Data Analysis

Practical Works 3
Multivariate regression

Younes Makhlouf
GL3

1. Forming the matrix X as in (3) by reading data directly from the Excel file and adding the intercept column filled by 1s to it.

```python
import numpy as np
import pandas as pd
from scipy.stats import f

# 1
file_path = 'TutorialCigarette.xlsx'

df = pd.read_excel(file_path, header=1, sheet_name="Feuil1",
                   usecols=['Tar (mg)', 'Nicotine (mg)', 'Weight (g)', 'Carbon Monoxide(mg)'])
# Renaming the columns to simplify
df.rename(columns={'Tar (mg)': 'tar', 'Nicotine (mg)': 'nicotine', 'Weight (g)': 'weight',
                   'Carbon Monoxide(mg)': 'carbon_monoxide'}, inplace=True)

print(df.head())

# Selecting the predictor variables
X = df[['weight', 'tar', 'nicotine']].copy()

# Adding a column of ones to account for the intercept (β4)
X['intercept'] = 1
X = X[['weight', 'tar', 'nicotine', 'intercept']]

# Converting the DataFrame to a NumPy array for later matrix operations
X_matrix = X.values
y = df['carbon_monoxide'].values

print(X_matrix)
```

```
[[ 0.9853 14.1     0.86    1.    ]
 [ 1.0938 16.      1.06    1.    ]
 [ 1.165  29.8     2.03    1.    ]
 [ 0.928   8.      0.67    1.    ]
 [ 0.9462  4.1     0.4     1.    ]
 [ 0.8885 15.      1.04    1.    ]
 [ 1.0267  8.8     0.76    1.    ]
 [ 0.9225 12.4     0.95    1.    ]
 [ 0.9372 16.6     1.12    1.    ]
 [ 0.8858 14.9     1.02    1.    ]
 [ 0.9643 13.7     1.01    1.    ]
 [ 0.9316 15.1     0.9     1.    ]
 [ 0.9705  7.8     0.57    1.    ]
 [ 1.124  11.4     0.78    1.    ]
 [ 0.8517  9.      0.74    1.    ]
 [ 0.7851  1.      0.13    1.    ]
 [ 0.9186 17.      1.26    1.    ]
 [ 1.0395 12.8     1.08    1.    ]
 [ 0.9573 15.8     0.96    1.    ]
 [ 0.9106  4.5     0.42    1.    ]
 [ 1.007  14.5     1.01    1.    ]
 [ 0.9806  7.3     0.61    1.    ]
 [ 0.9693  8.6     0.69    1.    ]
 [ 0.9496 15.2     1.02    1.    ]
 [ 1.1184 12.      0.82    1.    ]]
```

2. Determining the slope vector β

```
# 2
X_transpose = X.T
beta = np.linalg.inv(X_transpose.dot(X)).dot(X_transpose).dot(y)
print("Slope vector β:", beta)
```

```
Slope vector β: [-0.13048185  0.96257386 -2.63166111  3.20219002]
```

3. Computing the prediction vector y∗

```
# 3
y_pred = X.dot(beta)
print("Predicted values of y:", y_pred)
```

```
Predicted values of y: 0      14.382689
1      15.671090
2      26.392608
3       9.018481
4       5.972616
5      14.787937
6       9.538812
7      12.517658
8      16.111168
9      14.744665
10     13.605650
11     15.247003
12      9.083587
13     11.976175
14      9.806794
15      3.720207
16     16.130192
17     12.545305
18     15.759552
19      6.309658
20     14.370138
21      8.495715
22      9.538003
23     15.025113
24     12.449183
```

4. Building up the ANOVA table.

```python
# 4
# Given
# y: the actual values of the dependent variable
# y_pred: the predicted values based on the regression model
# X: the matrix of predictors

n = len(y)  # Number of observations
p = X.shape[1] - 1  # Number of predictors, excluding the intercept

# Computing SST, SSR, and SSE
SST = np.sum((y - np.mean(y)) ** 2)
SSR = np.sum((y_pred - np.mean(y)) ** 2)
SSE = np.sum((y - y_pred) ** 2)

# Calculating the degrees of freedom
dfR = p
dfE = n - p - 1
dfT = n - 1

# Mean Squares
MSR = SSR / dfR
MSE = SSE / dfE

# F-statistic
F = MSR / MSE

P_value = f.sf(F, dfR, dfE)  # This computes the P-value for the F-statistic

# Constructing the ANOVA table
anova_table = pd.DataFrame(
    {"Source": ["Regression", "Residual", "Total"], "Sum of Squares": [SSR, SSE, SST], "df": [dfR, dfE, dfT],
     "Mean Square": [MSR, MSE, np.nan], "F": [F, np.nan, np.nan], "P-value": [P_value, np.nan, np.nan]})

print(anova_table)
```

	Source	Sum of Squares	df	Mean Square	F	P-value
0	Regression	495.257814	3	165.085938	78.983834	1.328810e-11
1	Residual	43.892586	21	2.090123	NaN	NaN
2	Total	539.150400	24	NaN	NaN	NaN

5. Deduce the determination coefficient R2.

```
# 5
R_squared = 1 - (SSE / SST)
print("Coefficient of determination (R^2):", R_squared)
```

```
Coefficient of determination (R^2): 0.9185893479475058
```

Comment: An R^2 value of 0.91859 is quite high, suggesting that approximately 91.86% of the variance in the dependent variable, carbon monoxide content, can be explained by the independent variables (weight, nicotine, sand tar) in our model. This is a powerful indication that the model fits the data well and that the selected predictors are strongly related to the outcome variable.

6. Compute the correlation matrix of the predictor variables.

```
# 6
predictors = df[['weight', 'tar', 'nicotine']]

# Compute the correlation matrix
correlation_matrix = predictors.corr()

print("Correlation matrix of the predictor variables:")
print(correlation_matrix)
```

```
Correlation matrix of the predictor variables:
            weight       tar  nicotine
weight    1.000000  0.490765  0.500183
tar       0.490765  1.000000  0.976608
nicotine  0.500183  0.976608  1.000000
```

Given the very high correlation between `tar` and `nicotine` (0.976608 very close to 1), we might consider running the regression analysis without one of these variables to see how it affects the model's performance.

7. Propose a reduced multi-linear model and determine its parameters and its ANOVA analysis.

```python
# 7
# Selecting the predictors for the reduced model and the response variable
X_reduced = df[['weight', 'tar']]
X_reduced['intercept'] = 1  # Adding the intercept term

# Convert to a NumPy array for matrix operations
X_reduced = X_reduced[['intercept', 'weight', 'tar']].values  # Ensure the intercept is the first column

y = df['carbon_monoxide'].values  # The dependent variable remains unchanged

# Calculating the regression coefficients for the reduced model
beta_reduced = np.linalg.inv(X_reduced.T.dot(X_reduced)).dot(X_reduced.T).dot(y)

print("Coefficients for the reduced model:", beta_reduced)

# Compute predictions for the reduced model
y_pred_reduced = X_reduced.dot(beta_reduced)

# Calculate residues
residuals = y - y_pred_reduced
```

```python
# Calculate sums of squares
SST = np.sum((y - np.mean(y)) ** 2)
SSR = np.sum((y_pred_reduced - np.mean(y)) ** 2)
SSE = np.sum(residuals ** 2)

# Degrees of freedom
n = len(y)  # Number of observations
p = 2  # Number of predictors in the reduced model
dfR = p
dfE = n - p - 1
dfT = n - 1

# Mean squares
MSR = SSR / dfR
MSE = SSE / dfE

# F-statistic
F = MSR / MSE
```

```python
# P-value for the F-statistic
P_value = f.sf(F, dfR, dfE)

# ANOVA Table for the reduced model
anova_table_reduced = pd.DataFrame(
    {"Source": ["Regression", "Residual", "Total"], "Sum of Squares": [SSR, SSE, SST], "df": [dfR, dfE, dfT],
     "Mean Square": [MSR, MSE, np.nan], "F": [F, np.nan, np.nan], "P-value": [P_value, np.nan, np.nan]})

print(anova_table_reduced)
```

```
Coefficients for the reduced model: [ 3.11433366 -0.42287387  0.8041891 ]
       Source  Sum of Squares  df  Mean Square           F       P-value
0  Regression      494.306381   2   247.153190  121.250733  1.318076e-12
1    Residual       44.844019  22     2.038365         NaN           NaN
2       Total      539.150400  24          NaN         NaN           NaN
```