****

**ANL252**

**PYTHON FOR DATA ANALYTICS**

**End-of-Course Assignment**

**Submission Date: 13 February 11:55hrs**

****

|  |  |
| --- | --- |
| **Name** | **PI number** |
| James Lai Hou Xian | Y1981480 |

Q1ai)

|  |
| --- |
| **Codes:**  #Importing the various packages  import numpy as np  import pandas as pd  import math  import sklearn  import sqlite3  import csv  #Reading ship.csv and replacing all "." into NaN  ship = pd.read\_csv("ship.csv", na\_values = ".")  ship |
| **Output:** |

Q1aii)

|  |
| --- |
| **Codes:**  #Renaming all the column labels  ship = ship.rename(columns = {"T" : "types", "A" : "c\_years", "P" : "o\_periods", "MS" : "s\_months", "Y": "incidents"})  ship |
| **Output:** |

Q1aiii)

|  |
| --- |
| **Codes:**  #Using groupby function to calculate mean  shipgroup = round(ship.groupby(['types','o\_periods'],as\_index=False)[['s\_months','incidents']].mean())  shipgroup = shipgroup.astype(int)  shipgroup |
| **Output:** |

Q1aiv)

|  |
| --- |
| **Codes:**  #Using groupby multiple columns to replace missing values in s\_months  #Stated in question to use "types and o\_periods" as conditions. #.apply is used in combination with fillna to find the mean of all nan values  ship['s\_months'] = round(ship.groupby(['types','o\_periods'])['s\_months'].apply(lambda x:x.fillna(x.mean())))  #Using group by multiple columns to replace missing values in incidents  #Stated in question to use "types and o\_periods" as conditions. #.apply is used in combination with fillna to find the mean of all nan values  ship['incidents'] = round(ship.groupby(['types','o\_periods'])['incidents'].apply(lambda x:x.fillna(x.mean())))  #Declaring the entire dataframe as integer (stated in question)  ship = ship.astype(int)  #calling back ship dataframe  ship |
| **Output:** |

Q1av)

|  |
| --- |
| **Codes:**  #Saving incidents in ship dataframe into Y  Y = ship[['incidents']]  Y |
| **Output:** |

1bi)

|  |
| --- |
| **Codes:**  #Before Conversion  print(ship.dtypes)  #After conversion - changing the different variables into category variables  ship = ship.astype({'types':'category', 'c\_years': 'category', 'o\_periods':'category'})  print(ship.dtypes) |
| **Output:** |

Q1bii)

|  |
| --- |
| **Codes:**  #Converting all categorical variables into dummy variables and save them into df X  #Excluding both column labels "s\_months" and "incidents"  X = pd.get\_dummies(ship).drop(["s\_months","incidents"], axis=1)  X |
| **Output:** |

Q1biii)

|  |
| --- |
| **Codes:**  #Log-transformation as a way to scale down the variable "s\_months"  log\_s\_months = np.log(ship["s\_months"])  ship["log\_s\_months"] = log\_s\_months  X["log\_s\_months"] = log\_s\_months  X |
| **Output:** |

Q1c)

The premise of predicting unseen data is used to assess a model’s predictive power. In this context, the model produced was built from the original dataset (ship.csv). Without having a separate “testing” dataset, it may be insensible to split and test for the model’s predictive accuracy. Furthermore, the dataset provided only contains 40 rows of data, making it a small dataset. As such, it may be ineffective in evaluating model performance, as the obtained results may be biased.

Using the entire dataset for training purposes is a viable option due to data availability constraints. This is due to the model being able to analyse more data and producing more accurate results compared to splitting the dataset.

Q1d)

|  |
| --- |
| **Code:**  #Exporting ship as a CSV file and rename to ship\_prepared  ship.to\_csv('ship\_prepared.csv')  # generate a connection; will be created if it doesnt exist  con = sqlite3.connect('ship.db')  # produce a cursor object  cur = con.cursor()  ship.to\_sql('ship', con, if\_exists = "replace")  shipgroup.to\_sql('shipgroup', con, if\_exists = "replace")  # commit/save the changes to the sql database  con.commit()  # close the connection  con.close() |
| **Output:**  A successful export will be stored in the root folder. Screenshot illustrates the new csv filed prepared through the codes above. |

2a)

In scikit-learn (sklearn) library, the most applicable module in this context is the "sklearn.linear\_model", which is a class of the sklearn library. Generally, it is used to perform different functions related to machine learning using linear models. There are many linear models ranging from Poisson regressor, classification, and many more. The premise of these models is that it is based on training data, which computes the weight of each feature to form a model to estimate the target variable.

In context to this case study, the most appropriate linear\_model will be PoissonRegressor which creates a generalised linear model with a poisson distribution.

**Estimator**

An estimator is a type of object that can fit a model based on training data and infer properties from the new data. Generally, all estimators uses a fit method to perform the function of learning data through "estimator.fit(data)".

**Fit**

In "fit()", it allows the user to estimate the parameters that they required for their model. Within it, users have to input training data as arguments which can be either in a one-dimensional array (for unsupervised learning) or two-dimensional array (for supervised learning). There are two main parameters involved in "fit()" - X and Y. When applying this function, the parameters must be strictly observed to be dependent variables. For X, it must consist of an array-like shape; often the training data array. Similarly, Y has an array-like shape where user inputs an array to retrieve their targeted values.

**Predict**

Under the predict() function, it often leverages on trained datasets to predict the values of untrained data. The official syntax follows "prediction = .predict(data)". The parameters involved only consist of a single argument which contains only the data that is to be tested.

2b)

|  |
| --- |
| **Code:**  #Converting X dataframe into numpy  X = X.to\_numpy()  print(X.shape)  #Converting Y dataframe into numpy  Y = Y.to\_numpy()  print(Y.shape)  #Converting Y into a one-dimensional array  Y = Y.flatten()  Y  #Using the data from above, fit into the poisson model  from sklearn import linear\_model  clf = linear\_model.PoissonRegressor()  clf.fit(X, Y)  coeffi\_df = pd.DataFrame(clf.coef\_)  #Putting column labels  coeffi\_df.columns = ["Coefficient"]  #Inputting row labels  coeffi\_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"]  #Transposing the dataframe for presentation  coeffi\_df.transpose() |
| **Output:** |

2c)

|  |
| --- |
| **Code:**  #Getting deviance (Y)  def dev\_value(clf, X, y): #clf - poisson regression model, X - independent variable, y - dependent variable  pred\_y\_value = clf.predict(X)  dummy\_value = 0  for y\_value\_actual, y\_value\_expected in zip (y, pred\_y\_value):  if y\_value\_actual == 0:  dummy\_value += y\_value\_expected  else:  dummy\_value += y\_value\_actual \* np.log(y\_value\_actual / y\_value\_expected) - (y\_value\_actual - y\_value\_expected)  return 2\*dummy\_value  dev\_value\_final = dev\_value(clf, X, Y)  print(f"The deviance of Y is: {dev\_value\_final:.4f} (4 d.p)") |
| **Output:** |