LECTURE 6

PYTHON PROGRAMMING

STATISTICS

DR. PRAPASSORN TANTIPHANWADI
INDUSTRIAL ENGINEERING, FACULTY OF ENGINEERING AT KHAMPAENGSAEN
DECEMBER 2565

CONTENT

- Random variables
- Sampling Distributions
- Hypothesis testing
- Simple and Multiple Linear Regression

https://www.w3schools.com/python/module_statistics.asp

DISTRIBUTION

a) Mean

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

```
In [1]: import numpy as np

In [2]: x = np.arange(10)

In [3]: np.mean(x)
Out[3]: 4.5

In [4]: xWithNan = np.hstack( (x, np.nan) )     # append nan

In [5]: np.mean(xWithNan)
Out[5]: nan

In [6]: np.nanmean(xWithNaN)
Out[6]: 4.5
```

DISTRIBUTION

b) Median

```
In [7]: np.median(x)
Out[7]: 4.5
```

c) Mode

```
In [8]: from scipy import stats

In [9]: data = [1, 3, 4, 4, 7]

In [10]: stats.mode(data)
Out[10]: (array([4]), array([ 2.]))
```

d) Geometric Mean

In some situations the geometric mean can be useful to describe the location of a distribution. It can be calculated via the arithmetic mean of the log of the values.

$$mean_{geometric} = \left(\prod_{i=1}^{N} x_i\right)^{1/N} = exp\left(\frac{\sum_i ln(x_i)}{n}\right)$$

```
In [11]: x = np.arange(1,101)

In [12]: stats.gmean(x)
Out[12]: 37.992689344834304
```

DISTRIBUTION - VARIATION

a) Range

```
range = np.ptp(x)
```

b) Percentiles

$$CDF(x) = \int_{-\infty}^{x} PDF(x)dx$$

$$\mathbf{P}(a \leq X \leq b) = \int_{a}^{b} PDF(x)dx = CDF(b) - CDF(a)$$

- To find the range which includes 95 % of the data, one has to find the 2.5th and the 97.5th percentile of the sample distribution.
- The 50th percentile is the *median*.
- Also important are the *quartiles*, i.e., the 25th and the 75th percentile. The difference between them is called the *inter-quartile range* (*IQR*).

DISTRIBUTION

c) Standard Deviation and Variance

$$var = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1} \qquad s = \sqrt{var}$$

```
In [1]: data = np.arange(7,14)

In [2]: np.std(data, ddof=0)
Out[2]: 2.0

In [3]: np.std(data, ddof=1)
Out[3]: 2.16025
```

population

sample

Note: In *pandas*, the default for the calculation of the standard deviation is set to *ddof=1*.

d) Standard Error

For normally distributed data, the sample standard error of the mean (SE or SEM) is

$$SEM = \frac{s}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}} \cdot \frac{1}{\sqrt{n}}$$

e) Confidence Intervals

$$ci = mean \pm std * N_{PPF}\left(\frac{1-\alpha}{2}\right)$$

DISTRIBUTION

PARAMETERS DESCRIBING THE FORM OF A DISTRIBUTION

a) Location

A *location parameter* x_0 determines the location or shift of a distribution: $f_{x0}(x) = f(x - x_0)$

b) Scale

S=scale

The *scale parameter* describes the width of a probability distribution. $f_s(x) = f(x/s)/s$

where f is the density of a standardized version of the density.

c) Shape Parameters

Skewness Distributions are *skewed* if they depart from symmetry if the standard deviation is more than half the mean.

Such an asymmetry is referred to as *positive skewness*. The opposite, negative skewness, is rare.

Kurtosis is a measure of the "peakedness" of the probability distribution

DISTRIBUTION

- *Cumulative distribution function* (*CDF*): gives the probability of obtaining a value smaller than the given value.
  Example: What is the chance that a man is less than 165 cm tall?
- *Survival Function* (*SF*) = 1 − CDF: gives the probability of obtaining a value larger than the given value. It can also be interpreted as the proportion of data "surviving" above a certain value.
  Example: What is the chance that a man is larger than 165 cm?
- *Percentile Point Function* (*PPF*): the inverse of the CDF. The PPF answers the question "Given a certain probability, what is the corresponding input value for the CDF?"
  Example: Given that I am looking for a man who is smaller than 95 % of all other men, what size does the subject have to be?
- *Inverse Survival Function* (*ISF*): the name says it all.
  Example: Given that I am looking for a man who is larger than 95 % of all other men, what size does the subject have to be?
- Random Variate Sample (*RVS*): random variates from a given distribution. (A *variable* is the general type, a *variate* is a specific number.)

Note: In Python, the way of working with istribution functions is a two-step procedure:
• In the first step, you create the distribution (e.g., nd = stats.norm()).
• In the second step, you decide which function you want to use from this distribution, and calculate the function value for the desired x-input (e.g., y = nd.cdf(x)).

DISTRIBUTION

```
import numpy as np
from scipy import stats

myDF = stats.norm(5,3)     # Create the frozen distribution

x = np.linspace(-5, 15, 101)
y = myDF.cdf(x)          # Calculate the corresponding CDF
```

DISCRETE DISTRIBUTION

BERNOULLI DISTRIBUTION

```
In [1]: from scipy import stats
In [2]: p = 0.5
In [3]: bernoulliDist = stats.bernoulli(p)
In [4]: p_tails = bernoulliDist.pmf(0)
In [5]: p_heads = bernoulliDist.pmf(1)
```

we can simulate 10 Bernoulli trials with

```
In [6]: trials = bernoulliDist.rvs(10)

In [7]: trials
Out[7]: array([0, 0, 0, 1, 0, 0, 0, 1, 1, 0])
```

In `In[6]`, *rvs* stands for *random variates*.

DISCRETE DISTRIBUTION

BINOMIAL DISTRIBUTION

$$P[X = k] = p^k (1-p)^{n-k} \quad 0 \le p \le 1, \quad n \in \mathbf{N} \quad \text{where } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

First we generate the frozen distribution function, for example for four coin tosses:

```
In [1]: from scipy import stats
In [2]: import numpy as np

In [3]: (p, num) = (0.5, 4)
In [4]: binomDist = stats.binom(num, p)
```

	Mean	Variance
Binomial	$n \cdot p$	$np(1-p)$
Poisson	λ	λ

and then we can calculate, e.g., the probabilities how often heads come up during those four tosses, given by the PMF for the values zero to four:

```
In [5]: binomDist.pmf(np.arange(5))
Out[5]: array([ 0.0625,  0.25  ,  0.375 ,  0.25  ,  0.0625])
```

For example, the chance that heads never comes up is about 6 %, the chance that it comes up exactly once is 25 %, etc.

HYPOTHESIS TESTING

Typical Analysis Procedure

- Data Screening and Outliers
- Normality Check

QQ-Plots	The "Q" in QQ-plot stands for *quantile*. The quantiles of a given data set are plotted against the quantiles of a reference distribution, typically the standard normal distribution.
PP-Plots	Plot the CDF (cumulative-distribution-function) of a given data set against the CDF of a reference distribution.
Probability Plots	Plot the ordered values of a given data set against the quantiles of a reference distribution.

- Tests for Normality

HYPOTHESIS TESTING

- Tests for Normality

Below the output of the *Python*-module `C7_1_checkNormality.py` is shown, which checks 1000 random variates from a normal distribution for normality. Note that while for the full data set all tests correctly indicate that the underlying distribution is normal, the effects of extreme values strongly depend on the type of test if only the first 100 random variates are included.

```
p-values for all 1000 data points: ----------------
Omnibus                 0.913684
Shapiro-Wilk            0.558346
Lilliefors              0.569781
Kolmogorov-Smirnov      0.898967
```

Code: "ISP_checkNormality.py"

check graphically, as well as with different quantitative tests, if a given distribution is normal.

HYPOTHESIS TESTING

Test of one mean

If we have the hypothesis that the population we draw our samples from has a mean value of x_1 and a standard deviation of σ, and the actual population has a mean value of $x_1 + D$ and the same standard deviation, we can find such a difference with a minimum sample number of

$$n = \frac{(z_{1-\alpha/2} + z_{1-\beta})^2}{d^2} \qquad z = \frac{x - \mu}{\sigma}$$

and $d = \frac{D}{\sigma}$ the effect size.

In words, if the real mean has a value of x_1, we want to detect this correctly in at least $1 - \alpha\%$ of all tests; and if the real mean is shifted by D or more, we want to detect this with a likelihood of at least $1 - \beta\%$.

HYPOTHESIS TESTING

Test of One mean

```
In [1]: from statsmodels.stats import power

In [2]: nobs = power.tt_ind_solve_power(
                    effect_size = 0.5, alpha =0.05, power=0.8 )

In [3]: print(nobs)
Out[3]: 63.76561177540974
```

tells us that if we compare two groups with the same number of subjects and the same standard deviation, require an $\alpha = 0.05$ and a test power of 80 %, and we want to detect a difference between the groups that is half the standard deviation, we need to test 64 subjects in each group.

HYPOTHESIS TESTING

Test of One mean

Similarly,

```
In [4]: effect_size = power.tt_ind_solve_power(
                      alpha =0.05, power=0.8, nobs1=25 )

In [5]: print(effect_size)
Out[5]: 0.8087077886680407
```

tells us that if we have an $\alpha = 0.05$, a test power of 80 %, and 25 subjects in each group, then the smallest difference between the groups is 81 % of the sample standard deviation.

The corresponding command for one sample *t*-tests is `tt_solve_power`.

HYPOTHESIS TESTING

Test Between Two Different Populations

For finding a difference between two normally distributed means, with standard deviations of σ_1 and σ_2, the minimum number of samples we need in each group to detect an absolute difference D is

$$n_1 = n_2 = \frac{(z_{1-\alpha/2} + z_{1-\beta})^2(\sigma_1^2 + \sigma_2^2)}{D^2}.$$

Code: "ISP_sampleSize.py'

HYPOTHESIS TESTING

		Condition		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive**	False Positive (Type I error)	**Positive predictive value=** $\frac{\Sigma \text{True Positive}}{\Sigma \text{Test Outcome Positive}}$
	Test Outcome Negative	False Negative (Type II error)	**True Negative**	**Negative predictive value=** $\frac{\Sigma \text{True Negative}}{\Sigma \text{Test Outcome Negative}}$
		Sensitivity = $\frac{\Sigma \text{True Positive}}{\Sigma \text{Condition Positive}}$	**Specificitivity =** $\frac{\Sigma \text{True Negative}}{\Sigma \text{Condition Negative}}$	

		Condition		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive** (TP) = 25	False Positive (FP) = 175	**Positive predictive value=** = TP / (TP+FP) = 25 / (25+175) = 12.5%
	Test Outcome Negative	False Negative (FN) = 10	**True Negative** (TN) = 2000	**Negative predictive value=** = TN / (FN+TN) = 2000 / (10+2000) = 99.5%
		Sensitivity = = TP / (TP+FN) = 25 / (25+10) = 71%	**Specificitivity =** = TN / (FP+TN) = 2000 / (175+2000) = 92%	

- False positive rate (α) = type I error = $1 - \textit{specificity} = \frac{FP}{FP+TN} = \frac{175}{175+2000} = 8\,\%$
- False negative rate (β) = type II error = $1 - \textit{sensitivity} = \frac{FN}{TP+FN} = \frac{10}{25+10} = 29\,\%$
- Power = sensitivity = $1 - \beta$
- $\textit{positive likelihood ratio} = \frac{\textit{sensitivity}}{1-\textit{specificity}} = \frac{71\,\%}{1-92\,\%} = 8.9$
- $\textit{negative likelihood ratio} = \frac{1-\textit{sensitivity}}{\textit{specificity}} = \frac{1-71\,\%}{92\,\%} = 0.32$

HYPOTHESIS TESTING

One Sample t-Test for a Mean Value

As an example we take 100 normally distributed data, with a mean of 7 and with a standard deviation of 3. What is the chance of finding a mean value at a distance of 0.5 or more from the mean?

Answer: The probability from the t-test in the example is 0.057, and from the normal distribution 0.054.

In *Python*, test statistic and *p*-value for the one sample *t*-test can be calculated with

```
t, pVal = stats.ttest_1samp(data, checkValue)
```

Code: "ISP_oneGroup.py"[1]: Sample analysis for one group of continuous data.

HYPOTHESIS TESTING

Paired t-Test

Therefore the two tests `stats.ttest_1samp` and `stats.ttest_rel` provide the same result (apart from minute numerical differences):

```
In [1]: import numpy as np
In [2]: from scipy import stats

In [3]: np.random.seed(1234)
In [4]: data = np.random.randn(10)+0.1
In [5]: data1 = np.random.randn(10)*5  # dummy data
In [6]: data2 = data1 + data    # same group-difference as "data"

In [7]: stats.ttest_1samp(data, 0)
Out[7]: (-0.1245849229873140l, 0.9035904508547089)

In [8]: stats.ttest_rel(data2, data1)
Out[8]: (-0.1245849229873135, 0.9035904508547089)
```

Paired t-tests can detect differences that would be insignificant otherwise. For this example all the inner-subject differences are positive, and the paired t-test yields a p-value of p < 0:001, while the unpaired t-test leads to p=0.81

t-Test between Independent Groups

```
t_statistic, pVal = stats.ttest_ind(group1, group2)
```

HYPOTHESIS TESTING

The command `random.seed(123)` initializes the random number generator with the number 123, which ensures that two consecutive runs of this code produce the same result, corresponding to the numbers given above.

a) Classical *t*-Test

Let us take performance measurements from a racing team, on two different occasions. During Race1, the members of the team achieve a score of [79, 100, 93, 75, 84, 107, 66, 86, 103, 81, 83, 89, 105, 84, 86, 86, 112, 112, 100, 94], and during Race2 a score of [92, 100, 76, 97, 72, 79, 94, 71, 84, 76, 82, 57, 67, 78, 94, 83, 85, 92, 76, 88].

```
import numpy as np
from scipy import stats

# Generate the data
np.random.seed(123)
race_1 = np.round(np.random.randn(20)*10+90)
race_2 = np.round(np.random.randn(20)*10+85)

# t-test
(t, pVal) = stats.ttest_rel (race_1, race_2)

# Show the result
print('The probability that the two distributions '
      'are equal is {0:5.3f} .'.format(pVal))
```

which produce

```
The probability that the two distributions are equal is 0.033
```

HYPOTHESIS TESTING

b) Statistical Modeling

Expressed as a statistical model, we assume that the difference between the first and the second race is simply a constant value. (The null hypothesis would be that this value is equal to zero.) This model has one parameter: the constant value. We can find this parameter, as well as its confidence interval and a lot of additional information, with the following *Python* code:

```
import pandas as pd
import statsmodels.formula.api as sm

np.random.seed(123)
df = pd.DataFrame({'Race1': race_1, 'Race2':race_2})

result = sm.ols(formula='I(Race2-Race1) ~ 1', data=df).fit()

print(result.summary())
```

ANOVA

b) Example: One-Way ANOVA

As an example, take the red cell folate levels ($\mu g/l$) in three groups of cardiac bypass patients given different levels of nitrous oxide ventilation (Amess et al. 1978), described in the *Python* code example below. In total 22 patients were included in the analysis.

The null hypothesis of ANOVAs is that all groups come from the same population. A test whether to keep or reject this null hypothesis can be done with

```
from scipy import stats
F_statistic, pVal = stats.f_oneway(group_1, group_2, group_3)
```

ANOVA

ISP_anovaOneway.py

statsmodels:

```
import pandas as pd
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

df = pd.DataFrame(data, columns=['value', 'treatment'])
model = ols('value ~ C(treatment)', df).fit()
anovaResults = anova_lm(model)
print(anovaResults)
```

where the numerical values are in the first column of the array `data`, and the (categorical) group variable in the second column. This produces the following output:

```
                DF     SS          MS      F    p(>F)
C(treatment)     2  15515.76  7757.88   3.71  0.043
Residual        19  39716.09  2090.32    NaN    NaN
```

TWO-WAY ANOVA

Code: "ISP_anovaTwoway.py"

```
import pandas as pd
from C2_8_getdata import getData
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Get the data
data = getData('altman_12_6.txt', subDir='..\Data\data_altman')

# Bring them in DataFrame-format
df = pd.DataFrame(data, columns=['hs', 'fetus', 'observer'])

# Determine the ANOVA with interaction
formula = 'hs ~ C(fetus) + C(observer) + C(fetus):C(observer)'
lm = ols(formula, df).fit()
anovaResults = anova_lm(lm)

print(anovaResults)
```

TWO-WAY ANOVA

```
print(anovaResults)
```

This leads to the following result:

	df	sum_sq	mean_sq	F	PR(>F)
C(fetus)	2	324.00	162.00	2113.10	1.05e-27
C(observer)	3	1.19	0.39	5.21	6.497-03
C(fetus):C(observer)	6	0.56	0.09	1.22	3.29e-01
Residual	24	1.84	0.07	NaN	NaN

In words: While—as expected—different fetuses show highly significant differences in their head size ($p < 0.001$), also the choice of the observer has a significant effect ($p < 0.05$). However, no individual observer was significantly off with any individual fetus ($p > 0.05$).

LINEAR REGRESSION

Example 1: Simple Linear Regression

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i,$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \\ 1 & x_6 \\ 1 & x_7 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix}$$

Code: "ISP_fitLine.py'

Example 1: Line Fit with Confidence Intervals

For univariate distributions, the confidence intervals based on the standard deviation indicate the interval that we expect to contain 95 % of the data, and the confidence intervals based on the standard error of the mean indicate the interval that contains the true mean with 95 % probability. We also have these two types of confidence intervals (one for the data, and one for the fitted parameters) for line fits,

LINEAR REGRESSION

Example: Tobacco and Alcohol in the UK

```
In [1]: import numpy as np
   ...: import pandas as pd
   ...: import matplotlib as mpl
   ...: import matplotlib.pyplot as plt
   ...: import statsmodels.formula.api as sm
   ...: from sklearn.linear_model import LinearRegression
   ...: from scipy import stats
```

ISP_multipleRegression.py

LINEAR REGRESSION

Example: Tobacco and Alcohol in the UK

```
In [2]: data_str = '''Region Alcohol Tobacco
   ...: North 6.47 4.03
   ...: Yorkshire 6.13 3.76
   ...: Northeast 6.19 3.77
   ...: East_Midlands 4.89 3.34
   ...: West_Midlands 5.63 3.47
   ...: East_Anglia 4.52 2.92
   ...: Southeast 5.89 3.20
   ...: Southwest 4.79 2.71
   ...: Wales 5.27 3.53
   ...: Scotland 6.08 4.51
   ...: Northern_Ireland 4.02 4.56'''
   ...:
   ...: # Read in the data. Note that for Python 2.x,
   ...: # you have to change the "import" statement
   ...: from io import StringIO
   ...: df = pd.read_csv(StringIO(data_str), sep=r'\s+')
   ...:
```

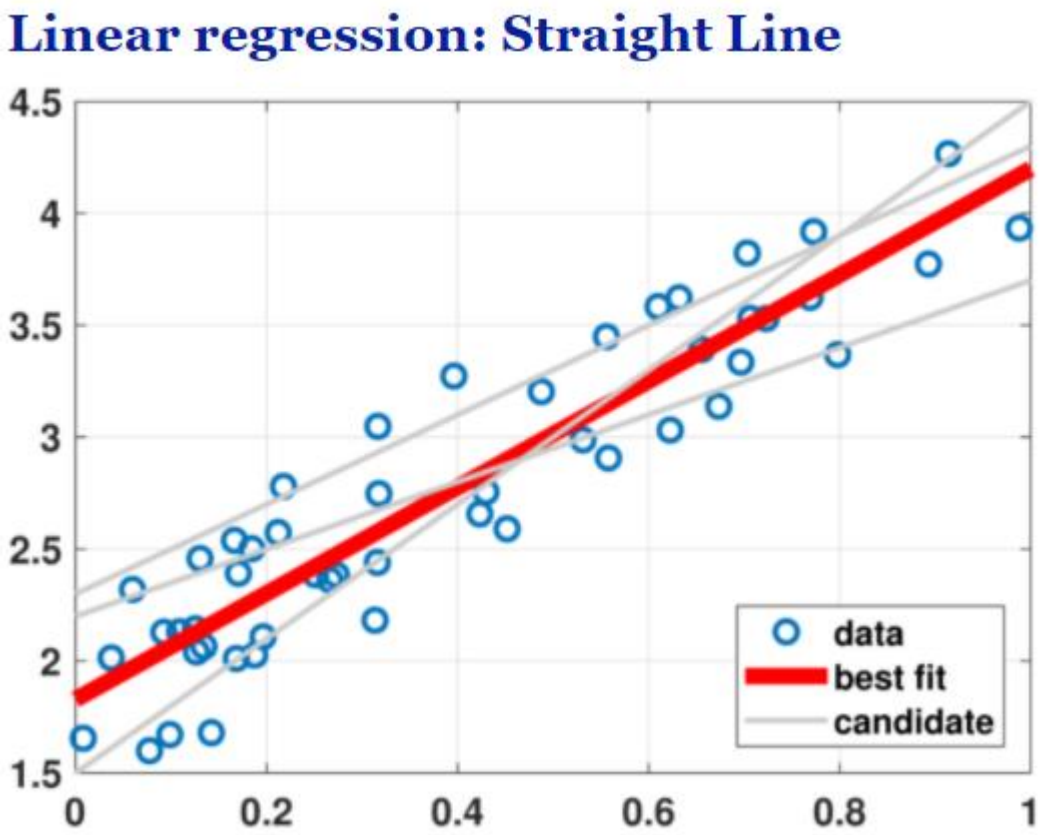
LINEAR REGRESSION

Example: Tobacco and Alcohol in the UK

```
In [3]: # Plot the data
   ...: df.plot('Tobacco', 'Alcohol', style='o')
   ...: plt.ylabel('Alcohol')
   ...: plt.title('Sales in Several UK Regions')
   ...: plt.show()
```

ADDITIONS

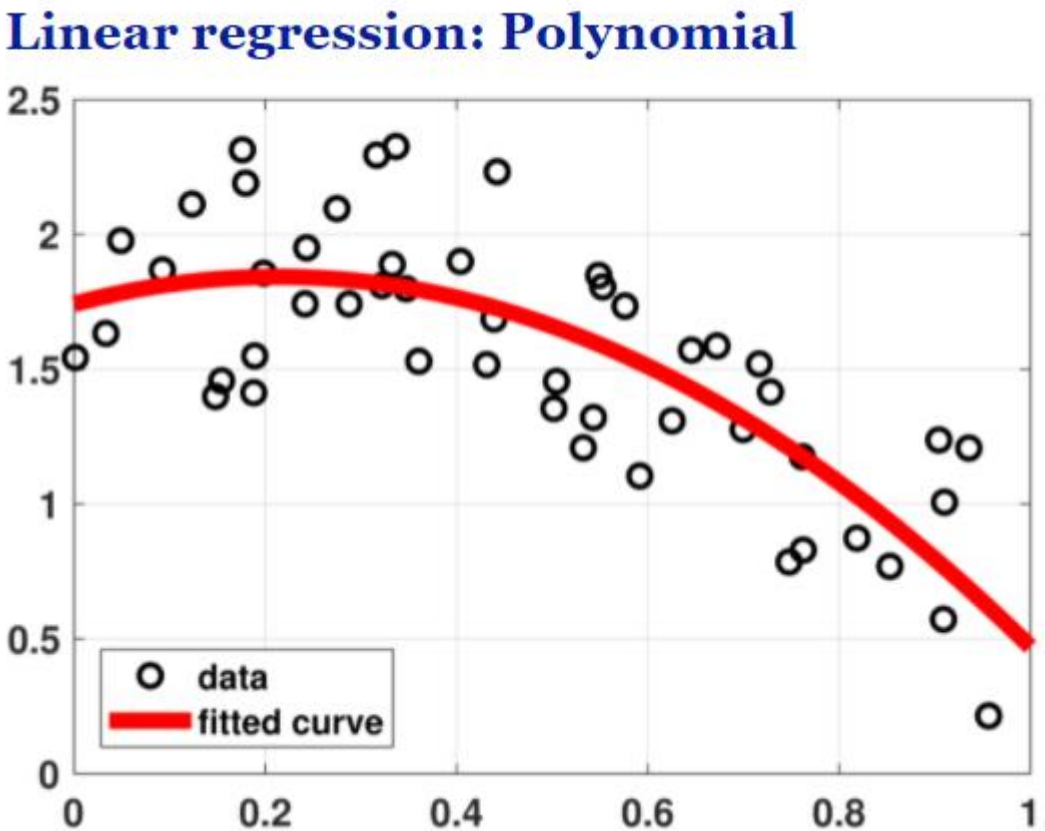
Regression




```
# Python code to fit data points using a straight line
import numpy as np
import matplotlib.pyplot as plt

N = 50
x = np.random.rand(N)
a = 2.5                                  # true parameter
b = 1.3                                  # true parameter
y = a*x + b + 0.2*np.random.randn(N)     # Synthesize training data

X = np.column_stack((x, np.ones(N)))     # construct the X matrix
theta = np.linalg.lstsq(X, y, rcond=None)[0] # solve y = X theta

t = np.linspace(0,1,200)                 # interpolate and plot
yhat = theta[0]*t + theta[1]
plt.plot(x,y,'o')
plt.plot(t,yhat,'r',linewidth=4)
```

ADDITIONS

Regression

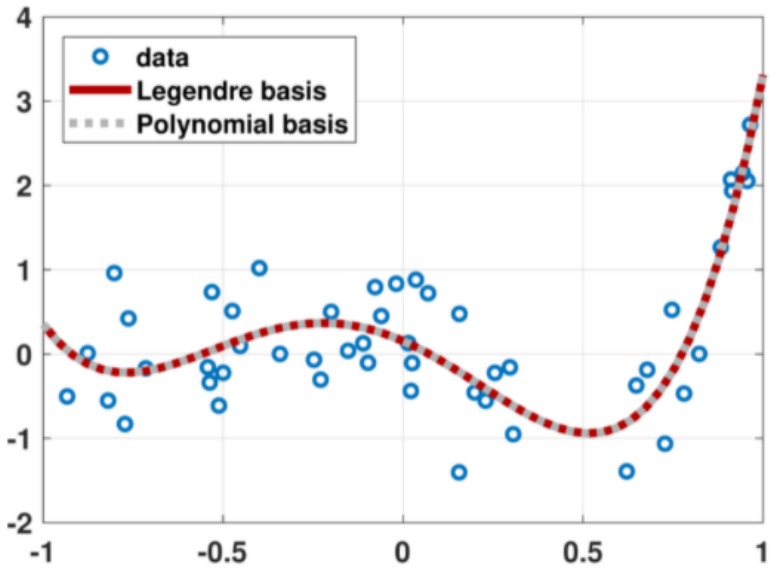


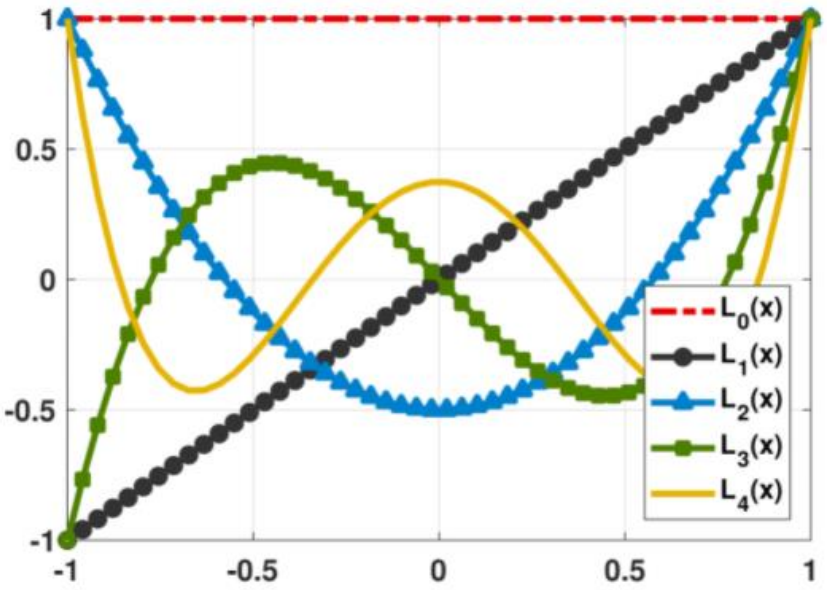

```
# Python code to fit data using a quadratic equation
import numpy as np
import matplotlib.pyplot as plt

N = 50
x = np.random.rand(N)
a = -2.5
b = 1.3
c = 1.2
y = a*x**2 + b*x + c + 0.2*np.random.randn(N)

X = np.column_stack((np.ones(N), x, x**2))
theta = np.linalg.lstsq(X, y, rcond=None)[0]
t = np.linspace(0,1,200)
yhat = theta[0] + theta[1]*t + theta[2]*t**2
plt.plot(x,y,'o')
plt.plot(t,yhat,'r',linewidth=4)
```

ADDITIONS

Regression

Legendre Polynomial



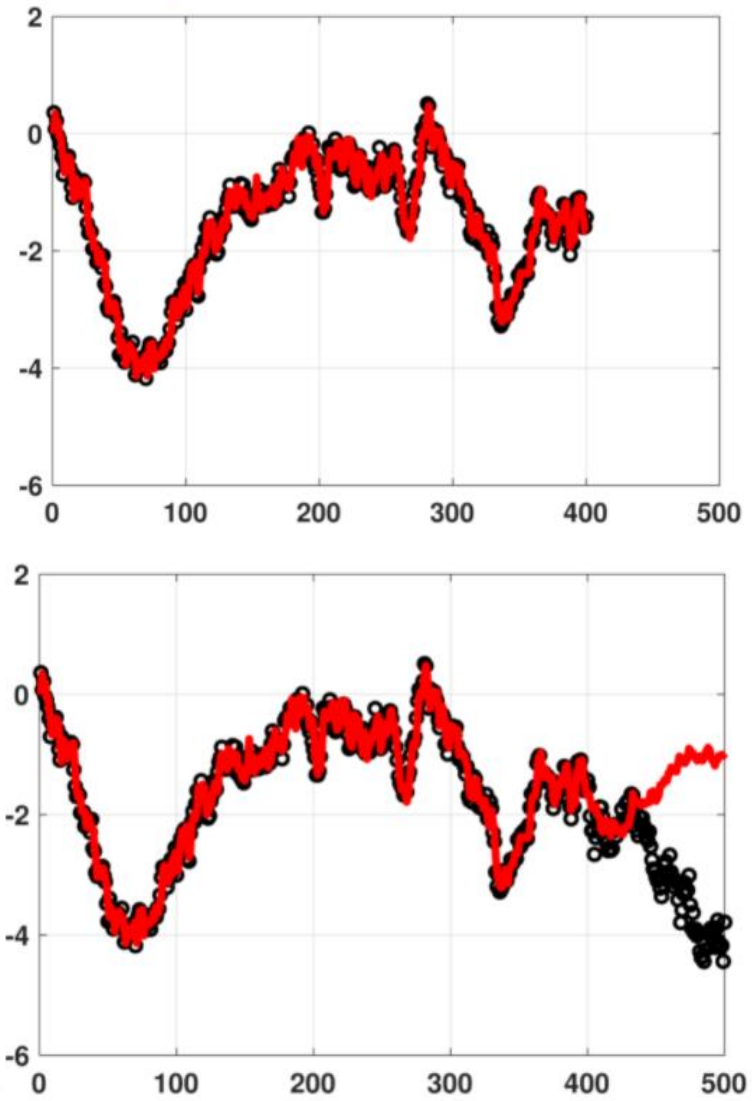



```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_legendre

N = 50
x = np.linspace(-1,1,N)
a = np.array([-0.001, 0.01, 0.55, 1.5, 1.2])
y = a[0]*eval_legendre(0,x) + a[1]*eval_legendre(1,x) + \
    a[2]*eval_legendre(2,x) + a[3]*eval_legendre(3,x) + \
    a[4]*eval_legendre(4,x) + 0.2*np.random.randn(N)

X = np.column_stack((eval_legendre(0,x), eval_legendre(1,x), \
                     eval_legendre(2,x), eval_legendre(3,x), \
                     eval_legendre(4,x)))
theta = np.linalg.lstsq(X, y, rcond=None)[0]
t     = np.linspace(-1, 1, 50);
yhat  = theta[0]*eval_legendre(0,t) + theta[1]*eval_legendre(1,t) + \
        theta[2]*eval_legendre(2,t) + theta[3]*eval_legendre(3,t) + \
        theta[4]*eval_legendre(4,t)
plt.plot(x,y,'o',markersize=12)
plt.plot(t,yhat, linewidth=8)
plt.show()
```

ADDITIONS

Regression

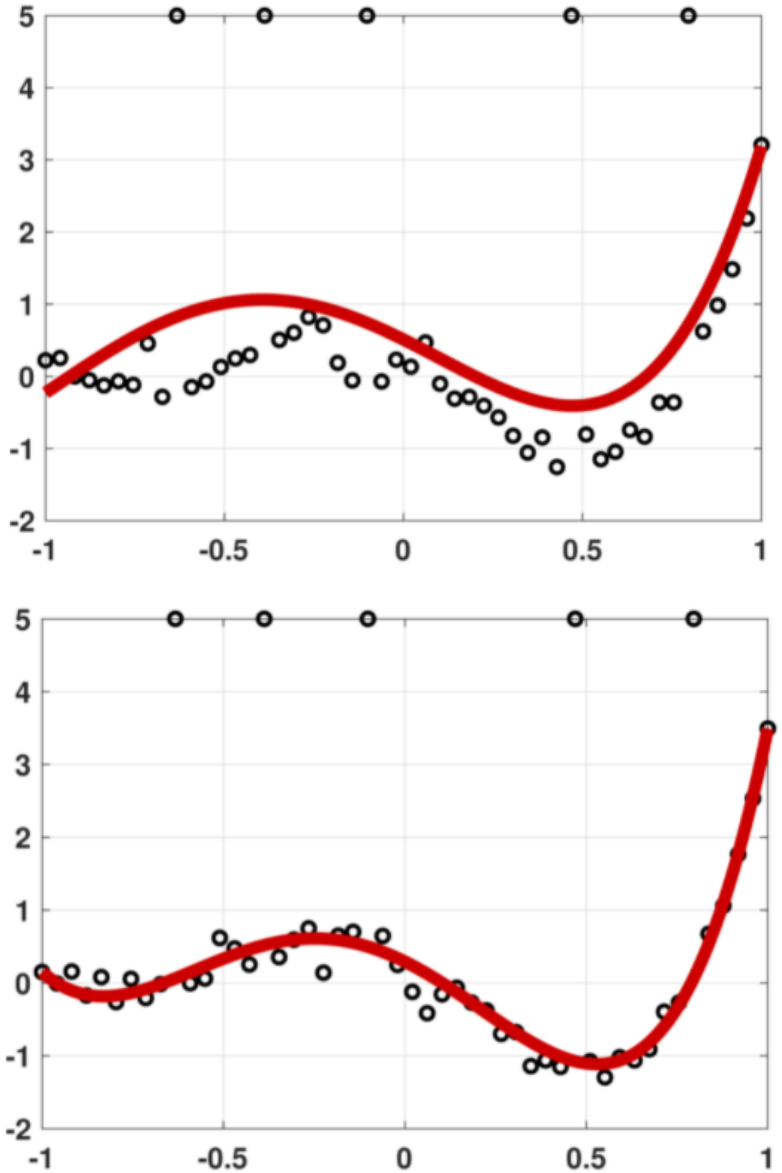
Auto-regressive model




```
# Python code for auto-regressive model
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import toeplitz

N = 500
y = np.cumsum(0.2*np.random.randn(N)) + 0.05*np.random.randn(N)

L = 100
c = np.hstack((0, y[0:400-1]))
r = np.zeros(L)
X = toeplitz(c,r)
theta = np.linalg.lstsq(X, y[0:400], rcond=None)[0]
yhat  = np.dot(X, theta)
plt.plot(y[0:400], 'o')
plt.plot(yhat[0:400],linewidth=4)
```

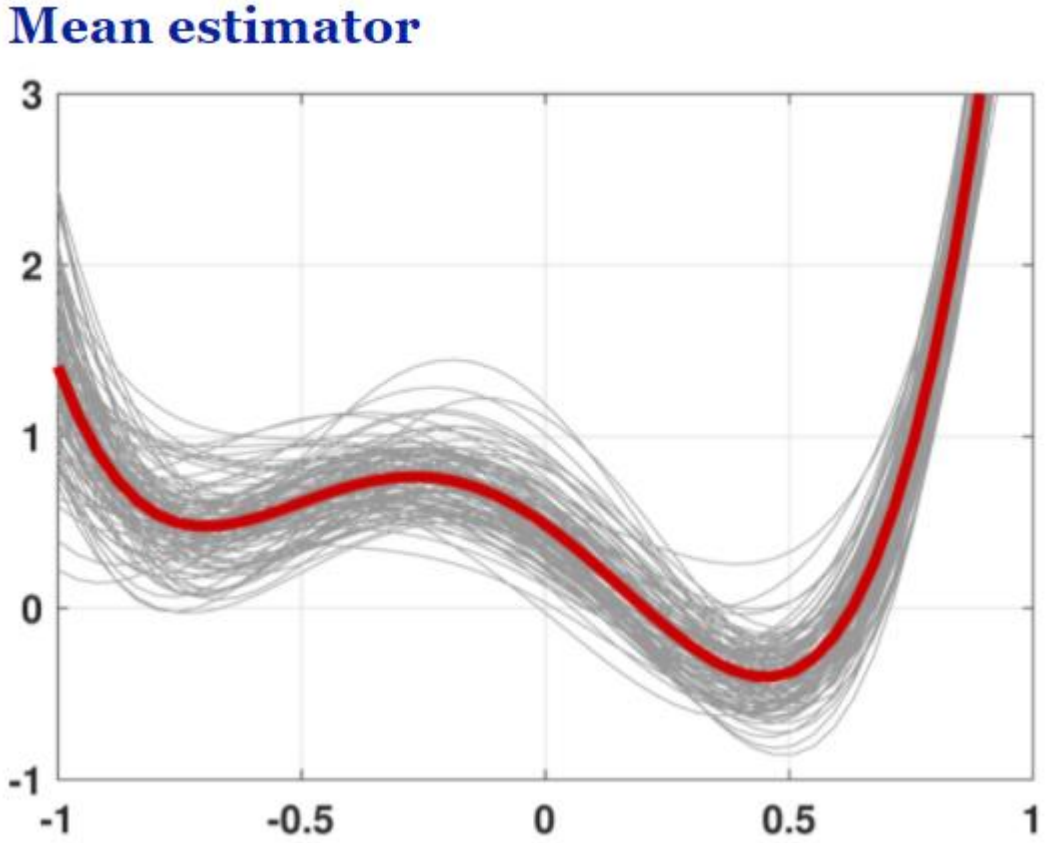
ADDITIONS

Robust regression by linear programming



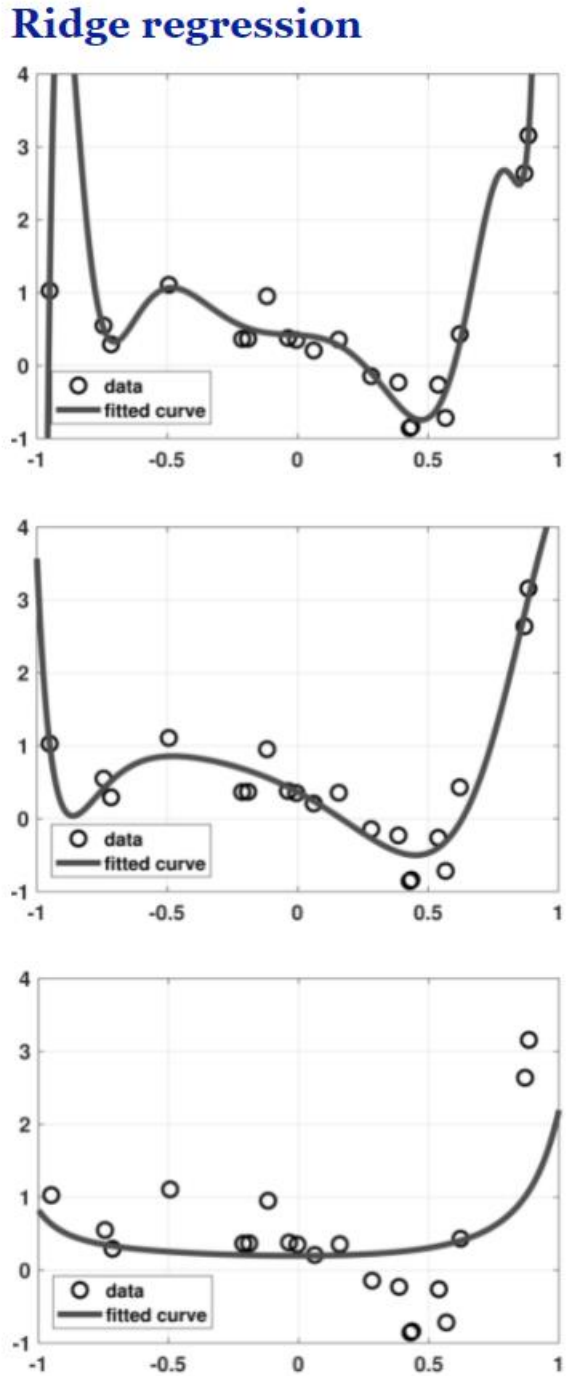

```
# Python code to demonstrate robust regression
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_legendre
from scipy.optimize import linprog

N = 50
x = np.linspace(-1,1,N)
a = np.array([-0.001, 0.01, 0.55, 1.5, 1.2])
y = a[0]*eval_legendre(0,x) + a[1]*eval_legendre(1,x) + \
    a[2]*eval_legendre(2,x) + a[3]*eval_legendre(3,x) + \
    a[4]*eval_legendre(4,x) + 0.2*np.random.randn(N)
idx = [10,16,23,37,45]
y[idx] = 5
X = np.column_stack((np.ones(N), x, x**2, x**3, x**4))
A = np.vstack((np.hstack((X, -np.eye(N))), np.hstack((-X, -np.eye(N)))))
b = np.hstack((y,-y))
c = np.hstack((np.zeros(5), np.ones(N)))
res = linprog(c, A, b, bounds=(None,None), method="revised simplex")
theta = res.x
t    = np.linspace(-1,1,200)
yhat = theta[0]*np.ones(200) + theta[1]*t + theta[2]*t**2 + \
       theta[3]*t**3 + theta[4]*t**4
plt.plot(x,y,'o',markersize=12)
plt.plot(t,yhat, linewidth=8)
plt.show()
```

ADDITIONS




```
# Python code to visualize the average predictor
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_legendre
np.set_printoptions(precision=2, suppress=True)

N = 20
x = np.linspace(-1,1,N)
a = np.array([0.5, -2, -3, 4, 6])
yhat = np.zeros((50,100))
for i in range(100):
  y = a[0] + a[1]*x    + a[2]*x**2 + \
           a[3]*x**3 + a[4]*x**4 + 0.5*np.random.randn(N)
  X = np.column_stack((np.ones(N), x, x**2, x**3, x**4))

  theta = np.linalg.lstsq(X, y, rcond=None)[0]
  t = np.linspace(-1,1,50)
  Xhat = np.column_stack((np.ones(50), t, t**2, t**3, t**4))
  yhat[:,i] = np.dot(Xhat, theta)
  plt.plot(t, yhat[:,i], c='gray')
plt.plot(t, np.mean(yhat, axis=1), c='r', linewidth=4)
```

ADDITIONS

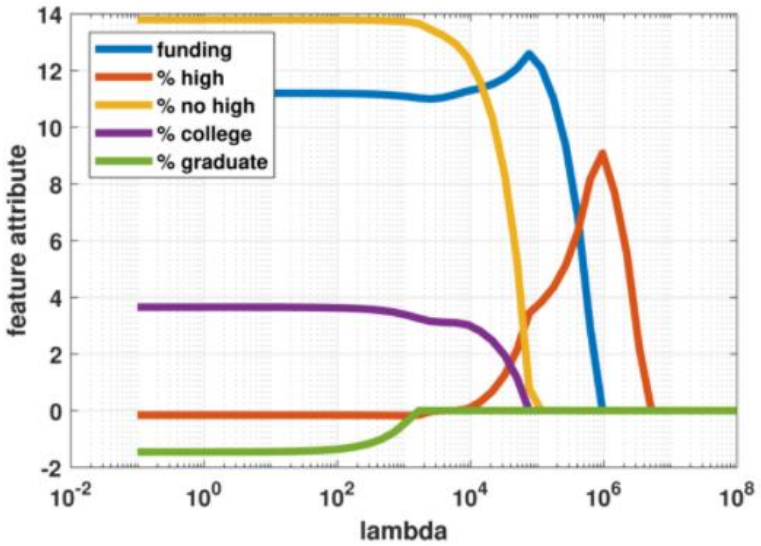


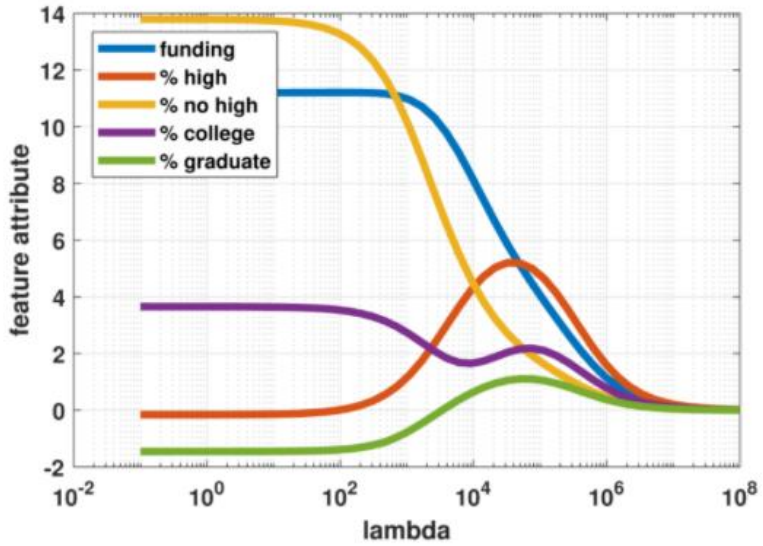

```
# Python code to demonstrate a ridge regression example
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_legendre
np.set_printoptions(precision=2, suppress=True)

N = 20
x = np.linspace(-1,1,N)
a = np.array([0.5, -2, -3, 4, 6])
y = a[0] + a[1]*x    + a[2]*x**2 + \
         a[3]*x**3 + a[4]*x**4 + 0.2*np.random.randn(N)

d = 20
X = np.zeros((N, d))
for p in range(d):
  X[:,p] = x**p

lambd = 0.1
A = np.vstack((X, np.sqrt(lambd)*np.eye(d)))
b = np.hstack((y, np.zeros(d)))
theta = np.linalg.lstsq(A, b, rcond=None)[0]

t    = np.linspace(-1, 1, 500)
Xhat = np.zeros((500,d))
for p in range(d):
  Xhat[:,p] = t**p
yhat = np.dot(Xhat, theta)

plt.plot(x,y,'o',markersize=12)
plt.plot(t,yhat, linewidth=4)
plt.show()
```

ADDITIONS

LASSO regression

Data download: ch7_data_crime.txt (1KB)

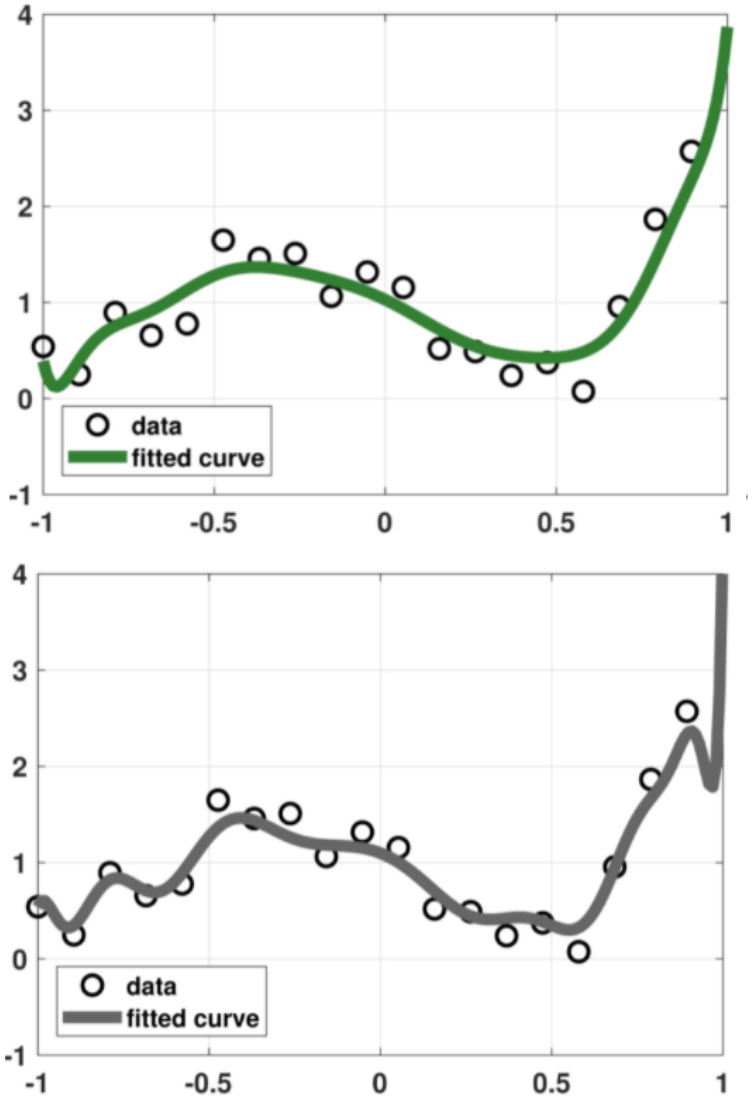





```
# Python

import cvxpy as cvx
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt("/content/ch7_data_crime.txt")
y = data[:,0]
X = data[:,2:7]
N,d = X.shape

lambd_set = np.logspace(-1,8,50)
theta_store = np.zeros((d,50))
for i in range(50):
  lambd = lambd_set[i]
  theta = cvx.Variable(d)
  objective   = cvx.Minimize( cvx.sum_squares(X*theta-y) \
              + lambd*cvx.norm1(theta) )
# objective   = cvx.Minimize( cvx.sum_squares(X*theta-y) \
              + lambd*cvx.sum_squares(theta) )
  prob        = cvx.Problem(objective)
  prob.solve()
  theta_store[:,i] = theta.value

for i in range(d):
  plt.semilogx(lambd_set, theta_store[i,:])
```

ADDITIONS

LASSO vs Ridge




```
# Python code to demonstrate overfitting and LASSO
import cvxpy as cvx
import numpy as np
import matplotlib.pyplot as plt

# Setup the problem
N = 20
x = np.linspace(-1,1,N)
a = np.array([1, 0.5, 0.5, 1.5, 1])
y = a[0]*eval_legendre(0,x) + a[1]*eval_legendre(1,x) + \
    a[2]*eval_legendre(2,x) + a[3]*eval_legendre(3,x) + \
    a[4]*eval_legendre(4,x) + 0.25*np.random.randn(N)

# Solve LASSO using CVX
d = 20
lambd = 1
X = np.zeros((N, d))
for p in range(d):
  X[:,p] = eval_legendre(p,x)

theta       = cvx.Variable(d)
objective   = cvx.Minimize( cvx.sum_squares(X*theta-y) \
                    + lambd*cvx.norm1(theta) )
prob        = cvx.Problem(objective)
prob.solve()
thetahat = theta.value

# Plot the curves
t = np.linspace(-1, 1, 500)
Xhat = np.zeros((500,d))
for p in range(P):
  Xhat[:,p] = eval_legendre(p,t)
yhat = np.dot(Xhat, thetahat)
plt.plot(x, y, 'o')
plt.plot(t, yhat, linewidth=4)
```

ADDITIONS - HYPOTHESIS

Compute confidence interval

Given α, compute the width of the confidence interval:

$$\epsilon \geq \Phi^{-1}\left(1-\frac{\alpha}{2}\right)$$

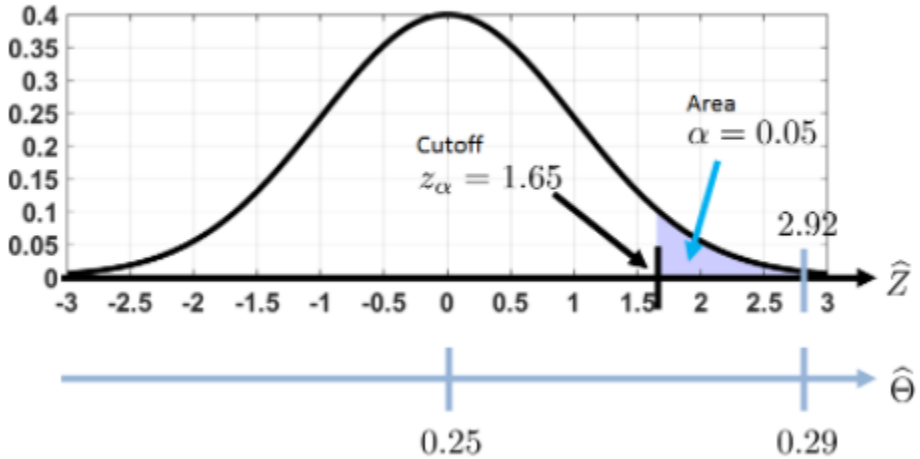
```
# Python code to compute the width of the confidence interval
import scipy.stats as stats
alph = 0.05;
mu   = 0; sigma = 1; # Standard Gaussian
epsilon = stats.norm.ppf(1-alph/2, mu, sigma)
print(epsilon)
```

Construct a confidence interval from data

```
# Python code to generate a confidence interval
import numpy as np
import scipy.stats as stats
x = np.array([72, 69, 75, 58, 67, 70, 60, 71, 59, 65])
N         = x.size
Theta_hat = np.mean(x) # Sample mean
S_hat     = np.std(x)  # Sample standard deviation
nu        = x.size-1   # degrees of freedom
alpha     = 0.05       # confidence level
z    = stats.t.ppf(1-alph/2, nu)
CI_L = Theta_hat-z*S_hat/np.sqrt(N)
CI_U = Theta_hat+z*S_hat/np.sqrt(N)
print(CI_L, CI_U)
```

ADDITIONS – HYPOTHESIS TESTING

Estimate Z-value

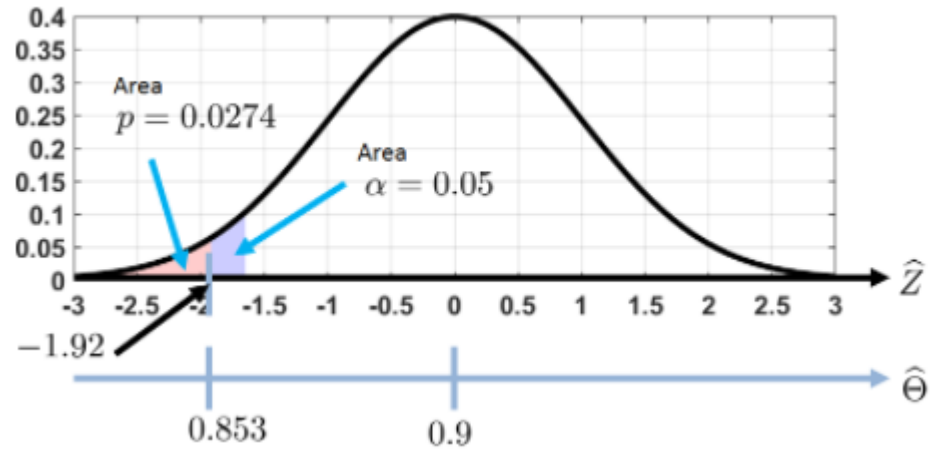



```
# Python command to estimate the Z_hat value
import numpy as np
Theta_hat = 0.29                    # Your estimate
theta     = 0.25                    # Your hypothesis
N         = 1000                    # Number of samples
sigma     = np.sqrt(theta*(1-theta)) # Known standard deviation
Z_hat = (Theta_hat - theta)/(sigma / np.sqrt(N))
print(Z_hat)
```

Compute critical value

```
# Python code to compute the critical value
import scipy.stats as stats
alpha = 0.05
z_alpha = stats.norm.ppf(1-alpha, 0, 1)
```

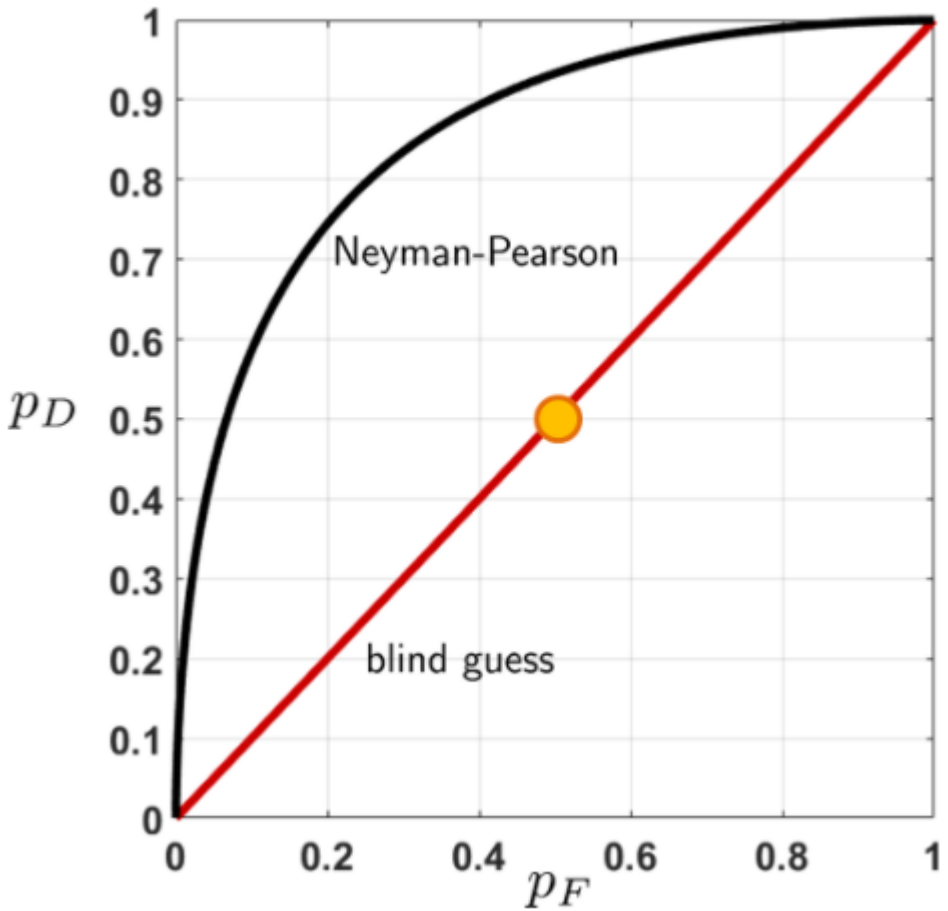
Compute p-value




```
# Python code to compute the p-value
import scipy.stats as stats
p = stats.norm.cdf(-1.92,0,1)
```

ADDITIONS – HYPOTHESIS TESTING

Plot an ROC curve



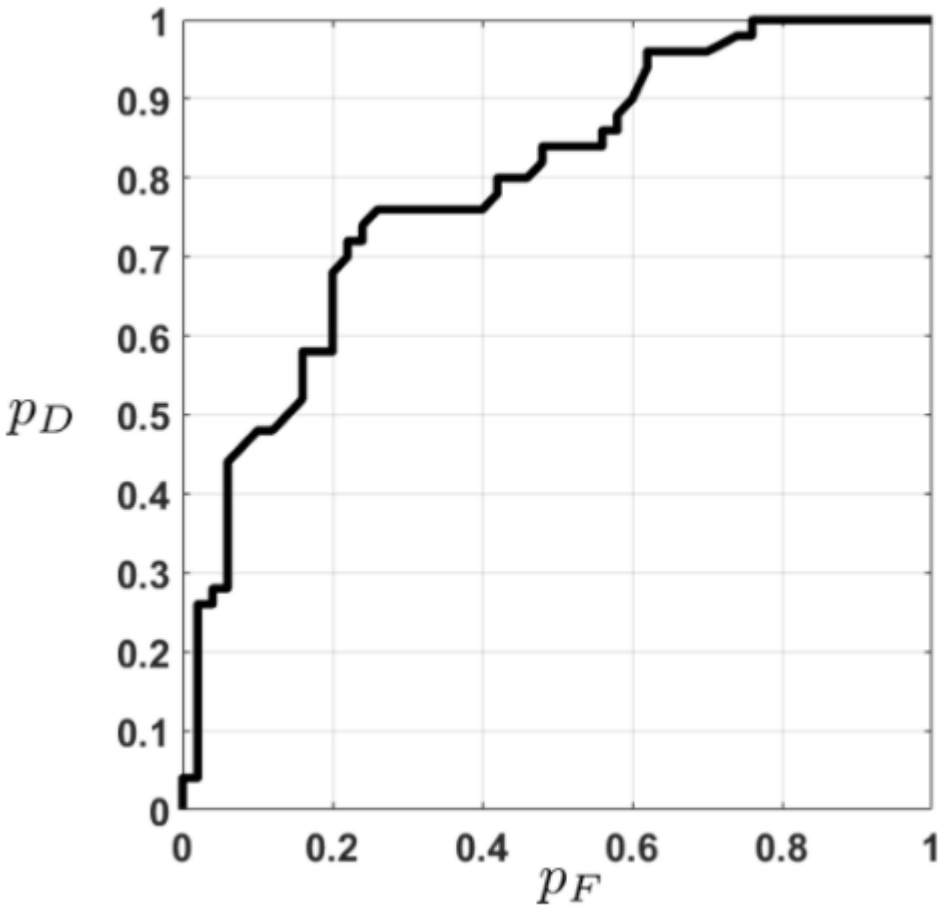

```
# Python code to plot ROC curve
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

sigma = 2; mu = 3;
alphaset = np.linspace(0,1,1000)
PF1 = np.zeros(1000); PD1 = np.zeros(1000)
PF2 = np.zeros(1000); PD2 = np.zeros(1000)
for i in range(1000):
  alpha = alphaset[i]
  PF1[i] = alpha
  PD1[i] = alpha
  PF2[i] = alpha
  PD2[i] = 1-stats.norm.cdf(stats.norm.ppf(1-alpha)-mu/sigma)
plt.plot(PF1,PD1)
plt.plot(PF2,PD2)
```

Computer area under curve

```
# Python
auc1 = np.sum(PD1 * np.append(0, np.diff(PF1)))
auc2 = np.sum(PD2 * np.append(0, np.diff(PF2)))
```

ADDITIONS – HYPOTHESIS TESTING

ROC on real data



Data download: ch9_ROC_example_data.txt (790KB)

```
# Python code to generate an empirical ROC curve
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
scores = np.loadtxt('ch9_ROC_example_data.txt')
labels = np.append(np.ones(50), np.zeros(50))
tau = np.linspace(0,1,1000)
PF = np.zeros(1000)
PD = np.zeros(1000)
for i in range(1000):
  idx = scores<= tau[i]
  predict = np.zeros(100)
  predict[idx] = 1
  true_positive  = 0;   true_negative  = 0
  false_positive = 0;   false_negative = 0
  for j in range(100):
    if (predict[j]==1) and (labels[j]==1): true_positive  += 1
    if (predict[j]==1) and (labels[j]==0): false_positive += 1
    if (predict[j]==0) and (labels[j]==1): false_negative += 1
    if (predict[j]==0) and (labels[j]==0): true_negative  += 1
  PF[i] = false_positive/50
  PD[i] = true_positive/50
plt.plot(PF, PD)
```

HOMEWORK – HYPOTHESIS TESTING

One sample t-test

Example :- you have 10 ages and you are checking whether avg age is 30 or not. (check code below for that using python)

```
from scipy.stats import ttest_1samp
import numpy as np


ages = np.genfromtxt("ages.csv")


print(ages)


ages_mean = np.mean(ages)
print(ages_mean)
tset, pval = ttest_1samp(ages, 30)


print("p-values",pval)


if pval < 0.05:    # alpha value is 0.05 or 5%
   print(" we are rejecting null hypothesis")
else:
  print("we are accepting null hypothesis")
```

HOMEWORK

Two sampled T-test

Example : is there any association between week1 and week2 (code is given below in python)

```
from scipy.stats import ttest_ind
import numpy as np


week1 = np.genfromtxt("week1.csv",  delimiter=",")
week2 = np.genfromtxt("week2.csv",  delimiter=",")


print(week1)
print("week2 data :-\n")
print(week2)
week1_mean = np.mean(week1)
week2_mean = np.mean(week2)


print("week1 mean value:",week1_mean)
print("week2 mean value:",week2_mean)


week1_std = np.std(week1)
week2_std = np.std(week2)


print("week1 std value:",week1_std)
print("week2 std value:",week2_std)


ttest,pval = ttest_ind(week1,week2)
print("p-value",pval)


if pval <0.05:
  print("we reject null hypothesis")
else:
  print("we accept null hypothesis")
```

HOMEWORK

Paired sampled t-test

H0 :- means difference between two sample is 0

H1:- mean difference between two sample is not 0

```
import pandas as pd
from scipy import stats
df = pd.read_csv("blood_pressure.csv")


df[['bp_before','bp_after']].describe()


ttest,pval = stats.ttest_rel(df['bp_before'], df['bp_after'])
print(pval)


if pval<0.05:
    print("reject null hypothesis")
else:
    print("accept null hypothesis")
```

HOMEWORK

one-sample Z test.

```
import pandas as pd
from scipy import stats
from statsmodels.stats import weightstats as stests


ztest ,pval = stests.ztest(df['bp_before'], x2=None, value=156)
print(float(pval))


if pval<0.05:
    print("reject null hypothesis")
else:
    print("accept null hypothesis")
```

HOMEWORK

Two-sample Z test

H0 : mean of two group is 0

H1 : mean of two group is not 0

Example : we are checking in blood data after blood and before blood data.(code in python below)

```
ztest ,pval1 = stests.ztest(df['bp_before'], x2=df['bp_after'],
value=0,alternative='two-sided')
print(float(pval1))


if pval<0.05:
    print("reject null hypothesis")
else:
    print("accept null hypothesis")
```

HOMEWORK

One Way F-test(Anova)

Example : there are 3 different category of plant and their weight and need to check whether all 3 group are similar or not (code in python below)

```
df_anova = pd.read_csv('PlantGrowth.csv')
df_anova = df_anova[['weight','group']]

grps = pd.unique(df_anova.group.values)
d_data = {grp:df_anova['weight'][df_anova.group == grp] for grp in
grps}

F, p = stats.f_oneway(d_data['ctrl'], d_data['trt1'], d_data['trt2'])


print("p-value for significance is: ", p)


if p<0.05:
    print("reject null hypothesis")
else:
    print("accept null hypothesis")
```

HOMEWORK

Two Way F-test

```
import statsmodels.api as sm
from statsmodels.formula.api import ols


df_anova2 =
pd.read_csv("https://raw.githubusercontent.com/Opensourcefordatascien
ce/Data-sets/master/crop_yield.csv")


model = ols('Yield ~ C(Fert)*C(Water)', df_anova2).fit()
print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f})
= {model.fvalue: .3f}, p = {model.f_pvalue: .4f}")


res = sm.stats.anova_lm(model, typ= 2)
res
```

HOMEWORK

Multiple regression

Year	Month	Interest_Rate	Unemployment_Rate	Stock_Index_Price
2017	12	2.75	5.3	1464
2017	11	2.5	5.3	1394
2017	10	2.5	5.3	1357
2017	9	2.5	5.3	1293
2017	8	2.5	5.4	1256
2017	7	2.5	5.6	1254
2017	6	2.5	5.5	1234
2017	5	2.25	5.5	1195
2017	4	2.25	5.5	1159
2017	3	2.25	5.6	1167
2017	2	2	5.7	1130
2017	1	2	5.9	1075
2016	12	2	6	1047
2016	11	1.75	5.9	965
2016	10	1.75	5.8	943
2016	9	1.75	6.1	958
2016	8	1.75	6.2	971
2016	7	1.75	6.1	949
2016	6	1.75	6.1	884
2016	5	1.75	6.1	866
2016	4	1.75	5.9	876
2016	3	1.75	6.2	822
2016	2	1.75	6.2	704
2016	1	1.75	6.1	719

HOMEWORK

Multiple regression

```
import pandas as pd

Stock_Market = {'Year': [2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016],
                'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1],
                'Interest_Rate': [2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75],
                'Unemployment_Rate': [5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1,6.1,5.9,6.2,6.2,6.1],
                'Stock_Index_Price': [1464,1394,1357,1293,1256,1254,1234,1195,1159,1167,1130,1075,1047,965,943,958,971,949,884,866,876,822,704,719]
                }

df = pd.DataFrame(Stock_Market,columns=['Year','Month','Interest_Rate','Unemployment_Rate','Stock_Index_Price'])

print (df)
```

Checking for Linearity

In our example, you may want to check that a linear relationship exists between the:

- Stock_Index_Price (dependent variable) and Interest_Rate (independent variable)
- Stock_Index_Price (dependent variable) and Unemployment_Rate (independent variable)

HOMEWORK

Multiple regression

Checking for Linearity

```
import pandas as pd
import matplotlib.pyplot as plt

Stock_Market = {'Year': [2017,2017,2017,2017,2017,2017,2017,2017,2017
                'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1
                'Interest_Rate': [2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,
                'Unemployment_Rate': [5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,
                'Stock_Index_Price': [1464,1394,1357,1293,1256,1254,1234,1
                }

df = pd.DataFrame(Stock_Market,columns=['Year','Month','Interest_Rate'

plt.scatter(df['Interest_Rate'], df['Stock_Index_Price'], color='red')
plt.title('Stock Index Price Vs Interest Rate', fontsize=14)
plt.xlabel('Interest Rate', fontsize=14)
plt.ylabel('Stock Index Price', fontsize=14)
plt.grid(True)
plt.show()
```

HOMEWORK

Multiple regression

And for the second case, you can use this code in order to plot the relationship between the Stock_Index_Price and the Unemployment_Rate: Scatter diagram

```
import pandas as pd
import matplotlib.pyplot as plt

Stock_Market = {'Year': [2017,2017,2017,2017,2017,2017,2017,2017,2017,2017
                'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1],
                'Interest_Rate': [2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.
                'Unemployment_Rate': [5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5
                'Stock_Index_Price': [1464,1394,1357,1293,1256,1254,1234,1195,1
                }

df = pd.DataFrame(Stock_Market,columns=['Year','Month','Interest_Rate','Uner

plt.scatter(df['Unemployment_Rate'], df['Stock_Index_Price'], color='green')
plt.title('Stock Index Price Vs Unemployment Rate', fontsize=14)
plt.xlabel('Unemployment Rate', fontsize=14)
plt.ylabel('Stock Index Price', fontsize=14)
plt.grid(True)
plt.show()
```

HOMEWORK

Multiple regression

```
import pandas as pd
from sklearn import linear_model
import statsmodels.api as sm

Stock_Market = {'Year': [2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2016,2016,201
                'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1],
                'Interest_Rate': [2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75,1.75,1.75,1.7
                'Unemployment_Rate': [5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1
                'Stock_Index_Price': [1464,1394,1357,1293,1256,1254,1234,1195,1159,1167,1130,1075,1047,
                }

df = pd.DataFrame(Stock_Market,columns=['Year','Month','Interest_Rate','Unemployment_Rate','Stock_Ir

X = df[['Interest_Rate','Unemployment_Rate']] # here we have 2 variables for multiple regression. If you
Y = df['Stock_Index_Price']

# with sklearn
regr = linear_model.LinearRegression()
regr.fit(X, Y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

# prediction with sklearn
New_Interest_Rate = 2.75
New_Unemployment_Rate = 5.3
print ('Predicted Stock Index Price: \n', regr.predict([[New_Interest_Rate ,New_Unemployment_Rate]]))

# with statsmodels
X = sm.add_constant(X) # adding a constant

model = sm.OLS(Y, X).fit()
predictions = model.predict(X)

print_model = model.summary()
print(print_model)
```

THE END