

**ANL252 (Online)**

**Python for Data Analytics**

# **End-of-Course Assessment**

**July 2021 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Geekger Ee** | **W2071283** |

**Tutorial Group: ­­­­­­­­­­­­ T 09**

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

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



1.

1. import pandas as pd

import numpy as np

import sqlite3

con = sqlite3.connect('ship.db')

1. ship = pd.read\_csv("ship.csv", na\_values ='.', na\_filter =True)

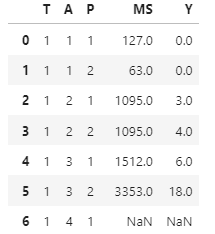


Figure 1: Example of Missing Values Converted from “.” To “Nan” in First 6 Rows.

1. updated\_ship = ship.rename(columns={

"T": "types",

"A": "c\_years",

"P":"o\_periods",

"MS":"s\_months",

"Y":"incidents"},   
 inplace = True)

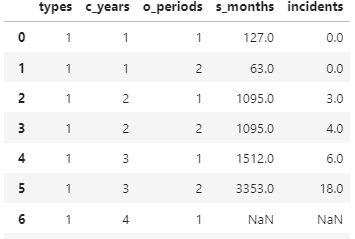


Figure 2: Renamed Variables

1. shipgroup = ship.groupby(['types', 'o\_periods']).agg({'s\_months': ['mean'],'incidents': ['mean']})

shipgroup.columns = ['s\_months\_mean', 'incident\_mean']

shipgroup['s\_months\_mean'] =shipgroup['s\_months\_mean'].apply(lambda x: round(x))

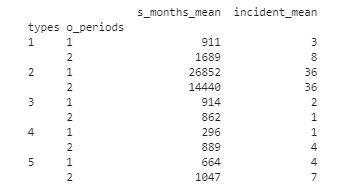
shipgroup['incident\_mean'] =shipgroup['incident\_mean'].apply(lambda x: round(x))  
  


Figure 3: Average Values Rounded to Nearest Integer According to Two Types of Categories

1. #Replacing the missing values of s\_months and incidents according to the mean based on the type and o\_period

#fillna is a way to fillup null values

#groupby is a way to group the columns by in order to perform the task of getting th mean based on type and period

#transform is used to calcualte the mean

ship['s\_months'].fillna(ship.groupby(['types','o\_periods'])['s\_months'].transform('mean'), inplace=True)

ship['incidents'].fillna(ship.groupby(['types','o\_periods'])['incidents'].transform('mean'), inplace=True)

#rounding of both s\_months and incidents to a whole number integer using lambda round method as other methods will not round up or round down properly

ship['s\_months'] =ship['s\_months'].apply(lambda x: round(x))

ship['incidents'] =ship['incidents'].apply(lambda x: round(x))

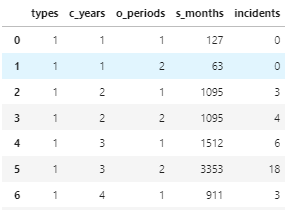


Figure 4: Example of Missing Value Filled According to the Average Value of the Two Categories Rounded to the Nearest Integer in Row 6.

1. Y = pd.DataFrame(ship['incidents'])

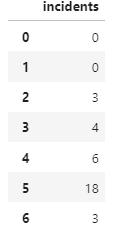


Figure 5: Variable Incidents Saved as a Dataframe Named Y

1. convert\_ship = {'types':'category',

'c\_years':'category',

'o\_periods':'category'}

ship= ship.astype(convert\_ship)

ship.dtypes

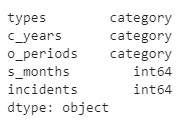


Figure 6: After Data Type Conversion of the Variables

1. X = pd.get\_dummies(ship, columns = ['types','c\_years','o\_periods'])

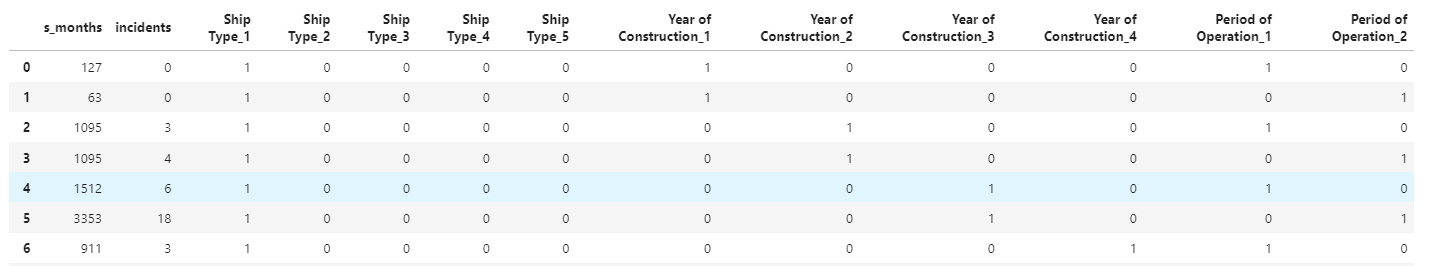


Figure 7: Converted Categorical Variables to Dummy Variables Saved as Dataframes X

1. logdata = np.log(ship['s\_months'])

ship['log\_s\_months'] = logdata

X['log\_s\_months'] =logdata

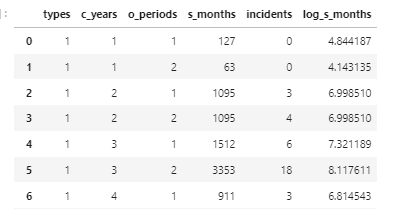


Figure 8: Transformed Variable Attached to the DataFrame ship

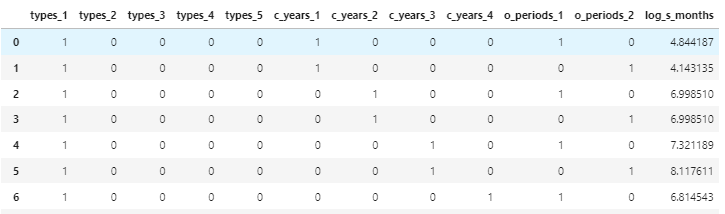


Figure 9: Transformed Variable Attached to the DataFrame X

1. Training and testing split procedures are used to predict the performance of the model and there are instances where it is not sensible to do so when you have a small dataset. According to our data frame, it only contains 40 rows of information which is insufficient when we split it into the training and testing dataset. Resulting in the testing dataset being ineffective in evaluating the model’s performance as testing requires around 30% of the dataset or the default proportion which is 25%. Small datasets make it difficult for the training dataset model to learn effectively in order to map the inputs and outputs as they are unable to identify what values are real and what values are random.
2. ship.to\_csv('ship\_prepared.csv', index=False)

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

ship.to\_sql("ship", con, if\_exists='append')

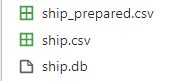


Figure 10: DataFrame ship Saved in CSV File and Database as Table

2.

1. from sklearn.linear\_model import PoissonRegressor

The above line is the corresponding scikit-learn module which needs to be imported to perform the Poison regression which is part of the linear model.

The module in this case would be linear\_model from the above line which is the algorithm.

The estimator would be PoissonRegressor which is the required algorithm we need to learn from our data.

When you call upon the fit method, it is to train your data to fit the model used where X represents the training data, Y represents the column we wish to target, and sample\_weight refers to whether your estimator is focusing on the samples.

Predict method lastly is used to predict certain data using Poisson regression which is a type of linear model The parameters involved are X where it represents the training data used in the fit method

1. from sklearn.linear\_model import PoissonRegressor

from sklearn import preprocessing

from collections import defaultdict

poissonmodel = PoissonRegressor()

X.drop(columns=['s\_months','incidents'], axis=1, inplace=True)

train = X.to\_numpy()

target = Y.to\_numpy()

target= target.ravel()

poissonmodel.fit(train, target)

poissonmodel.coef\_

expectedvalueY=poissonmodel.predict(train)

expectedvalueY

#Present coefficients with labels as a table  
table=defaultdict(int)

x=0

for val in poissonmodel.coef\_:

table['β'+str(x)]=val

x=x+1

df=pd.DataFrame(table.items(), columns=['β', 'Coefficient'])

df.index=['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']

df

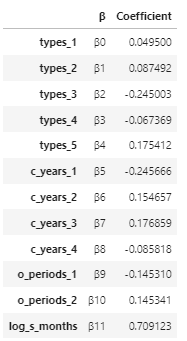


Figure 11: DataFrame to present the coefficients with labels

#value to add up the deviance value   
value=0

#loops through the length of target which is 40 and performs the formula where i represent the index to use

for i in range(len(target)):

if(target[i]==0):

D=expectedvalueY[i]

value+=D

else:

D=target[i]\*np.log(target[i]/expectedvalueY[i])-(target[i]-(expectedvalueY[i]))

value+=D

deviance=value\*2

print(deviance)



Figure 12: Computed Deviance using the formula above