

**ANL252 (Online)**

**Python for Data Analytics**

# **End Course Assignment**

**January 2023**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **ANG DING YI** | **B2211211** |

**Tutorial Group: ­­­­­­­­­­­­ TV 05**

**Instructor’s Name: Dr. KUMAR MUNISH**

**Submission Date: 02/03/2023**

**Content Page**

|  |  |
| --- | --- |
| Content | Page |
| Content Page | 2 |
| Question 1 | 3 |
| Question 2 | 4 |
| Question 3 | 7 |
| Question 4 | 9 |
| Question 5 | 13 |
| Question 6 | 16 |

# **Question 1**

Figure 1: Missing value counts per column

Table

Description automatically generated

The pandas dataframe was read with the missing values listed above taken into consideration. With that, the number of missing values per column was computed, and the variables that contain missing values are: Claim\_ID, Actual and Terms as seen in the above screenshot. See Python code below.

|  |
| --- |
| # Q1  # Specify the possible na values  na\_values = [' ', 'Unkn', '???']  # Read in dataset as a pandas dataframe, with the specified na values taken into consideration  df = pd.read\_csv('ECA.csv', na\_values = na\_values, na\_filter=True)  # Check and state the variables that contains missing values.  We see the variables `Claim\_ID`, `Actual` and `Terms` contains missing values from the output of the code below  df.isnull().sum(axis=0) |

# **Question 2**

As mentioned in question 1, there are 3 variables with missing data: Claim\_ID (5 missing values), Actual (1677 missing values) and Terms (7 missing values).

Firstly, to treat the 5 missing values in the `Claim\_ID` column, a dummy ID was created and imputed as the `Claim\_ID` for each row with missing `Claim\_ID` value.

Rationale: According to the appendix in the question, Claim\_ID represents a unique identifier of the claim. Since the purpose of Claim\_ID is just to identify a claim, and each claim needs to have a unique identifier, we can create a dummy ID that is unique for each of the 5 claims that have missing `Claim\_ID`

Secondly, mode imputation was done to treat the 1677 missing values in the `Actual` column. Each of the missing `Actual` values was replaced with the mode date of the `Actual` column.

Rationale: As `Actual` are dates of claim settlement, we are unable to do mean imputation as they are not numeric values. In this case, we opt for mode imputation.

Thirdly, to treat the 7 missing values in `Terms` column, mode imputation was done. Each of the missing `Terms` values was replaced with the mode category of the `Terms` column.

Rationale: As `Terms` are internal terms and condition code which are categorical values, we are unable to do mean imputation as they are not numeric values. In this case, we opt for mode imputation. See Python Code below.

|  |
| --- |
| # Q2  # Treat the missing values  # First, treat the missing values in `Claim\_ID`.  # According to the appendix, Claim ID is unique identifier of the claim. Hence, each claim needs to have a unique identifier. We can create a dummy unique ID for each of these 5 claims that have missing `Claim\_ID`  # Find the rows with missing `Claim\_ID`  idx\_missing\_claimid = np.where(df.Claim\_ID.isnull())[0]  print('Index of rows with missing Claim\_ID - Before :', idx\_missing\_claimid)  # Since there are only 5 rows with missing Claim ID, we can manually impute a dummy claim ID for each of these rows.  for i in range(len(idx\_missing\_claimid)):  df.loc[idx\_missing\_claimid[i], 'Claim\_ID'] = i  # Let's check if there is still any missing claim\_id  idx\_missing\_claimid\_after = np.where(df.Claim\_ID.isnull())[0]  print('Index of rows with missing Claim\_ID - After :', idx\_missing\_claimid\_after)  # Next, treat the missing values in `Actual` (date of claim settlement)  # Find the rows with missing `Actual`  idx\_missing\_actual = np.where(df.Actual.isnull())[0]  print('Index of rows with missing Actual - Before :', idx\_missing\_actual)  # As `Actual` are dates of claim settlement, we are unable to do mean imputation since they are not numeric values.  # In this case, we opt for mode imputation.  # Find mode of `Actual` column and impute into the rows with missing `Actual` values  actual\_mode = df.Actual.mode()[0]  print('Mode of `Actual` column', actual\_mode) # 12/11/2021 0:00  for i in range(len(idx\_missing\_actual)):  df.loc[idx\_missing\_actual[i], 'Actual'] = actual\_mode  # Let's check if there is still any missing `Actual`  idx\_missing\_actual\_after = np.where(df.Actual.isnull())[0]  print('Index of rows with missing Actual - After :', idx\_missing\_actual\_after)  # Next, treat the missing values in `Terms` (Internal terms and condition code)  # Find the rows with missing `Terms`  idx\_missing\_terms = np.where(df.Terms.isnull())[0]  print('Index of rows with missing Terms - Before :', idx\_missing\_terms)  # As `Terms` are internal terms and condition code, we are unable to do mean imputation as they are not numeric values.  # In this case, we opt for mode imputation.  # Find mode of `Terms` column and impute into the rows with missing `Terms` values  terms\_mode = df.Terms.mode()[0]  print('Mode of `Terms` column', terms\_mode)  for i in range(len(idx\_missing\_terms)):  df.loc[idx\_missing\_terms[i], 'Terms'] = terms\_mode  # Let's check if there is still any missing `Terms`  idx\_missing\_terms\_after = np.where(df.Terms.isnull())[0]  print('Index of rows with missing Actual - After :', idx\_missing\_terms\_after) |

# **Question 3**

There are 3 other data preparation tasks are required for further data analysis.

Firstly, upon studying the data, I found that the date columns Actual, Planned and Created have date formats that are unstandardized and different. For the analysis, having these columns in a standardized date format is better. Hence, I converted these date columns into pd.datetime type with pandas.to\_datetime() function, with the standardized date format of: YYYY-mm-dd.

Secondly, duplicated data are redundant and are not useful for data analysis. Hence, i dropped duplicated rows with pandas.drop\_duplicates() function. In this step, 3 duplicated rows are dropped.

Thirdly, upon looking at the dataset, I found that in the column `Amount`, the values are string type and each value also have different decimal points. Also, one of the row has ‘OO’ as zeros instead of ‘00’ which is clearly a data entry error. Hence, to further clean and process this column, I converted all the values in this column to float type, with 2 decimal places since claim amounts (which are monetary values) should always have 2 decimal places. This is done with the functions: astype(float) and round(2). See Python code below:

|  |
| --- |
| # Q3  # Explain and implement three (3) other data preparation tasks required for further analysis of the data.  # (1) There are date columns with unstandardized date format, E.g the columns `Planned`, `Actual` and `Created` have date formats that are different from each other.  # To have a standardized date format, I convert date columns into pd.datetime type, with standardized date format of YYYY-mm-dd.  df['Planned'] = pd.to\_datetime(df['Planned'], format='%d/%m/%Y')  df['Actual'] = pd.to\_datetime(df['Actual'], format= '%d/%m/%Y %H:%M')  df['Created'] = pd.to\_datetime(df['Created'], format = '%Y%m%d')  # Check the columns after converting into pd.datetime standardized date format  display(df[['Planned','Actual','Created']])  # (2) Duplicated data are not useful/helpful for the analysis. So we check if there are any duplicated data and drop them.  print('Before drop duplicate data, no. of rows:', df.shape[0])  df = df.drop\_duplicates().reset\_index(drop=True)  print('After drop duplicate data, no. of rows:', df.shape[0])  # (3) Looking at the column `Amount`, the values are in strings and have different decimal points.  # Also, row 3697 has 'OO' as zeros instead of '00'  # As it is a claim amount, the datatype should be a float type, and have 2 dp.  # We will clean and convert all the values to float, with 2 decimal place as monetary claims always have only 2 dp.  df.loc[3697, 'Amount'] = '1762.00'  df['Amount'] = pd.to\_numeric(df['Amount']).round(2)  # After the above 3 data preparation tasks are done, view the final df  display(df) |

# **Question 4**

After the data was cleaned, I analysed the data and found 3 insights into the corporate claims processing of the insurance company.

Insight 1:

As there were many claims made by different claimants, I wanted to analyse and gain an insight into who were the top 5 claimants based on their total claim amount.

Figure 2: Top 5 claimants and their total claim amount

Icon

Description automatically generated

A bar chart was plotted in figure 2 to display the top 5 claimants and their total claim amount out of 3058 claimants. As seen from the bar chart, among all the claimants available in the dataset, the top 5 claimants are Hayley Dennis, David Anthony, Robert Martin, Troy Phillips and Taylor Campbell (in the order highest to lowest claim amount)

Insight 2:

As there 66 unique Terms in the dataset, I wanted to analyse and gain insight of which were the top 10 most frequent terms code that appear among the claims.

Figure 3: Top 10 most frequent terms code

Chart, bar chart

Description automatically generated

A bar chart was plotted in figure 3 to display the top 10 most frequent terms code that appeared in the dataset. As observed, the Top 10 most frequent terms code (most to least frequent)are AD23, AB27, CB91, CD89, BE01, AE14, De16, DD29, CB37, EC05. AD23 is the highest, appearing 10106 times among the 24,210 claims in the dataset.

Insight 3:

As there are 2 unique Category code in the dataset, I wanted to analyse and gain insight of which of these 2 is the most frequently appearing category.

Figure 4: Category Code count among the claims

Chart

Description automatically generated

A bar chart was plotted in figure 4 to display the total count of each of the Category that appear in the dataset. As seen, ‘AT’ is the most frequent category, with almost 99% (24205 out of 24210) of the claims having ‘AT’ category.

See below for Python code.

|  |
| --- |
| # Q4  # Analyse the data and describe three (3) insights into the corporate claims processing of the insurance company, with at least one (1) supporting visualization created to illustrate each insight.  # Analysis 1: Determine who are the top 5 claimants and their total claim amount (overall)  get\_top\_5\_claimants = df.groupby('Name').aggregate({'Amount':'sum', 'Name': 'max'})  get\_top\_5\_claimants = get\_top\_5\_claimants.sort\_values(by='Amount', ascending=False)[:5]  display(get\_top\_5\_claimants)  ax = get\_top\_5\_claimants[['Amount','Name']].plot(kind='bar', y='Amount', x='Name', figsize=(10,5), title='Top 5 claimants and their total claim amount', rot=0)  ax.set\_ylabel('Claim amount (in millions)')  plt.show()  # Insight 1: Top 5 claimants are Hayley Dennis, David Anthony, Robert Martin, Troy Phillips, Taylor Campbell  # Analysis 2: Determine which is the top 10 most frequent Terms code among the claims  get\_top\_10\_claim\_term = df.groupby('Terms').size().sort\_values(ascending=False)[:10]  ax2 = get\_top\_10\_claim\_term.plot(kind='bar', title = 'Top 10 most frequent terms code among the claims', figsize=(10,5), rot=0)  ax2.bar\_label(ax2.containers[0])  ax2.set\_ylabel('Count')  plt.show()  # Insight 2: Top 10 most frequent terms code (most to least frequent): AD23, AB27, CB91, CD89, BE01, AE14, De16, DD29, CB37, EC05  # Analysis 3: Determine which is the most frequent Category code among the claims  get\_top\_10\_claim\_cat = df.groupby('Category').size().sort\_values(ascending=False)[:10]  ax3 = get\_top\_10\_claim\_cat.plot(kind='bar', title = 'Count of category code among the claims', figsize=(10,5), rot=0)  ax3.bar\_label(ax3.containers[0])  ax3.set\_ylabel('Count')  plt.show()  # Insight 3: Top most frequent category code: AT |

# **Question 5**

Before we can perform linear regression modelling to predict the delay in days (between Planned and Actual date), we need to do further data pre-processing.

There are a few pre-processing steps to be done.

1. Create a new column `delay\_in\_days` which represents the delay in days between the planned and actual date as this was not previously created / available in the existing dataset.As we have converted the columns `Planned` and `Actual` into pd.datetime type, we can directly take the difference between these 2 columns to get the `delay\_in\_days` value.
2. Split df into X (containing predictors) and Y (target variable) To do so, we just drop the target variable (delay\_in\_days) from the df, to represent X and set Y to be the column with the target variable (delay in days)
3. Drop Claim\_ID, Policy No, date and name columns in X as there are too many distinct values and not meaningful predictors. To do so, we can just input these column names into the pandas.DataFrame.drop() function.
4. Before passing the dataframe X into the linear regression model, one hot encoding on the remaining categorical variables such as `Terms`, `Category`, `Type`, `Paid` needs to be carried out. To do so, we can use the pandas.to\_dummies function.
5. Next, we have to split X, y into train and test sets (75% train, 25% Test) so that we can train the model with the train set, and evaluate the performance of the linear regression model on the test set. To do so, the train\_test\_split() function from sklearn is used.

After completing the 5 pre-processing steps, a linear regression model was fitted on the training set using the following codes:

|  |
| --- |
| model = LinearRegression()  model.fit(X\_train, y\_train) |

See python code below:

|  |
| --- |
| # Q5  # Perform linear regression modelling to predict the delay in days (between the Planned and Actual date) in processing the claims, explaining the approach taken, including any further data pre-processing needed for modelling.  # Before we can perform linear regression modelling to predict the delay in days (between Planned and Actual date), we need to do further data pre-processing.  # There are a few preprocessing steps:  # (1) create new column `delay\_in\_days` which represents the delay in days between planned and actual date  df['delay\_in\_days'] = (df['Actual'] - df['Planned']).dt.days  df['delay\_in\_days'] = df['delay\_in\_days'].astype(int)  # Positive value --> claim is delayed / later than planned date.  # Negative value --> Claim is earlier than planned date.  # (2) Split df into X (containing predictors) and Y (target variable)  X = df.drop('delay\_in\_days', axis=1)  Y = df['delay\_in\_days']  # (3) Drop date and name columns in X (as there are too many distinct values and not very meaningful predictors)  X = df.drop(['Actual','Planned','Created', 'Name'], axis=1)  # (4) Do One hot encoding on the remaining categorical variables  X = pd.get\_dummies(data=X)  # (5) Split X, y into train and test sets (75% train, 25% Test)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=0)  X\_train = X\_train.reset\_index(drop=True)  X\_test = X\_test.reset\_index(drop=True)  # Perform the linear regression modelling  model = LinearRegression()  model.fit(X\_train, y\_train)  # Get the variables required for the equation of the LR modelling and results  # summary of the model  print('model intercept :', model.intercept\_)  print('model coefficients : ', model.coef\_)  # construct and print the linear regression formula  def get\_regression\_formula(model, X):  formula = [f"{model.intercept\_:.2f} "]  for i, var in enumerate(X.columns.values):  coef = model.coef\_[i]  coef\_abs = abs(coef)  if coef\_abs < 0.1:  continue  formula.append(f"{'+' if coef > 0 else '-'} {coef\_abs:.2f} \* {var} ")  return f"delay\_in\_days = {''.join(formula)}"  print(get\_regression\_formula(model, X\_train))  # print the results - train and test set MSE  y\_train\_preds = model.predict(X\_train)  y\_test\_preds = model.predict(X\_test)  print('MSE train set:', round(mean\_squared\_error(y\_train, y\_train\_preds),2))  print('MSE test set', round(mean\_squared\_error(y\_test, y\_test\_preds),2)) |

# **Question 6**

After completing the model as in question 5, we can compute the performance of the model. We will use the mean\_squared\_error as the performance metric as this is a regression problem.

Using the following codes:

|  |
| --- |
| y\_train\_preds = model.predict(X\_train)  y\_test\_preds = model.predict(X\_test)  print('MSE train set:', round(mean\_squared\_error(y\_train, y\_train\_preds), 2))  print('MSE test set', round(mean\_squared\_error(y\_test, y\_test\_preds),2)) |

The MSE of the train and test sets are 96.53 and 107.38 respectively. It seems that the MSE of the test set is close to that of the train set, hence the model is performing relatively well.

To obtain the values of the coefficients and intercept of the linear regression model, the following codes can be used:

|  |
| --- |
| print('model intercept :', model.intercept\_)  print('model coefficients : ', model.coef\_) |

The resultant intercept and coefficient of the model is given as:

|  |
| --- |
| model intercept : 8.022096236023344  model coefficients : [-9.21694629e-05 -7.80762321e+01 7.80762321e+01 -8.34796705e+01  8.34796705e+01 -3.47637651e-01 1.70473413e-10 -4.23468234e-01  -9.59052908e+00 6.61243236e+01 -3.04385058e+00 1.85070427e-01  6.13544952e+01 2.86295943e-01 -7.94830514e+00 -1.14621256e+00  -1.98708847e+01 6.08634645e+01 -2.79807377e-01 1.75594192e+01  2.14959524e+00 6.21592885e+00 -1.45654506e+01 3.31234424e-02  -7.24293318e+00 1.05658493e+01 -8.29398919e+00 -8.25081778e+00  -3.53560662e+00 -4.35309584e-02 -2.14654960e-11 7.36949745e-01  -4.01719927e-01 -6.56718779e+00 7.96164252e+00 1.15353618e+00  -5.85853449e+00 -2.93122840e+01 2.70625882e+00 2.69059622e+00  -1.15995134e+00 2.78126925e+00 -5.87102430e-01 -1.28785871e-12  6.72736897e+00 -7.36938951e+00 3.78196809e+01 -1.93743176e+00  -5.68347777e+00 -3.55585434e+00 -5.59414060e+00 -4.31315546e+01  -3.54046194e-01 -8.37044407e+00 -1.01842649e+01 -5.36364596e+00  1.73432316e+00 -1.85648142e+01 3.00204306e-13 -9.77570778e-03  7.75864948e+00 1.04220481e+01 1.36664385e+00 -1.55036375e+01  -3.73785853e+00 4.19048614e+00 2.27915679e+00 -9.54404380e+00  -2.83358274e+00 -3.90154454e+01 -6.44296467e+00 3.72830257e-01  -3.72830257e-01 -7.70742590e-01 7.70088473e-01 -4.14729095e+00  0.00000000e+00 3.77511480e+00 3.72830257e-01] |

And the final linear regression equation is given as:

delay\_in\_days = 8.02 - 0.00 \* Amount - 78.08 \* Paid\_No + 78.08 \* Paid\_Yes - 83.48 \* Category\_AT + 83.48 \* Category\_XT - 0.35 \* Terms\_AA01 + 0.00 \* Terms\_AA31 - 0.42 \* Terms\_AA75 - 9.59 \* Terms\_AB09 + 66.12 \* Terms\_AB12 - 3.04 \* Terms\_AB27 + 0.19 \* Terms\_AD23 + 61.35 \* Terms\_AD49 + 0.29 \* Terms\_AD58 - 7.95 \* Terms\_AD59 - 1.15 \* Terms\_AE14 - 19.87 \* Terms\_AE69 + 60.86 \* Terms\_AE81 - 0.28 \* Terms\_AE92 + 17.56 \* Terms\_BA01 + 2.15 \* Terms\_BA02 + 6.22 \* Terms\_BB38 - 14.57 \* Terms\_BC27 + 0.03 \* Terms\_BC75 - 7.24 \* Terms\_BC81 + 10.57 \* Terms\_BD15 - 8.29 \* Terms\_BD99 - 8.25 \* Terms\_BE01 - 3.54 \* Terms\_BE30 - 0.04 \* Terms\_BE78 - 0.00 \* Terms\_BE82 + 0.74 \* Terms\_CA18 - 0.40 \* Terms\_CA90 - 6.57 \* Terms\_CB04 + 7.96 \* Terms\_CB21 + 1.15 \* Terms\_CB37 - 5.86 \* Terms\_CB54 - 29.31 \* Terms\_CB89 + 2.71 \* Terms\_CB91 + 2.69 \* Terms\_CB99 - 1.16 \* Terms\_CC03 + 2.78 \* Terms\_CC48 - 0.59 \* Terms\_CD89 - 0.00 \* Terms\_CD97 + 6.73 \* Terms\_CE21 - 7.37 \* Terms\_CE32 + 37.82 \* Terms\_CE76 - 1.94 \* Terms\_DA17 - 5.68 \* Terms\_DA57 - 3.56 \* Terms\_DA58 - 5.59 \* Terms\_DB29 - 43.13 \* Terms\_DB75 - 0.35 \* Terms\_DC36 - 8.37 \* Terms\_DC39 - 10.18 \* Terms\_DC90 - 5.36 \* Terms\_DD19 + 1.73 \* Terms\_DD24 - 18.56 \* Terms\_DD29 + 0.00 \* Terms\_DD52 - 0.01 \* Terms\_DE16 + 7.76 \* Terms\_DE49 + 10.42 \* Terms\_DE61 + 1.37 \* Terms\_EA68 - 15.50 \* Terms\_EB64 - 3.74 \* Terms\_EC05 + 4.19 \* Terms\_EC21 + 2.28 \* Terms\_EC44 - 9.54 \* Terms\_ED32 - 2.83 \* Terms\_ED62 - 39.02 \* Terms\_ED69 - 6.44 \* Terms\_EE56 + 0.37 \* Region\_FVS - 0.37 \* Region\_LOC - 0.77 \* Type\_L001 + 0.77 \* Type\_L002 - 4.15 \* Type\_L003 - 0.00 \* Type\_L004 + 3.78 \* Type\_L005 + 0.37 \* Type\_0001