****

**ANL252**

**PYTHON FOR DATA ANALYTICS**

**Tutorial Group T09**

**ECA**

|  |  |
| --- | --- |
| **Name** | **PI Number** |
| **LU KAI WEN** | **J1911368** |

**Submission Date: 13 Sep 2021**

Table of Contents

[Question 1 (a)(i) Output for print(ship): 5](#_Toc82345899)

[Question 1 (a)(ii) Code: 6](#_Toc82345900)

[Question 1 (a)(ii) Output for print(ship): 7](#_Toc82345901)

[Question 1 (a)(iii) Code: 8](#_Toc82345902)

[Question 1 (a)(iii) Output for print(shipgroup): 8](#_Toc82345903)

[Question 1 (a)(iv) Code: 9](#_Toc82345904)

[Question 1 (a)(iv) Output for print(ship): 10](#_Toc82345905)

[Question 1 (a)(iv) Code continued: 11](#_Toc82345906)

[Question 1 (a)(iv) Output for print(ship\_alt): 11](#_Toc82345907)

[Question 1 (a)(iv) Code continued: 12](#_Toc82345908)-13

[Question 1 (a)(iv) Output for print(ship): 14](#_Toc82345909)

[Question 1 (a)(v) Code: 15](#_Toc82345910)

[Question 1 (a)(v) Output for print(Y): 16](#_Toc82345911)

[Question 1 (a)(v) CSV output for Y.csv: 17](#_Toc82345912)

[Question 1 (b)(i) Code: 18](#_Toc82345913)

[Question 1 (b)(i) Output for print(ship): 19](#_Toc82345914)

[Question 1 (b)(ii) Code: 20](#_Toc82345915)

[Question 1 (b)(ii) Output for print(X): 21](#_Toc82345916)

[Question 1 (b)(ii) CSV output for X.csv: 22](#_Toc82345917)

[Question 1 (b)(iii) Code: 23](#_Toc82345918)

[Question 1 (b)(iii) Output for print(a): 24](#_Toc82345919)

[Question 1 (b)(iii) Output for print(x): 25](#_Toc82345920)

[Question 1 (b)(iii) Output for print(x) continued: 26](#_Toc82345921)

[Question 1 (b)(iii) Output for print(x) continued: 27](#_Toc82345922)

[Question 1 (c): 28](#_Toc82345923)

[Question 1 (d) Code: 28](#_Toc82345924)

[Question 1 (d) CSV output for ship\_prepared.csv: 29](#_Toc82345925)

[Question 1 (d) CSV output for x\_prepared.csv: 30](#_Toc82345926)

[Question 2 (a): 31](#_Toc82345927)-32

[Question 2 (b) Code: 33](#_Toc82345928)

[Question 2 (b) Output for print(np.shape(Y)): 33](#_Toc82345929)

[Question 2 (b) Code continued: 34](#_Toc82345930)

[Question 2 (b) Output for print(Y): 34](#_Toc82345931)

[Question 2 (b) Code continued: 34](#_Toc82345932)

[Question 2 (b) Output for clf.coef\_: 35](#_Toc82345933)

[Question 2 (b) Code continued: 35](#_Toc82345934)

[Question 2 (b) Output for clf.get\_params(): 35](#_Toc82345935)

[Question 2 (b) Code continued: 36](#_Toc82345936)

[Question 2 (b) Output for print(df): 37](#_Toc82345937)

[Question 2 (b) CSV output for Poisson\_Regression\_Coefficients.csv: 37](#_Toc82345938)

[Question 2 (c) Code: 38](#_Toc82345939)-39

[Question 2 (c) Output for print(Ey.sum()): 39](#_Toc82345940)

[References: 40](#_Toc82345941)

**Question 1 (a)(i) Code:**

import numpy as np

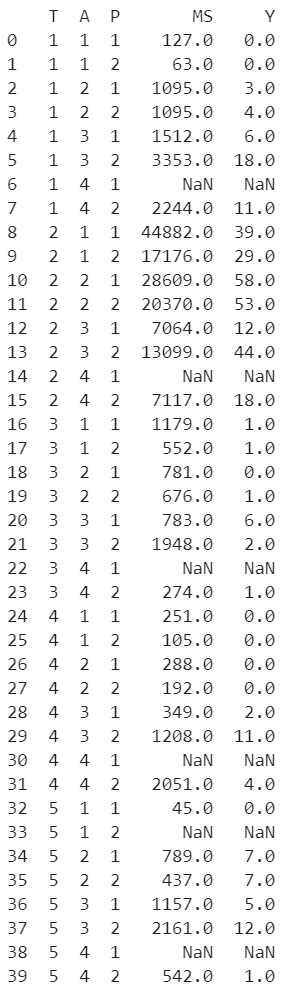
import pandas as pd

#Question 1(a)(i)

ship = pd.read\_csv('ship.csv',delimiter = ',',na\_values={"MS":["."], "Y":["."]}) #reads the original ship csv and replaces the "." as NaN values

print(ship)

# Question 1 (a)(i) Output for print(ship):



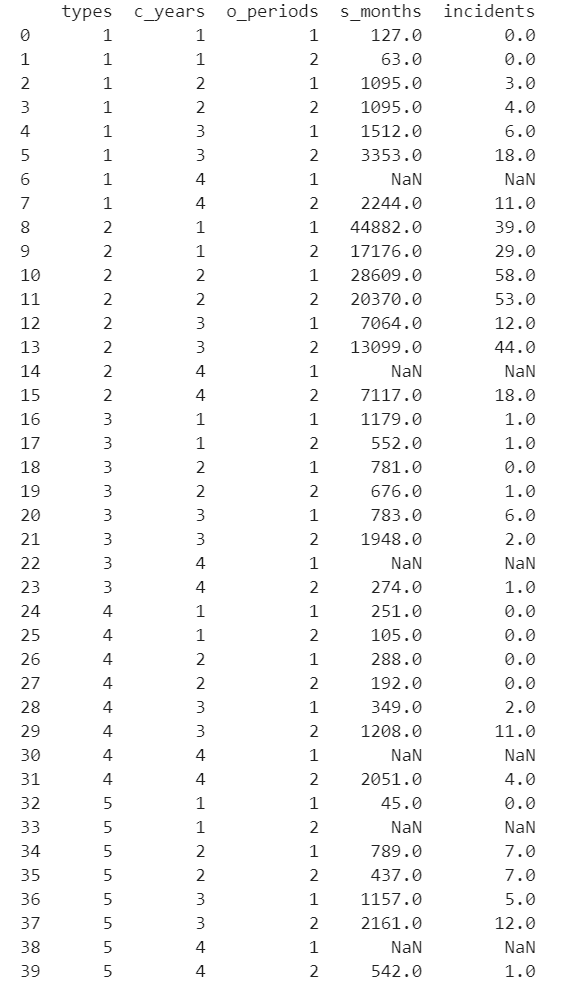
# Question 1 (a)(ii) Code:

#Question 1(a)(ii)

ship = ship.rename(columns={"T":"types", "A":"c\_years", "P":"o\_periods", "MS":"s\_months", "Y":"incidents"}) #rename the columns to appropriate names describing the nature of variables

print(ship)

# Question 1 (a)(ii) Output for print(ship):



# Question 1 (a)(iii) Code:

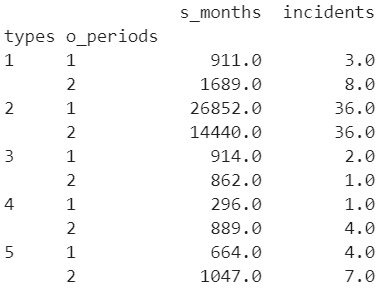
#Question (1)(a)(iii)

ship\_crosstypes = ship.groupby(['types','o\_periods']) #segregated the 'types' and 'o\_period' columns in ship

shipgroup = ship\_crosstypes.mean().round().drop(columns=["c\_years"]) #took the average of s\_months and incidents

print(shipgroup)

# Question 1 (a)(iii) Output for print(shipgroup):



# Question 1 (a)(iv) Code:

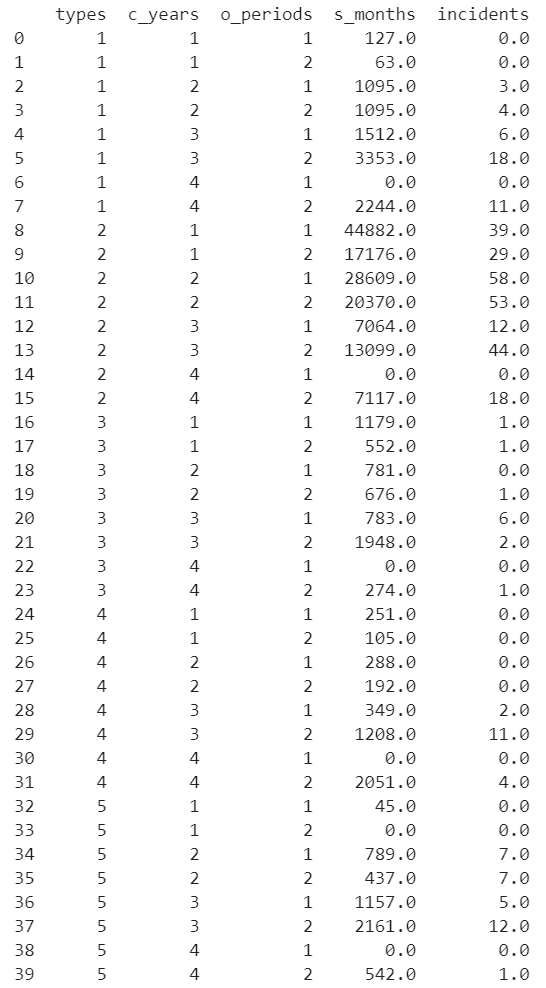
#Question (1)(a)(iv)

ship\_alt = ship.groupby(['types','o\_periods']).mean().round() #segregated the 'types' and 'o\_period' columns in ship.csv

ship = ship.fillna(0) #replaced the NaN values with 0 for iteration later

print(ship)

# Question 1 (a)(iv) Output for print(ship):



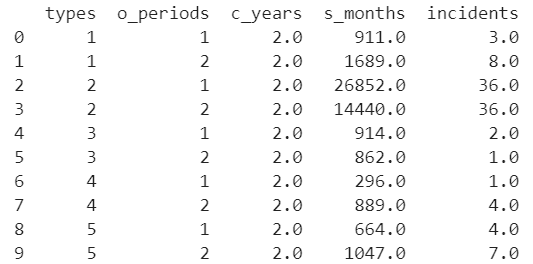
# Question 1 (a)(iv) Code continued:

#Question (1)(a)(iv) cont.

ship\_alt = ship\_alt.reset\_index() #reset the index numbering

print(ship\_alt)

# Question 1 (a)(iv) Output for print(ship\_alt):



# Question 1 (a)(iv) Code continued:

#Question (1)(a)(iv) cont.

for i,j in ship.iterrows(): #iterate through ship.csv

t1 = j['s\_months'] #construct labels for s\_months, types and o\_periods columns

t2 = j['types']

t3 = j['o\_periods']

if t1 == 0: #if a particular element is empty, then execute the following

for k,l in ship\_alt.iterrows(): #iterate through the alt ship.csv

t4 = l['types'] #construct labels

t5 = l['o\_periods']

t6 = l['s\_months']

if t2 == t4 and t3 == t5: #if the types are the same AND o\_periods are the same then copy the mean value

#of s\_months from ship\_alt.csv to ship.csv

ship.at[i,'s\_months'] = t6

for i,j in ship.iterrows(): #performs the same function except this copies the mean value of incidents

t1 = j['incidents']

t2 = j['types']

t3 = j['o\_periods']

if t1 == 0:

for k,l in ship\_alt.iterrows():

t4 = l['types']

t5 = l['o\_periods']

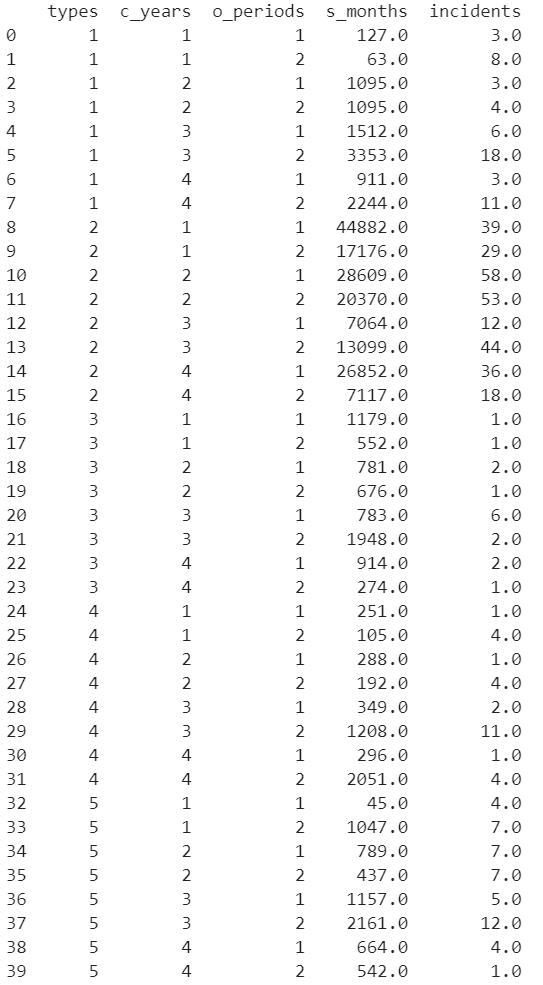
t6 = l['incidents']

if t2 == t4 and t3 == t5:

ship.at[i,'incidents'] = t6

print(ship)

# Question 1 (a)(iv) Output for print(ship):



# Question 1 (a)(v) Code:

#Question (1)(a)(v)

Y = pd.DataFrame(columns=['incidents']) #constructs an empty DataFrame

for i,j in ship.iterrows(): #iterate through the new ship.csv with mean values

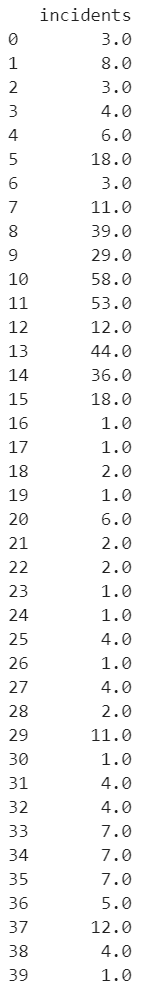
temp = j['incidents']

Y.at[i,'incidents'] = temp #copies "incidents" to Y

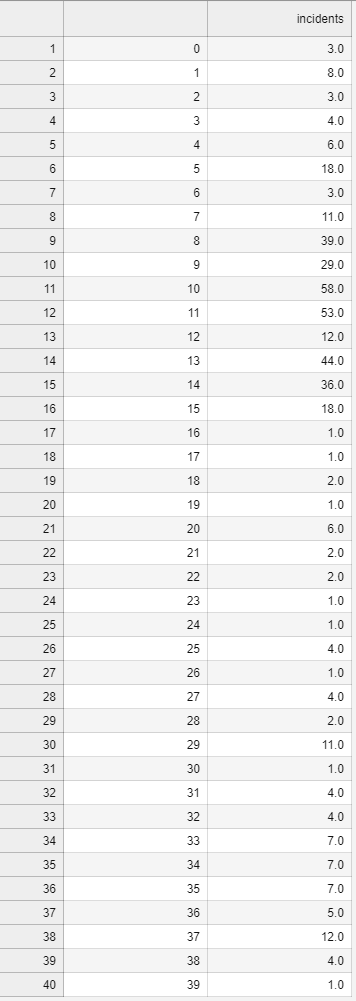
print(Y)

Y.to\_csv("Y.csv") #saves the csv file for later use

# Question 1 (a)(v) Output for print(Y):



# Question 1 (a)(v) CSV output for Y.csv:



# Question 1 (b)(i) Code:

#Question (1)(b)(i)

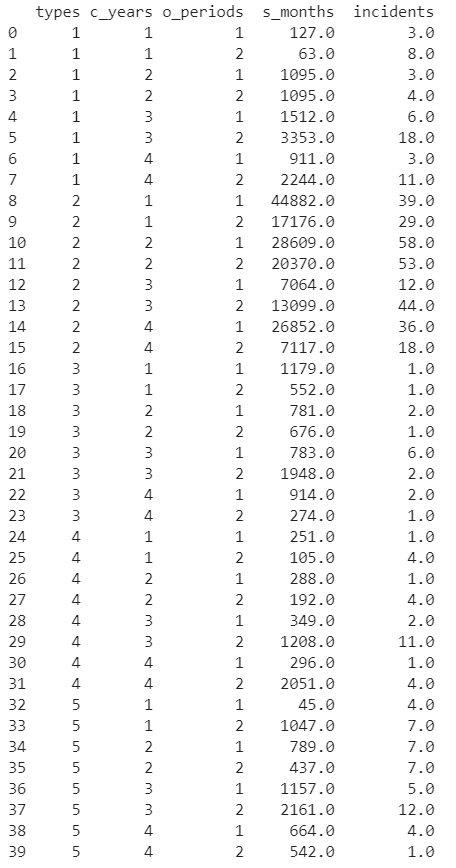
ship['types'] = ship['types'].astype('category') #converts interested columns into categorical variables

ship['c\_years'] = ship['c\_years'].astype('category')

ship['o\_periods'] = ship['o\_periods'].astype('category')

print(ship)

# Question 1 (b)(i) Output for print(ship):



# Question 1 (b)(ii) Code:

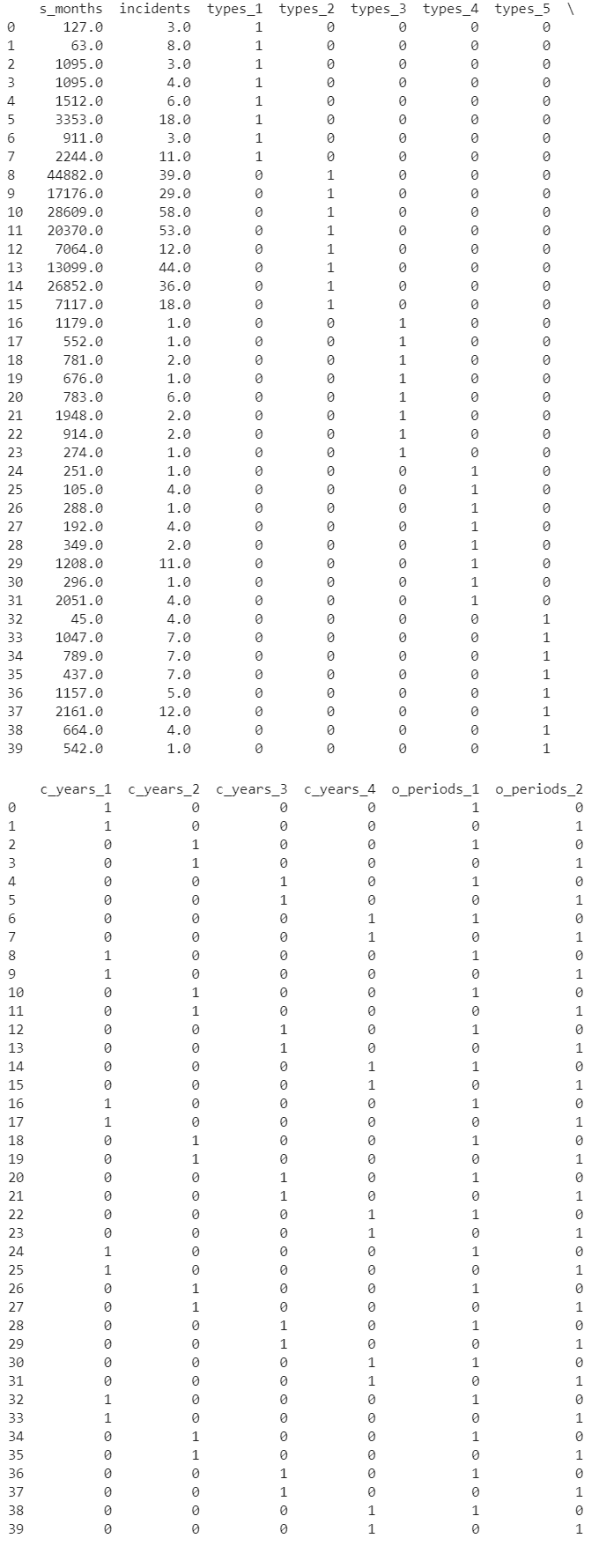
#Question 1(b)(ii)

X = pd.get\_dummies(ship,columns=['types','c\_years','o\_periods']) #converts categorical variables to dummy variables

print(X)

X.to\_csv("X.csv") #saves dummy variables as X.csv

# Question 1 (b)(ii) Output for print(X):

****

# Question 1 (b)(ii) CSV output for X.csv:

# Question 1 (b)(iii) Code:

#Question 1(b)(iii)

log\_s\_months = pd.DataFrame(np.log(ship.s\_months)) #performs a log-transformation of s\_months in ship.csv

log\_s\_months = log\_s\_months.rename(columns ={'s\_months':'log\_s\_months'}) #renames s\_months as log\_s\_months

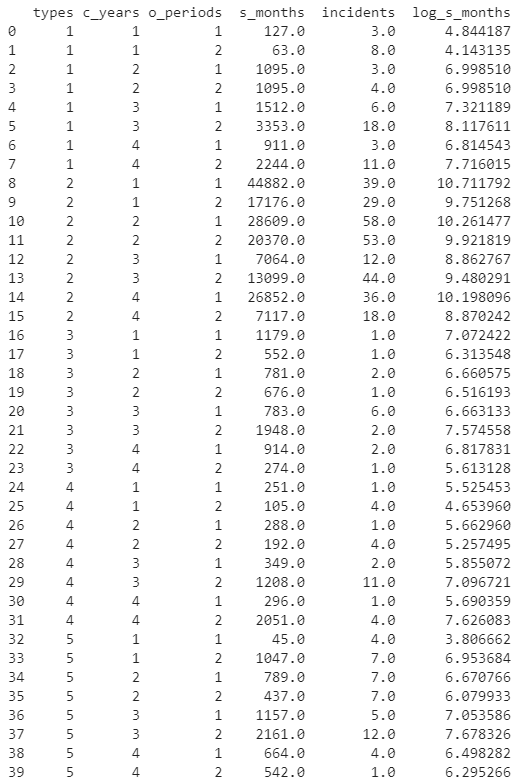
a = pd.concat([ship,log\_s\_months],axis=1) #attaches log\_s\_months to ship.csv

print(a)

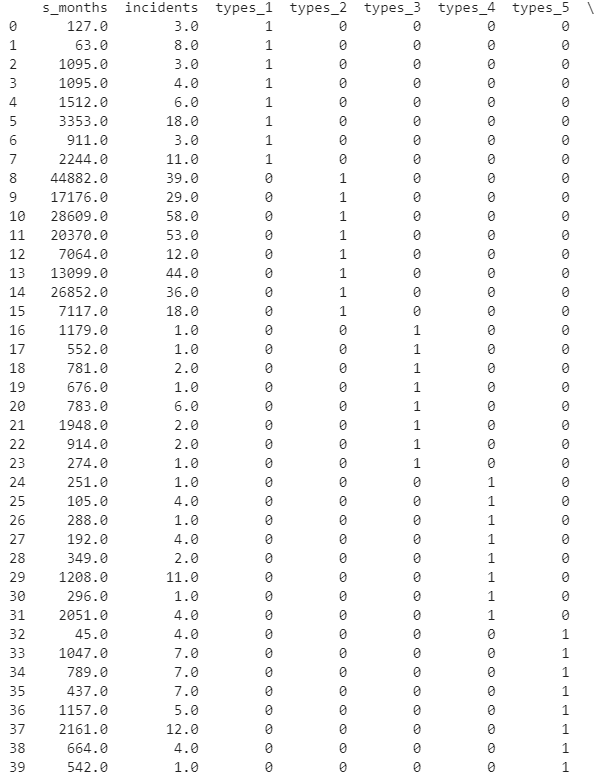
x = pd.concat([X,log\_s\_months],axis=1) #attaches log\_s\_months to X.csv

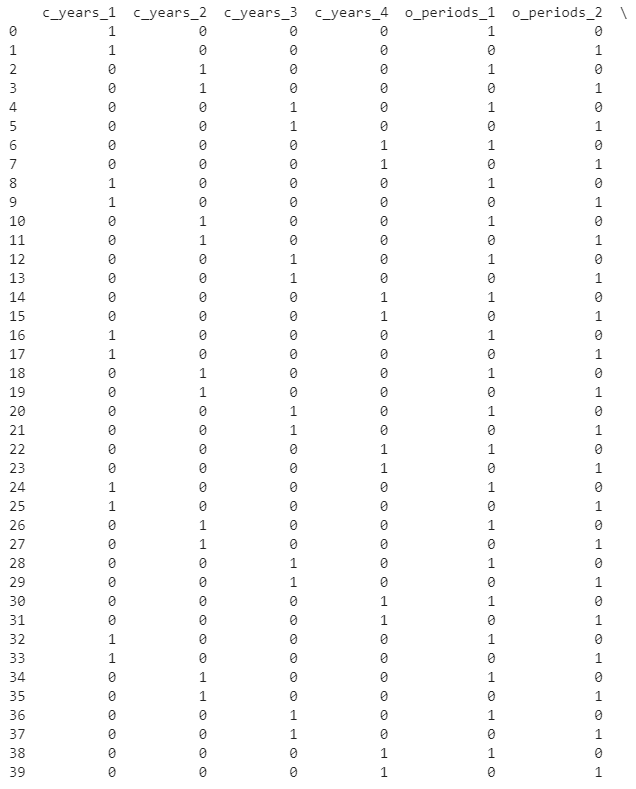
print(x)

# Question 1 (b)(iii) Output for print(a):

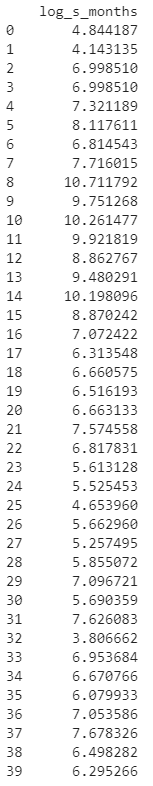


# Question 1 (b)(iii) Output for print(x):



Question 1 (b)(iii) Output for print(x) continued:

# Question 1 (b)(iii) Output for print(x) continued:



# Question 1 (c):

It is not sensible to split the data sets here into training and testing data sets due to stratification. Ship\_prepared.csv has multiclass classifications (with 5 different classes: types, c\_years, o\_periods, s\_months and incidents). A general train-test split would randomly split the data set with total disregard of the distribution and/or weightage of each class. In this scenario, the train set and test set will have different data distributions. Applying a model on each set will perform poor and fail the validation test results.

# Question 1 (d) Code:

#Question 1(d)

a.to\_csv('ship\_prepared.csv') #prepares ship\_prepared.csv

x.to\_csv('x\_prepared.csv') #prepares x\_prepared.csv

from sqlalchemy import create\_engine

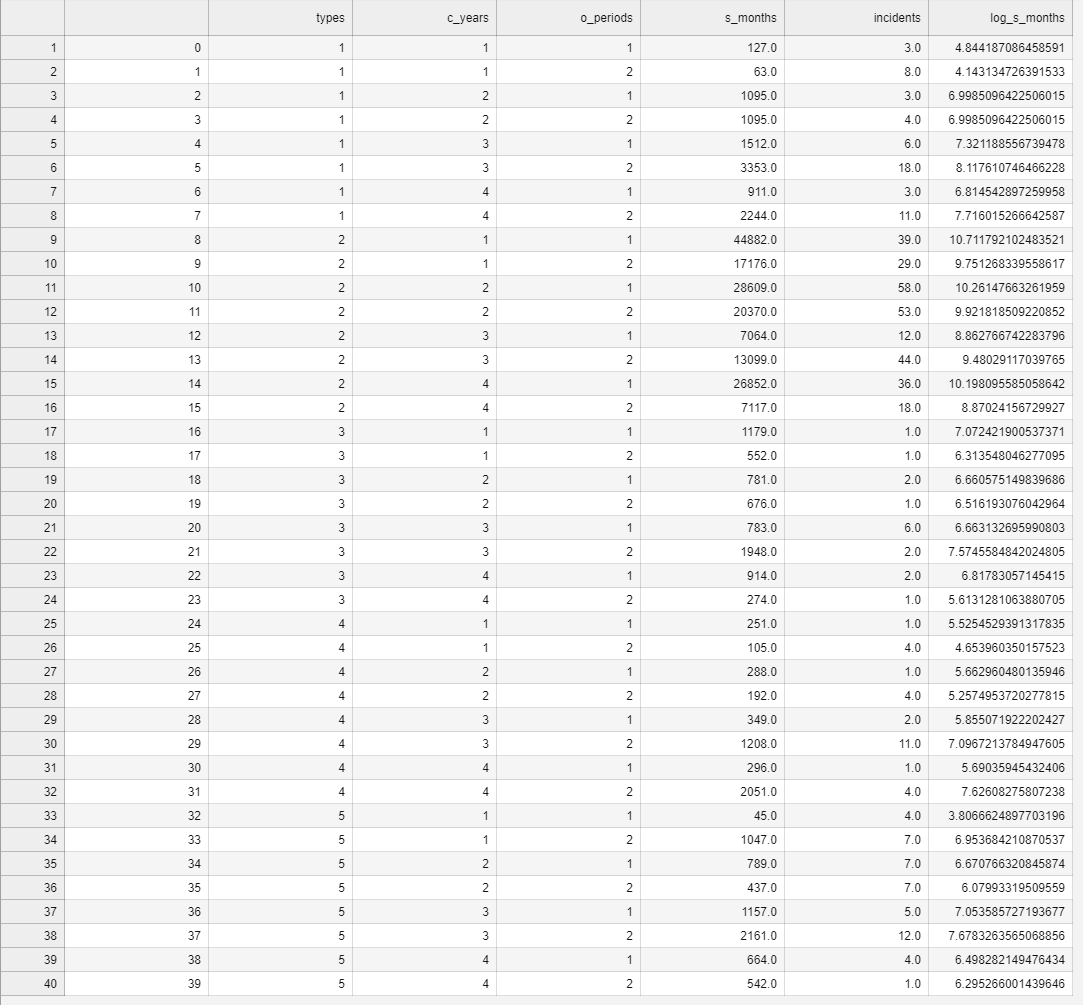
engine = create\_engine('sqlite:///ship.db',echo=True)

sqlite\_connection = engine.connect()

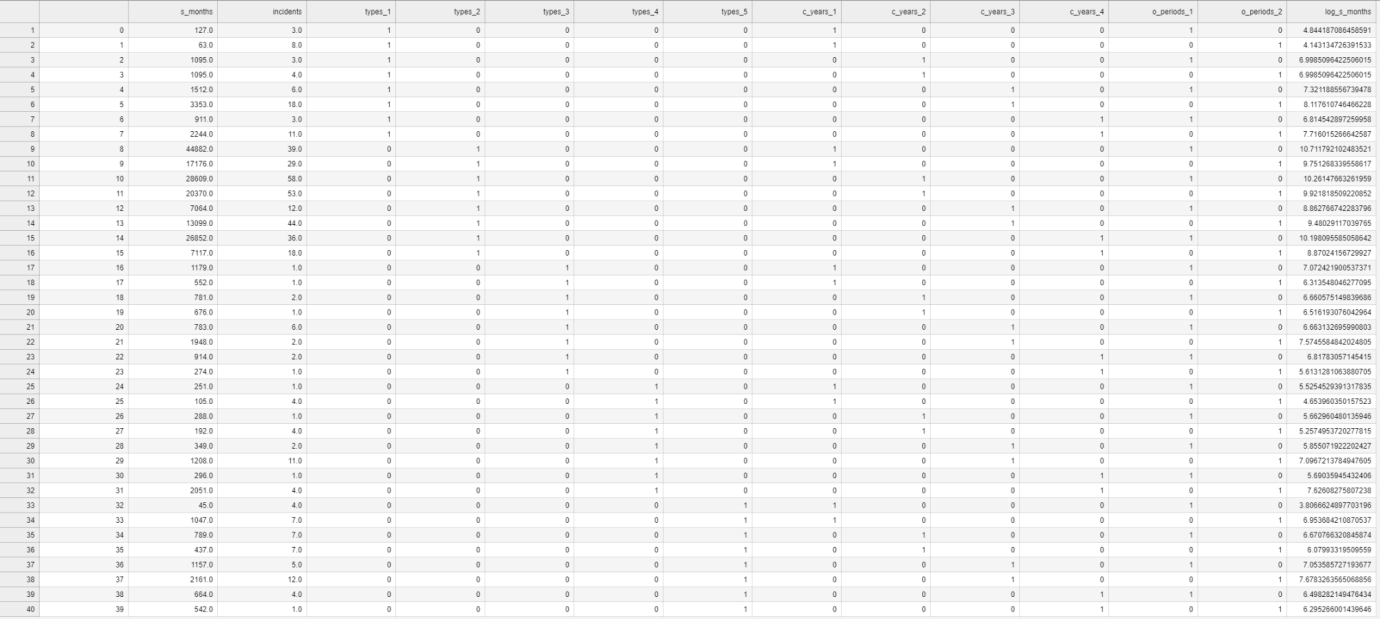
sqlite\_table = "ship.db"

a.to\_sql(sqlite\_table,sqlite\_connection,if\_exists='fail') #exports ship\_prepared.csv as ship.db in tabular form

# Question 1 (d) CSV output for ship\_prepared.csv:



# Question 1 (d) CSV output for x\_prepared.csv:



# Question 2 (a):

The Poisson regressor module is a generalised linear model (GLM) with a built-in lock-link function that outputs values as a function of input variables x and inverse function h such that:

Where represents the deviance of a set of distribution. The deviance is represented as a minimisation problem. For Poisson regression, the target domain of y is with a unit deviance given by:

The Poison regression is popular for data sets with non-negative integer values.

The Poisson regressor module uses the least-squares estimator that approximates the values of . The least-squares method minimises the sum of residual squares – corroborated by the fact the deviance is represented by a minimisation problem. As we do not assume a normal distribution, the least-squares method is justified by the Gauss-Markov theorem. The Gauss-Markov theorem states that if assumptions are met (e.g. linearity, random, non-collinearity, exogeneity and homoscedasticity), then the least-squares method for regression coefficients produces the best linear unbiased estimate.

The fit function from Poisson regression (i.e. linear\_model.PoissonRegressor()) minimises the residual sum of squares between data points in a dataset. The fit function essentially outputs a solution to the minimisation problem (explained as the deviance above):

The solution to the minimisation problem is expressed as the coefficients (.

The predict function is basically uses the GLM to output predicted values using a linear combination of observed values.

The default parameters of the Poisson regressor are as follows:

fit intercept = bool; max\_iter = 100; warm start = False

The parameter is a constant that determines the regularisation strength. Thus, = 0 is an unpenalized GLM.

The fit intercept is a constant that applies a bias to the linear predictor, changing the predicted value .

Max\_iter determines the number of repetition the Poisson regression analysis will conduct. A higher number of iteration would result in a greater accuracy but at the expense of longer computational time.

Warm start, if True, reuses the previous solution to minimise the residual sum of squares in the next data point. It reduces computational time at the cost of accuracy.

# Question 2 (b) Code:

#Question 2(b)

import pandas as pd

from sklearn import linear\_model

import numpy as np

from sklearn import preprocessing

clf = linear\_model.PoissonRegressor() #names the Poisson regressor

Y = pd.read\_csv("Y.csv",index\_col=0) #imports Y.csv

Y = Y.to\_numpy() #converts Y from DataFrame to numpy array

Y = Y.ravel() #flattens Y.csv into a 1D array

print(np.shape(Y))

# Question 2 (b) Output for print(np.shape(Y)):



# Question 2 (b) Code continued:

#Question 2(b) cont.

print(Y)

X = pd.read\_csv("X.csv",index\_col=0) #imports X.csv

scaler = preprocessing.StandardScaler().fit(X) #standardises the data, removing the mean and performs normalisation

scaler.mean\_

scaler.scale\_

X\_scaled = scaler.transform(X) #renames X.csv

# Question 2 (b) Output for print(Y):



# Question 2 (b) Code continued:

#Question 2(b) cont.

clf.fit(X\_scaled,Y) #performs Poisson regression

clf.coef\_ #obtains the coefficients

# Question 2 (b) Output for clf.coef\_:

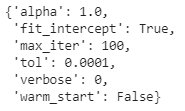


# Question 2 (b) Code continued:

#Question 2(b) cont.

clf.get\_params() #the parameters of the estimated model

# Question 2 (b) Output for clf.get\_params():



# Question 2 (b) Code continued:

#Question 2(b) cont.

df = pd.DataFrame(clf.coef\_)

df.columns = ['coefficients\_value'] #extracts the coefficients of Poisson regression

foo = []

for i in range(len(clf.coef\_)):

foo.append('b'+str(i)) #creates a list b0, b1, b2... bx

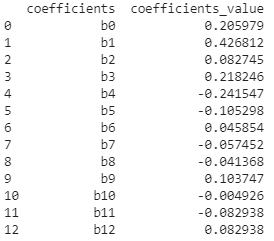
df\_index = pd.DataFrame(foo,columns=['coefficients'])

df = pd.concat([df\_index,df],axis=1)#concatenates index and coefficients

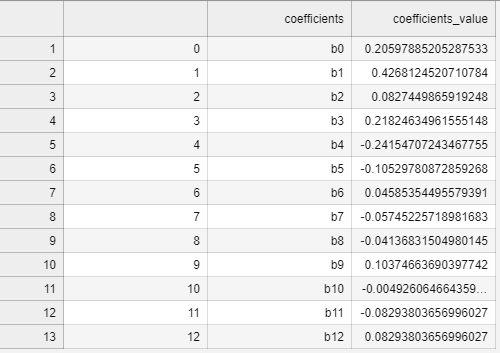
print(df)

df.to\_csv('Poisson\_Regression\_Coefficients.csv') #saved as csv to present coefficients with corresponding labels

# Question 2 (b) Output for print(df):



# Question 2 (b) CSV output for Poisson\_Regression\_Coefficients.csv:



# Question 2 (c) Code:

#Question 2(c)

a = pd.DataFrame(X\_scaled) #creates DataFrame

b = pd.DataFrame(a.sum()) #sums the value of X\_scaled

b.reset\_index(level=0,inplace=True) #resets the index after sum() fx

b.columns = ['variable','sum'] #renames the column in Dataframe b as 'variable' and 'sum'

c = pd.concat([b,df],axis=1) #concatenates

e = c[['sum','coefficients\_value']] #extracts 2 columns of sum and coeff\_value

for i,j in e.iterrows(): #iterates through extracted DataFrame e

t1 = j['sum']

t2 = j['coefficients\_value']

if t1 > 1000: #removes the element in the first row and first column

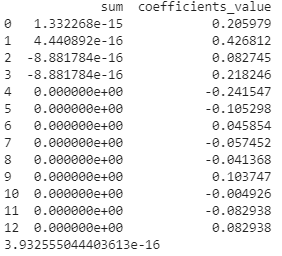
e.at[i,'sum'] = 1 #replaces with 1

print(e)

Ey = 2\* e['sum'] \* e['coefficients\_value'] #takes the product of 'sum' and 'coefficient value columns'

print(Ey.sum())

# Question 2 (c) Output for print(Ey.sum()):



# References:

1. Bhattacharyya, M. (2019, December 04). *3 Things You Need To Know Before You Train-Test Split*. Retrieved from <https://towardsdatascience.com/3-things-you-need-to-know-before-you-train-test-split-869dfabb7e50>
2. Volpi, G. F. (2019, August 07). *6 amateur mistakes I’ve made working with train-test splits*. Retrieved from <https://towardsdatascience.com/6-amateur-mistakes-ive-made-working-with-train-test-splits-916fabb421bb>
3. Solawetz, J. (2020, September 04). *The Train, Validation, Test Split and Why You Need It*. Retrieved from <https://blog.roboflow.com/train-test-split/>
4. scikit-learn. (n.d.). *1.1. Linear Models*. Retrieved from: <https://scikit-learn.org/stable/modules/linear_model.html>
5. scikit-learn. (n.d.). *sklearn.linear\_model.PoissonRegressor*. Retrieved from: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PoissonRegressor.html>