**Lab 04**

|  |  |
| --- | --- |
| Name: | 劉邦彬 |
| Student ID: | R11631033 |
| Total Score: |  |

**Note:**

Most of the explanations in this lab is optional. However, giving reasonable explanations to your answer or programs will earn you partial credits when your answer is incorrect.

1. **Multiple Choice (10 points, 5 points each question)**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Answer | Explanation (Optional) | Score |
| 1 | (c) |  |  |
| 2 | (a) |  |  |

1. **Programming Exercise (30 points)**

|  |  |  |
| --- | --- | --- |
| # | Explanation (Mandatory for #1) | Score |
| 1 | What percentage of legendary Pokémon have an Attack value greater than 150? legendary = (df.Legendary == True)  legendary\_150 = (df.Legendary == True) & (df.Attack> 150)  percent = (len(df[legendary\_150].index)/len(df[legendary].index))\*100  print(f"percentage of legendary Pokémon have an Attack value greater than 150 is " , percent)  Output:  percentage of legendary Pokémon have an Attack value greater than 150 is 12.307692307692308 What percentage of non-legendary Pokémon have an Attack value greater than 150?  non\_legendary = (df.Legendary == False)  non\_legendary\_150 = (df.Legendary == False) & (df.Attack >150)  percent2 = (len(df[non\_legendary\_150].index)/len(df[non\_legendary].index))\*100  print("percentage of non-legendary Pokémon have an Attack value greater than 150 is" ,percent2)  Output:  percentage of non-legendary Pokémon have an Attack value greater than 150 is 1.3605442176870748  Provide a brief description of your findings.  legendary Pokémon is strong  Identify the Pokémon that appears as an outlier in the lower right corner.  #4  x = df['Defense']  y = df['Attack']  plt.scatter(x,y)  df.loc[(df['Defense'] > 200) & (df['Attack'] < 25)]  Output: |  |
| 2 | def pokemon\_type1\_count(pokemon):      """      compute the number of pokemons for each type1      Args:          pokemon (pd.DataFrame) : pokemon dataframe      Returns:          Dict[str, int] : dictionary of pokemon types and their counts      """      ans = {}      # TODO\_2      value = pokemon['Type 1']      ans = value.value\_counts().to\_dict()      return ans  Output: |  |
| 3 | def average\_attack\_type(pokemon):      """      compute the average attack for each type1      Args:          pokemon (pd.DataFrame) : pokemon dataframe      Returns:          Dict[str, int] : dictionary of pokemon types and their average attack      """      ans = {}      # TODO\_3      aver\_atk1 = pokemon.groupby('Type 1' )['Attack'].sum()      aver\_atk2 = pokemon.groupby('Type 2' )['Attack'].sum()      count1 = pokemon.groupby('Type 1' )['#'].count()      count2 = pokemon.groupby('Type 2' )['#'].count()      avg = aver\_atk1 + aver\_atk2      cnt\_avg = count1 + count2      a = avg/cnt\_avg      ans = a.to\_dict()      return ans  Output: |  |

1. **Data Analysis and Visualization for Climate Change (60 points)**

|  |  |  |
| --- | --- | --- |
| # | Explanation | Score |
| 1 | df\_state = pd.read\_csv('GlobalLandTemperaturesByState.csv', delimiter=',', encoding='utf-8')  for country in df\_state['Country'].unique():      print(country)  Output: |  |
| 2 | countries\_to\_remove = ['Brazil', 'Russia']  def cleanse\_country\_data(df):      """      Remove countries in the countries\_to\_remove list from the dataframe df\_state      and simplify the country names that include additional abbreviation.      kwargs:          country\_data (pd.DataFrame) : the input dataframe to preprocess      return:          pd.DataFrame : the preprocessed dataframe      """      # TODO\_2.1      df['Country'] = df['Country'].str.replace('United States (US)', 'United States')      df['Country'] = df['Country'].str.replace('Canada (CA)', 'Canada')      to\_remove = df[df['Country'].isin(countries\_to\_remove) ].index      df.drop(to\_remove, inplace=True)      return df  Output(i):      Input(ii):  def drop\_missing\_values(df):      """      Drop rows with at least one missing value from an input dataframe.      args:          df (pd.DataFrame) : an input dataframe      returns:          pd.DataFrame : a subset of df where rows with missing values in any column are removed.      """      # TODO\_2.2      df= df.dropna()      return df  Output(i): |  |
| 3 | # Show the key statistics such as the minimum value, maximum value, average (mean),  # and standard deviation of the "AverageTemperature" column in df\_state.  # TODO\_3  stat = df\_state['AverageTemperature'].describe()  print('minimum value' , stat['min'])  print('maximum value' , stat['max'])  print('average (mean)' , stat['mean'])  print('standard deviation' , stat['std'])  print(stat)  Output: |  |
| 4 | def remove\_outliers(df, col):      """      Remove any row whose data at a given column is considered outlier according to the IQR rule.          args:          df (pd.DataFrame) : an input dataframe where outlier rows should be removed          col (str) : the column name to check for outlier      return:          pd.DataFrame : a subset of the input dataframe after outlier rows are removed      """      # TODO\_4      Q1 = df[col].quantile(0.25)      Q3 = df[col].quantile(0.75)      IQR = Q3 - Q1      df = (df.query(f"not ({col} <= {Q1-1.5\*IQR}) | ({col} >= {Q3+1.5\*IQR})"))      return df  Output: |  |