

**ANL252**

**Python for Data Analytics**

**End-of-Course Assessment**

**July 2021 Presentation**

|  |  |
| --- | --- |
| Name: | Tan Yi Hao |
| Student Number: | H2070659 |
| Submissions Date: | 7th September 2021 |

**Question 1(a)(i)**

|  |
| --- |
| import sqlite3  import pandas as pd  import numpy as np  # Read in "ship.csv" as pandas DataFrame called "ship"  ship = pd.read\_csv("ship.csv", na\_values=".") |
|  |

**Calendar

Description automatically generated** **Calendar

Description automatically generated**

**Question 1(a)(ii)**

|  |
| --- |
| # rename column names  ship = ship.rename(columns={  'T': 'types',  'A': 'c\_years',  'P': 'o\_periods',  'MS': 's\_months',  'Y': 'incidents'  }) |
|  |

**Table

Description automatically generated Table

Description automatically generated**

**Question 1(a)(iii)**

|  |
| --- |
| # group by every combination of "type" and "o\_periods", and calculate their means  shipgroup = ship.groupby(by = ["types", "o\_periods"]).mean().round(0)  shipgroup |
|  |

**Table

Description automatically generated**

**Question 1(a)(iv)**

|  |
| --- |
| # loop through rows that have missing values  for index, row in ship[ship['s\_months'].isnull()].iterrows():  # replace s\_month with the mean of other rows with same type and operational periods, rounding to 2 decimal places  ship.loc[index, 's\_months'] = ship[(ship['types']==row['types']) & (ship['o\_periods']==row['o\_periods'])]['s\_months'].mean().round(2)  # replace incidents with the mean of other rows with same type and operational periods, rounding to 2 decimal places  ship.loc[index, 'incidents'] = ship[(ship['types']==row['types']) & (ship['o\_periods']==row['o\_periods'])]['incidents'].mean().round(2)  ship |
|  |

**Table

Description automatically generated** **Table

Description automatically generated**

**Question 1(a)(v)**

|  |
| --- |
| # save the target variable "incidents" into a df Y, into 2 decimal places  Y = ship['incidents'].round(2)  Y |
|  |

**Table

Description automatically generated**

**Question 1(b)(i)**

|  |
| --- |
| # data type conversion to categorical  ship['types'] = ship.types.astype('category')  ship['c\_years'] = ship.types.astype('category')  ship['o\_periods'] = ship.types.astype('category')  ship |
|  |

**Table

Description automatically generated** **Table

Description automatically generated**

**Question 1(b)(ii)**

|  |
| --- |
| # get dummy variables and save the result as X  X = pd.get\_dummies(ship)  X |
|  |

**Table

Description automatically generated**

**Table

Description automatically generated with medium confidence**

**Question 1(b)(iii) (ship output)**

|  |
| --- |
| # log transformation  ship['log\_s\_months'] = np.log(ship['s\_months'])  X['log\_s\_months'] = np.log(X['s\_months'])  ship |
|  |

**Table

Description automatically generatedTable

Description automatically generated**

**Question 1(b)(iii) (X output)**

|  |
| --- |
| # log transformation  ship['log\_s\_months'] = np.log(ship['s\_months'])  X['log\_s\_months'] = np.log(X['s\_months'])  X |
|  |

**Table

Description automatically generated**

**Question 1(c.)**

When the dataset is parted into train and test sets, the dataset is very limited; thus, it is not wise to split the dataset even further. Similarly, there will not be enough data in the test set to access the model presentation. This would likely cause a bad to normal result. Based on the current dataframe, the model was based on the original data, which would not make it sensible to do a split, and thereby not holding back any additional data (Wu, 2021). Therefore, it is recommended to use the entire dataset for training purposes.

**Question 1(d)**

|  |
| --- |
| # save the prepared DataFrame "ship" as a new csv  ship.to\_csv('ship\_prepared.csv', index = False) |
|  |

|  |
| --- |
| # create database called ship.db  conn = sqlite3.connect("ship.db")  cursor = conn.cursor()  # export the DataFrame to the database as tables  ship.to\_sql("ship", conn, if\_exists = "replace", index = False)  conn  cursor |
|  |

**Question 2(a)**

We will start by using Poisson regression model in the scikit-learn module. Poission distribution is a distribution that tells us how many numbers of times an event will occur within a specific period. The module to use will be: class sklearn.linear\_model.PoissonRegressor() and the following are its estimators:

(*\**, *alpha=””*, *fit\_intercept=True/False*, *max\_iter=””* *tol=””*, *warm\_start=True/False*, *verbose=””*) (JMLR, 2011). Alpha is a constant that takes any integer values, and its default value is 1. Fit\_intercept specifies whether a constant should be added to the linear predictor. The values are either True or False. Max\_iter determines the maximum number of iterations, and its default value is 100.

Secondly, for the fit() function, it takes 2 perimeters into consideration and is used to fit or train the model by giving the training data as input to the function. The parameter X usually contains all the independent variables, and Y consists of all the target values (Wu, 2021).

Finally, the predict() function is used to predict the target values for the feature matrix X, and only requires one parameter. It will return the predicted values for the respective X samples of data when using the function, and will usually be used to pass the training samples data to get the predictors.

**Question 2(b)**

|  |
| --- |
| #consider DataFrames X and Y generated in Question 1 and to convert to numpy array  #Using poisson regressor model as shown in the offical website  import numpy as np  import pandas as pd  X = numpy.array(X)  Y = numpy.array(Y)  Y = numpy.ravel(Y)  from sklearn import linear\_model  clf = linear\_model.PoissonRegressor()  # Generating a table or a DataFrame to present the coefficients with the corresponding labels  clf.fit(X, Y)  clf.score(X, Y)  PoissonRegressor() |
|  |

**Question 2(c)**

|  |
| --- |
| # Without using score() function of the scikit-learn package, we will use numpy to find logs  import numpy as np  n = int(input("Enter n: "))  Y = float(input("Enter Y: "))  expected\_value\_Y = float(input("Enter expected value of Y: "))  # Using np.log to get log  # If Y = 0, np.log(Y/expected\_value\_Y) = 0  if Y == 0:  D = ((Y \* 0)) - (Y - expected\_value\_Y)  else:  for i in range(1,n+1):  D = ((Y \* np.log(Y/expected\_value\_Y))) - (Y - expected\_value\_Y)  D=2\*D  print(f"D = {D:.2f}") |
|  |

A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generated

**REFERENCES**

Wu, K. Y. (2021). *ANL252 Python for data analytics (study guide)*. Singapore University of

Social Sciences.

[Scikit-learn: Machine Learning in Python](http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html), Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

**APPENDIX 1: SCREENSHOT OF CODE**

**QNS 1a(i):**

**Graphical user interface, text, application, email

Description automatically generated**

**QNS 1a(ii):**

**Graphical user interface, text, application, email

Description automatically generated**

**QNS 1a(iii):**

**A picture containing text

Description automatically generated**

**APPENDIX 1: SCREENSHOT OF CODE**

**QNS 1a(iv):**

**Text

Description automatically generated with medium confidence**

**QNS 1a(v):**

**Graphical user interface, text, application

Description automatically generated**

**QNS 1b(i):**

Graphical user interface, text, application

Description automatically generated

**APPENDIX 1: SCREENSHOT OF CODE**

**QNS 1b(ii):**

Graphical user interface, text, application, chat or text message

Description automatically generated

**QNS 1b(iii):**

Graphical user interface, text, application

Description automatically generated

**QNS 1d:**

Graphical user interface, text, application, email

Description automatically generated

**APPENDIX 1: SCREENSHOT OF CODE**

**QNS 2B:**

**Qns 2C:**

**Graphical user interface, text, application, email

Description automatically generated**