the records are calculated at a record level (all 40 records from "Y\_list" are mapped to all corresponding 40 records from "clf.predict(x)" are calculated at a positional level)

D\_list1 = []

for Yi, Ui in zip(Y\_list, clf.predict(X)):

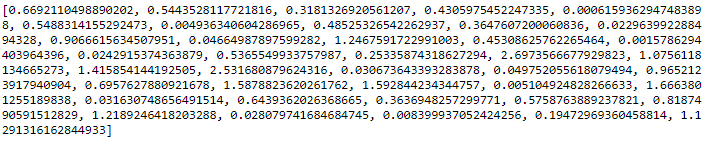
if Yi == 0:

D\_list1.append(0 - (Yi - Ui))

else:

D\_list1.append(Yi\*np.log(Yi/Ui) - (Yi - Ui))

print(D\_list1)



# Summation of the all the deviance of each record and muliplied by 2 to derive the deviance of the model

D = round(2 \* sum(D\_list1),4)

print(D)



**References**

1. Brownlee, J. (2020, August 26). *Train-Test Split for Evaluating Machine Learning Algorithms*. Machine Learning Mastery. https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/
2. *1.1. Linear Models — scikit-learn 0.24.2 documentation*. (2007). Scikit-Learn. https://scikit-learn.org/stable/modules/linear\_model.html
3. *sklearn.linear\_model.PoissonRegressor — scikit-learn 0.24.2 documentation*. (2007). Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.PoissonRegressor.html#sklearn.linear\_model.PoissonRegressor