

**End-of-Course Assessment – July Semester 2021**

|  |  |
| --- | --- |
| **Name** | **Ang Chung Siong Derek** |
| **SUSS PI No.** | **N2072594** |
| **Date Of Submission** | **12/9/2021** |

**Question 1**

**(A)**

**(i).** Reading the given data.

Code:   
  
import pandas as pd

import numpy as np

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

**Output:**

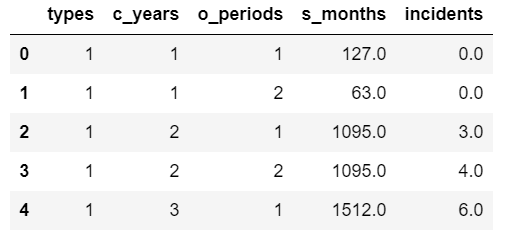
|  | **T** | **A** | **P** | **MS** |  | **Y** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1 | 1 | 127.0 |  | 0.0 |
| **1** | 1 | 1 | 2 | 63.0 |  | 0.0 |
| **2** | 1 | 2 | 1 | 1095.0 |  | 3.0 |
| **3** | 1 | 2 | 2 | 1095.0 |  | 4.0 |
| **4** | 1 | 3 | 1 | 1512.0 |  | 6.0 |

**(ii).**  Names of the columns were changed as instructed.

Code:   
ship=ship.rename(columns={"T":'types', "A":'c\_years', "P":'o\_periods', "MS":'s\_months', "Y":'incidents'})

ship.head()

**Output:**

****

**(iii).** The averages of s\_months and incidents were calculated for each types and o\_periods combinations and saved in a dataframe named ship group as instructed using the following codes.

avg\_incidents=[]

avg\_s\_months=[]

for t in range(1,6):

for p in range(1,3):

avg\_s=round(ship[(ship["types"]==t) & (ship["o\_periods"]==p)]["s\_months"].mean())

avg\_i=round(ship[(ship["types"]==t) & (ship["o\_periods"]==p)]["incidents"].mean())

avg\_s\_months.append(avg\_s)

avg\_incidents.append(avg\_i)

types=[1,1,2,2,3,3,4,4,5,5]

o\_periods=[1,2,1,2,1,2,1,2,1,2]

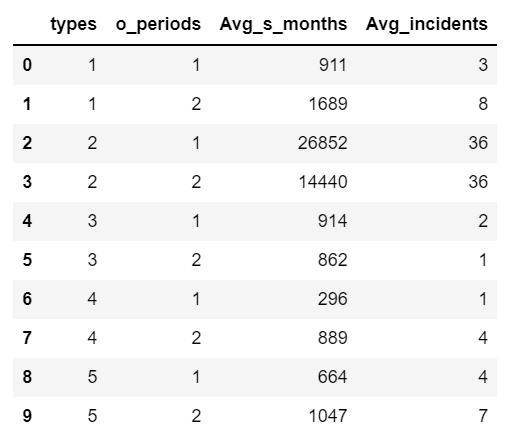
shipgroup=np.array([types,o\_periods,avg\_s\_months,avg\_incidents])

shipgroup=np.transpose(shipgroup)

shipgroup=pd.DataFrame(shipgroup, columns=["types","o\_periods","Avg\_s\_months", "Avg\_incidents"])

shipgroup

**Output:**



This output shows clearly that averages of s\_months and incidents for each combination of types and o\_periods were calculated. It can be observed that the average service months for type 1 and service period 1 is 911 and average number of incidents for this combination of variables are 3. Similarly, other rows of shipgroup can be read.

**(iv).** Here we are supposed to fill the missing values with corresponding average value saved in dataframe shipgroup. Codes are commented using # as expected in python.

miss\_val\_rindex=ship.isnull().any(axis=1) # finding the location of missing values.

# Now we will save the index of rows where missing values were found in a variable.

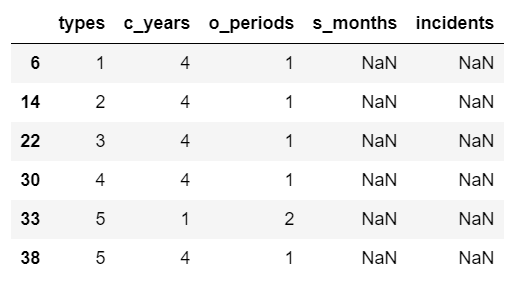
miss\_val\_rindex=miss\_val\_rindex[miss\_val\_rindex==True].index

# Now we will show the rows having missing values.

ship.loc[miss\_val\_rindex]

## Here we found the rows in which data is missing. The data is abstracted from dataframe ship. Therefore, the data shown has ciolumns just like the dataframe ship.

**Output:**



miss\_s\_months=[] # We are creating an empty list for saving average values of service of months corresponding to each types and o\_periods combinations.

miss\_incidents=[] # Here we are creating an empty list for saving average values of missing values of incidents from data shipgroup for each combination of types and o\_periods.

for k in miss\_val\_rindex: # counter k can take value from the index of rows containing missing data i.e., 6, 14, 22 and so on

t=ship.iloc[k]["types"] # Here we are finding the value of the variable types in the row with index k

p=ship.iloc[k]["o\_periods"] # Here we are finding the value of variable o\_periods for row index k

# Now we will find the corresponding average value of s\_month and incidents from dataframe shipgroup and append in the lists that were created earlier.

miss\_s\_months.append(shipgroup[(shipgroup['types']==t) & (shipgroup['o\_periods']==p)]['Avg\_s\_months'].mean())

miss\_incidents.append(shipgroup[(shipgroup['types']==t) & (shipgroup['o\_periods']==p)]['Avg\_incidents'].mean())

idx=miss\_val\_rindex # saving index number of rows containing missing data in a variable idx

# Now will create data frame named missing\_s\_months and missing\_incidents

missing\_s\_months=pd.DataFrame(miss\_s\_months, index=idx, columns=["s\_months"])

missing\_incidents=pd.DataFrame(miss\_incidents, index=idx, columns=["incidents"])

pd.set\_option('mode.chained\_assignment',None)

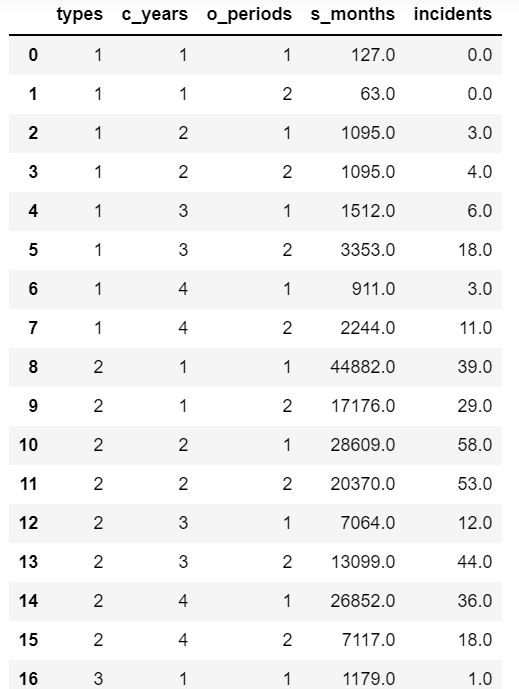
## Let’s fill the missing data as required.

ship["s\_months"][idx]=missing\_s\_months["s\_months"][idx]

ship["incidents"][idx]=missing\_incidents["incidents"][idx]

ship # Let’s view the filled data

**Output:**

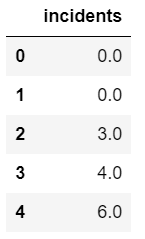
  
 Note: Only 17 rows i.e., up to index number 16 are shown here although there are 40 rows.

**(v).** Now we are creating a dataframe named Y containing data incidents from ship

Y=pd.DataFrame(ship["incidents"], columns=["incidents"])

Y.head() # Let’s view the data

**Output:**



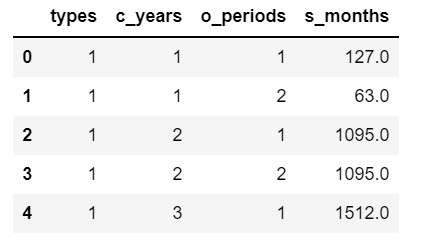
**(B).**

**(i).** In this part, we will change the type of variables. All the variables in ship dataset are numerical in nature but actually, types, c\_years and o\_periods are categorical not interval or ratio. The following codes transforms the expected categorical variables from numerical to categorical.   
  
ship\_pr=ship.astype({"types": "category", "c\_years":"category", "o\_periods" :"category"})

X\_pr=ship\_pr.iloc[:,0:4]

X\_pr.head()

**Output:**

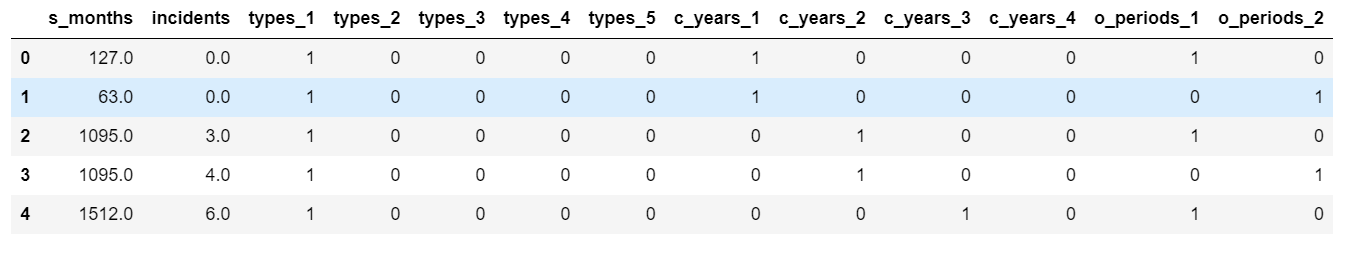


**(ii).** We can create dummy variables using the following codes.

X=pd.get\_dummies(ship\_pr)

X.head()

**Output:**



(iii). Let’s transform s\_months into log(s\_months) and save it in dataframe ship and X both.

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

X\_pr["log\_s\_months"]=np.log(ship["s\_months"])

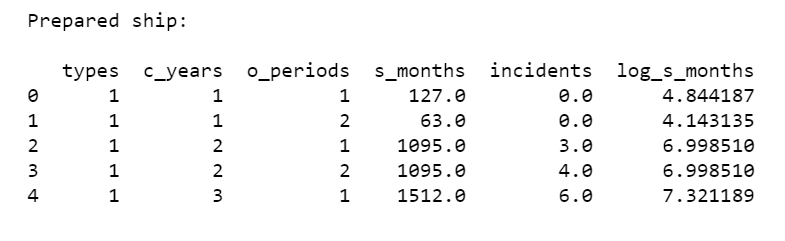
print("Prepared ship data:\n")

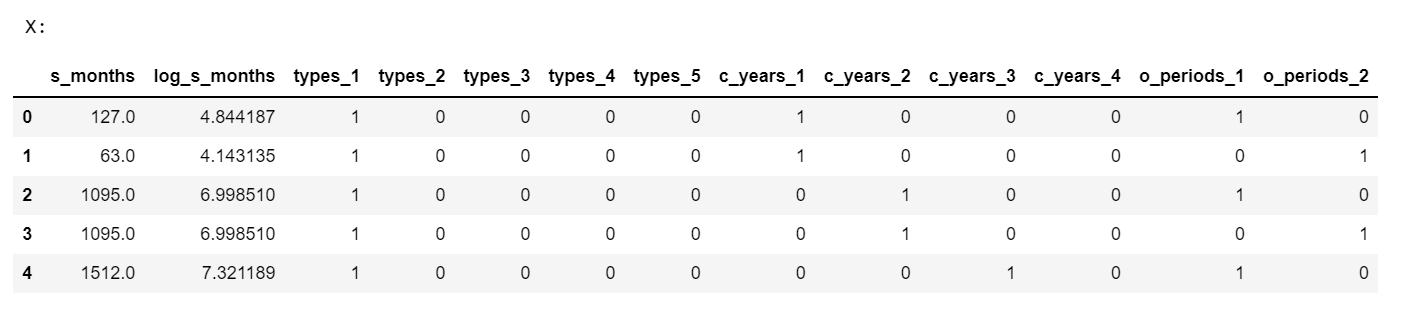
print(ship.head())

print("\n \nPrepared X data : ")

X=pd.get\_dummies(X\_pr)

X.head()

**Output:   
**

****

**(C).**

Observing the dataset ship, one can conclude that number of incidents depend on both the ship types as well as construction of years and operation periods. Since these parameters are not distributed uniformly throughout the dataset therefore it will not be wise to split the dataset and thus we will not split given dataset into training set and test set but will use whole data to train the model.

**(D).**

ship.to\_csv("ship\_prepared.csv") # saving ship to a new dataset

import sqlite3

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

curs=conn.cursor()

ship.to\_sql("ship",conn, if\_exists='replace', index=True)

**Question 2**

**(A)**

PoissonRegressor is used to train a Poisson regression model. PoissonRegressor is a part of generalized linear model and thus is loaded from the scikit learn library linear\_model. It syntax is PoissonRegressor(\*, alpha, fit\_intercept, max\_iter, tol, warm\_star, verbose). Here alpha is a constant by which the penalty term is multiplied. For PoissonRegression model alpha is 1. And for generalized linear model with no penalty term alpha is zero.

Other arguments are summarized below:

**max\_iter** is argument that specify the maximum number of iterations after which calculations will stopped if solution is not found in prior iterations. It is user defined and its default value is 100 if user does not specify.

Usually, a large number of iterations give expected result.

**tol** is the value of tolerance in float. And its default value is 0.001.

**fit\_intercept** is boolean. The deault value of this parameter is True and it simply adds a constant term in the poisson model. However, if the situation does not allow a constant term in the model then it can be set to False. The model will not have a constant term.

**X:** {array-like, sparse matrix} of n features i.e., X is an array of independent variables.

**y:** 1-D array of dependent variables

Besides it we can use .score to get the deviance which also takes arguments as X and y in array just like for the model making.

**(B).**

from sklearn import linear\_model

poisson\_model = linear\_model.PoissonRegressor(alpha=1e-3, max\_iter=20000)

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

X\_tr=X.values

X\_tr[:, 0:2]=sc.fit\_transform(X\_tr[:, 0:2])

X\_tr[:3, :3]poisson.fit(X\_train[:, 1:],Y.values.ravel())

poisson\_model.fit(X\_tr[:, 1:],Y.values.ravel())

Coefficients\_df = pd.DataFrame(poisson\_model.coef\_, columns=["Coefficients"], index=X.iloc[:,1:].columns)

Coefficients\_df

**Output:**



**(C).**

Y\_hat1=pd.DataFrame(poisson\_model.predict(X\_tr[:, 1:]), columns=["E(Y)"])

Y\_hat=round(Y\_hat1)

Y\_hat.head()

**Output:**

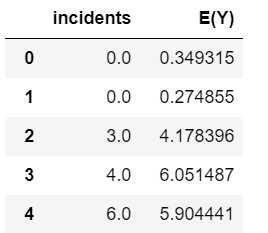
****

Y\_and\_Y\_hat1=pd.concat([ Y, Y\_hat1], axis=1)

Dev\_data=Y\_and\_Y\_hat1 ## Dev\_data is the data for the calculations of deviance

Dev\_data.head()

**Output:**

****

Dev\_data["Y log[Y/E(Y)]"]=np.zeros(40)

for i in range(40):

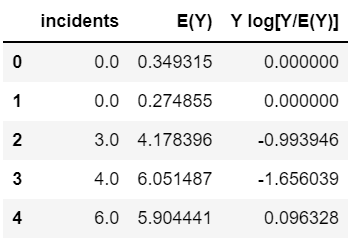
if Dev\_data.loc[i]["incidents"]==0:

Dev\_data.loc[i]["Y log[Y/E(Y)]"]=0

else: Dev\_data.loc[i]["Y log[Y/E(Y)]"]=Dev\_data["incidents"].loc[i]\*np.log(Dev\_data["incidents"][i]/Dev\_data["E(Y)"][i])

Dev\_data.head()

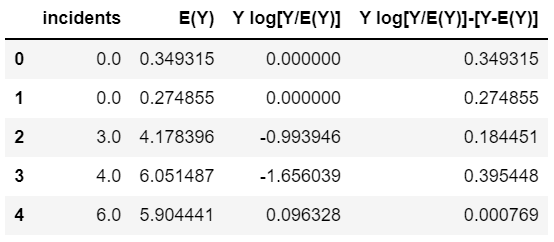
**Output:**

****

Dev\_data["Y log[Y/E(Y)]-[Y-E(Y)]"]=Dev\_data["Y log[Y/E(Y)]"]-(Dev\_data["incidents"]-Dev\_data["E(Y)"])

Dev\_data.head()

**Output:**



Deviance=2\*Dev\_data["Y log[Y/E(Y)]-[Y-E(Y)]"].sum()

print( f"Deviance : {Deviance}")

**Output:**

Deviance : 40.47630419962101

However, the deviance using the score function of scikit learn is given below:

poisson\_model.score(X\_tr[:, 1:], Y.values.ravel())

**Output:**

0.9413995350636287

The formula used is as per instructions. The formula is given below:

**The difference in deviances using score() and calculations using the given formula might be due to the incorrectness of the formula. I think expe(E(Y)) should be used in place of E(Y).**

