Week 10 – Session 2

DW 10.009 – Introduction to Python Programming

Week 10 Breakdown

- Session 1: Introduction to Data Science
    - Introduction to Numpy
    - Core ideas about data science
    - Data Manipulation and Visualization
- Session 2: Introduction to regression
    - Key parameters for regression
    - Linear regression
    - Multiple linear regression
- Session 3: About classification
    - Key parameters for classification
    - K-NN Classification

Week 10 Breakdown

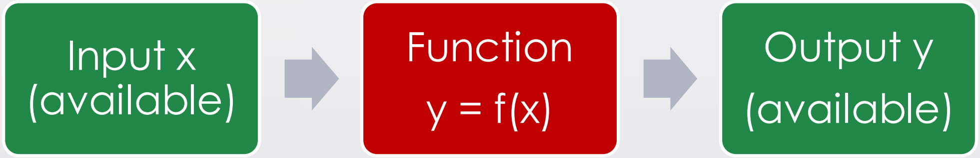
- Session 1: Introduction to Data Science
  - Introduction to Numpy
  - Core ideas about data science
  - Data Manipulation and Visualization
- Session 2: Introduction to regression
  - Key parameters for regression
  - Linear regression
  - Multiple linear regression
- Session 3: About classification
  - Key parameters for classification
  - K-NN Classification

Linear regression 101

Key concepts about linear regression

Linear regression: core idea

- Linear regression is a typical example of a data science problem, where we look for a function that connects inputs and outputs in our data.



- Hypothesis: in the linear regression approach, we assume that the missing function f is a linear function.

$$y = f(x) = a_0 + a_1 x$$

Linear regression: core idea

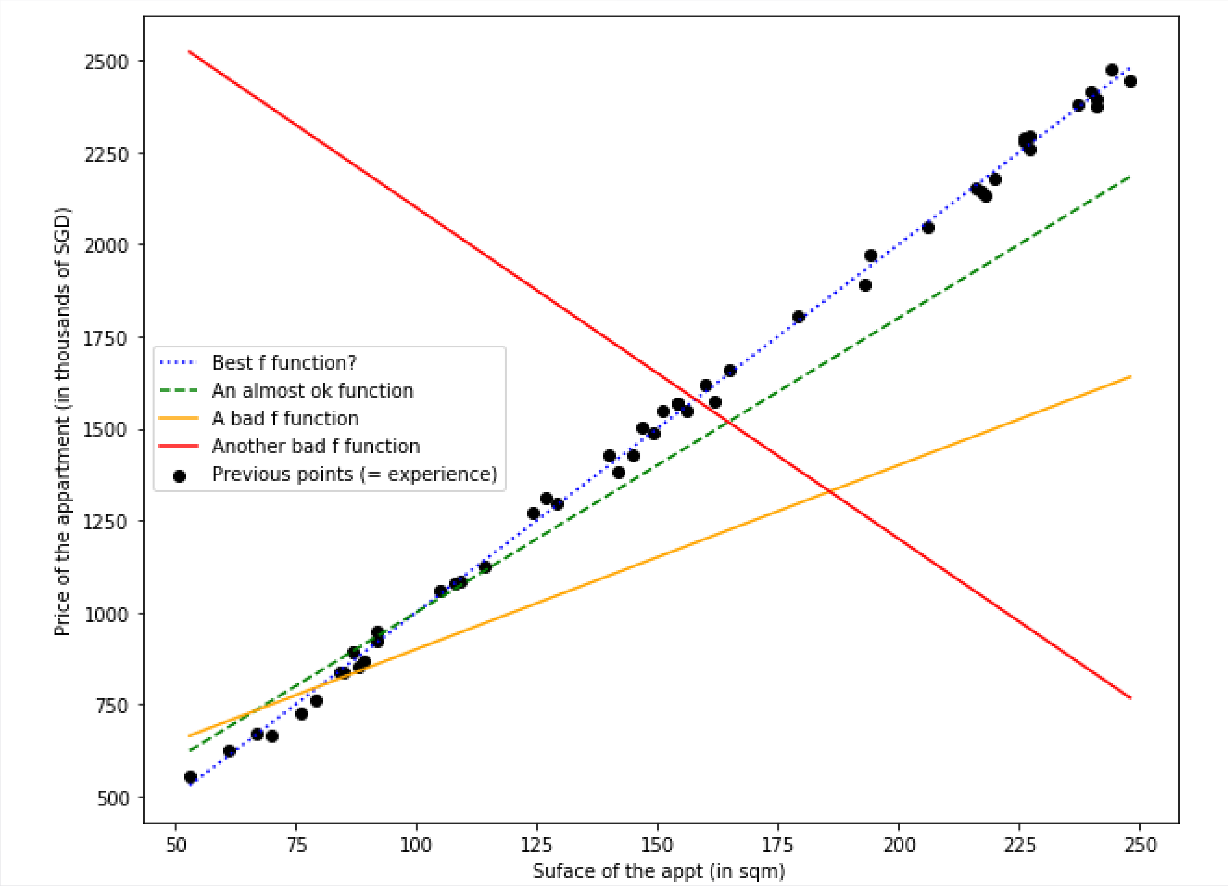
- **Hypothesis:** in the linear regression approach, we assume that the missing function f is a **<u>linear</u>** function.

$$y = f(x) = a_0 + a_1 x$$

- **Objective:** find the coefficient values (a_0, a_1) that fit the data (x, y) in our record/experience, in the best possible way.

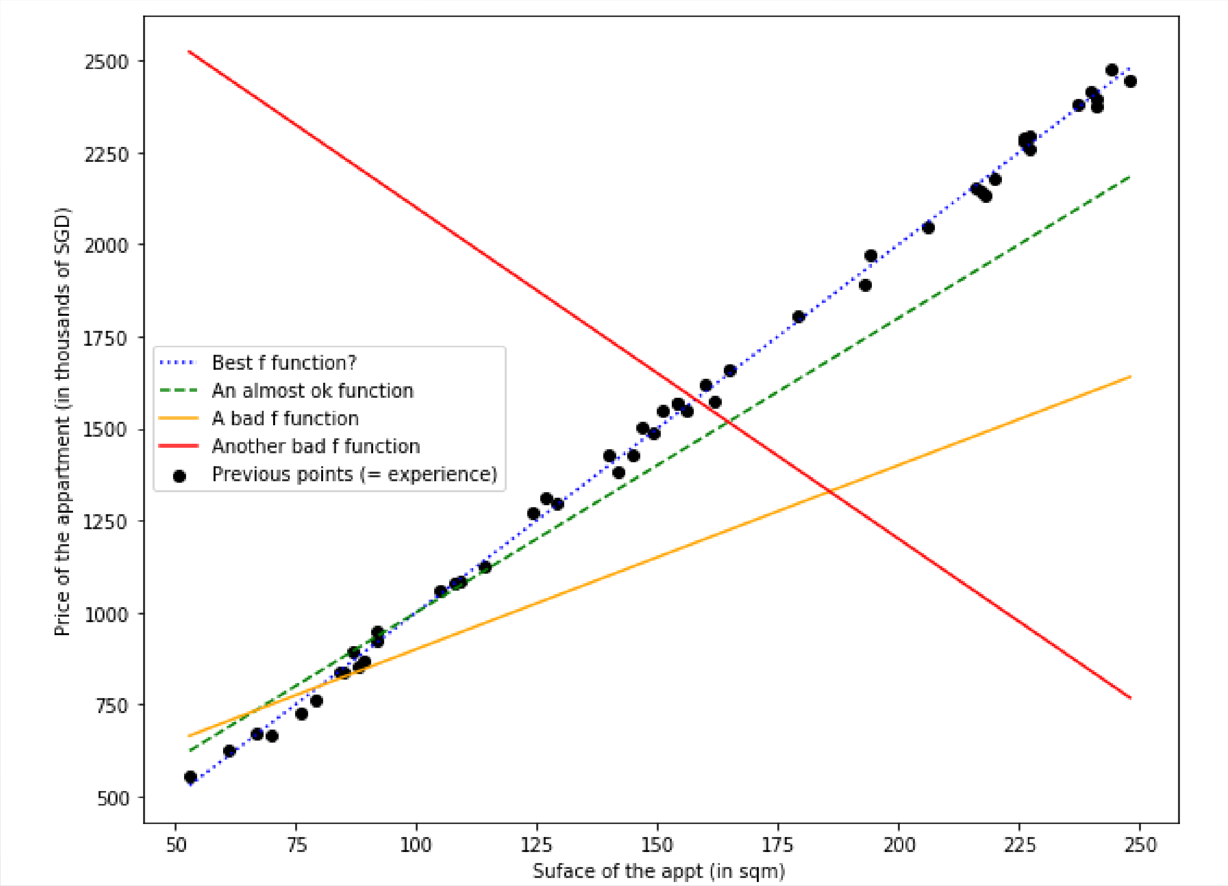
Linear regression: core idea

- But how do we define the **« best »** function f?
- Why is the blue function our best candidate and why are the red/yellow ones bad functions here?



Linear regression: core idea

- But how do we define the **« best »** function f?
- Why is the blue function our best candidate and why are the red/yellow ones bad functions here?
- Need some sort of a **performance measure** for the « quality » of the function, or its ability to fit the data!



But before that: train and test samples!

- **Objective:** We want to use the data in our record/experience to find the best function to fit our data.

- **General rule of data science: do not test** the **accuracy** of the function/model on the same samples you used to calculate the function in the first place.

Train and test samples example

- Split your data into training and testing sets.
- **Training samples:** samples used to decide on which function to use.
- **Test samples:** samples used to measure the accuracy of your proposed solution.

Train and test samples example

- Split your data into training and testing sets.
- **Training samples:** samples used to decide on which function to use.
- **Test samples:** samples used to measure the accuracy of your proposed solution.

- In the case of computer vision, with the cats/birds example
- **Training samples:** images used to train the AI to recognize cats/birds.
- **Test samples:** images our AI has never seen before, used to measure the accuracy of our AI

```python
#Show the data set
print(data.shape)
print(data)
```

```
(50, 2)
[[  53.          554.11031423]
 [  61.          625.22641092]
 [  67.          669.45843036]
 [  70.          668.06457267]
 [  76.          724.3742697 ]
 [  79.          759.67839012]
 [  84.          836.96868258]
 [  85.          836.93525736]
 [  87.          893.425784  ]
 [  88.          855.25445011]
 [  89.          866.89844538]
 [  92.          923.32979144]
 [  92.          947.34748427]
 [ 105.         1060.14656536]
 [ 108.         1081.32750774]
 [ 109.         1083.30988896]
 [ 114.         1124.57510573]
 [ 124.         1271.05508988]
 [ 127.         1310.86755257]
 [ 129.         1298.03866349]
 [ 140.         1428.60129896]
```

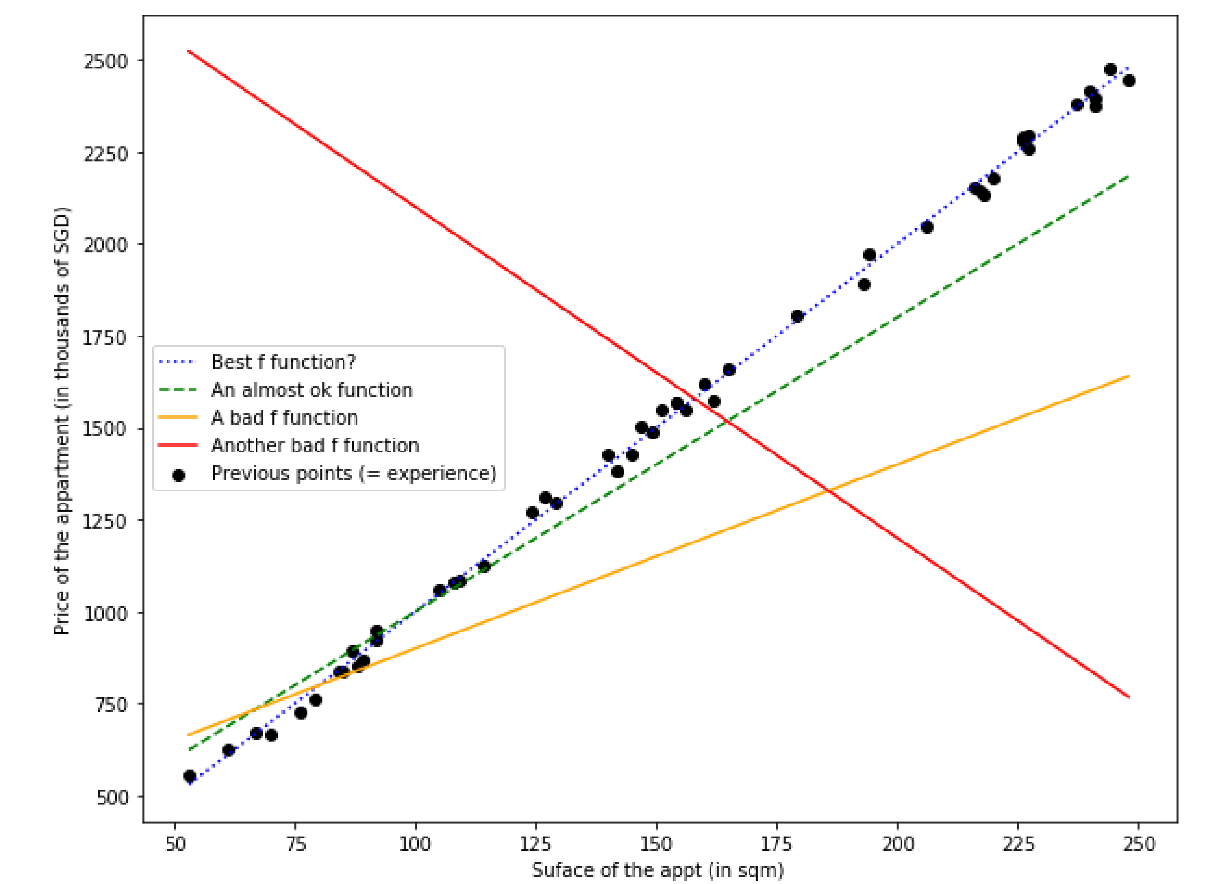
```python
# Import Train and test split from sklearn
from sklearn.model_selection import train_test_split

# Separate x and y
x_data = data[:,[0]]
y_data = data[:,[1]]

# Do the splitting
percentage_for_test = 0.4
random_seed = 42
x_train, x_test, y_train, y_test = train_test_split(x_data, \
                                                    y_data, \
                                                    test_size = percentage_for_test, \
                                                    random_state = random_seed)
```

```python
# Prints training data (for verification)
print("-- TRAIN SAMPLES")
#print(x_train)
print(x_train.shape)
print("-")
#print(y_train)
print(y_train.shape)

# Prints test data (for verification)
print("-- TEST SAMPLES")
#print(x_test)
print(x_test.shape)
print("-")
#print(y_test)
print(y_test.shape)
```

```
-- TRAIN SAMPLES
(30, 1)
-
(30, 1)
-- TEST SAMPLES
(20, 1)
-
(20, 1)
```

Introducing two performance metrics for regression: the mean square error and R2 score

- We can measure the performance of the proposed function in two ways.
- **Mean square error (MSE):** the closer it gets to zero, the better the function is.
- **R2 score (R2):** the closer it gets to 1, the better the function is.



Creating a Linear Regression model

- The sklearn library provides functions to find the best linear function f, that fits any set of training samples.

$$y = f(x) = a_0 + a_1 x$$

- It all revovles around a LinearRegression object.
    - Its attributes, after regression, give the function f coefficients.

```
# Import linear regression model
from sklearn import linear_model

# Initialize a LinearRegression object regr
regr = linear_model.LinearRegression()

# Find the best function using our training data
regr.fit(x_train, y_train)

# Print some regr object attributes (once regression is complete)
print(regr.coef_) # a1 coefficient
print(regr.intercept_) # a0 coefficient
```

```
[[9.97696503]]
[2.3099459]
```

Using your linear regression model on test samples and performance evaluation

- Once our regression is complete, we have a **regr** LinearRegression object.
- Use it on your test samples x_test, store the result in y_pred!
- Compute the MSE and R2 scores, to check the performance of your Linear Regression!

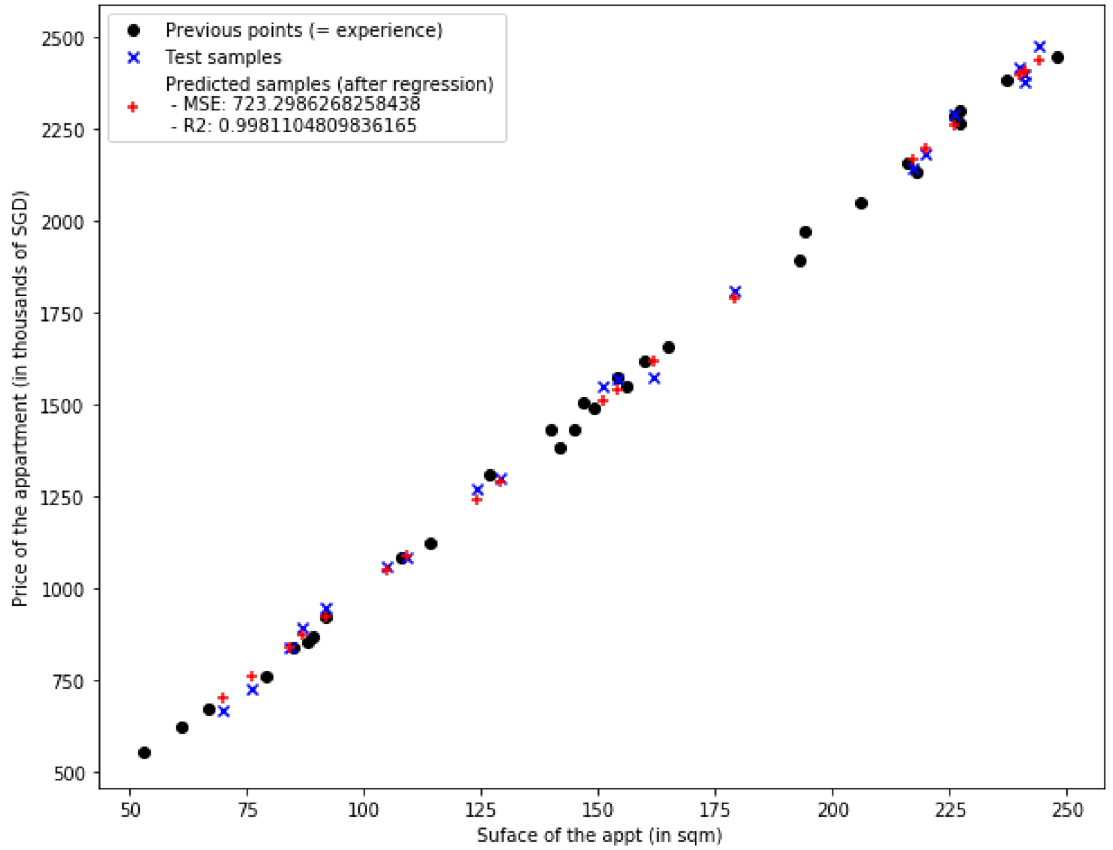
```
# Compute the values we would obtain for y,
# if we used our function on the test samples x_test
# Store it in y_pred
y_pred = regr.predict(x_test)

# Compute the MSE and R2 scores,
# using y_pred and y_test
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
MSE = mean_squared_error(y_pred, y_test)
R2 = r2_score(y_pred, y_test)
print(MSE)
print(R2)
```

```
723.2986268258438
0.9981104809836165
```

Final results display

```
predict_string = 'Predicted samples (after regression) \n - MSE: {} \n - R2: {}'.format(MSE, R2)
fig = plt.figure(figsize = (10,8))
plt.scatter(x_train, y_train, color = 'black', label = 'Previous points (= experience)')
plt.scatter(x_test, y_test, color = 'blue', marker = 'x', label = 'Test samples')
plt.scatter(x_test, y_pred, color = 'red', marker = '+', label = predict_string)
plt.legend(loc = 'best')
plt.xlabel('Suface of the appt (in sqm)')
plt.ylabel('Price of the appartment (in thousands of SGD)')
plt.show()
```

- At the end, plot your training samples, testing samples and prediction samples on the same graph!
- Add the MSE and R2 as well!



To recap

1. **Step 1:** Load data, and use a scatter plot, to check your data.
2. **Step 2:** Use the train_test_split, to split your record/experience into training (x_train, y_train) and testing (x_test, y_test) samples.
3. **Step 3:** Use the linear regression model from sklearn, and compute the fucntion coefficients, which have the optimal MSE and R2 performance.
4. **Step 4:** Predict your samples using this function on your x_test samples and store it in y_pred.
5. **Step 5:** Compute the MSE and R2 by using y_pred and y_test.
6. **Step 6:** Display your final results!

Let us practice with Q5

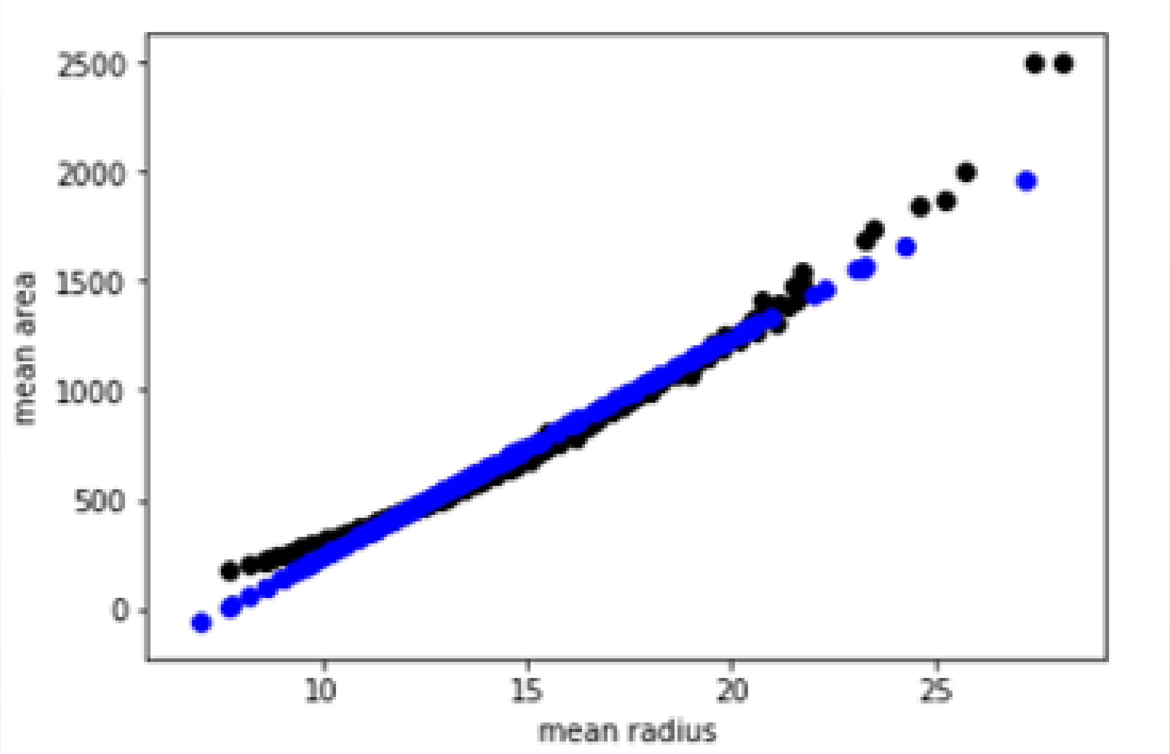
Q5 – Our first linear regression.

Q5: Your turn to play!

1. **Step 1:** Load data, and use a scatter plot, to check your data.
2. **Step 2:** Use the train_test_split, to split your record/experience into training (x_train, y_train) and testing (x_test, y_test) samples.
3. **Step 3:** Use the linear regression model from sklearn, and compute the fucntion coefficients, which have the optimal MSE and R2 performance.
4. **Step 4:** Predict your samples using this function on your x_test samples and store it in y_pred.
5. **Step 5:** Compute the MSE and R2 by using y_pred and y_test.
6. **Step 6:** Display your final results!

Recap: previous activity

- In the previous activity, we used a simple linear regression to try and fit our data.
- Not the best, because our data did not seem to have a linear trend, but a quadratic one.



Polynomial regression

- **Hypothesis:** in the polynomial regression approach, we assume that the missing function f is a **<u>polynomial</u>** function of degree **n**.

$$y = f(x) = a_0 + \sum_{i}^{n} a_i x^i$$

- **Objective:** find the coefficient values $(a_0, a_1, \dots a_n)$ that fit the data (x, y) in our record/experience, in the best possible way.

What changes?

- We just reuse the linear regression, but with multiple **polynomial features**.
- Basically, multiple x columns, for each power of x.
  - One for x, one for x^2, one for x^3, etc.
- And that's it!

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(order, include_bias=False)
c = poly.fit_transform(x)
c_train, c_test, y_train, y_test = train_test_split( c , y , test_size = size, random_state = seed)
regr = linear_model.LinearRegression()
regr.fit(c_train, y_train)
y_pred = regr.predict(c_test)
```

Let us practice with Q6

Q6 – Our first polynomial regression.

Q6: our first polynomial model

- As before with Q4…
  - but this time, we can have any polynomial function for f,
  - and have to specify the order of the polynomial fucntion in the arguments of our linear regression function.
- Almost same steps 1-6, but…
  - Compute polynomial features before the train/test split.
  - Use the train/test splits on the polynomial features.
  - Linear regression on the polynomial train/test samples.

Conclusion

- Today we covered two typical examples of linear and polynomial regression.
- If more time, let us go back to Q2 and Q3!

Let us practice a bit

Problem set 10 – Q2 & Q3 (5-number summary and normalization)

Q2: Five-number summary

- The **five-number summary**, is an informative function about data, listing
  - The **minimal** value in a given array
  - The **maximal** value in a given array
  - The **median** value in a given array
  - The **first quarter percentile** value in a given array
  - The **third quarter percentile** value in a given array

For Q2: Min, Max, Mean, Median, Percentile

- Numpy has functions for finding the
  - **Minimal** value,
  - **Maximal** value,
  - **Mean** value,
  - **Median** values,
  - Etc.
- For any given array, containing data.

```python
# Minimal value
print(np.min(matrix))
```

```
5.052808826566668e-06
```

```python
# Maximal value
print(np.max(matrix))
```

```
0.999945892478096
```

```python
# Mean value
print(np.mean(matrix))
```

```
0.5024344386819765
```

```python
# Median value
print(np.median(matrix))
```

```
0.5042077048148175
```

```python
# n%-percentile value: value of the element,
# which is greater than n% of the samples in matrix
n = 25
print(np.percentile(matrix, n))
```

```
0.2509906081803283
```

Let us practice a bit

Problem set 10 – Q2 & Q3 (5-number summary and normalization)

Q3: data normalization

- Data normalization is a typical operation in Machine Learning.
  - It re-scales the data, so that the minimal value in the data will become 0.
  - And the maximal value will become 1.

Input x1	Input x2
1	10
2	6
3	2
4	4
5	0

Input x1 (normalized)	Input x2 (normalized)
0	1
0.25	0.6
0.5	0.2
0.75	0.4
1	0

Q3: data normalization

- Data normalization is a typical operation in Machine Learning.
  - It re-scales the data, so that the minimal value in the data will become 0.
  - And the maximal value will become 1.
- Q3: write a function that receives a data array, and normalize the columns of the array one-by-one.

Input x1	Input x2
1	10
2	6
3	2
4	4
5	0

Input x1 (normalized)	Input x2 (normalized)
0	1
0.25	0.6
0.5	0.2
0.75	0.4
1	0