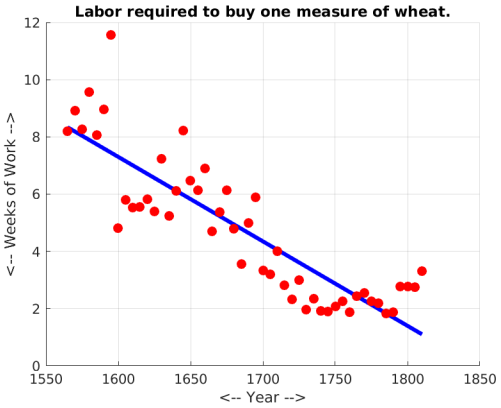
# Linear Models of Data
# ML_2022: Machine Learning

https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/linear_lab/linear_lab.pdf




*John Playfair was able to estimate a linear relationship for the price of wheat over time.*

**Variance reduction!**

*We will practice techniques for determining a linear model that approximates a set of data.*

- *We start with the classic Fahrenheit/Celsius formula, and just two data values;*
- *We estimate the price of a used Ford based on mileage, from 23 data values;*
- *We look at a model of medical bills, which depend on many factors;*
- *We try a linear model for Mexico's population, and then replace it by splitting the domain and creating two separate models;*

## 1 Copying the data:

Each of the exercises will be carried out on a particular datafile. These datafiles are available on the *datasets* page at the class website:
**https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/datasets/datasets.html**

You might go ahead now and download them all:

- *two_temperatures_data.txt*
- *ford_data.txt*
- *insurance_data.txt*
- *mexico_population_data.txt*

## 2 Prepare for Exercise 1:

There is a simple linear relationship between the Celsius ($C^o$) and Fahrenheit ($F^o$) temperature scales. Ordinarily, this is written as a pair of functional formulas:

$$C = \frac{5}{9}(F - 32)$$
$$F = \frac{9}{5}C + 32$$

but it could also be written as a single linear relationship:

$$160 - 5F + 9C = 0$$

Let's concentrate on the familiar form of linear equations, where `x` values represent independent variables, and `y` is the dependent variable. Then, letting $x_0 = 1, x_1 = F, y = C$, our formula reads,

$$y = -\frac{160}{9}x_0 + \frac{5}{9}x_1$$

Letting `c` be the coefficient vector `c=[-160/9, 5/9]`, then for any single set of `x` data, we can compute `y` as the dot product of the coefficients and `x`:

$$x'c = y \qquad \text{(Linear system with one data value)}$$

But suppose we didn't know this relationship... then how could we go about discovering it?

To begin with, assume we have two data items, in an array `data`

```
data = [ [ 32,    0 ],
         [ 212, 100 ] ]
```

Listing 1: Two temperature data points

Create the array `X` from a column of 1's followed by all but the last column of data:

```
X = np.c_ [ np.ones(2), data[:,0:-1] ]  # Read this line carefully!
y = data[:,-1]
```

Listing 2: Creating X and y

We want to find `c` by solving the following singular linear system:

$$Xc = y \qquad \text{(Linear system with multiple data values)}$$

If we can solve this system, then our original data is modeled by

```
y = c[0] + c[1] * x[0]
```

and if we have a new value of temperature `F`, we can compute the corresponding Celsius temperature `C` by applying the formula

```
C = c[0] + c[1] * F
```

In the following exercise, we will start with the smallest amount of data, two items, and try out several equivalent methods of determining the linear coefficients `c`. In this case, because we only have two data points, the line will fit our data exactly. Moreover, since it turns out that this linear relationship always holds between `F` and `C`, we can predict exactly the value of `C` corresponding to any `F` we choose. This happy case will be exceptional when we move on to the further exercises.

# 3 Exercise 1:

Write a program *exercise1.py*, which uses the two temperature data points, and solves for the linear system with multiple data values to get a vector `c`. Normalize this vector by dividing it by `c[0]/160`, in which case your result should be almost exactly `[160,9,-5]`.

- Read `data` from *two_temperatures_data.txt*;
- Plot the two data points as blue dots, as Fahrenheit versus Celsius
- Create the arrays `X` and `y` for handling the linear system;
- Solve $Xc = y$ for each of the following approaches, saving your results:
  1. `c1`: the normal equations;
  2. `c2`: the QR factorization;
  3. `c3`: the SVD pseudoinverse;
  4. `c4`: the numpy `linalg.lstsq()` function;
  5. `c5`: the scikit-learn function `LinearRegression()`;
- For each solution vector, compute and print the Mean Square Error;
- For 50 points $32 \leq F \leq 212$, compute `C = c[0] + c[1] * F`, plot your two data points as blue dots, and this linear model as a red line;
- Rescale your solutions so the first entry is 160. The coefficient of `y` was implicitly -1, so rescale that too, and print it as the final entry of `c`:

```
for c in [ c1, c2, c3, c4, c5 ]:
  s = 160 / c[0]
  cy = -1.0 * s
  c =     c * s
  print ( c, cy )
```

Your original five solutions c1 through c5 might have differed, but if we rescale each of them so that the first component is 160, their values should all be very close to [160, -5, 9]

# 4 Prepare for Exercise 2:

Our first example used just two data values, so of course we could find a linear relationship. Now we are going to look at situations in which there are many data values. We will take the data that John Playfair used to find a trend in the price of wheat. The file `playfair_data.txt` contains 50 records. Each record gives a year, the price of a measure of wheat in shillings, and the average weekly earnings of a mechanic in shillings. The ratio wheat price / earning gives the cost of wheat.

We will search for a relationship between the year, and the cost, of the form

$$\text{cost} = c_0 * 1 + c_1 * \text{year}$$

Since we have 50 data items, we will expect that if we can compute a best approximating coefficent vector `c`, we will no longer have a tiny value of the mean square error.

# 5 Exercise 2:

Write a program *exercise2.py* that uses Playfair's data to construct a linear model, as follows:

- Read `data` from *playfair_data.txt*;
- Plot year (data[:,0]) versus cost (data[:,1]/data[:,2]) as blue dots

- Create the arrays `X` and `y` for handling the linear system; Column 0 of X will be ones, and column 1 will be the year; The vector `y` is simply your cost data.
- Solve $Xc = y$ using the numpy `linalg.lstsq()` function;
- Print the value of `c`;
- Compute and print the Mean Square Error;
- For 50 points $1550 \leq year \leq 1810$, compute `cost = c[0] + c[1] * year`, plot your data points as blue dots, and this linear model as a red line;
- According to your linear model, what will be the price of wheat in 1850?

# 6 Prepare for Exercise 3:

In some cases, our data describes several factors which together seem to affect the value of some quantity we are interested in. So our model will no longer be a line, but a plane, or a hyperplane. Still, the problem of finding the linear coefficients `c` is formulated in the same way. If we have three variables $x_i$ that seem to contribute to the value of a quantity $y$ (so here `d=4`), and we seek a linear model, we write

$$y = c_0 * 1 + c_1 * x_0 + c_2 * x_1 + c_3 x_2$$

Our data file may contain `n` rows, with row $i$ containing four values $(x_{i,0}, x_{i,1}, x_{i,2}, y_i)$. Again, we create a matrix `X` with a column of 1's, followed by the `x` values, and a separate `y` vector, and solve for best coefficients `c` in the same way as before.

The fact that there are multiple (`d`) factors would seem to give us a better chance of coming up with a good (low error) formula for the `y` values. On the other hand, if `n`, the number of data records, is large, that means we have more and more equations to satisfy, which makes it likely our error will be large.

For the following example, we will try to estimate the size of a patient's hospital bill, using various facts about the patient.

The file *insurance_data.txt* includes 1338 records about medical insurance. Each record lists, for a given person, 7 factors: `age, sex, bmi, kids, smoker, region, bill`. We seek a linear formula for the medical charges `bill`, based on the values of the first five quantities. We will ignore the factor `region`, which records the region of the country.

Thus, we are trying to find values `c` so that a linear formula involving the factors can approximately predict the resulting hospital bill:

$$bill \approx c_0 + c_1\, age + c_2\, sex + c_3\, bmi + c_4\, kids + c_5\, smoker$$

# 7 Exercise 3:

Write a program *exercise3.py* that uses the insurance data to construct a linear model that predicts the size of a hospital bill.

- Read `data` from *insurance_data.txt*;
- Create the array `X` by a column of 1's followed by the values of columns 0, 1, 2, 3 and 4 (`age, sex, bmi, kids, smoker`);
- Create the array `y` from column 6 of the data;
- To get a feeling for the behavior of `y`, make a histogram;
- Solve $Xc = y$ using one of the techniques we have discussed;
- Print the value of `c`;
- Compute and print the Mean Square Error;

- Predict the bill for a patient with `age=19, sex=1, bmi=24.6, kids=1, smoker=0`. Note that the actual bill was $1837.23;
- Predict the bill for a patient with `age=34, sex=0, bmi=31.92, kids=1, smoker=1`. Note that the actual bill was $37,701.

The magnitude and sign of coefficient $c_i$ tells us something about the imporance of factor $x_i$. In particular, negative coefficients suggest this factor reduces the bill. If you have any experience of medical billing, it should be no surprise that the data is rather difficult to model accurately, but at least we have a ball park estimate.

# 8 Prepare for Exercise 4:

We will use the Mexican population data to construct a linear model that predicts the population given the year, and measure MSE, the mean square error. From the plot, we will see that the linear model is pretty far from the curve. However, the curve seems to break into two separate straight lines. We compute the mean square error (MSE), a measure of how far our model is from the data.

If we inspect the graph, we may notice that we could do a much better approximation by breaking the data up into the intervals `[year[0],year[i]]` and `[year[i],year[n-1]]`. We create arrays `data1` and `data2`, then the corresponding arrays `X1, y1`, and `X2, y2`, solve for `c1` and `c2`, and plot the two lines over their separate intervals, getting errors `mse1` and `mse2`. Presumably `mse1+mse2` is much less than `mse`, indicating that our approximation is significantly better. Since this is a 2D problem, we can plot the data and the linear models for a visual proof.

We have to be careful in splitting our problem, naming and defining the new arrays, but in a sense the construction of the model is the same as before, except we do it twice, over two separate intervals.

# 9 Exercise 4:

Write a program *exercise4.py* that uses the Mexican population data to construct a linear model that predicts the population given the year, and measure MSE, the mean square error. From the plot, we will see that the linear model is pretty far from the curve. However, the curve seems to break into two separate straight lines.

- Read `data` from *mexico_population_data.txt*;
- Plot year `data[:,0]` versus population `data[:,1]` as blue dots
- Create the arrays `X` and `y` for handling the linear system; Column 0 of `X` will be ones, and column 1 will be the year; The vector `y` is the population data.
- Plot the data points as blue dots;
- Solve $Xc = y$ using one of the techniques we have discussed;
- Print the value of `c`;
- Compute and print `mse`, the Mean Square Error;
- Plot year versus population data as blue dots, and the linear model `pop=c[0]+c[1]*year` as a red line;
- Looking at the plot, decide on a value `i` so that the curve roughly splits into two straight lines at `year[i]`;
- Split `data` into two parts; `data1=data[0:i+1,:]` and `data2=data[i:,:]`. (Read these assignments carefully and think about what the Python statements are saying!)
- Set up `X1, y1`, and solve `c1` and compute `mse1`;
- Set up `X2, y2`, and solve `c2` and compute `mse2`;
- Print `mse` and `mse1+mse2` for comparison;

- Plot year versus population data as blue dots, and the linear model `pop=c1[0]+c1[1]*year` as a red line over the domain `year[0]` to `year[i]`, and the linear model `pop=c2[0]+c2[1]*year` as a green line over the domain `year[i]` to `year[n-1]`;

Presumably, `mse1+mse2` is much less than `mse`, suggesting that the second pair of models is superior. Just looking at the plot of the original data, we could guess that a single line would not be a great approximation to the shape that the data suggests. This is similar to our clustering efforts, where breaking the data up into smaller groups could produce results with dramatically smaller total variance.