Exercise 3

In this exercise, you will analyse a dataset obtained from the London transport system (TfL). The data is in a filled called `tfl_readership.csv` (comma-separated-values format). As in Exercise 2, we will load and view the data using `pandas`.

In [1]:
```
# If you are running this on Google Colab, uncomment and run the followin
# from google.colab import drive
# drive.mount('/content/drive')
```

In [2]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [3]:
```
# Load data
df_tfl = pd.read_csv('tfl_ridership.csv')
# If running on Google Colab change path to '/content/drive/MyDrive/IB-Da

df_tfl.head(13)
```

Out[3]:

	Year	Period	Start	End	Days	Bus cash (000s)	Bus Oyster PAYG (000s)	Bus Contactless (000s)	Bus One Day Bus Pass (000s)	Bus Day Travelcard (000s)
0	2000/01	P 01	01 Apr '00	29 Apr '00	29d	884	0	0	210	231
1	2000/01	P 02	30 Apr '00	27 May '00	28d	949	0	0	214	205
2	2000/01	P 03	28 May '00	24 Jun '00	28d	945	0	0	209	221
3	2000/01	P 04	25 Jun '00	22 Jul '00	28d	981	0	0	216	241
4	2000/01	P 05	23 Jul '00	19 Aug '00	28d	958	0	0	225	248
5	2000/01	P 06	20 Aug '00	16 Sep '00	28d	984	0	0	243	236
6	2000/01	P 07	17 Sep '00	14 Oct '00	28d	1001	0	0	205	216
7	2000/01	P 08	15 Oct '00	11 Nov '00	28d	979	0	0	199	221
8	2000/01	P 09	12 Nov '00	09 Dec '00	28d	971	0	0	184	212
9	2000/01	P 10	10 Dec '00	06 Jan '01	28d	912	0	0	192	211
10	2000/01	P 11	07 Jan '01	03 Feb '01	28d	943	0	0	193	186
11	2000/01	P 12	04 Feb '01	03 Mar '01	28d	975	0	0	194	210
12	2000/01	P 13	04 Mar '01	31 Mar '01	28d	974	0	0	186	204

13 rows × 26 columns

Each row of our data frame represents the average daily ridership over a 28/29 day period for various types of transport and tickets (bus, tube etc.). We have used the `.head()` command to display the top 13 rows of the data frame (corresponding to one year). Focusing on the "Tube Total" column, notice the dip in ridership in row 9 (presumably due to Christmas/New Year's), and also the slight dip during the summer (rows 4,5).

In [4]:
```
#df_tfl.sample(3)  #random sample of 3 rows
df_tfl.tail(3)  #last 3 rows
```

Out[4]:

	Year	Period	Start	End	Days	Bus cash (000s)	Bus Oyster PAYG (000s)	Bus Contactless (000s)	Bus One Day Bus Pass (000s)	Bus Day Travelcard (000s)
242	2018/19	P 09	11 Nov '18	08 Dec '18	28d	0	1110	1089	0	41
243	2018/19	P 10	09 Dec '18	05 Jan '19	28d	0	1001	949	0	38
244	2018/19	P 11	06 Jan '19	02 Feb '19	28d	0	1036	1075	0	30

3 rows × 26 columns

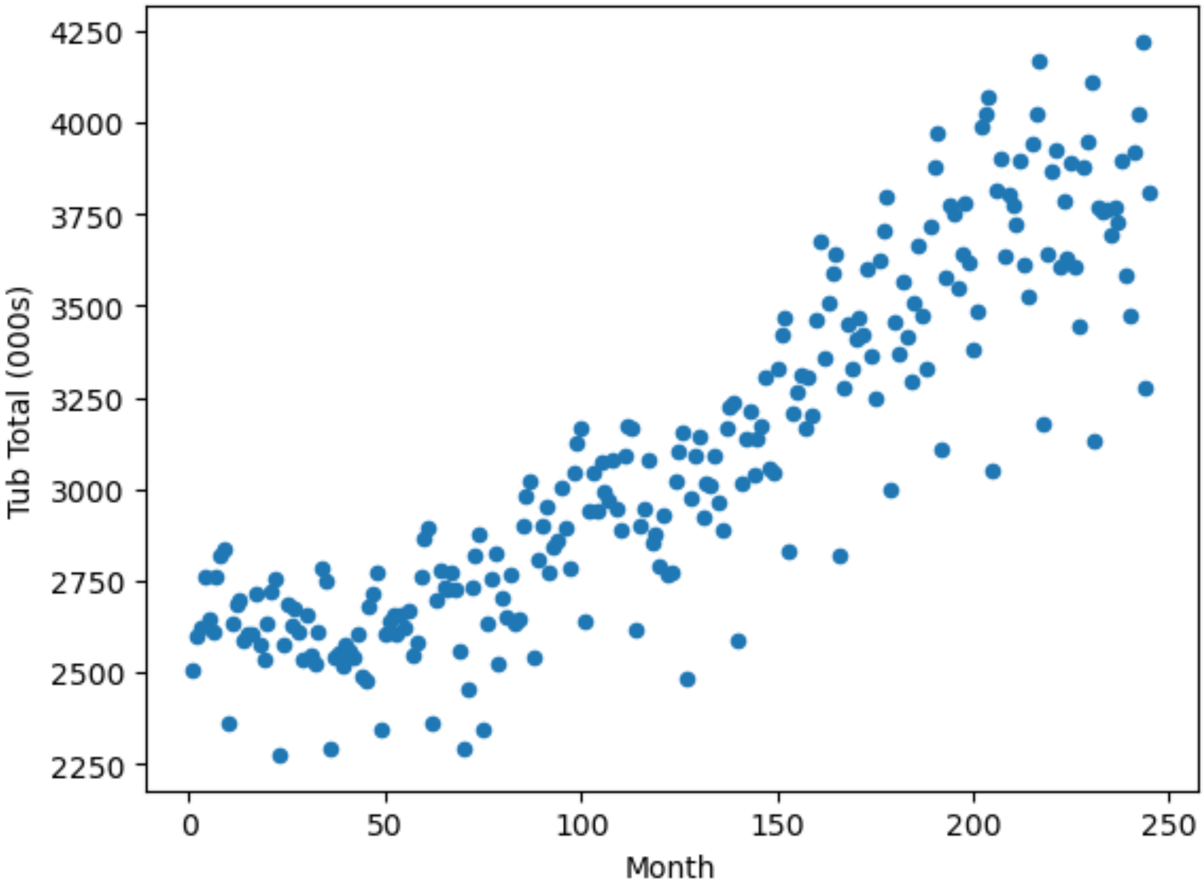
The dataframe contains $N = 245$ counting periods (of 28/29 days each) from 1 April 2000 to 2 Feb 2019. We now define a numpy array consisting of the values in the ' Tube Total (000s)' column:

In [5]:
```
yvals = np.array(df_tfl['Tube Total (000s)'])
N = np.size(yvals)
xvals = np.linspace(1,N,N) #an array containing the values 1,2....,N
```

We now have a time series consisting of points (x_i, y_i), for $i = 1, \ldots, N$, where y_i is the average daily rideship in counting period $x_i = i$.

3a) Plot the data in a scatterplot

In [6]:
```
plt.scatter(xvals, yvals, s=20)

plt.xlabel('Month')
plt.ylabel('Tub Total (000s)')
plt.savefig('TimevsTubTotal.pdf', bbox_inches = 'tight')
plt.show()
```



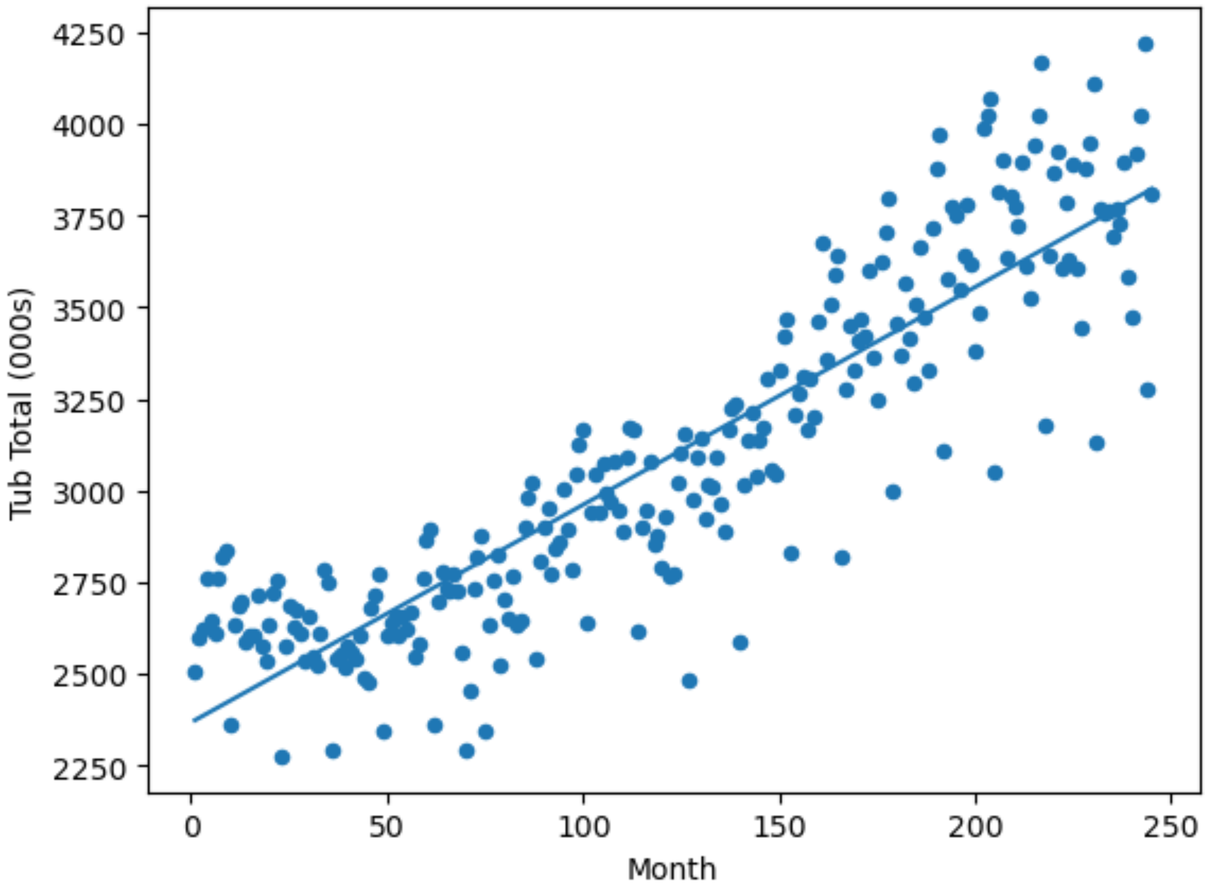
3b) Fit a linear model $f(x) = \beta_0 + \beta_1 x$ to the data

- Print the values of the regression coefficients β_0, β_1 determined using least-squares.
- Plot the fitted model and the scatterplot on the same plot.
- Compute and print the **MSE** and the R^2 coefficient for the fitted model.

All numerical outputs should be displayed to three decimal places.

In [7]:
```
def polyreg(x, y, k):
    x_priv = x
    y_priv = y

    all_ones = np.ones(np.shape(x_priv))

    columns = [all_ones]
    for power in range(1, k + 1):
        columns.append(x_priv ** power)
    X = np.column_stack(columns)

    y_values = y_priv

    beta = np.linalg.lstsq(X, y_values, rcond=None)[0]
    fit = X.dot(beta)
    resid = y_values - fit
    return [beta, fit, resid]

_, fit_0, _ = polyreg(xvals, yvals, 0)
beta_priv, fit_priv, resid_priv = polyreg(xvals, yvals, 1)

print("coefficients:", beta_priv)

SSE_0 = np.linalg.norm(yvals - fit_0)**2
SSE = np.linalg.norm(yvals - fit_priv)**2
MSE = SSE/(np.size(yvals))
R_square = np.round(1- SSE/SSE_0, decimals = 4)

plt.xlabel('Month')
plt.ylabel('Tub Total (000s)')
plt.scatter(xvals, yvals, s=20)
plt.plot(xvals, fit_priv)
plt.show()

print("MSE:", MSE)
print("R_square:", R_square)
```

```
coefficients: [2367.38176648    5.93899012]
```

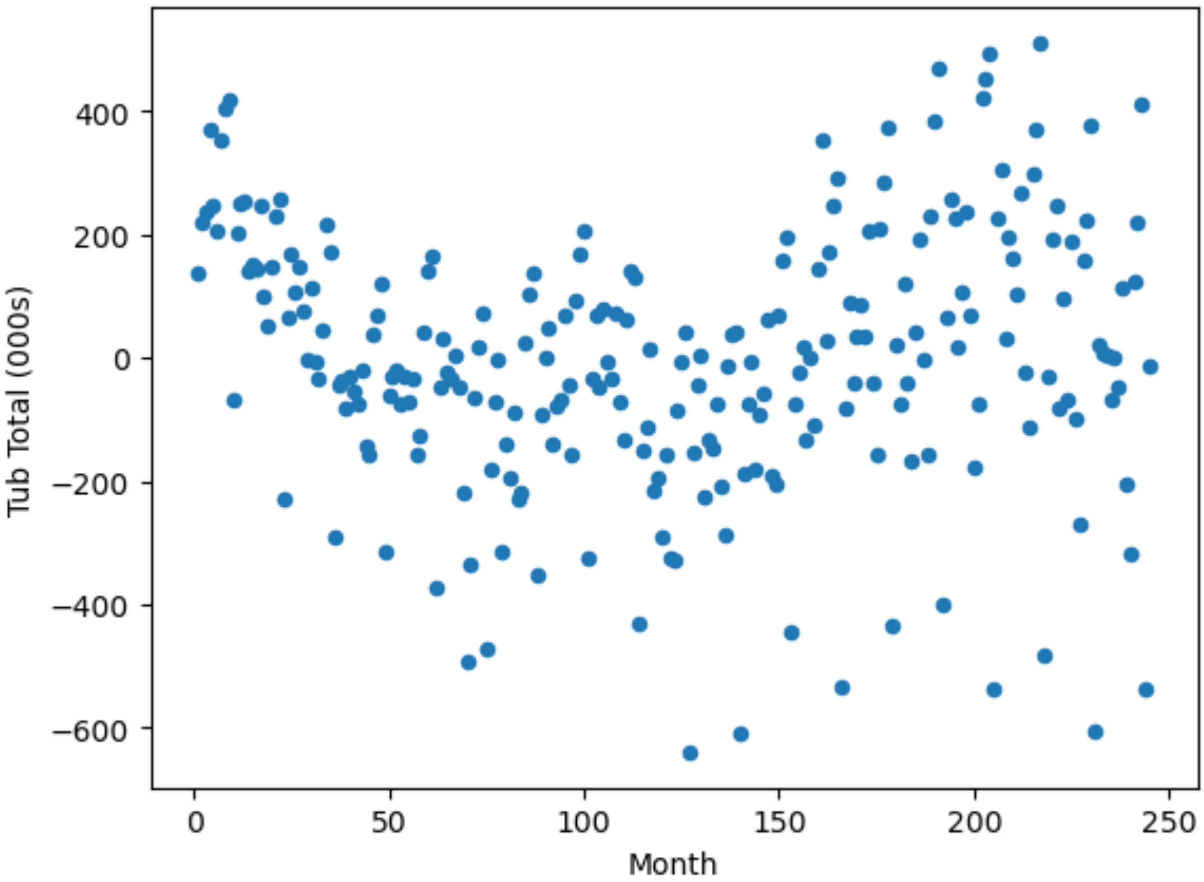


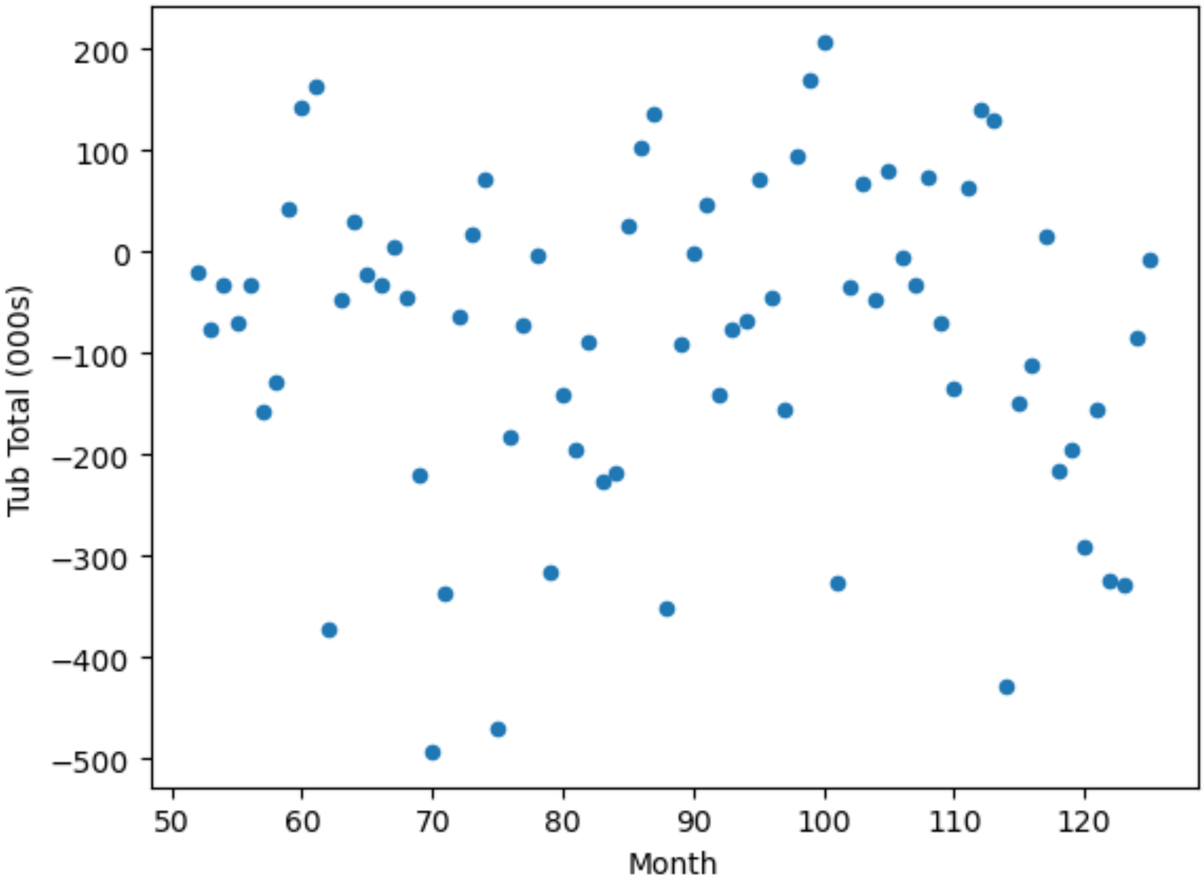

```
MSE: 45323.63592122835
R_square: 0.7956
```

3c) Plotting the residuals

- Plot the residuals on a scatterplot
- Also plot the residuals over a short duration and comment on whether you can discern any periodic components.

In [8]:
```
plt.xlabel('Month')
plt.ylabel('Tub Total (000s)')
plt.scatter(xvals, resid_priv, s=20)
plt.show()

plt.xlabel('Month')
plt.ylabel('Tub Total (000s)')
plt.scatter(xvals[51:125], resid_priv[51:125], s=20)
plt.show()
```



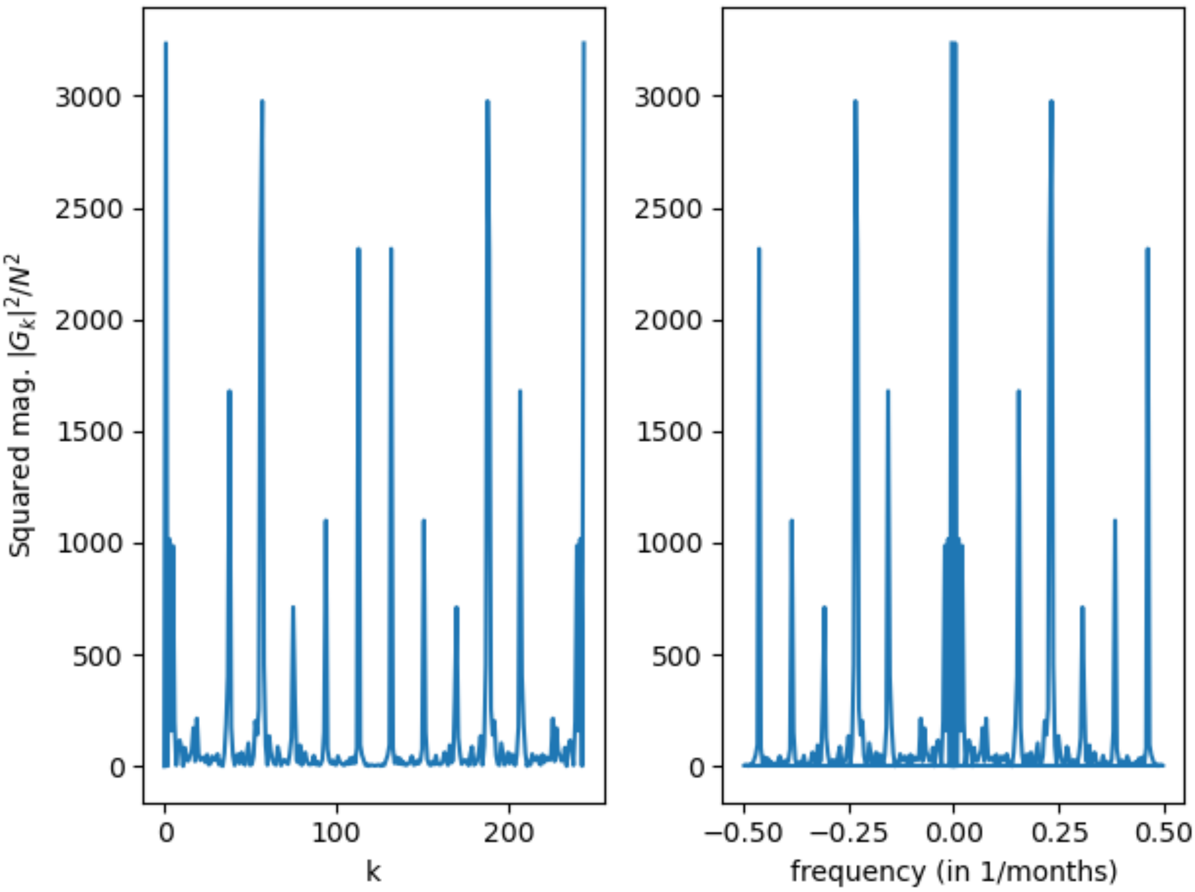


Somehow it looks like trignometric wave

3d) Periodogram

- Compute and plot the peridogram of the residuals. (Recall that the periodogram is the squared-magnitude of the DFT coefficients.)
- Identify the indices/frequencies for which the periogram value exceeds **50%** of the maximum.

In [9]:
```
N = np.size(xvals)
T = xvals[100] - xvals[99]  # Months

pgram = np.abs(np.fft.fft(resid_priv, N)/N)**2
indices = np.linspace(0, (N-1), num = N)
freqs_in_hz = np.fft.fftfreq(N)/T
freqs_in_rads = freqs_in_hz*2*math.pi

plt.subplot(121)
plt.plot(indices, pgram)
plt.xlabel('k')
plt.ylabel('Squared mag. $|G_k|^2/N^2$')
plt.subplot(122)
plt.plot(freqs_in_hz, pgram)
plt.xlabel('frequency (in 1/months)')  # Since units of T is years
plt.savefig('DFT_andPeridogram.pdf', bbox_inches = 'tight')
plt.tight_layout()
```



In [10]:
```
top_inds = indices[(pgram > 0.2*np.max(pgram))]
top_freqs_hz = freqs_in_hz[(pgram > 0.2*np.max(pgram))]
print('Top indices:', top_inds, ' Top frequencies in Hz:', top_freqs_hz)
```

```
Top indices: [  1.   3.   5.  38.  56.  57.  75.  94. 113. 132. 151. 170.
188. 189.
 207. 240. 242. 244.]  Top frequencies in Hz: [ 0.00408163  0.0122449
0.02040816  0.15510204  0.22857143  0.23265306
  0.30612245  0.38367347  0.46122449 -0.46122449 -0.38367347 -0.30612245
 -0.23265306 -0.22857143 -0.15510204 -0.02040816 -0.0122449  -0.00408163]
```

3e) To the residuals, fit a model of the form

$\beta_{1_s}\sin(\omega_1 x) + \beta_{1_c}\cos(\omega_1 x) + \beta_{2_s}\sin(\omega_2 x) + \beta_{2_c}\cos(\omega_2 x) + \ldots + \beta_{K_s}\sin(\omega_K x) +$

The frequencies $\omega_1, \ldots, \omega_K$ in the model are those corresponding to the indices identified in Part 2c. (Hint: Each of the sines and cosines will correspond to one column in your X-matrix.)

- Print the values of the regression coefficients obtained using least-squares.

All numerical outputs should be displayed to three decimal places.

In [11]:
```
def trig_fit(position):
    w = 2*math.pi*top_freqs_hz[position]

    XT = np.vstack((np.sin(w*xvals), np.cos(w*xvals)))
    X = np.transpose(XT)

    beta_sc = np.linalg.inv(XT.dot(X)).dot(XT).dot(resid_priv) # Calculat
    fit_sc = X.dot(beta_sc)
    return w, beta_sc, fit_sc

w_list = []
s_list = []
c_list = []
sc_fit_total = [0*len(top_inds)]

for i in range(len(list(top_inds))):
    w, beta, fit = trig_fit(i)
    sc_fit_total += fit
    w_list.append(w)
    s_list.append(beta[0])
    c_list.append(beta[1])
    X = np.column_stack([w_list, s_list, c_list])

print(X)
print(sc_fit_total)
```

```
[[ 2.56456543e-02 -5.12528880e+01  1.01555806e+02]
 [ 7.69369629e-02  6.30415324e+01  9.71853944e+00]
 [ 1.28228272e-01  5.84062863e+01  2.31288730e+01]
```

```
 [ 9.74534864e-01  6.16276582e+01 -5.40056189e+01]
 [ 1.43615664e+00 -1.55806757e+01 -9.47973260e+01]
 [ 1.46180230e+00  8.16586950e+01  7.23810630e+01]
 [ 1.92342407e+00  4.59908596e+01 -2.70296679e+01]
 [ 2.41069151e+00  6.26280394e+01 -2.17896528e+01]
 [ 2.89795894e+00  3.24722696e+01  9.05889318e+01]
 [-2.89795894e+00 -3.24722696e+01  9.05889318e+01]
 [-2.41069151e+00 -6.26280394e+01 -2.17896528e+01]
 [-1.92342407e+00 -4.59908596e+01 -2.70296679e+01]
 [-1.46180230e+00 -8.16586950e+01  7.23810630e+01]
 [-1.43615664e+00  1.55806757e+01 -9.47973260e+01]
 [-9.74534864e-01 -6.16276582e+01 -5.40056189e+01]
 [-1.28228272e-01 -5.84062863e+01  2.31288730e+01]
 [-7.69369629e-02 -6.30415324e+01  9.71853944e+00]
 [-2.56456543e-02  5.12528880e+01  1.01555806e+02]]
[ 5.14293832e+02  5.15352203e+02  2.55859228e+02  4.00948384e+02
  2.28240171e+02  2.97137192e+02  3.36016243e+02  3.79437376e+02
  7.21439796e+02  2.25258048e+02  7.15377813e+02 -1.40358054e+02
  3.40094881e+02  5.68744797e+02  5.43677320e+02  3.01353989e+02
  4.27017019e+02  1.60353449e+02  1.63261203e+02  2.60290566e+02
  3.17365648e+02  5.74920178e+02 -6.27669109e+01  4.67717478e+02
 -2.88650279e+02  1.14975893e+02  2.64812397e+02  2.34135196e+02
  3.56255649e+01  1.69540581e+02 -1.59422453e+02 -1.89964704e+02
 -1.16838114e+00  1.05260379e+02  3.18356982e+02 -4.22148211e+02
  1.83859180e+02 -4.30596232e+02 -6.69745149e+01  3.27353335e+01
  2.89134183e+01 -9.56341367e+01  7.24939988e+01 -2.93195068e+02
 -3.38332937e+02 -4.70555253e+01  1.16110322e+02  2.94640499e+02
 -5.44110271e+02  1.30423213e+02 -3.46240916e+02 -3.11656713e+01
  1.31083038e+00  7.45584224e+00 -6.40565866e+01  1.15544094e+02
 -3.06013659e+02 -3.90184811e+02 -2.88256940e+01  1.59383464e+02
  2.81930152e+02 -6.76055310e+02  3.59226190e+01 -3.26838903e+02
 -7.89286746e+01 -1.33356150e+02 -1.33186676e+02 -1.67305134e+02
  1.36663995e+01 -4.62361166e+02 -5.85062036e+02 -1.59109360e+02
  5.36702654e+01  1.36765233e+02 -9.19642095e+02 -1.60989273e+02
 -3.93562759e+02 -1.87101643e+02 -3.02468122e+02 -2.82072936e+02
 -2.55777356e+02 -4.61010955e+01 -5.38870167e+02 -6.67848069e+02
 -1.58098056e+02  9.84107334e+01  1.75587617e+02 -9.46159658e+02
 -1.28692699e+02 -2.19680401e+02 -3.67343074e+01 -1.99424168e+02
 -1.53158654e+02 -7.12190551e+01  1.62595621e+02 -3.43910325e+02
 -4.86313790e+02  8.12549436e+01  3.52458428e+02  4.09395826e+02
 -7.92613605e+02  4.44261662e+01  5.37397906e+01  1.82607183e+02
 -5.87028785e+01 -2.49969495e+01  6.59183455e+01  2.82659092e+02
 -2.64583181e+02 -4.52304332e+02  1.26560523e+02  3.69026660e+02
  3.85669048e+02 -9.09185883e+02 -8.49468832e+01 -4.52262323e+00
  6.11090014e+01 -2.60753543e+02 -2.44257162e+02 -1.52831548e+02
  4.96084828e+01 -5.15594718e+02 -7.27098711e+02 -1.28443520e+02
  9.86424605e+01  1.14420609e+02 -1.22480018e+03 -3.87071047e+02
 -2.09124555e+02 -1.60490287e+02 -5.11247957e+02 -4.68338083e+02
 -3.42263462e+02 -1.13676365e+02 -6.41386285e+02 -8.28797003e+02
 -1.83217150e+02  5.29374329e+01  1.11929923e+02 -1.22471907e+03
 -3.57341764e+02 -7.42960207e+01 -2.05486922e+01 -3.78011953e+02
 -3.01574293e+02 -1.49376635e+02  9.84839250e+01 -3.87176544e+02
 -5.56555908e+02  1.06331074e+02  3.17171692e+02  4.04155952e+02
 -9.40994647e+02 -8.53933086e+01  2.53285748e+02  2.81057843e+02
 -1.09598118e+02 -3.75355248e+01  9.18209433e+01  3.17049391e+02
 -1.50991695e+02 -3.30605807e+02  3.04613306e+02  4.46711601e+02
  5.43897132e+02 -8.15593624e+02 -1.32118349e+00  3.60545421e+02
```

```
 3.54804189e+02 -6.57332405e+01 -1.41398221e+00  9.50201893e+01
 2.97904363e+02 -1.32252808e+02 -3.03012946e+02  3.08970336e+02
 3.88279118e+02  5.25865388e+02 -8.06173494e+02 -1.65305865e+01
 3.78557352e+02  3.70765969e+02 -4.10167782e+01  4.29566631e+01
 1.22350614e+02  3.23012800e+02 -3.58820051e+01 -1.70979454e+02
 4.25092175e+02  4.43911569e+02  6.40850629e+02 -6.38276389e+02
 1.22922434e+02  5.34987935e+02  5.25314008e+02  1.25861558e+02
 2.19039947e+02  2.57613919e+02  4.33954123e+02  1.35153088e+02
 1.84970147e+01  5.64368012e+02  4.84424545e+02  7.19090568e+02
-5.18870632e+02  1.78282987e+02  5.63028216e+02  5.27355161e+02
 1.24616093e+02  2.03774619e+02  1.71980091e+02  3.01301483e+02
 5.52751567e+01 -4.93931342e+01  4.31062998e+02  2.38687047e+02
 5.16814447e+02 -6.59122379e+02 -2.21725508e+01  3.34073974e+02
 2.93165972e+02 -8.15813797e+01  9.97643410e+00 -6.97973915e+01
 4.07210815e+01 -1.11865897e+02 -1.63902178e+02  2.83243103e+02
 8.69513230e+00  3.72815542e+02 -6.88128249e+02 -7.78993475e+01
 2.72587302e+02  2.63052927e+02 -4.39067860e+01  8.94581766e+01
-1.81366197e+01  9.14645270e+01  5.33518492e+01  6.90505806e+01
 4.86140661e+02  1.24925971e+02  5.73902664e+02 -3.67821198e+02
 1.99501895e+02]
```

3f) The combined fit

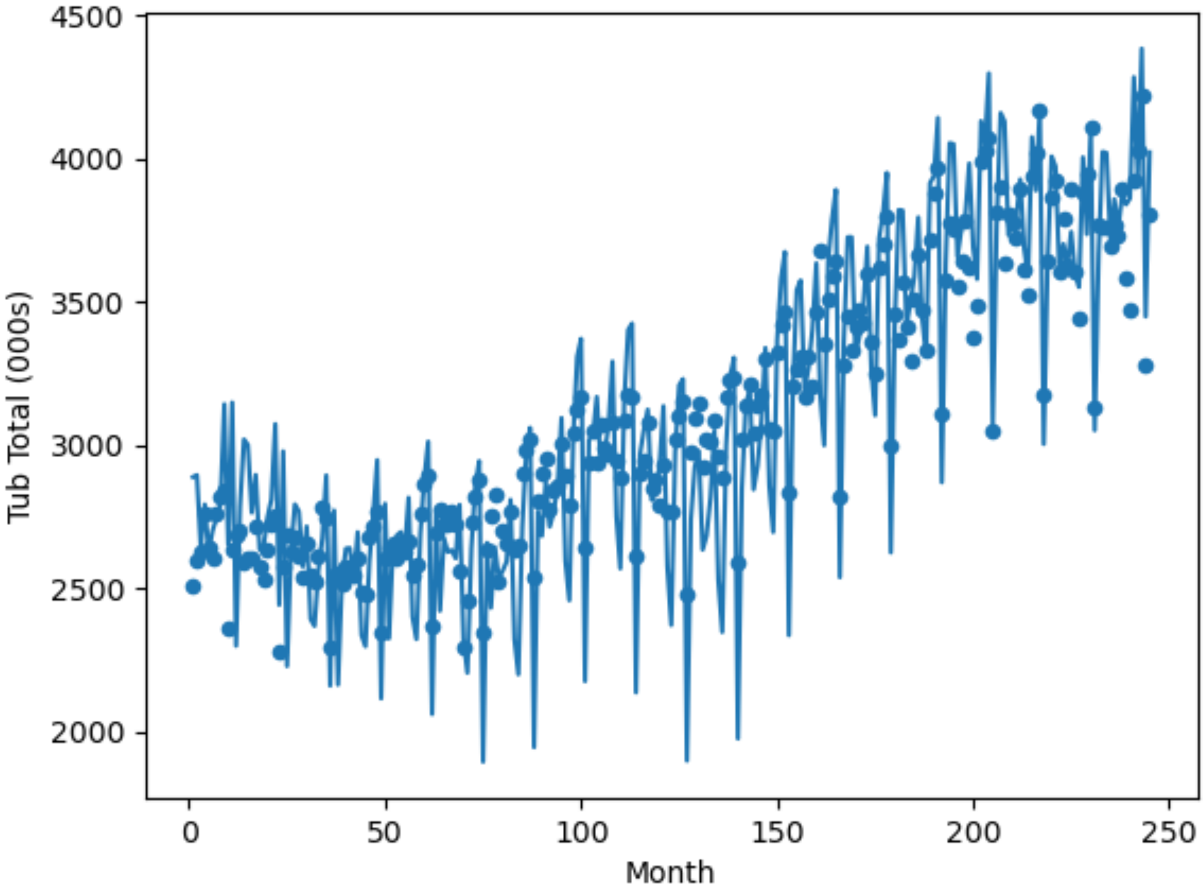
- Plot the combined fit together with a scatterplot of the data
- Compute and print the final **MSE** and R^2 coefficient. Comment on the improvement over the linear fit.

The combined fit, which corresponds to the full model

$$f(x) = \beta_0 + \beta_1 x + \beta_{s1}\sin(\omega_1 x) + \beta_{c1}\cos(\omega_1 x) + \ldots + \beta_{sk}\sin(\omega_k x) + \beta_{ck}\cos(\omega_k x$$

can be obtained by adding the fits in parts 2b) and 2e).

In [12]:
```
combined_fit = sc_fit_total + fit_priv

_, fit_0, _ = polyreg(xvals, yvals, 0)

SSE_0 = np.linalg.norm(yvals - fit_0)**2
SSE = np.linalg.norm(yvals - combined_fit)**2
MSE = SSE/(np.size(yvals))
R_square = np.round(1- SSE/SSE_0, decimals = 4)

plt.xlabel('Month')
plt.ylabel('Tub Total (000s)')
plt.scatter(xvals, yvals, s=20)
plt.plot(xvals, combined_fit)
plt.show()

print("MSE:", MSE)
print("R_square:", R_square)
```




```
MSE: 45323.63592122875
R_square: 0.7956
```

The MSE and R_square of this model barely sees any improvement from the linear fit, as the data itself has too many feature to be used in modeling its behavior.