# 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`.

```
# If you are running this on Google Colab, uncomment and run the following lines; otherwise ignore this cell
# from google.colab import drive
# drive.mount('/content/drive')
```

```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Load data
df_tfl = pd.read_csv('tfl_ridership.csv')
# If running on Google Colab change path to '/content/drive/MyDrive/IB-Data-Science/Exercises/tfl_ridership.csv'

df_tfl.head(13)
```

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

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).

```
#df_tfl.sample(3)  #random sample of 3 rows
df_tfl.tail(3)  #last 3 rows
```

| | 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) | ... | Tube Contactless (000s) | Tube Day Travelcard (000s) | Se Travel (€ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **242** | 2018/19 | P 09 | 11 Nov '18 | 08 Dec '18 | 28d | 0 | 1110 | 1089 | 0 | 41 | ... | 1399 | 249 | |
| **243** | 2018/19 | P 10 | 09 Dec '18 | 05 Jan '19 | 28d | 0 | 1001 | 949 | 0 | 38 | ... | 1110 | 242 | |
| **244** | 2018/19 | P 11 | 06 Jan '19 | 02 Feb '19 | 28d | 0 | 1036 | 1075 | 0 | 30 | ... | 1310 | 204 | |

3 rows × 26 columns

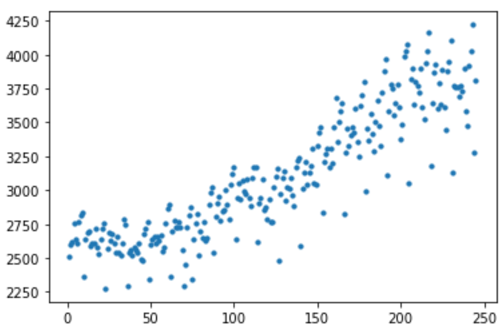
The dataframe contains $N = 245$ counting periods (of 28/29 days each) from 1 April 2000 to 2 Feb 2019. We now define a numpy array consisting of the values in the ' Tube Total (000s)' column:

```
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 tube rideship in counting period $x_i = i$.

## 2a) Plot the data in a scatterplot

```
#Your code for scatterplot here
plt.scatter(xvals, yvals, s=10)
plt.show()
```



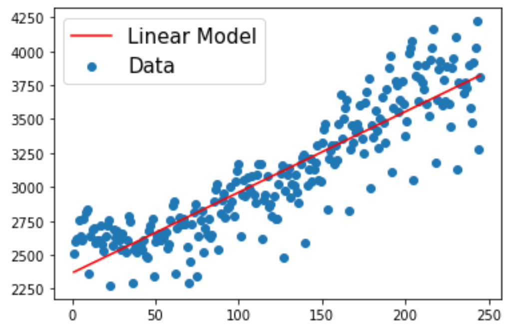

## 2b) Fit a linear model $f(x) = \beta_0 + \beta_1 x$ to the data

- Print the values of the regression coefficients $\beta_0, \beta_1$ 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.

```
def polyreg(data_x, data_y, k): #Simialr function to that in exercise 1
    # Your code here
    # The function should return the the coefficient vector beta, the fit, and the vector of residuals
```

```
        all_ones = np.ones(np.shape(xvals))

        N = np.shape(data_x)[0]

        if k >= N:
            k = N - 1

        else:
            k = k

        data_x_array = []
        for i in range(k+1): # working it out for every vaule of k
                x_calc = data_x ** i
                if i == 0:
                     data_x_array.append(all_ones)
                else:
                     data_x_array.append(x_calc)

        x2_trans = np.vstack(data_x_array)
        x2 = np.transpose(x2_trans)

        beta_lin = np.linalg.lstsq(x2, data_y, rcond=None)[0]
        fit_lin = x2.dot(beta_lin)
        resid = data_y - fit_lin


        return beta_lin, fit_lin, resid

    beta_lin, fit_lin, resid = polyreg(xvals, yvals, 1) #Getting the values for when k is 1

    plt.scatter(xvals, yvals, label = 'Data') # Plotting the scatter with the linear fit
    plt.plot(xvals, fit_lin, 'r', label = 'Linear Model')
    plt.legend(fontsize = 15)
    plt.show()

    all_ones = np.ones(np.shape(xvals))
    fit_0 = np.mean(yvals) * all_ones # If k is zero

    SSE_0 = np.linalg.norm(yvals - fit_0)**2 # Calculating the SSEs
    SSE_1 = np.linalg.norm(yvals - fit_lin)**2

    MSE_1 = SSE_1 / (np.size(yvals)) # Calculating the MSEs

    R_1 = 1 - SSE_1 / SSE_0 # Calculating R^2

    print('Beta_0 = ', np.round(beta_lin[0], 3)) #Printing all the values
    print('Beta_1 = ', np.round(beta_lin[1], 3))
    print('SSE_1 = ', np.round(SSE_1, 3))
    print('MSE_1 = ', np.round(MSE_1, 3))
    print('R_1^2 = ', np.round(R_1, 3))
```



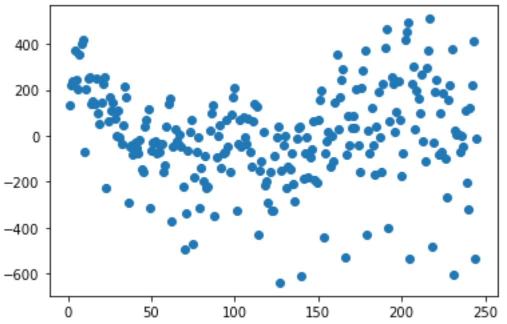

```
Beta_0 =  2367.382
Beta_1 =  5.939
SSE_1 =  11104290.801
MSE_1 =  45323.636
R_1^2 =  0.796
```

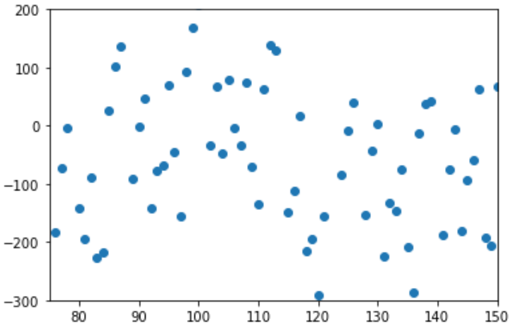
## 2c) Plotting the residuals

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

```
# Your code here
plt.scatter(xvals, resid) # Plotting the scatter
plt.show()

plt.scatter(xvals, resid)
```

```
plt.axis([75, 150, -300, 200]) # Zooming in on some of the scatterplot
plt.show()
```



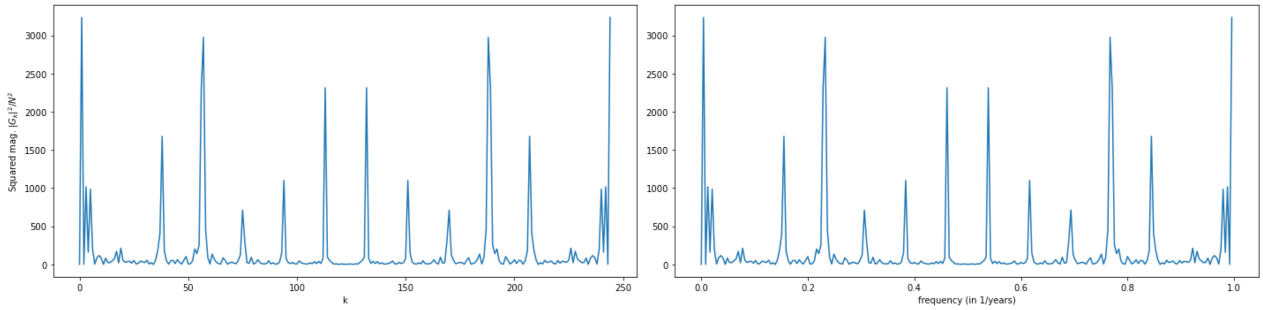


Looking at both the overall residual scatter plot and the zoomed in one, there is no evidence of any periodic component.

## 2d) 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.

```
# Your code to compute and plot the histogram

T = xvals[76] - xvals[75]  # This can be the time interval between any two successive values.

# Note that T is in years

# Compute the squared magnitudes of the DFT coefficients -- this is known as the "periodogram"
pgram = np.abs(np.fft.fft(resid, N)/N)**2 #We normalize by N, but this is optional
indices = np.linspace(0, (N-1), num = N)
freqs_in_hz = indices/(N*T)
freqs_in_rads = freqs_in_hz*2*math.pi

plt.figure(figsize = (20, 5))
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/years)')  # Since units of T is years
plt.tight_layout()
plt.show()
```



```
# Your code to identify the indices for which the periodogram value exceeds 50% of the maximum

top_inds = indices[(pgram > 0.5*np.max(pgram))]
top_freqs_hz = freqs_in_hz[(pgram > 0.5*np.max(pgram))]
print('Top indices:', top_inds, ' Top frequencies in Hz:', top_freqs_hz)
```

```
Top indices: [  1.  38.  56.  57. 113. 132. 188. 189. 207. 244.]  Top frequencies in Hz: [0.00408163 0.15510204 0.22857143 0.232653
 0.76734694 0.77142857 0.84489796 0.99591837]
```

## 2e) To the residuals, fit a model of the form

$$\beta_{1s}\sin(\omega_1 x) + \beta_{1c}\cos(\omega_1 x) + \beta_{2s}\sin(\omega_2 x) + \beta_{2c}\cos(\omega_2 x) + \ldots + \beta_{Ks}\sin(\omega_K x) + \beta_{Kc}\cos(\omega_K x).$$

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.
- Compute and print the final **MSE** and $R^2$ coefficient. Comment on the improvement over the linear fit.

All numerical outputs should be displayed to three decimal places.

```
# Your code here
w = 2*math.pi*top_inds[0]/(N*T)  # Freq. in rad/s corresponding to k* is w = 2.pi.k*/(NT)
N_top_inds = int(len(top_inds) / 2 - 1)

# Now form the X matrix with columns sin(wx) and cos(wx) for x in dates. First define its transpose
x = np.vstack((np.sin(w*xvals), np.cos(w*xvals))) # Stacking in vertical array - single column

for i in range (N_top_inds):
    w = 2*math.pi*top_inds[i+1]/(N*T)  # Freq. in rad/s corresponding to k* is w = 2.pi.k*/(NT)
    x = np.vstack((x,np.sin(w*xvals)))
    x = np.vstack((x,np.cos(w*xvals)))


x_trans = np.transpose(x) # Transposing to get horizontal array - single row

beta_sc = np.linalg.inv(x.dot(x_trans)).dot(x).dot(resid) # Calculating beta and it's range
beta_range = int(len(beta_sc)/2)

for i in range(beta_range):
    print('beta_',i+1,'s', ', beta_',i+1,'c:', np.round(beta_sc[2*i:2*i+2],3))
```

```
beta_ 1 s , beta_ 1 c: [-51.253 101.556]
beta_ 2 s , beta_ 2 c: [ 61.628 -54.006]
beta_ 3 s , beta_ 3 c: [-15.581 -94.797]
beta_ 4 s , beta_ 4 c: [81.659 72.381]
beta_ 5 s , beta_ 5 c: [32.472 90.589]
```

## 2f) 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.

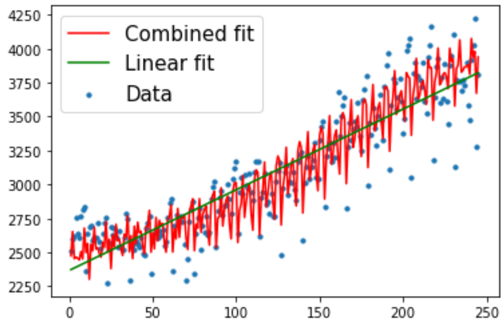
The combined fit, which corresponds to the full model

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

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

```
# Your code here
fit_sc = x_trans.dot(beta_sc) # Calcualting the fit and other values
resid_fit = resid - fit_sc
SSE_fit = np.linalg.norm(resid_fit) ** 2
MSE_fit = SSE_fit / np.size(yvals)
R2 = 1 - SSE_fit / SSE_0

fit_comb = fit_lin + fit_sc # The fit of the combined is the some of the other fits

plt.scatter(xvals, yvals, s=10, label = 'Data') # Plotting scatter and fits to see how they compare
plt.plot(xvals, fit_comb, 'r', label = 'Combined fit')
plt.plot(xvals, fit_lin, 'g', label = 'Linear fit')
plt.legend(fontsize = 15)
plt.show()

print('Linear fit') # Printing the values optained from the linear fit to compare to the combined fit
print('SSE_1 = ', np.round(SSE_1, 3))
```

```
print('MSE_1 = ', np.round(MSE_1, 3))
print('R_1^2 = ', np.round(R_1, 3))
print('')
print('Combined fit') # Printing the values of the combined fit
print('SSE = ', np.round(SSE_fit, 3))
print('MSE = ', np.round(MSE_fit, 3))
print('R^2 = ', np.round(R2, 3))
```




```
Linear fit
SSE_1 =  11104290.801
MSE_1 =  45323.636
R_1^2 =  0.796

Combined fit
SSE =  4972887.791
MSE =  20297.501
R^2 =  0.908
```

The linear fit shows the overall trend over all long time (the general trend). Whereas, with the combined fit, it shows the variations in the data over a short period of time (it can change very quickly).

The combined fit is better than the linear one as the MSE is reduced by over a half and the R^2 value is closer to one.