ITMAL

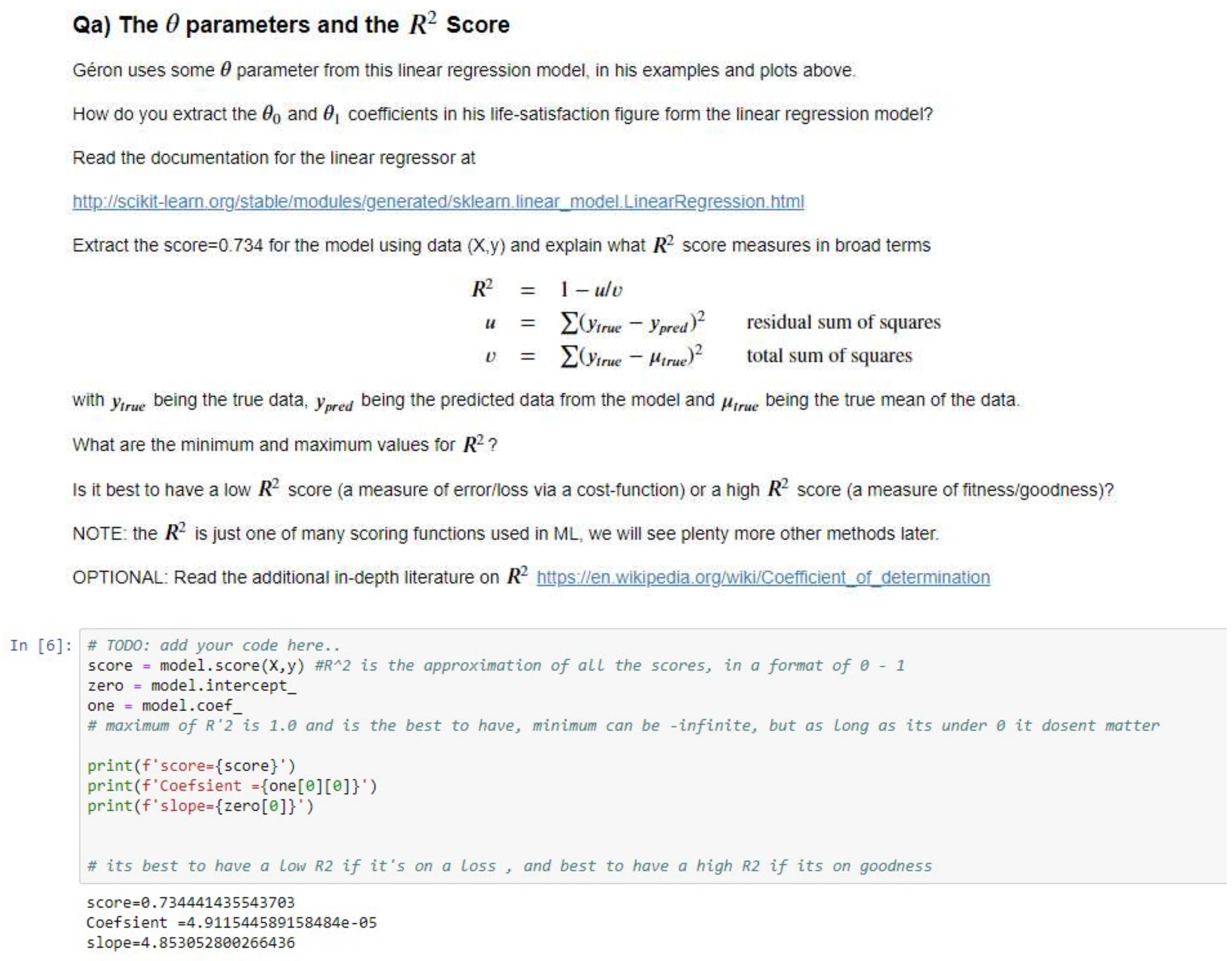
Hand-in: Journal 1

Af Gruppe 12

Indhold

Leksion 1 - intro.ipynb, Qa+b+c.:

Qa:



Qa) The θ parameters and the R^2 Score

Géron uses some θ parameter from this linear regression model, in his examples and plots above.

How do you extract the θ_0 and θ_1 coefficients in his life-satisfaction figure form the linear regression model?

Read the documentation for the linear regressor at

http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Extract the score=0.734 for the model using data (X,y) and explain what R^2 score measures in broad terms

$$\begin{aligned} R^2 &= 1 - u/v \\ u &= \sum(y_{true} - y_{pred})^2 \quad \text{residual sum of squares} \\ v &= \sum(y_{true} - \mu_{true})^2 \quad \text{total sum of squares} \end{aligned}$$

with y_{true} being the true data, y_{pred} being the predicted data from the model and μ_{true} being the true mean of the data.

What are the minimum and maximum values for R^2 ?

Is it best to have a low R^2 score (a measure of error/loss via a cost-function) or a high R^2 score (a measure of fitness/goodness)?

NOTE: the R^2 is just one of many scoring functions used in ML, we will see plenty more other methods later.

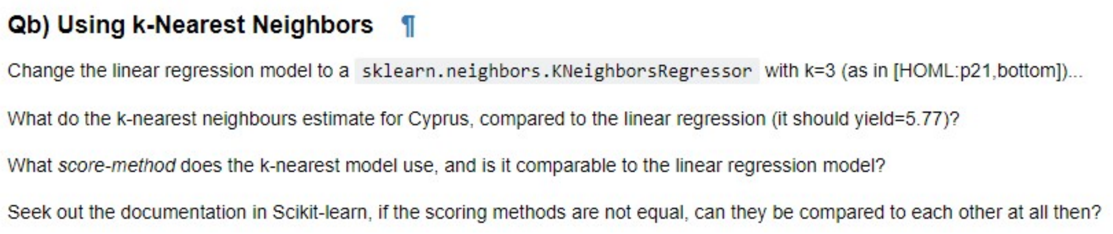
OPTIONAL: Read the additional in-depth literature on R^2 https://en.wikipedia.org/wiki/Coefficient_of_determination

In [6]:
```python
# TODO: add your code here..
score = model.score(X,y) #R^2 is the approximation of all the scores, in a format of 0 - 1
zero = model.intercept_
one = model.coef_
# maximum of R'2 is 1.0 and is the best to have, minimum can be -infinite, but as long as its under 0 it dosent matter

print(f'score={score}')
print(f'Coefsient ={one[0][0]}')
print(f'slope={zero[0]}')


# its best to have a low R2 if it's on a loss , and best to have a high R2 if its on goodness
```

```
score=0.734441435543703
Coefsient =4.911544589158484e-05
slope=4.853052800266436
```

Forklaring: Maximum of R'2 (score) is 1.0 and is the best to have, minimum can be -infinite, but as long as its under 0 it dosent matter. Its best to have a low R2 if it's on a loss , and best to have a high R2 if its on goodness

Qb:



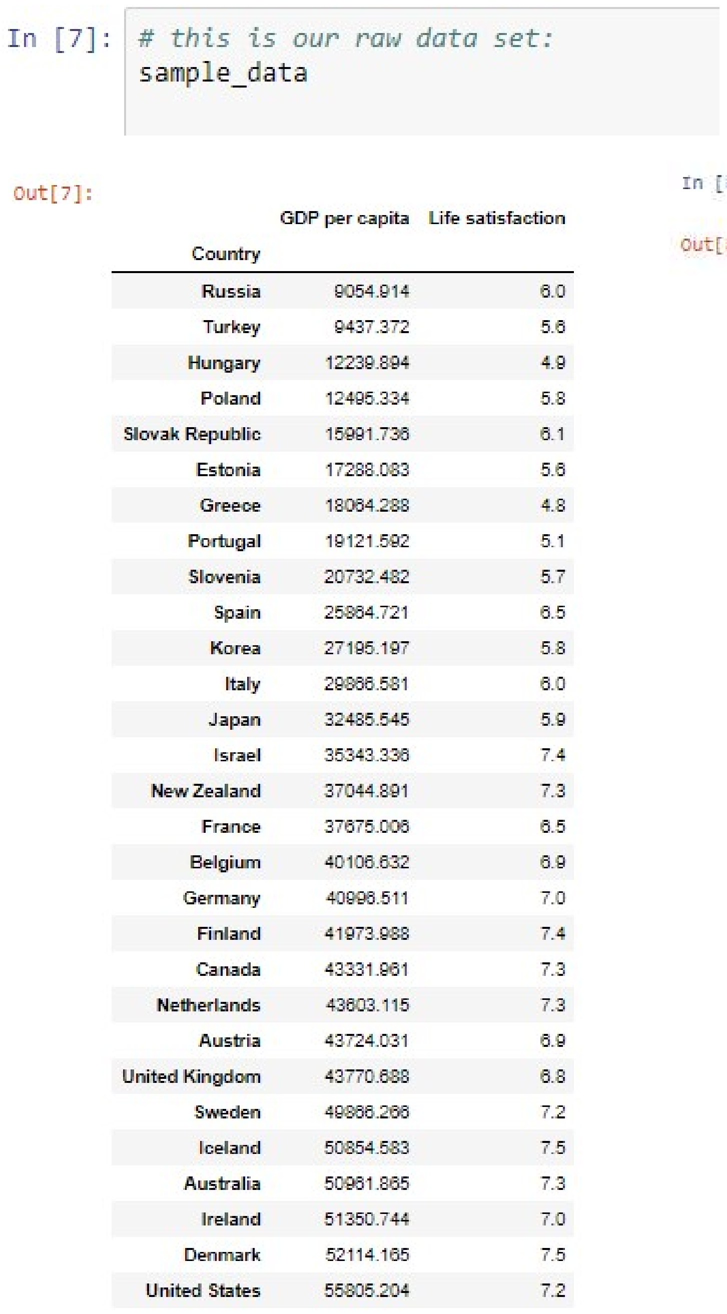
Qb) Using k-Nearest Neighbors ¶

Change the linear regression model to a `sklearn.neighbors.KNeighborsRegressor` with k=3 (as in [HOML:p21,bottom])...

What do the k-nearest neighbours estimate for Cyprus, compared to the linear regression (it should yield=5.77)?

What *score-method* does the k-nearest model use, and is it comparable to the linear regression model?

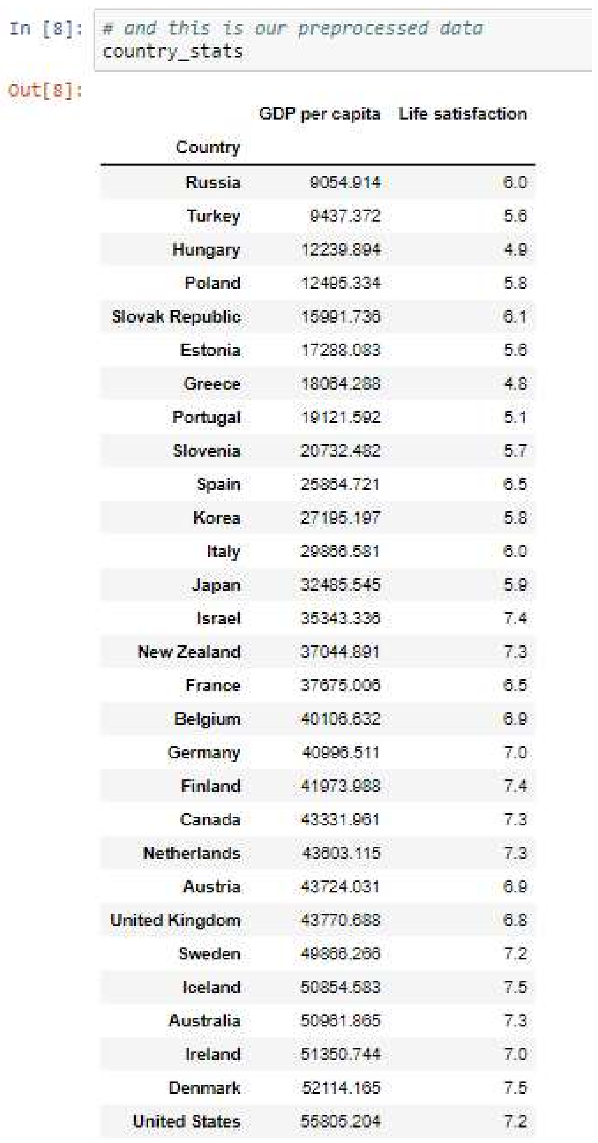
Seek out the documentation in Scikit-learn, if the scoring methods are not equal, can they be compared to each other at all then?



```
In [7]: # this is our raw data set:
        sample_data
```

Out[7]:

Country	GDP per capita	Life satisfaction
Russia	9054.914	6.0
Turkey	9437.372	5.6
Hungary	12239.894	4.9
Poland	12495.334	5.8
Slovak Republic	15991.736	6.1
Estonia	17288.083	5.6
Greece	18064.288	4.8
Portugal	19121.592	5.1
Slovenia	20732.482	5.7
Spain	25864.721	6.5
Korea	27195.197	5.8
Italy	29866.581	6.0
Japan	32485.545	5.9
Israel	35343.336	7.4
New Zealand	37044.891	7.3
France	37675.006	6.5
Belgium	40106.632	6.9
Germany	40996.511	7.0
Finland	41973.988	7.4
Canada	43331.961	7.3
Netherlands	43603.115	7.3
Austria	43724.031	6.9
United Kingdom	43770.688	6.8
Sweden	49866.266	7.2
Iceland	50854.583	7.5
Australia	50961.865	7.3
Ireland	51350.744	7.0
Denmark	52114.165	7.5
United States	55805.204	7.2



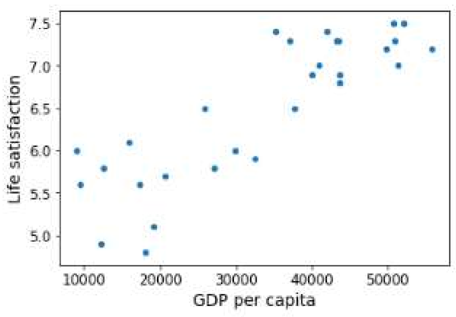
```
In [8]: # and this is our preprocessed data
        country_stats
```

Out[8]:

Country	GDP per capita	Life satisfaction
Russia	9054.914	6.0
Turkey	9437.372	5.6
Hungary	12239.894	4.9
Poland	12495.334	5.8
Slovak Republic	15991.736	6.1
Estonia	17288.083	5.6
Greece	18064.288	4.8
Portugal	19121.592	5.1
Slovenia	20732.482	5.7
Spain	25864.721	6.5
Korea	27195.197	5.8
Italy	29866.581	6.0
Japan	32485.545	5.9
Israel	35343.336	7.4
New Zealand	37044.891	7.3
France	37675.006	6.5
Belgium	40106.632	6.9
Germany	40996.511	7.0
Finland	41973.988	7.4
Canada	43331.961	7.3
Netherlands	43603.115	7.3
Austria	43724.031	6.9
United Kingdom	43770.688	6.8
Sweden	49866.266	7.2
Iceland	50854.583	7.5
Australia	50961.865	7.3
Ireland	51350.744	7.0
Denmark	52114.165	7.5
United States	55805.204	7.2

```
In [9]: # Prepare the data
        X = np.c_[country_stats["GDP per capita"]]
        y = np.c_[country_stats["Life satisfaction"]]

        print("X.shape=",X.shape)
        print("y.shape=",y.shape)

        # Visualize the data
        country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
        plt.show()

        # Select and train a model (linear regression or k-nearest neighbours)

        # TODO: add code here...

        model2 = sklearn.neighbors.KNeighborsRegressor(3)
        model2.fit(X,y)
        score = model2.score(X,y)
        predict = model2.predict(X_new)

        print(f'score={score}')
        print(f'predict for Cyprus={predict[0][0]}')

        # as we can see, with kNeighborsRegressor, the score increases by about 12%, so this must means it's more accurate
        # kNeighborsRegressor looks at the nearest to it self as specified in with k, by this, its still linear in its own local space.
        # so yes, it can be compared, however, it adapts more to what is immediatly near the data being run
```

```
X.shape= (29, 1)
y.shape= (29, 1)
```



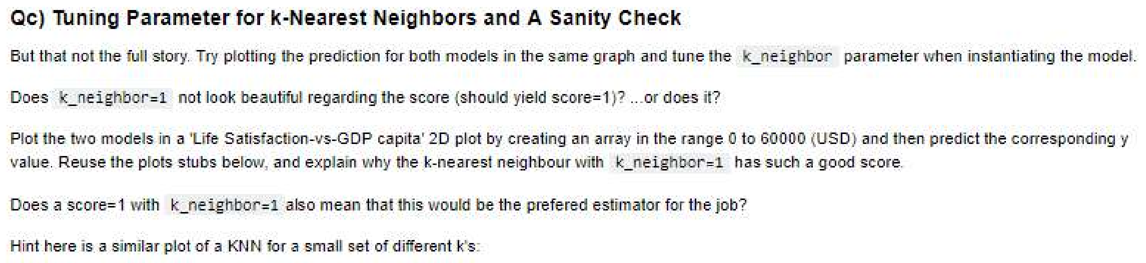
```
score=0.8525732853499179
predict for Cyprus=5.766666666666666
```

Forklaring: As we can see, with kNeighborsRegressor, the score increases by about 12%, so this must means it's more accurate

kNeighborsRegressor looks at the nearest to it self as specified in with k, by this, its still linear in its own local space.

So yes, it can be compared, however, it adapts more to what is immediatly near the data being run

Qc:



Qc) Tuning Parameter for k-Nearest Neighbors and A Sanity Check

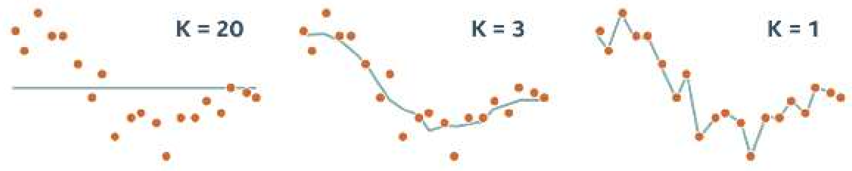
But that not the full story. Try plotting the prediction for both models in the same graph and tune the `k_neighbor` parameter when instantiating the model.

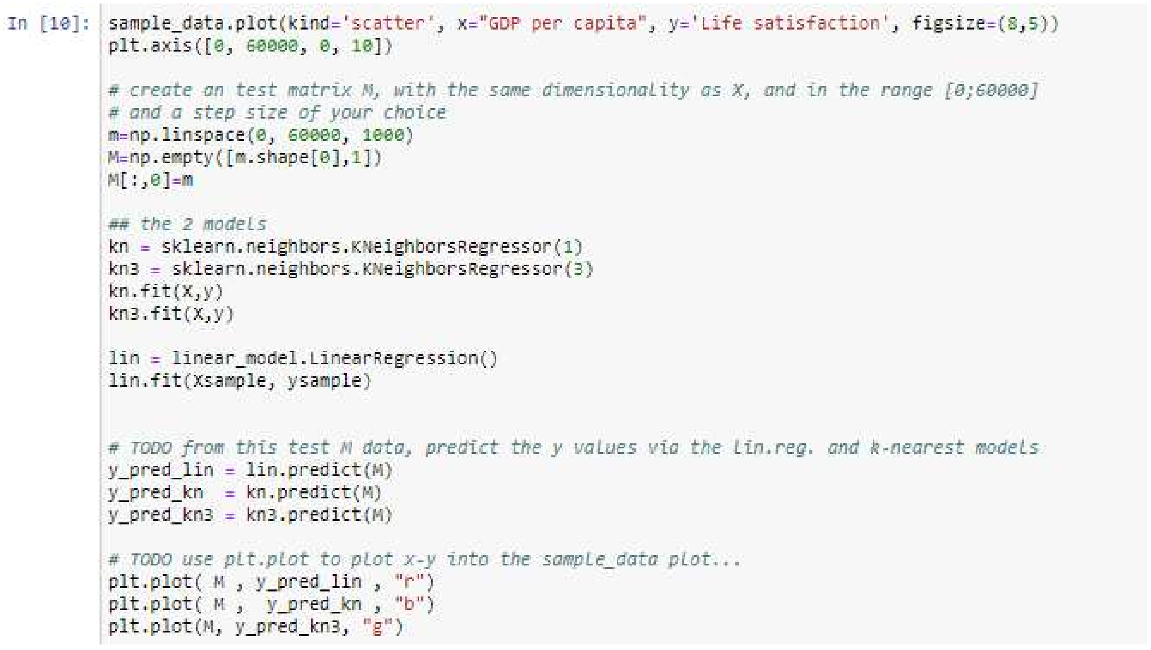
Does `k_neighbor=1` not look beautiful regarding the score (should yield score=1)? ...or does it?

Plot the two models in a 'Life Satisfaction-vs-GDP capita' 2D plot by creating an array in the range 0 to 60000 (USD) and then predict the corresponding y value. Reuse the plots stubs below, and explain why the k-nearest neighbour with `k_neighbor=1` has such a good score.

Does a score=1 with `k_neighbor=1` also mean that this would be the prefered estimator for the job?

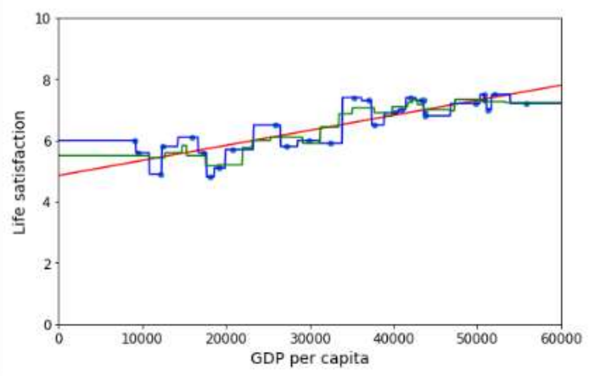
Hint here is a similar plot of a KNN for a small set of different k's:





```
In [10]: sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(8,5))
         plt.axis([0, 60000, 0, 10])

         # create an test matrix M, with the same dimensionality as X, and in the range [0;60000]
         # and a step size of your choice
         m=np.linspace(0, 60000, 1000)
         M=np.empty([m.shape[0],1])
         M[:,0]=m

         ## the 2 models
         kn = sklearn.neighbors.KNeighborsRegressor(1)
         kn3 = sklearn.neighbors.KNeighborsRegressor(3)
         kn.fit(X,y)
         kn3.fit(X,y)

         lin = linear_model.LinearRegression()
         lin.fit(Xsample, ysample)


         # TODO from this test M data, predict the y values via the lin.reg. and k-nearest models
         y_pred_lin = lin.predict(M)
         y_pred_kn  = kn.predict(M)
         y_pred_kn3 = kn3.predict(M)

         # TODO use plt.plot to plot x-y into the sample_data plot...
         plt.plot( M , y_pred_lin , "r")
         plt.plot( M ,  y_pred_kn , "b")
         plt.plot(M, y_pred_kn3, "g")
```

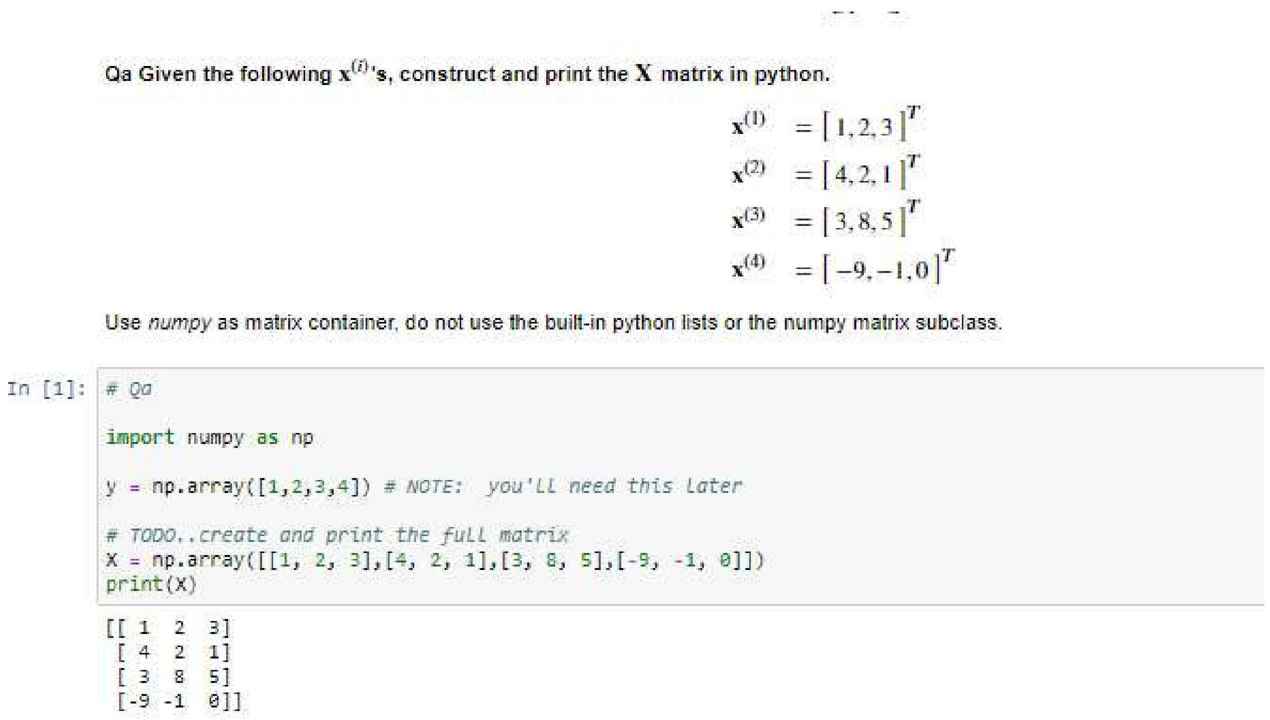
```
Out[10]: [<matplotlib.lines.Line2D at 0x14362e158d0>]
```



Forklaring: In our dataset, the knearst with 1 neighbor, we get at good prediction, seen in the plot with as the blue line. It has more straight line which isn´t a good tool to tell us of data point not in the dataset though. with 3 neightbors we have a better average, but misses a few actual data points.

Leksion 2 - math.ipynb, Qa+b+c+d+e :

Qa:



Qa Given the following $\mathbf{x}^{(i)}$'s, construct and print the $\mathbf{X}$ matrix in python.

$$\begin{aligned}
\mathbf{x}^{(1)} &= [1,2,3]^T \\
\mathbf{x}^{(2)} &= [4,2,1]^T \\
\mathbf{x}^{(3)} &= [3,8,5]^T \\
\mathbf{x}^{(4)} &= [-9,-1,0]^T
\end{aligned}$$

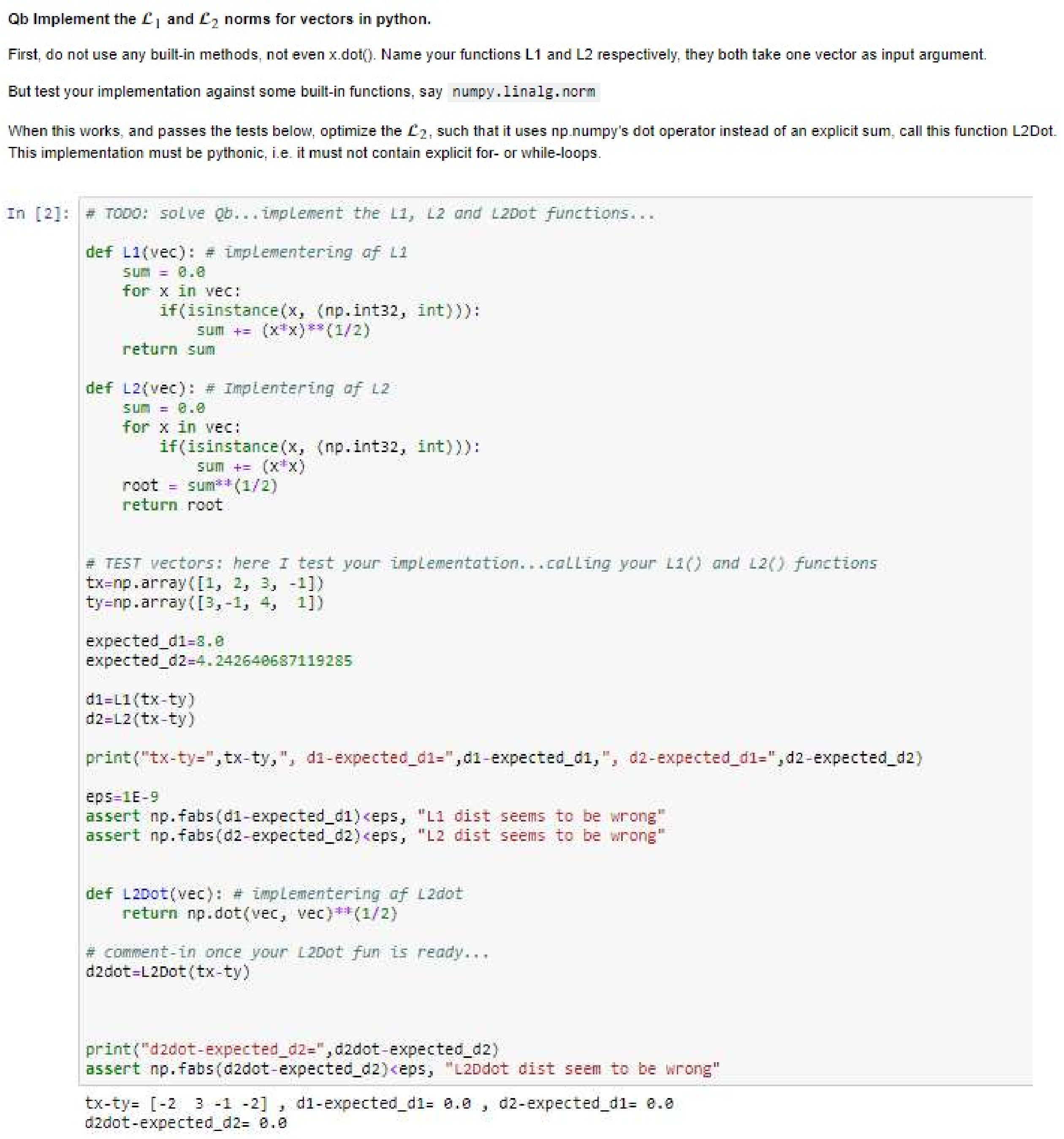
Use *numpy* as matrix container, do not use the built-in python lists or the numpy matrix subclass.

```
In [1]: # Qa

        import numpy as np

        y = np.array([1,2,3,4]) # NOTE:  you'll need this later

        # TODO..create and print the full matrix
        X = np.array([[1, 2, 3],[4, 2, 1],[3, 8, 5],[-9, -1, 0]])
        print(X)

[[ 1  2  3]
 [ 4  2  1]
 [ 3  8  5]
 [-9 -1  0]]
```

Forklaring: for x(1) til x(4) er arrayet X opsat ovenfor, ihenhold til opgavens beskrivelser.

Som vi kan se i udprintet overfor, er matricen opsat og udprintet på matrix form.

Qb:



Qb Implement the $\mathcal{L}_1$ and $\mathcal{L}_2$ norms for vectors in python.

First, do not use any built-in methods, not even x.dot(). Name your functions **L1** and L2 respectively, they both take one vector as input argument.

But test your implementation against some built-in functions, say `numpy.linalg.norm`

When this works, and passes the tests below, optimize the $\mathcal{L}_2$, such that it uses np.numpy's dot operator instead of an explicit sum, call this function L2Dot. This implementation must be pythonic, i.e. it must not contain explicit for- or while-loops.

```
In [2]: # TODO: solve Qb...implement the L1, L2 and L2Dot functions...

def L1(vec): # implementering af L1
    sum = 0.0
    for x in vec:
        if(isinstance(x, (np.int32, int))):
            sum += (x*x)**(1/2)
    return sum

def L2(vec): # Implentering af L2
    sum = 0.0
    for x in vec:
        if(isinstance(x, (np.int32, int))):
            sum += (x*x)
    root = sum**(1/2)
    return root


# TEST vectors: here I test your implementation...calling your L1() and L2() functions
tx=np.array([1, 2, 3, -1])
ty=np.array([3,-1, 4,  1])

expected_d1=8.0
expected_d2=4.242640687119285

d1=L1(tx-ty)
d2=L2(tx-ty)

print("tx-ty=",tx-ty,", d1-expected_d1=",d1-expected_d1,", d2-expected_d1=",d2-expected_d2)

eps=1E-9
assert np.fabs(d1-expected_d1)<eps, "L1 dist seems to be wrong"
assert np.fabs(d2-expected_d2)<eps, "L2 dist seems to be wrong"


def L2Dot(vec): # implementering af L2dot
    return np.dot(vec, vec)**(1/2)

# comment-in once your L2Dot fun is ready...
d2dot=L2Dot(tx-ty)



print("d2dot-expected_d2=",d2dot-expected_d2)
assert np.fabs(d2dot-expected_d2)<eps, "L2Ddot dist seem to be wrong"
```

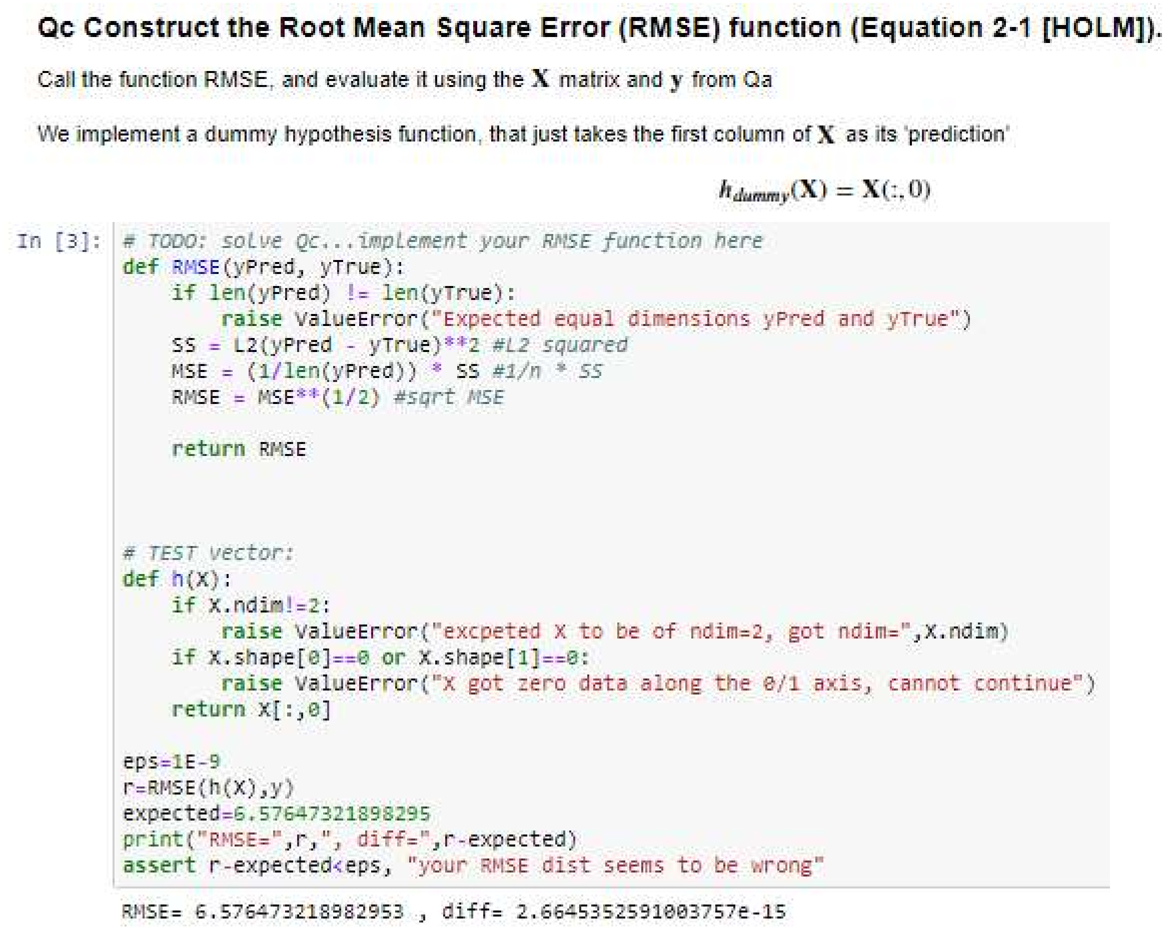
```
tx-ty= [-2  3 -1 -2] , d1-expected_d1= 0.0 , d2-expected_d1= 0.0
d2dot-expected_d2= 0.0
```

Forklaring: L1 er implementeret således at alle tal i arrayet, vendes til positive tal ved at opløfte tallet i anden potens, og derefter tage kvadrat roden af resultatet, for at få det tilbage til dets originale værdi i postiv form, hvorefter det summeres med samtlige x værdier.

L2 Gør det samme som L1 , udover tallet SKAL være i anden potens, og efter endt for lykke, skal der tages kvadrat roden af den samlede sum.

L2Dot Gør det samme som vores L2 funktion, men her bruges blot numpy's biblioteke

Qc:



Qc Construct the Root Mean Square Error (RMSE) function (Equation 2-1 [HOLM]).

Call the function RMSE, and evaluate it using the $\mathbf{X}$ matrix and $\mathbf{y}$ from Qa

We implement a dummy hypothesis function, that just takes the first column of $\mathbf{X}$ as its 'prediction'

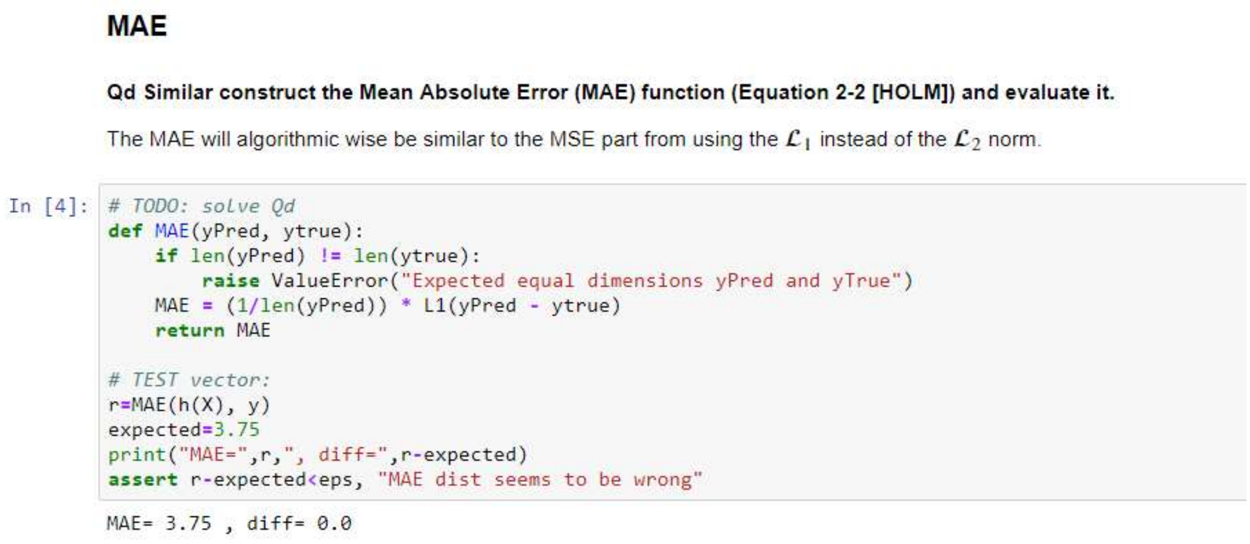
$$h_{dummy}(\mathbf{X}) = \mathbf{X}(:,0)$$

In [3]:
```python
# TODO: solve Qc...implement your RMSE function here
def RMSE(yPred, yTrue):
    if len(yPred) != len(yTrue):
        raise ValueError("Expected equal dimensions yPred and yTrue")
    SS = L2(yPred - yTrue)**2 #L2 squared
    MSE = (1/len(yPred)) * SS #1/n * SS
    RMSE = MSE**(1/2) #sqrt MSE

    return RMSE



# TEST vector:
def h(X):
    if X.ndim!=2:
        raise ValueError("excpeted X to be of ndim=2, got ndim=",X.ndim)
    if X.shape[0]==0 or X.shape[1]==0:
        raise ValueError("X got zero data along the 0/1 axis, cannot continue")
    return X[:,0]

eps=1E-9
r=RMSE(h(X),y)
expected=6.57647321898295
print("RMSE=",r,", diff=",r-expected)
assert r-expected<eps, "your RMSE dist seems to be wrong"
```

```
RMSE= 6.576473218982953 , diff= 2.6645352591003757e-15
```

Forklaring: RMSE er blevet opdelt så det er indlysende hvilke steps i functionen der gør hvad Vi raiser en error hvis ikke længden er ens på de 2, da dette vil konflicte

Qd:



MAE

Qd Similar construct the Mean Absolute Error (MAE) function (Equation 2-2 [HOLM]) and evaluate it.

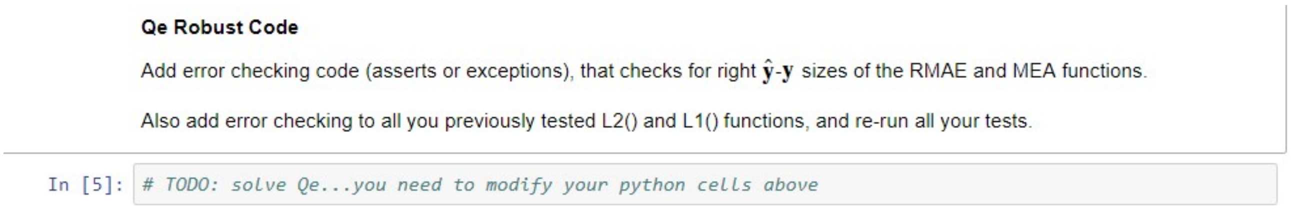
The MAE will algorithmic wise be similar to the MSE part from using the $\mathcal{L}_1$ instead of the $\mathcal{L}_2$ norm.

In [4]:
```
# TODO: solve Qd
def MAE(yPred, ytrue):
    if len(yPred) != len(ytrue):
        raise ValueError("Expected equal dimensions yPred and yTrue")
    MAE = (1/len(yPred)) * L1(yPred - ytrue)
    return MAE

# TEST vector:
r=MAE(h(X), y)
expected=3.75
print("MAE=",r,", diff=",r-expected)
assert r-expected<eps, "MAE dist seems to be wrong"
```

```
MAE= 3.75 , diff= 0.0
```

Forklaring: For at udregne MAE bruges som sagt L1 istedet for L2, udover dette er den angivede formel fulgt.

Qe:



Qe Robust Code

Add error checking code (asserts or exceptions), that checks for right $\hat{\mathbf{y}}$-$\mathbf{y}$ sizes of the RMAE and MEA functions.

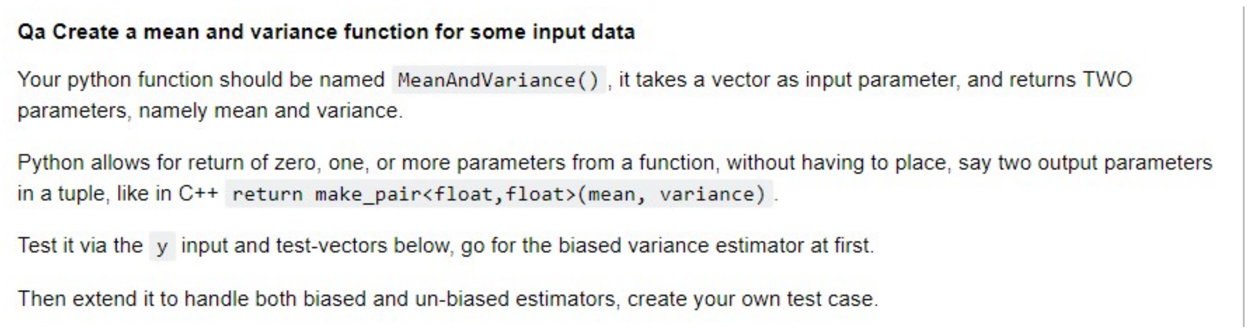
Also add error checking to all you previously tested L2() and L1() functions, and re-run all your tests.

In [5]:
```
# TODO: solve Qe...you need to modify your python cells above
```

Forklaring: Tilføjet error checks i alle opgaver…

Leksion 2 - statistics.ipynb, Qa+b:

Qa:



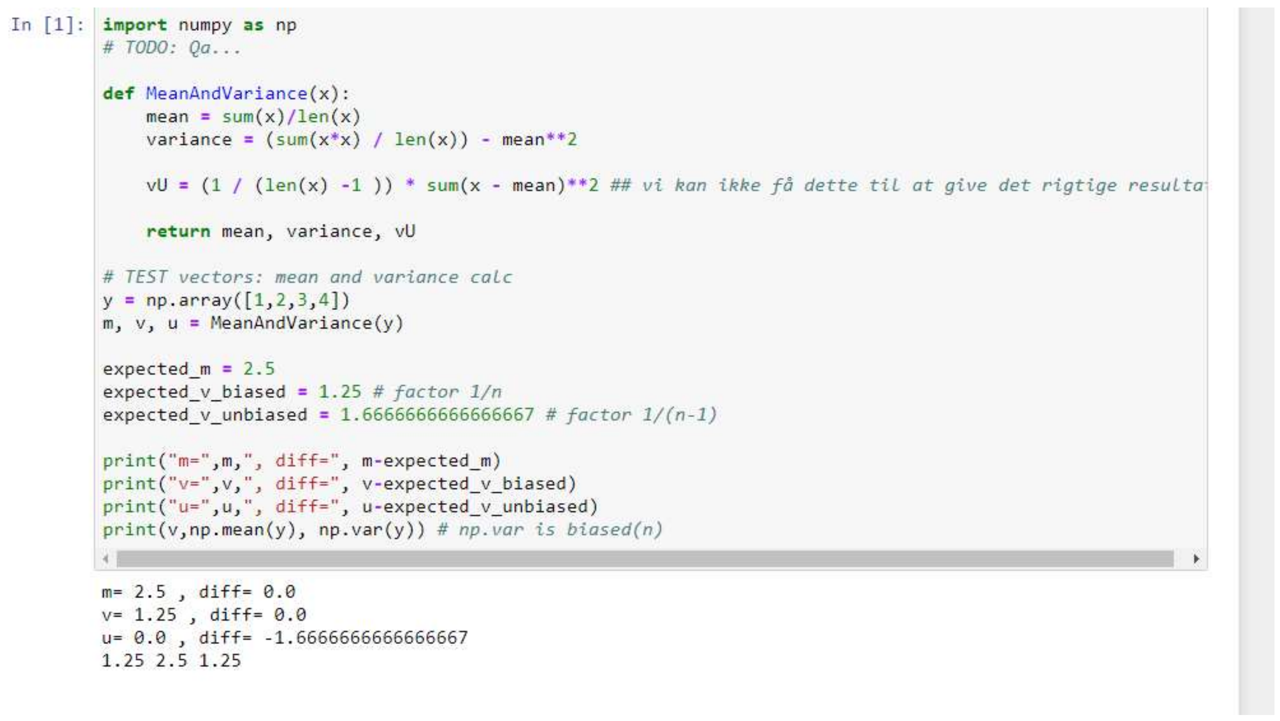
Qa Create a mean and variance function for some input data

Your python function should be named `MeanAndVariance()`, it takes a vector as input parameter, and returns TWO parameters, namely mean and variance.

Python allows for return of zero, one, or more parameters from a function, without having to place, say two output parameters in a tuple, like in C++ `return make_pair<float,float>(mean, variance)`.

Test it via the `y` input and test-vectors below, go for the biased variance estimator at first.

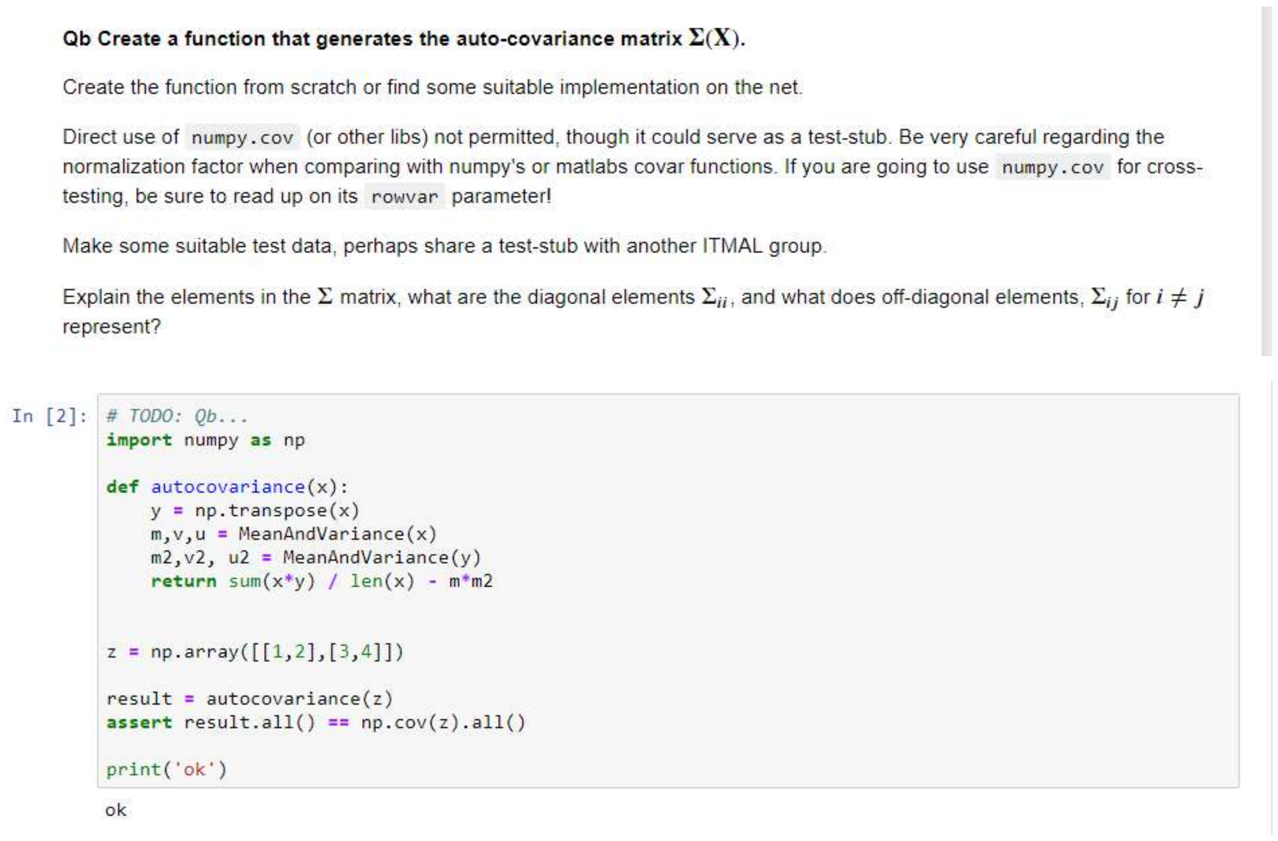
Then extend it to handle both biased and un-biased estimators, create your own test case.



```
In [1]: import numpy as np
        # TODO: Qa...

        def MeanAndVariance(x):
            mean = sum(x)/len(x)
            variance = (sum(x*x) / len(x)) - mean**2

            vU = (1 / (len(x) -1 )) * sum(x - mean)**2 ## vi kan ikke få dette til at give det rigtige resulta

            return mean, variance, vU

        # TEST vectors: mean and variance calc
        y = np.array([1,2,3,4])
        m, v, u = MeanAndVariance(y)

        expected_m = 2.5
        expected_v_biased = 1.25 # factor 1/n
        expected_v_unbiased = 1.6666666666666667 # factor 1/(n-1)

        print("m=",m,", diff=", m-expected_m)
        print("v=",v,", diff=", v-expected_v_biased)
        print("u=",u,", diff=", u-expected_v_unbiased)
        print(v,np.mean(y), np.var(y)) # np.var is biased(n)
```

```
m= 2.5 , diff= 0.0
v= 1.25 , diff= 0.0
u= 0.0 , diff= -1.6666666666666667
1.25 2.5 1.25
```

Qb:



Qb Create a function that generates the auto-covariance matrix $\Sigma(\mathbf{X})$.

Create the function from scratch or find some suitable implementation on the net.

Direct use of `numpy.cov` (or other libs) not permitted, though it could serve as a test-stub. Be very careful regarding the normalization factor when comparing with numpy's or matlabs covar functions. If you are going to use `numpy.cov` for cross-testing, be sure to read up on its `rowvar` parameter!

Make some suitable test data, perhaps share a test-stub with another ITMAL group.

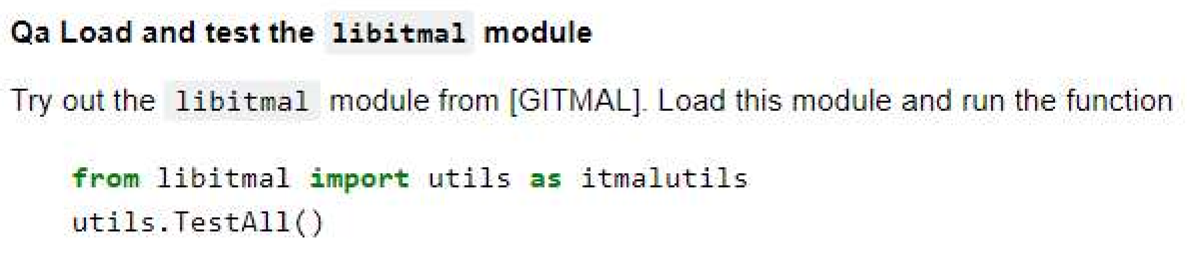
Explain the elements in the Σ matrix, what are the diagonal elements Σ_{ii}, and what does off-diagonal elements, Σ_{ij} for $i \neq j$ represent?

In [2]:
```
# TODO: Qb...
import numpy as np

def autocovariance(x):
    y = np.transpose(x)
    m,v,u = MeanAndVariance(x)
    m2,v2, u2 = MeanAndVariance(y)
    return sum(x*y) / len(x) - m*m2


z = np.array([[1,2],[3,4]])

result = autocovariance(z)
assert result.all() == np.cov(z).all()

print('ok')
```

```
ok
```

Forklaring: Vi har assertet ved fejl, derfor skulle det være ok hvis der kun vises ok

Leksion 3 - modules_and_classes.ipynb, Qa+b+c+e+f+g :

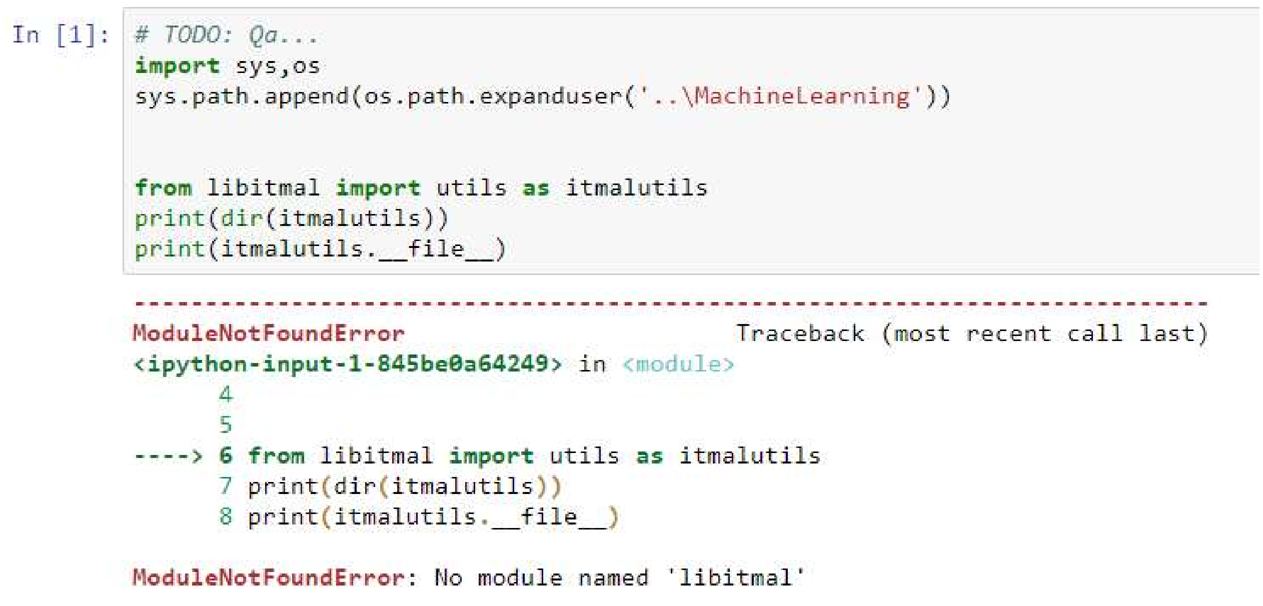


Qa Load and test the `libitmal` module

Try out the `libitmal` module from [GITMAL]. Load this module and run the function

```
from libitmal import utils as itmalutils
utils.TestAll()
```

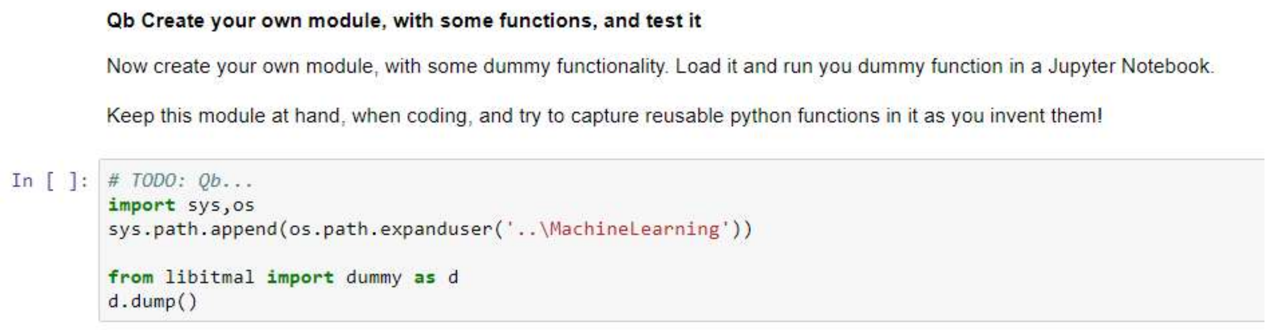
Qa:



```
In [1]: # TODO: Qa...
        import sys,os
        sys.path.append(os.path.expanduser('..\MachineLearning'))


        from libitmal import utils as itmalutils
        print(dir(itmalutils))
        print(itmalutils.__file__)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-845be0a64249> in <module>
      4
      5
----> 6 from libitmal import utils as itmalutils
      7 print(dir(itmalutils))
      8 print(itmalutils.__file__)

ModuleNotFoundError: No module named 'libitmal'
```

Qb:



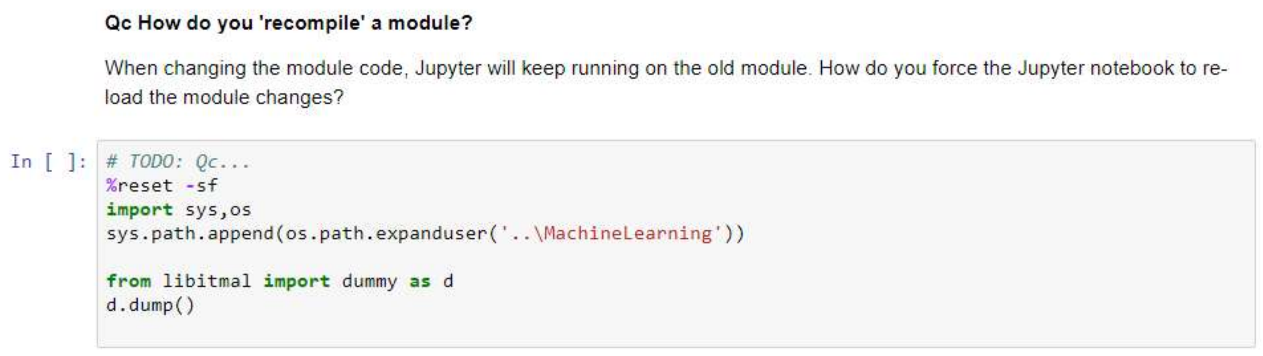
Qb Create your own module, with some functions, and test it

Now create your own module, with some dummy functionality. Load it and run you dummy function in a Jupyter Notebook.

Keep this module at hand, when coding, and try to capture reusable python functions in it as you invent them!

```
In [ ]: # TODO: Qb...
        import sys,os
        sys.path.append(os.path.expanduser('..\MachineLearning'))

        from libitmal import dummy as d
        d.dump()
```

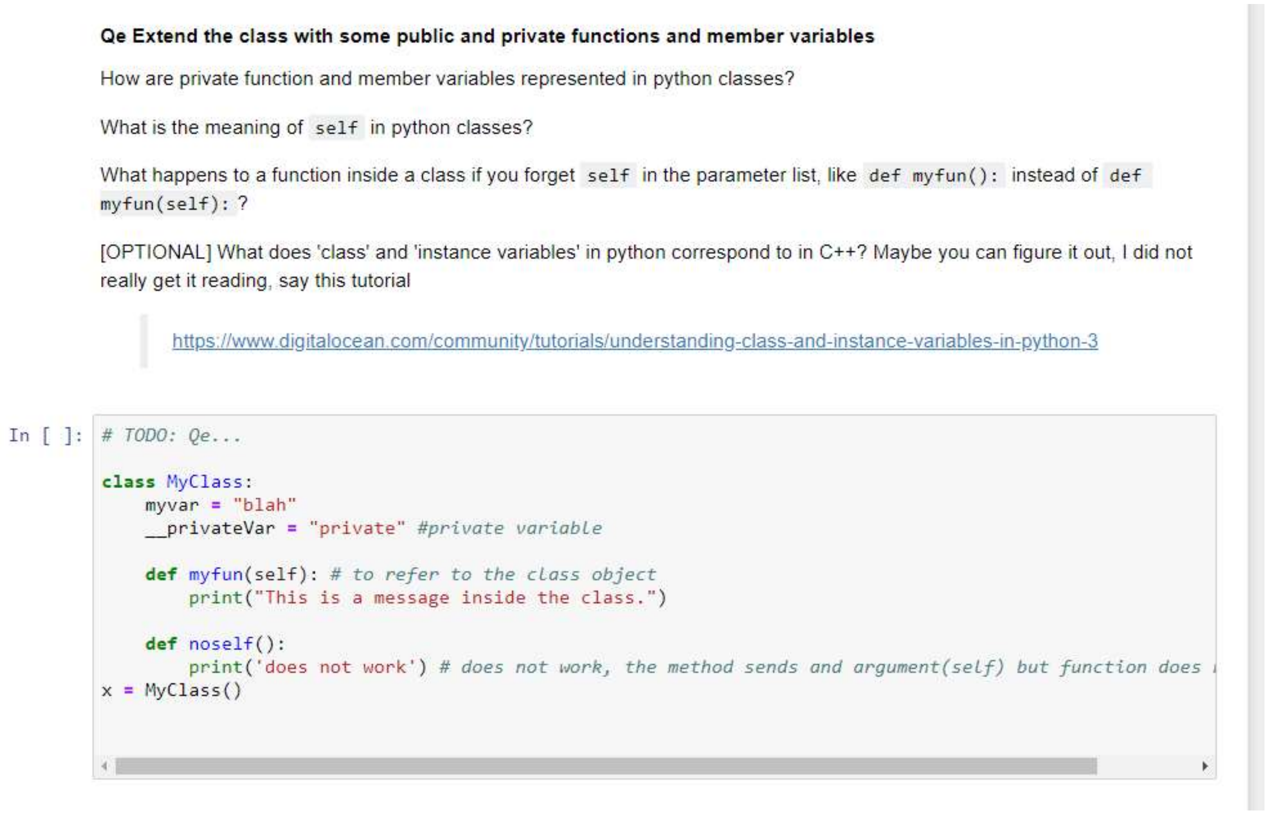
Qc:



Qc How do you 'recompile' a module?

When changing the module code, Jupyter will keep running on the old module. How do you force the Jupyter notebook to re-load the module changes?

In [ ]:
```
# TODO: Qc...
%reset -sf
import sys,os
sys.path.append(os.path.expanduser('..\MachineLearning'))

from libitmal import dummy as d
d.dump()
```



Qe Extend the class with some public and private functions and member variables

How are private function and member variables represented in python classes?

What is the meaning of `self` in python classes?

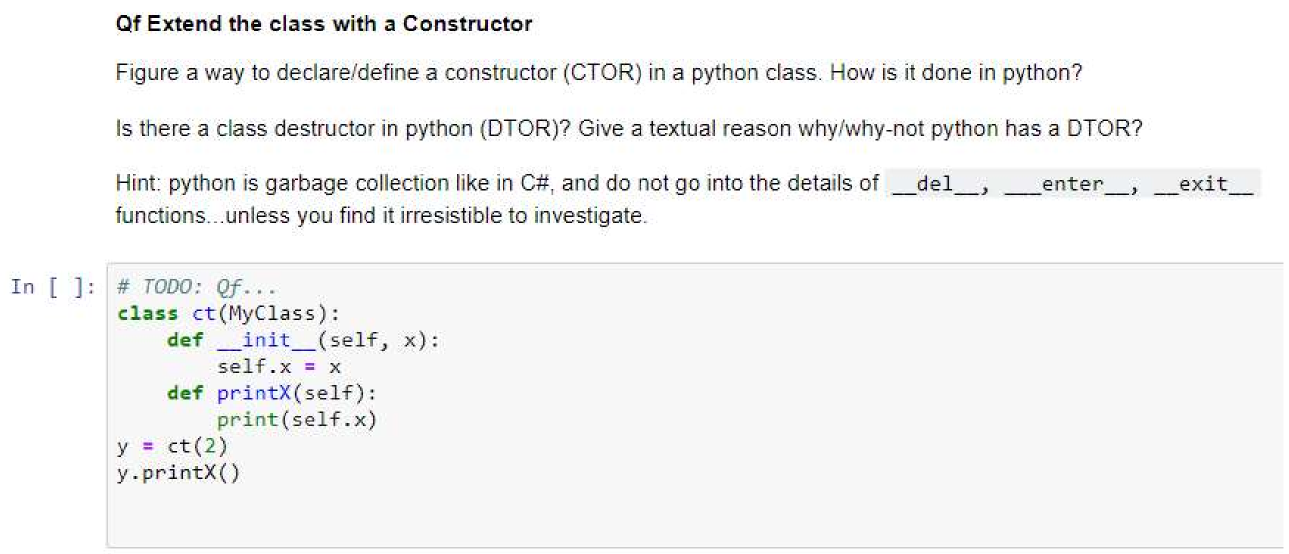
What happens to a function inside a class if you forget `self` in the parameter list, like `def myfun():` instead of `def myfun(self):` ?

[OPTIONAL] What does 'class' and 'instance variables' in python correspond to in C++? Maybe you can figure it out, I did not really get it reading, say this tutorial

> https://www.digitalocean.com/community/tutorials/understanding-class-and-instance-variables-in-python-3

In [ ]:
```
# TODO: Qe...

class MyClass:
    myvar = "blah"
    __privateVar = "private" #private variable

    def myfun(self): # to refer to the class object
        print("This is a message inside the class.")

    def noself():
        print('does not work') # does not work, the method sends and argument(self) but function does
x = MyClass()
```

Qf:



Qf Extend the class with a Constructor

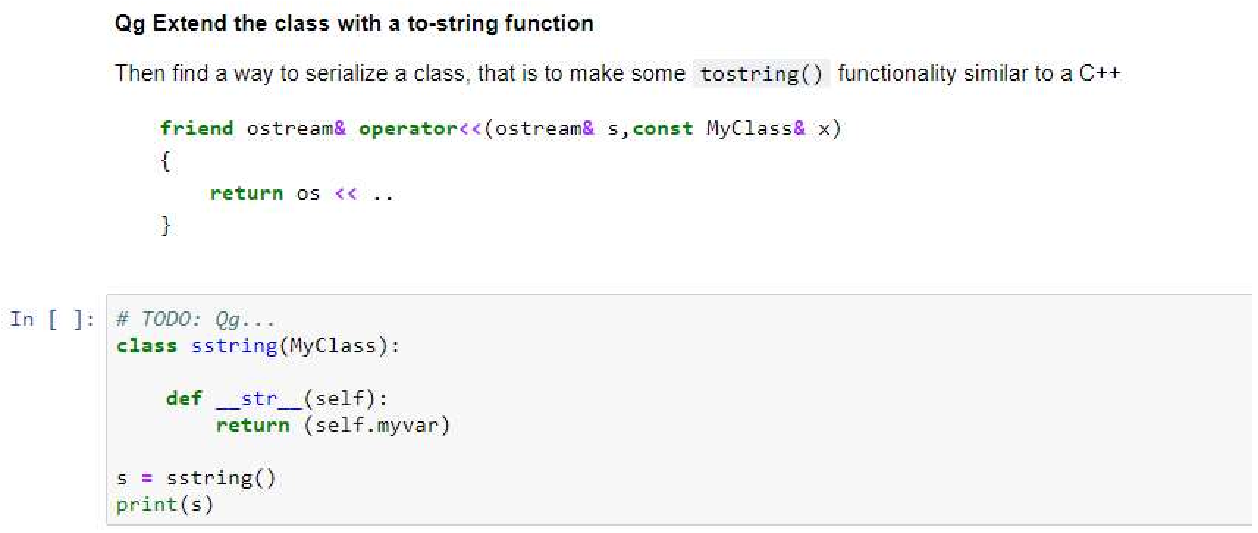
Figure a way to declare/define a constructor (CTOR) in a python class. How is it done in python?

Is there a class destructor in python (DTOR)? Give a textual reason why/why-not python has a DTOR?

Hint: python is garbage collection like in C#, and do not go into the details of `__del__, ___enter__, __exit__` functions...unless you find it irresistible to investigate.

```
In [ ]: # TODO: Qf...
        class ct(MyClass):
            def __init__(self, x):
                self.x = x
            def printX(self):
                print(self.x)
        y = ct(2)
        y.printX()
```

Qg:



Qg Extend the class with a to-string function

Then find a way to serialize a class, that is to make some `tostring()` functionality similar to a C++

```
friend ostream& operator<<(ostream& s,const MyClass& x)
{
    return os << ..
}
```

```
In [ ]: # TODO: Qg...
        class sstring(MyClass):

            def __str__(self):
                return (self.myvar)

        s = sstring()
        print(s)
```

Leksion 3 - datasets.ipynb, Qa+b+c+d+e+f+g:

Qa:

Qa Data load function

We begin with a 100% synthetic dataset called moon. It creates two interleaved half-moon like datasets, and is frequently used as an XOR-like problem set (especially in Deep Learning).

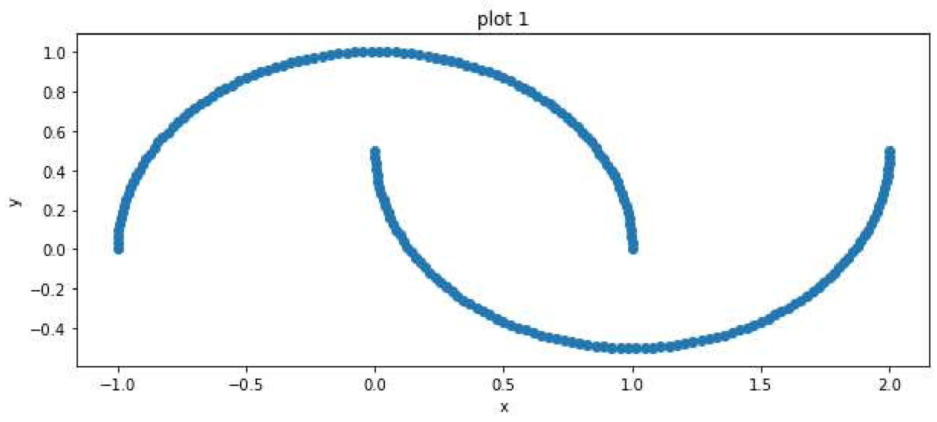
Create a `MOON_GetDataSet()` that generates moon data, based on Scikit-learn's `make_moon()` function.

Extend the `MOON_GetDataSet()` function signature to include some of the parameters found in `make_moon()`, like 'n_sample'.

Also create a `MOON_Plot()` function, that plots the data...good thing here is that the feature set is 2D and easy to handle!

```
In [1]: # TODO: Qa...

        # NOTE: some free help here regarding import clauses...
        %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        from sklearn.datasets import make_moons

        def MOON_GetDataSet(n_samples):
            # TODO: your code here...
            return make_moons(n_samples)
        def MOON_Plot(X, y, title="My title" , xlable="", ylabel=""):
            # TODO: some simple plot commands,

            plt.figure('one', figsize=(10,4))
            plt.scatter(X[:,0],X[:,1])
            plt.title(title)
            plt.xlabel(xlable)
            plt.ylabel(ylabel)
            plt.show()


        # TEST CODE:
        X, y=MOON_GetDataSet(n_samples=200)
        print("X.shape=",X.shape,", y.shape=",y.shape)
        MOON_Plot(X,y, "plot 1", 'x' , 'y')
```

```
X.shape= (200, 2) , y.shape= (200,)
```



Qb:

Qb Try it with a train-test split function

Now, use a train-test split function from Scikit-learn, that is able to split a `(X, y)` dataset tuple into a train-set and a test-set.

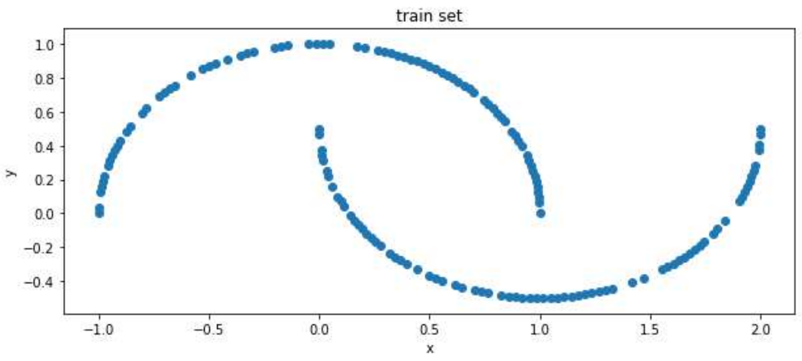
Plot the train and test data using your `MOON_Plot` function.

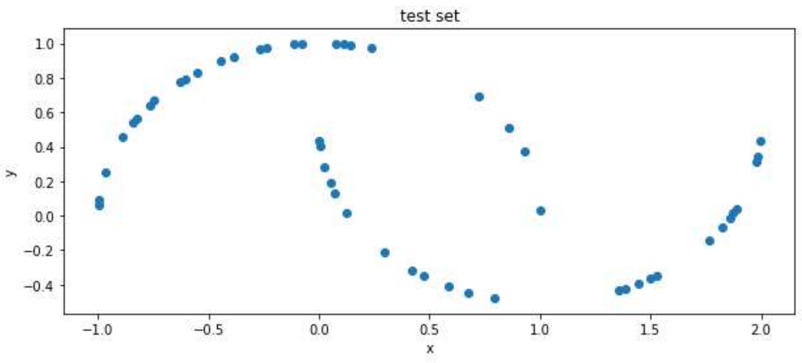
Extend the plot function to add a plot title and x- and y-labels to the plot, say as default parameters

```
def MOON_Plot(X, y, title="my title", xlable="", ylabel=""):
    # implementation here...
```

or similar. Use the titles "train" and "test" when plotting the train and test data.

```
In [ ]: # TODO: Qb....


        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X,y)

        MOON_Plot(X_train,y_train, 'train set' , 'x' , 'y')
        MOON_Plot(X_test, y_test, 'test set' , 'x' , 'y')
```





Qc:

Qc Data load function

Now for the MNIST data set, creating an easy to use data-loader, `MNIST_GetDataSet()` .

There are several ways of getting the MNIST dataset. You could base the data loader on the `fetch_mldata('MNIST original')` function or try to use the `keras.datasets.mnist.load_data()` function.

The later function pre-splits into a train-test set, and to be compatible with the former, you must concatenate the train-test and return a plain `X, y` set.

Also create a `MNIST_PlotDigit()` , that is able to plot a single digit from the dataset, and try to plot some of the digits in the dataset (set TEST CODE below).

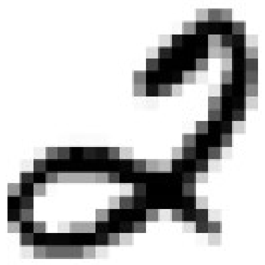
```
In [ ]: # fetch once , takes long
        from sklearn.datasets import fetch_openml
        from sklearn.datasets import fetch_mldata
        print('started')
        mnist = fetch_openml('mnist_784', version=1, cache=True)
        print('loaded')
```

```
started
```

```
In [ ]: # TODO: Qc...

        ## from keras.datasets import mnist
        # mnist = fetch_openml('mnist_784', version=1, cache=True)
        # mnist = fetch_mldata('MNIST original')


        def MNIST_PlotDigit(data):

            image = data.reshape(28, 28)
            # TODO: add plot functionality for a single digit...
            plt.imshow(image, cmap = plt.cm.binary,
            interpolation="nearest")
            plt.axis("off")
            plt.show()

        def MNIST_GetDataSet():
            # TODO: use mnist = fetch_mldata('MNIST original') or mnist.load_data(),
            #       but return as a single X-y tuple
            return (mnist['data'], mnist['target'])


        # TEST CODE:
        X, y = MNIST_GetDataSet()
        print("X.shape=",X.shape, ", y.shape=",y.shape)

        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=50000, shuffle=True, random_state=42) # random state set

        print("X_train.shape=",X_train.shape,", X_test.shape=",X_test.shape)
        MNIST_PlotDigit(X_train[3000])
```

```
X.shape= (70000, 784) , y.shape= (70000,)
```

```
X_train.shape= (50000, 784) , X_test.shape= (20000, 784)
```



Qd:

Qd Data load function

Creating the iris data loader, `IRIS_GetDataSet()`, this time we use the iris loader located in `sklearn.datasets.load_iris()`.

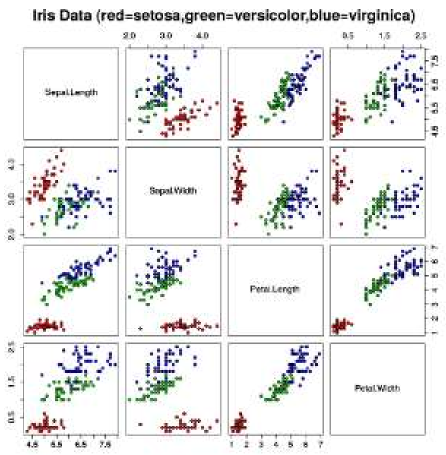
In [5]:
```
# TODO: Qd...
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
Y = iris.target
```

Qe:

Qe Examine the data via scatter plots

Now, as a data-scientist, it is always good to get some grasp on how your data looks like. For the iris data we now want to plot some of the features-against-some-of the other-features to see how they separate in the given 2D-feature space.

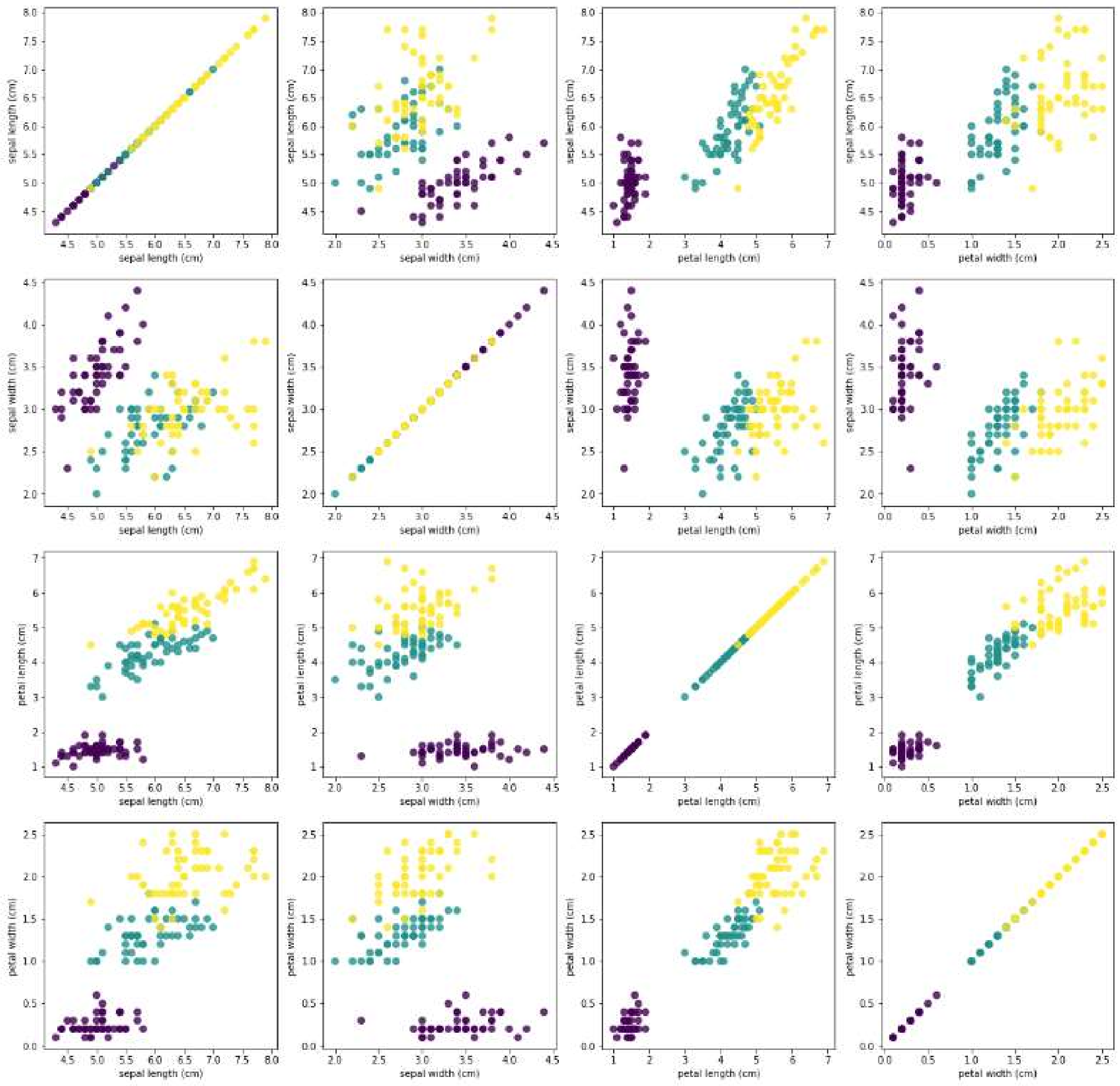
A scatter plot for all iris features against all other may look like



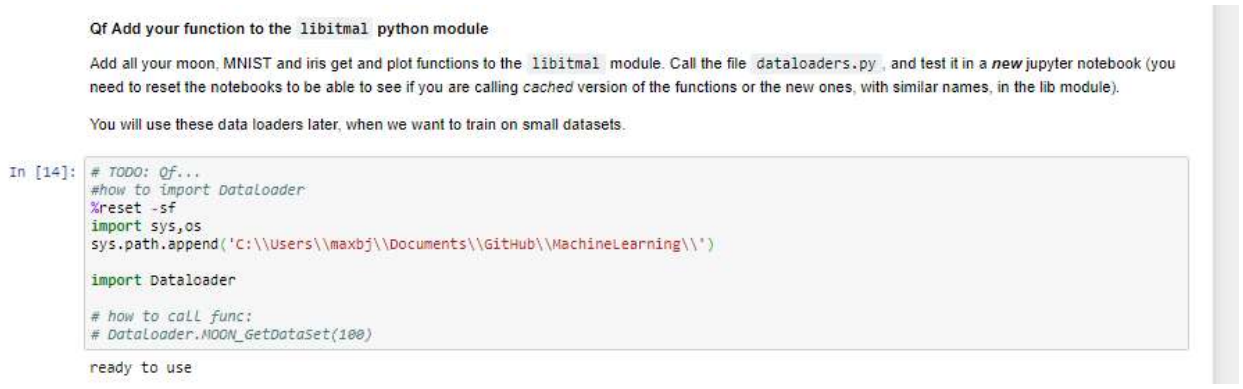
Create a plot function that takes just two feature dimensions and plots them in a 2D plot, and plot all features against all others (resembling the "Iris Data" scatter plot just above).

In [9]:
```
# TODO: Qe...
import matplotlib.pyplot as plt

def plot(k):
    size = len(k.data[0])
    plt.figure(figsize=(20,20))
    counter = 0;
    for z in range(0,size):
        for i in range(0,size):
            counter = counter + 1
            plt.subplot(4,4, counter)
            plt.scatter(k.data.T[i], k.data.T[z], c=k.target, s=50*1, alpha=0.8, cmap='viridis')
            plt.xlabel(k.feature_names[i])
            plt.ylabel(k.feature_names[z])

plot(iris)
```

Qe output:


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

Qf:



Qf Add your function to the `libitmal` python module

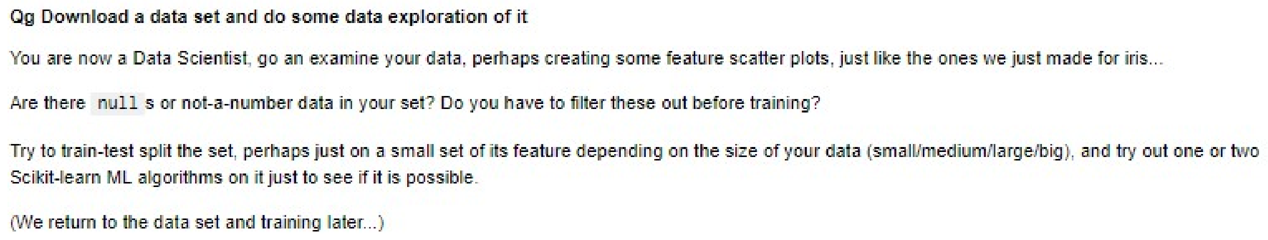
Add all your moon, MNIST and iris get and plot functions to the `libitmal` module. Call the file `dataloaders.py`, and test it in a ***new*** jupyter notebook (you need to reset the notebooks to be able to see if you are calling *cached* version of the functions or the new ones, with similar names, in the lib module).

You will use these data loaders later, when we want to train on small datasets.

```
In [14]: # TODO: Qf...
         #how to import Dataloader
         %reset -sf
         import sys,os
         sys.path.append('C:\\Users\\maxbj\\Documents\\GitHub\\MachineLearning\\')

         import Dataloader

         # how to call func:
         # Dataloader.MOON_GetDataSet(100)
```

```
ready to use
```

Qg:



Qg Download a data set and do some data exploration of it

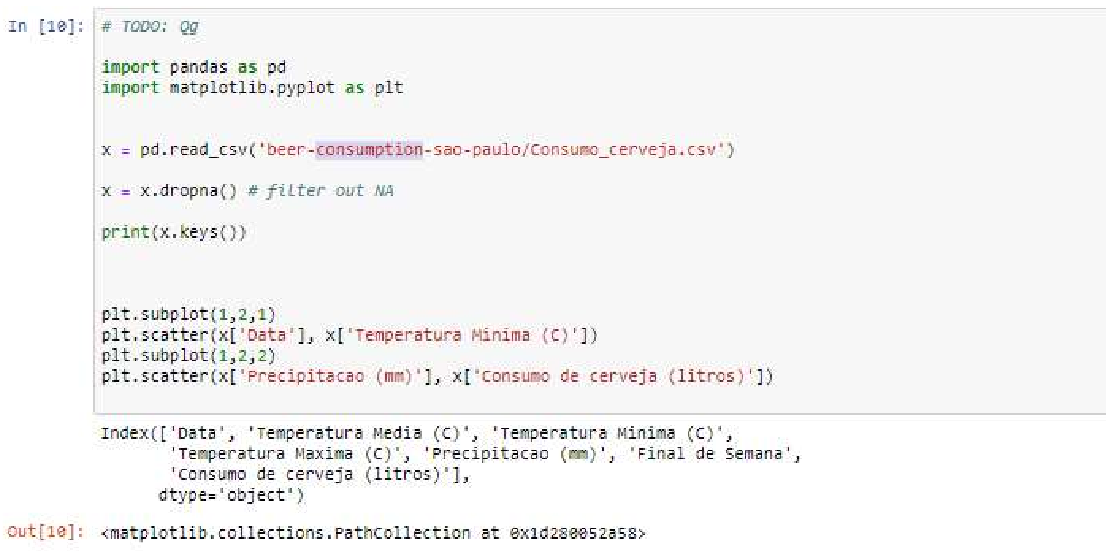
You are now a Data Scientist, go an examine your data, perhaps creating some feature scatter plots, just like the ones we just made for iris...

Are there `null` s or not-a-number data in your set? Do you have to filter these out before training?

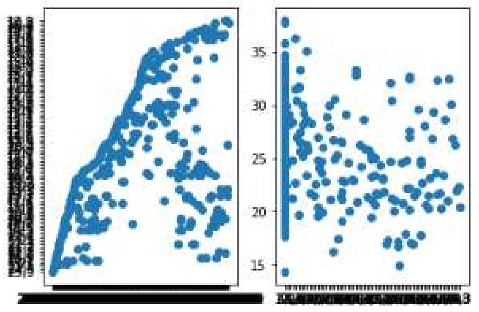
Try to train-test split the set, perhaps just on a small set of its feature depending on the size of your data (small/medium/large/big), and try out one or two Scikit-learn ML algorithms on it just to see if it is possible.

(We return to the data set and training later...)

Vi har valgt at bruge beer consumption eksempel som i opgave beskrivelsen:

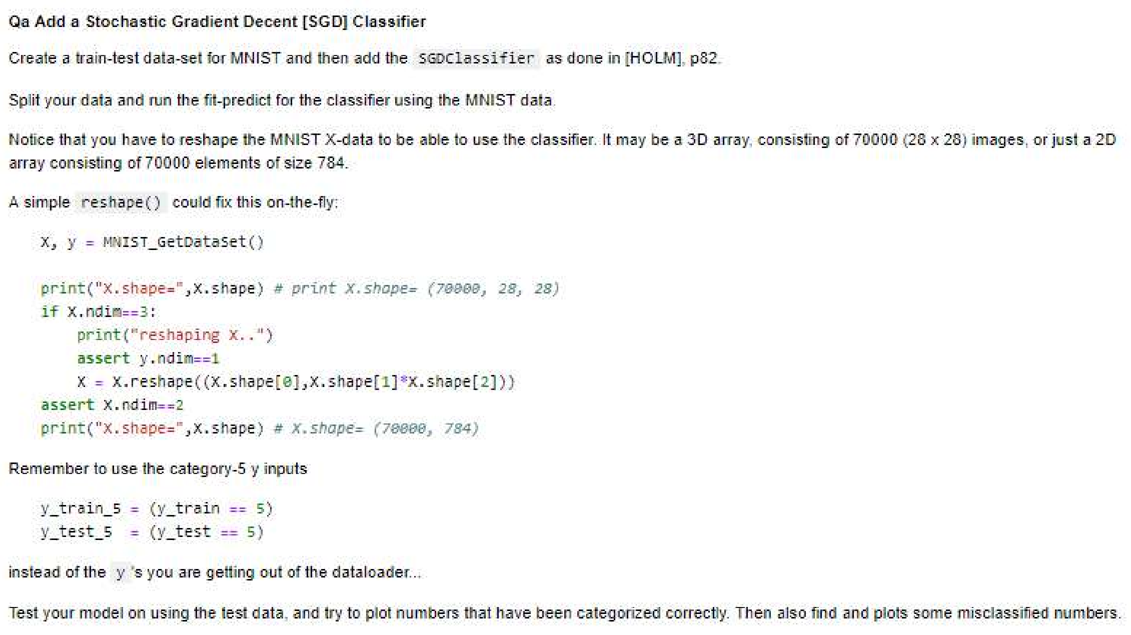


```
In [10]: # TODO: Qg

         import pandas as pd
         import matplotlib.pyplot as plt


         x = pd.read_csv('beer-consumption-sao-paulo/Consumo_cerveja.csv')

         x = x.dropna() # filter out NA

         print(x.keys())



         plt.subplot(1,2,1)
         plt.scatter(x['Data'], x['Temperatura Minima (C)'])
         plt.subplot(1,2,2)
         plt.scatter(x['Precipitacao (mm)'], x['Consumo de cerveja (litros)'])

         Index(['Data', 'Temperatura Media (C)', 'Temperatura Minima (C)',
                'Temperatura Maxima (C)', 'Precipitacao (mm)', 'Final de Semana',
                'Consumo de cerveja (litros)'],
               dtype='object')

Out[10]: <matplotlib.collections.PathCollection at 0x1d280052a58>
```



Leksion 3 - dummy_classifier.ipynb, Qa+b:

Qa:



Qa Add a Stochastic Gradient Decent [SGD] Classifier

Create a train-test data-set for MNIST and then add the `SGDClassifier` as done in [HOLM], p82.

Split your data and run the fit-predict for the classifier using the MNIST data.

Notice that you have to reshape the MNIST X-data to be able to use the classifier. It may be a 3D array, consisting of 70000 (28 x 28) images, or just a 2D array consisting of 70000 elements of size 784.

A simple `reshape()` could fix this on-the-fly:

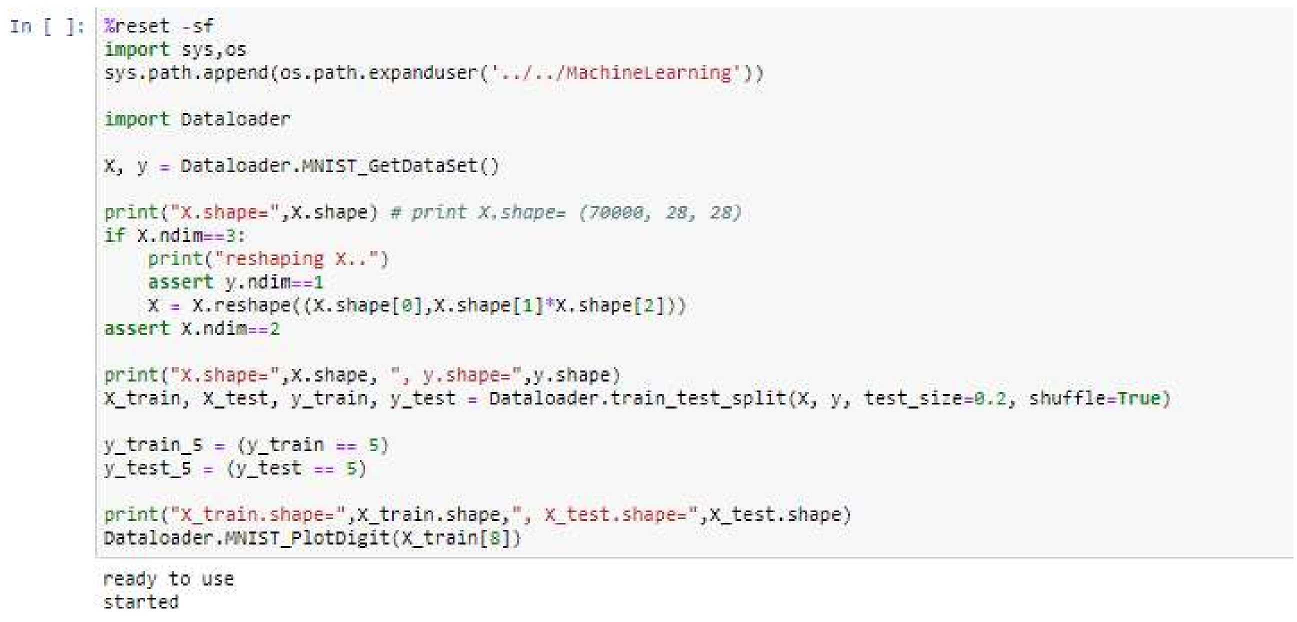
```
X, y = MNIST_GetDataSet()

print("X.shape=",X.shape) # print X.shape= (70000, 28, 28)
if X.ndim==3:
    print("reshaping X..")
    assert y.ndim==1
    X = X.reshape((X.shape[0],X.shape[1]*X.shape[2]))
assert X.ndim==2
print("X.shape=",X.shape) # X.shape= (70000, 784)
```

Remember to use the category-5 y inputs

```
y_train_5 = (y_train == 5)
y_test_5  = (y_test == 5)
```

instead of the `y` 's you are getting out of the dataloader...

Test your model on using the test data, and try to plot numbers that have been categorized correctly. Then also find and plots some misclassified numbers.



```
In [ ]: %reset -sf
        import sys,os
        sys.path.append(os.path.expanduser('../../MachineLearning'))

        import Dataloader

        X, y = Dataloader.MNIST_GetDataSet()

        print("X.shape=",X.shape) # print X.shape= (70000, 28, 28)
        if X.ndim==3:
            print("reshaping X..")
            assert y.ndim==1
            X = X.reshape((X.shape[0],X.shape[1]*X.shape[2]))
        assert X.ndim==2

        print("X.shape=",X.shape, ", y.shape=",y.shape)
        X_train, X_test, y_train, y_test = Dataloader.train_test_split(X, y, test_size=0.2, shuffle=True)

        y_train_5 = (y_train == 5)
        y_test_5 = (y_test == 5)

        print("X_train.shape=",X_train.shape,", X_test.shape=",X_test.shape)
        Dataloader.MNIST_PlotDigit(X_train[8])
```

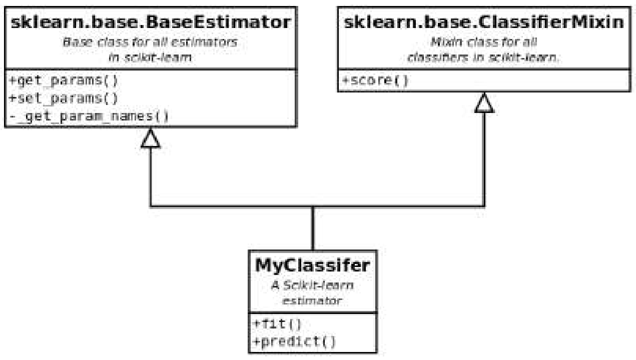
```
        ready to use
        started
```

Qb:

Qb Implement a dummy binary classifier

Follow the code found in [HOML], p84, but name you estimator `DummyClassifier` instead of `Never5Classifyer`.

Here our Python class knowledge comes into play. The estimator class hierarchy looks like


sklearn.base.BaseEstimator
Base class for all estimators in scikit-learn
+get_params()
+set_params()
-_get_param_names()
sklearn.base.ClassifierMixin
Mixin class for all classifiers in scikit-learn.
+score()
MyClassifer
A Scikit-learn estimator
+fit()
+predict()

All Scikit-learn classifiers inherit form `BaseEstimator` (and possible also `ClassifierMixin`), and they must have a `fit-predict` function pair (strangely not in the base class!) and you can actually find the `sklearn.base.BaseEstimator` and `sklearn.base.ClassifierMixin` python source code somewhere in you anaconda install dir, if you should have the nerves to go to such interesting details.

But surprisingly you may just want to implement a class that contains the `fit-predict` functions, ***without inheriting*** from the `BaseEstimator`, things still work due to the pythonic 'duck-typing': you just need to have the class implement the needed interfaces, obviously `fit()` and `predict()` but also the more obscure `get_params()` etc....then the class 'looks like' a `BaseEstimator` ...and if it looks like an estimator, it *is* an estimator (aka. duct typing).

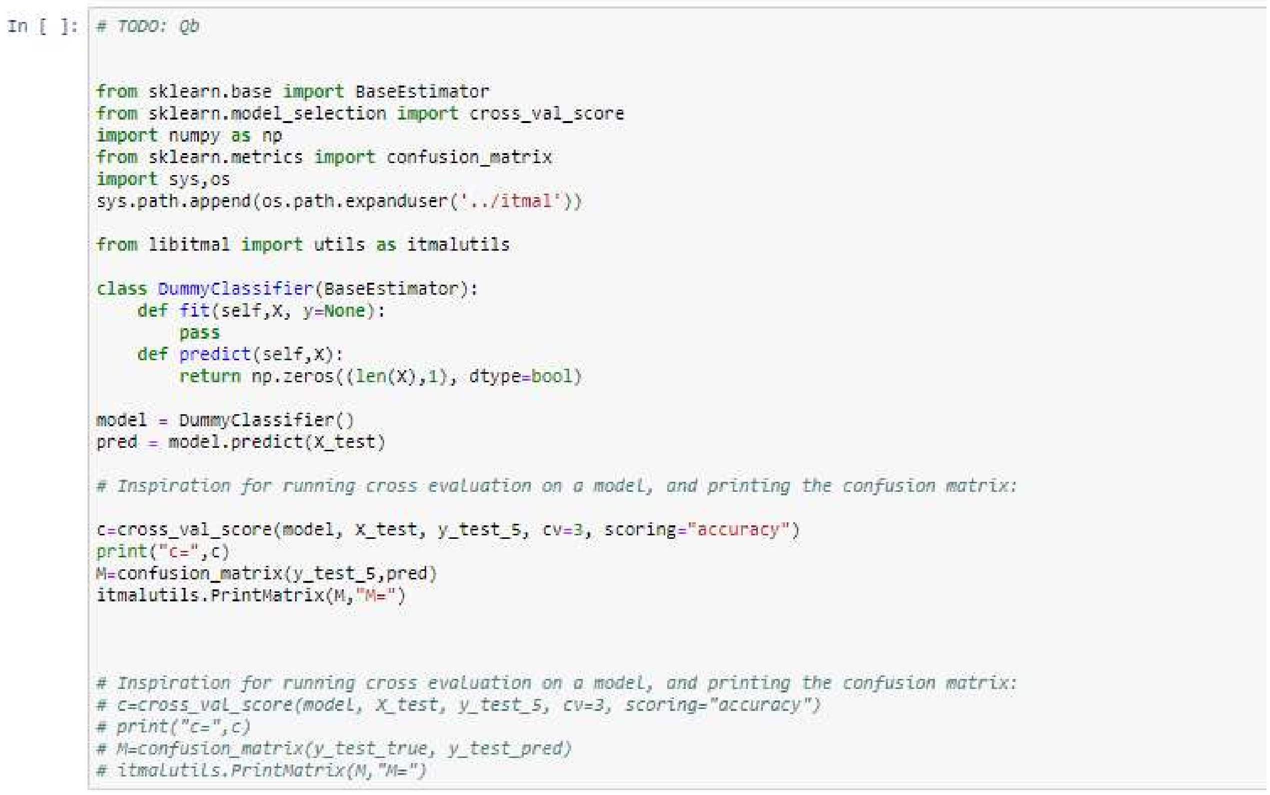
Templates in C++ also allow the language to use compile-time duck typing!

> https://en.wikipedia.org/wiki/Duck_typing

Call the fit-predict on a newly instantiated `DummyClassifier` object, and try to compare the confusion matrix for both the dummy and SDG classifier.

We will be discussing the confusion matrix next, but first, print the `y_test_5.shape` and count the numbers of `y_test_5==True` and `y_test_5==False` and see if you can find these numbers in the confusion matrix.

Qb fortsat:

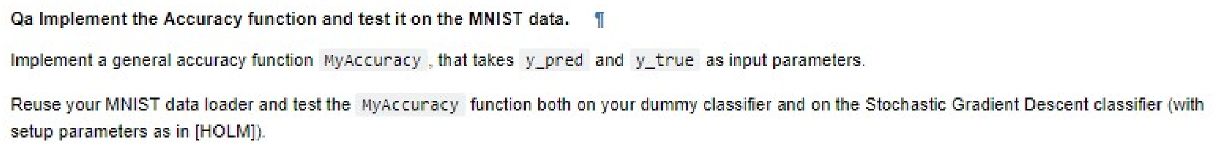


```
In [ ]: # TODO: Qb


from sklearn.base import BaseEstimator
from sklearn.model_selection import cross_val_score
import numpy as np
from sklearn.metrics import confusion_matrix
import sys,os
sys.path.append(os.path.expanduser('../itmal'))

from libitmal import utils as itmalutils

class DummyClassifier(BaseEstimator):
    def fit(self,X, y=None):
        pass
    def predict(self,X):
        return np.zeros((len(X),1), dtype=bool)

model = DummyClassifier()
pred = model.predict(X_test)

# Inspiration for running cross evaluation on a model, and printing the confusion matrix:

c=cross_val_score(model, X_test, y_test_5, cv=3, scoring="accuracy")
print("c=",c)
M=confusion_matrix(y_test_5,pred)
itmalutils.PrintMatrix(M,"M=")



# Inspiration for running cross evaluation on a model, and printing the confusion matrix:
# c=cross_val_score(model, X_test, y_test_5, cv=3, scoring="accuracy")
# print("c=",c)
# M=confusion_matrix(y_test_true, y_test_pred)
# itmalutils.PrintMatrix(M,"M=")
```

Vi ved ikke helt hvor det gik galt her...

Leksion 3 -: metrics.ipynb, Qa+b+c+d+e

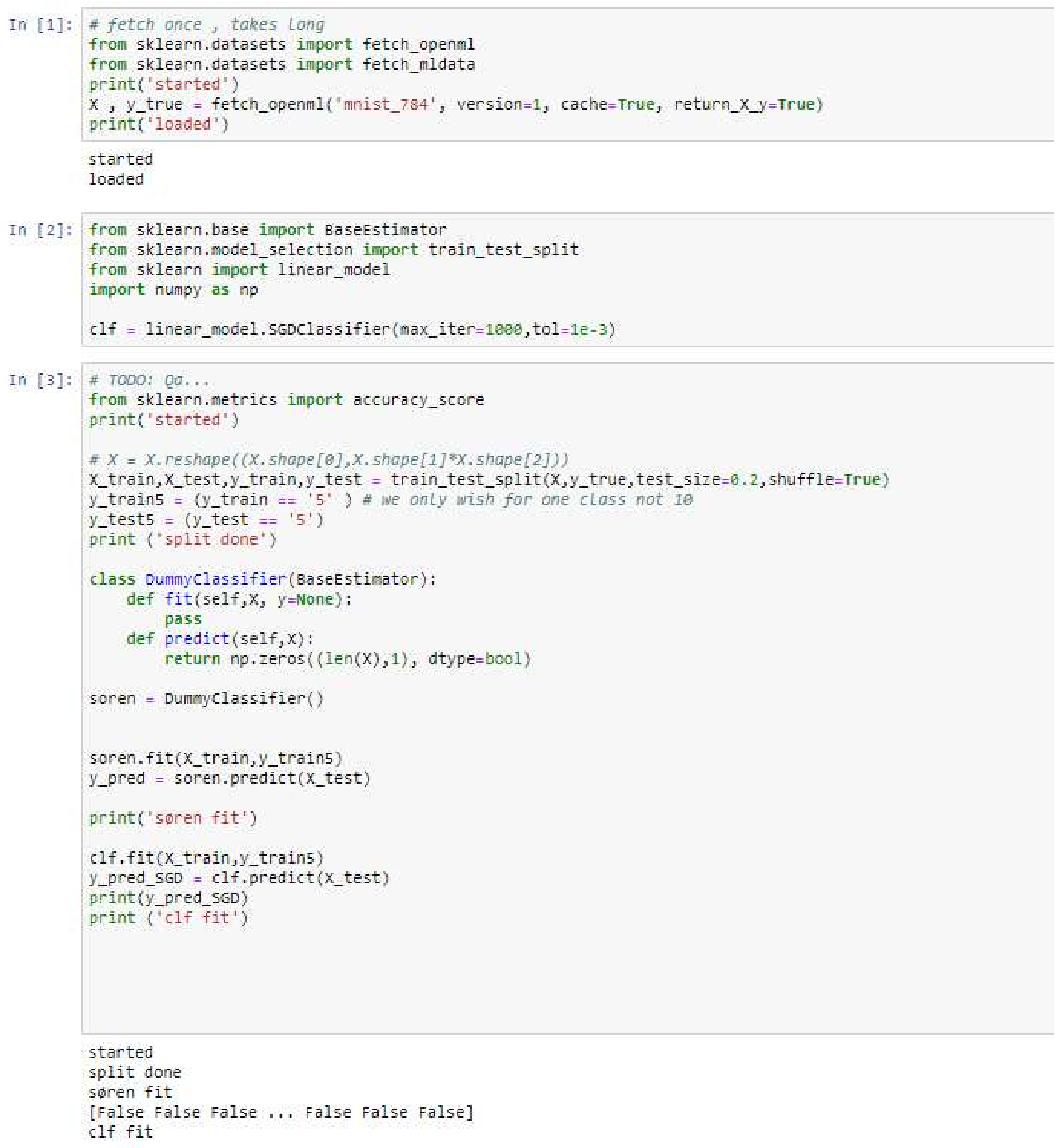
Qa:



Qa Implement the Accuracy function and test it on the MNIST data. ¶

Implement a general accuracy function `MyAccuracy`, that takes `y_pred` and `y_true` as input parameters.

Reuse your MNIST data loader and test the `MyAccuracy` function both on your dummy classifier and on the Stochastic Gradient Descent classifier (with setup parameters as in [HOLM]).



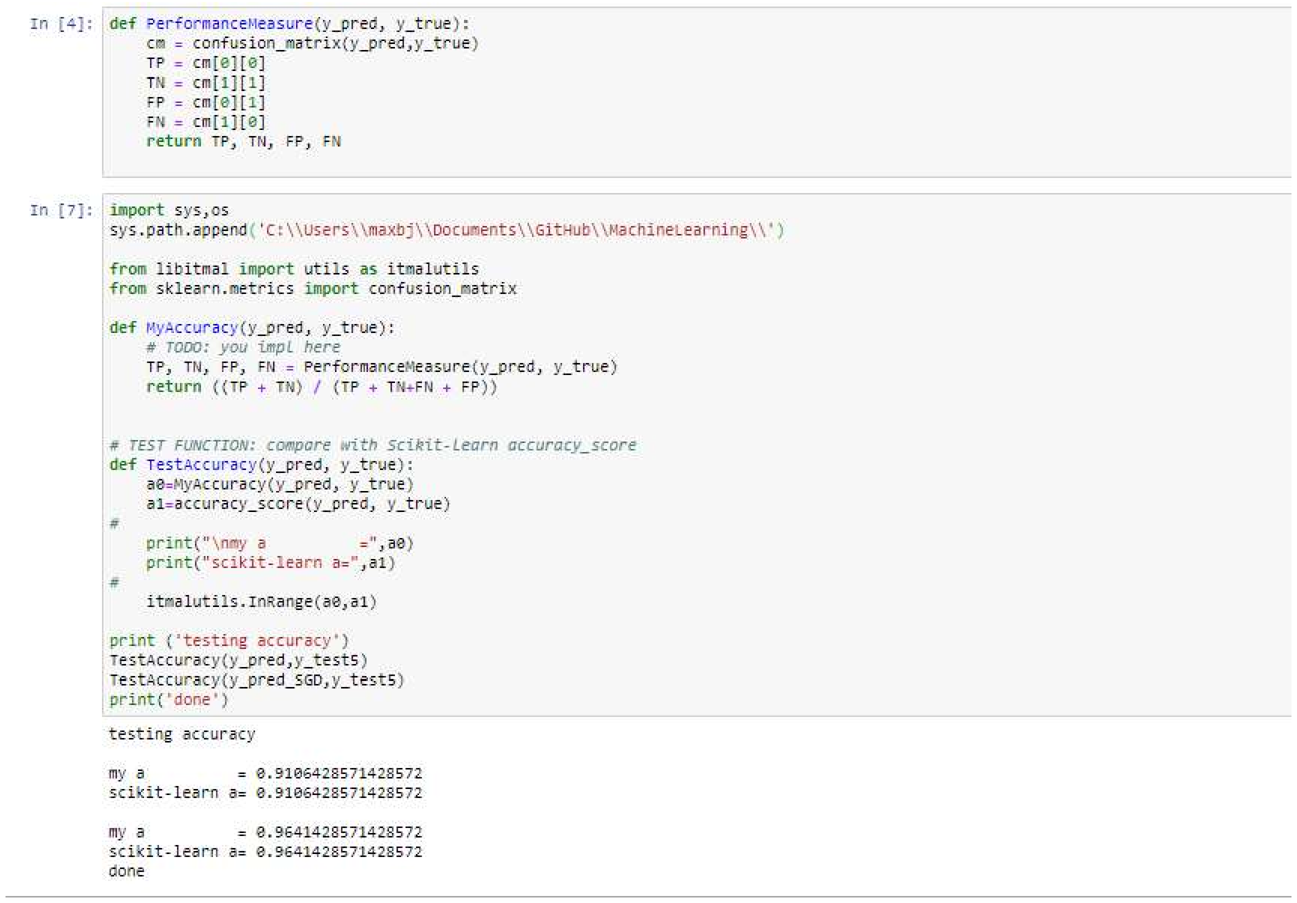
In [1]:
```
# fetch once , takes long
from sklearn.datasets import fetch_openml
from sklearn.datasets import fetch_mldata
print('started')
X , y_true = fetch_openml('mnist_784', version=1, cache=True, return_X_y=True)
print('loaded')
```

```
started
loaded
```

In [2]:
```
from sklearn.base import BaseEstimator
from sklearn.model_selection import train_test_split
from sklearn import linear_model
import numpy as np

clf = linear_model.SGDClassifier(max_iter=1000,tol=1e-3)
```

In [3]:
```
# TODO: Qa...
from sklearn.metrics import accuracy_score
print('started')

# X = X.reshape((X.shape[0],X.shape[1]*X.shape[2]))
X_train,X_test,y_train,y_test = train_test_split(X,y_true,test_size=0.2,shuffle=True)
y_train5 = (y_train == '5' ) # we only wish for one class not 10
y_test5 = (y_test == '5')
print ('split done')

class DummyClassifier(BaseEstimator):
    def fit(self,X, y=None):
        pass
    def predict(self,X):
        return np.zeros((len(X),1), dtype=bool)

soren = DummyClassifier()


soren.fit(X_train,y_train5)
y_pred = soren.predict(X_test)

print('søren fit')

clf.fit(X_train,y_train5)
y_pred_SGD = clf.predict(X_test)
print(y_pred_SGD)
print ('clf fit')
```

```
started
split done
søren fit
[False False False ... False False False]
clf fit
```

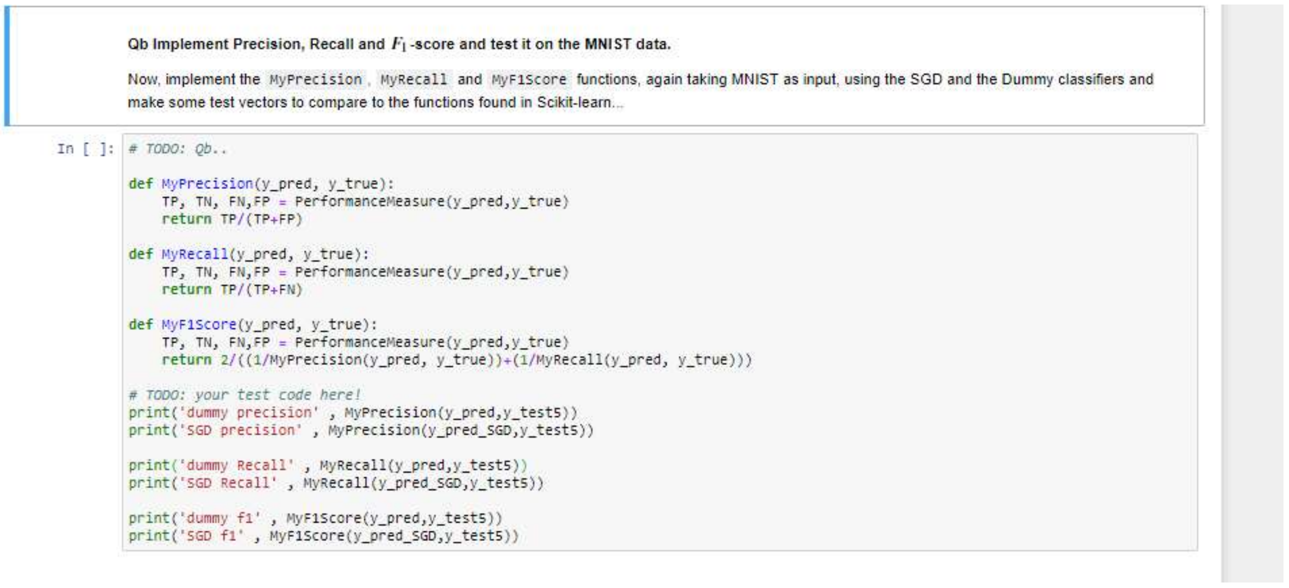


```
In [4]: def PerformanceMeasure(y_pred, y_true):
            cm = confusion_matrix(y_pred,y_true)
            TP = cm[0][0]
            TN = cm[1][1]
            FP = cm[0][1]
            FN = cm[1][0]
            return TP, TN, FP, FN
```

```
In [7]: import sys,os
        sys.path.append('C:\\Users\\maxbj\\Documents\\GitHub\\MachineLearning\\')

        from libitmal import utils as itmalutils
        from sklearn.metrics import confusion_matrix

        def MyAccuracy(y_pred, y_true):
            # TODO: you impl here
            TP, TN, FP, FN = PerformanceMeasure(y_pred, y_true)
            return ((TP + TN) / (TP + TN+FN + FP))


        # TEST FUNCTION: compare with Scikit-Learn accuracy_score
        def TestAccuracy(y_pred, y_true):
            a0=MyAccuracy(y_pred, y_true)
            a1=accuracy_score(y_pred, y_true)
        #
            print("\nmy a          =",a0)
            print("scikit-learn a=",a1)
        #
            itmalutils.InRange(a0,a1)

        print ('testing accuracy')
        TestAccuracy(y_pred,y_test5)
        TestAccuracy(y_pred_SGD,y_test5)
        print('done')
```

```
testing accuracy

my a          = 0.9106428571428572
scikit-learn a= 0.9106428571428572

my a          = 0.9641428571428572
scikit-learn a= 0.9641428571428572
done
```

Qb:

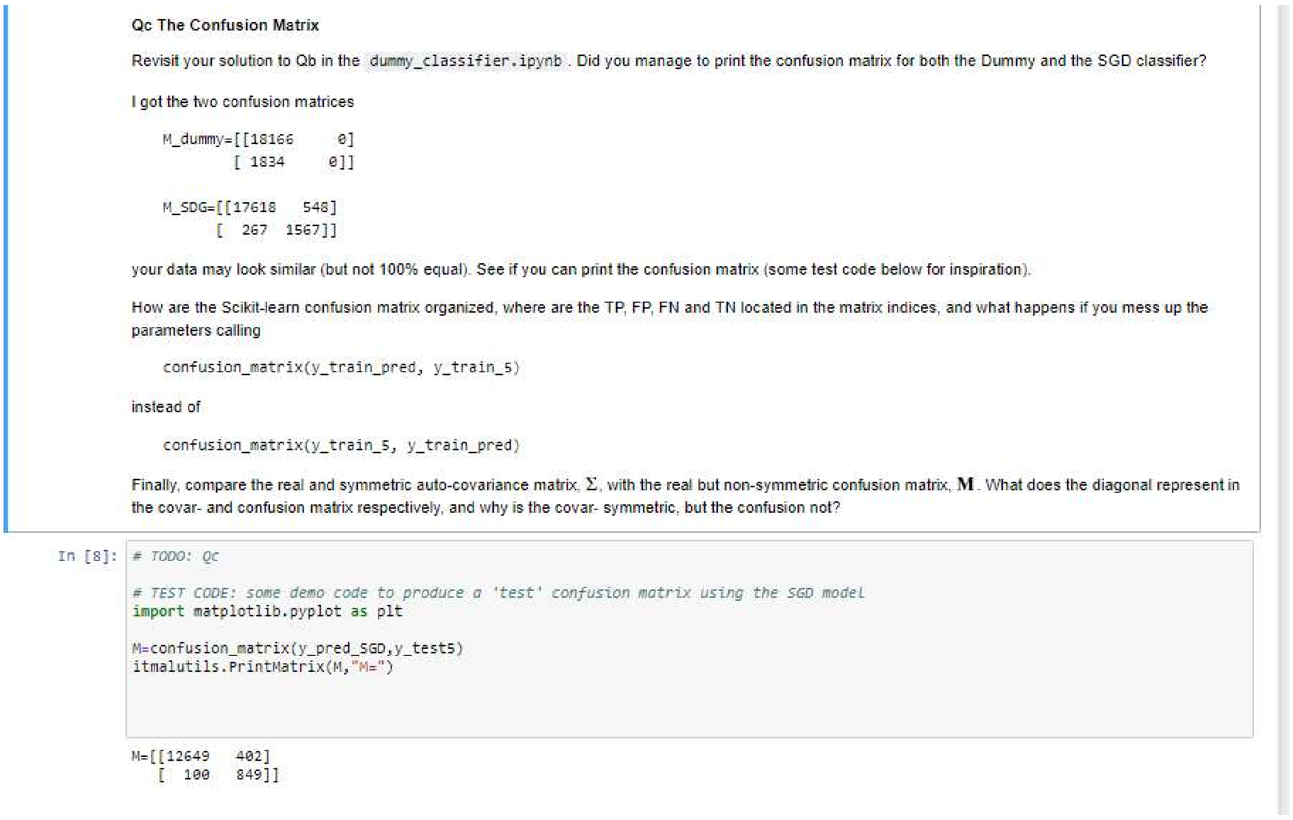


Qb Implement Precision, Recall and F_1-score and test it on the MNIST data.

Now, implement the `MyPrecision`, `MyRecall` and `MyF1Score` functions, again taking MNIST as input, using the SGD and the Dummy classifiers and make some test vectors to compare to the functions found in Scikit-learn...

```
In [ ]: # TODO: Qb..

        def MyPrecision(y_pred, y_true):
            TP, TN, FN,FP = PerformanceMeasure(y_pred,y_true)
            return TP/(TP+FP)

        def MyRecall(y_pred, y_true):
            TP, TN, FN,FP = PerformanceMeasure(y_pred,y_true)
            return TP/(TP+FN)

        def MyF1Score(y_pred, y_true):
            TP, TN, FN,FP = PerformanceMeasure(y_pred,y_true)
            return 2/((1/MyPrecision(y_pred, y_true))+(1/MyRecall(y_pred, y_true)))

        # TODO: your test code here!
        print('dummy precision' , MyPrecision(y_pred,y_test5))
        print('SGD precision' , MyPrecision(y_pred_SGD,y_test5))

        print('dummy Recall' , MyRecall(y_pred,y_test5))
        print('SGD Recall' , MyRecall(y_pred_SGD,y_test5))

        print('dummy f1' , MyF1Score(y_pred,y_test5))
        print('SGD f1' , MyF1Score(y_pred_SGD,y_test5))
```

Qc:



Qc The Confusion Matrix

Revisit your solution to Qb in the `dummy_classifier.ipynb`. Did you manage to print the confusion matrix for both the Dummy and the SGD classifier?

I got the two confusion matrices

```
M_dummy=[[18166     0]
         [ 1834     0]]

M_SDG=[[17618   548]
       [  267  1567]]
```

your data may look similar (but not 100% equal). See if you can print the confusion matrix (some test code below for inspiration).

How are the Scikit-learn confusion matrix organized, where are the TP, FP, FN and TN located in the matrix indices, and what happens if you mess up the parameters calling

```
confusion_matrix(y_train_pred, y_train_5)
```

instead of

```
confusion_matrix(y_train_5, y_train_pred)
```

Finally, compare the real and symmetric auto-covariance matrix, Σ, with the real but non-symmetric confusion matrix, $\mathbf{M}$. What does the diagonal represent in the covar- and confusion matrix respectively, and why is the covar- symmetric, but the confusion not?

In [8]:
```
# TODO: Qc

# TEST CODE: some demo code to produce a 'test' confusion matrix using the SGD model
import matplotlib.pyplot as plt

M=confusion_matrix(y_pred_SGD,y_test5)
itmalutils.PrintMatrix(M,"M=")
```

```
M=[[12649   402]
   [  100   849]]
```

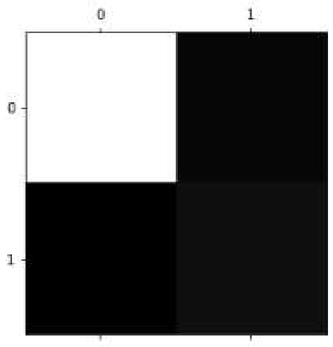
Qd:

Qd A Confusion Matrix Heat-map

Generate a *heat map* image for the confusion matrices, `M_dummy` and `M_SGD` respectively, getting inspiration from [HOML], pp96-97.

This heat map could be an important guide for you when analysing multiclass data in the future.

```
In [9]: # TODO: Qd
        plt.matshow(M,cmap=plt.cm.gray)
        plt.show() # her ses det flest TP, hvilket giver god mening når det 5 og ikke 5
```



Qe:

Qe Run a classifier on your data

Finally, try to run a classifier on the data-set you selected previously, perhaps starting with the SGD.

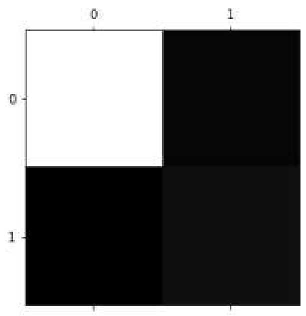
Is it possible to classify at all on your data, or do we need regression instead?

Are you able to do supervised learning, or are there no obvious `y_true` data in your set at all?

If your data is in the form, where you are able to do supervised-classification, could you produce a confusion matrix heatmap, then?

```
In [10]: # TODO: Qe...
         pred = clf.predict(X_test)

         M=confusion_matrix(pred,y_test5)
         itmalutils.PrintMatrix(M,"M=")

         plt.matshow(M,cmap=plt.cm.gray)
         plt.show()
```

```
M=[[12649   402]
   [  100   849]]
```



Wiki af gruppe 12 -Training and Testing Data :

https://blackboard.au.dk/webapps/Bb-wiki-BBLEARN/wikiView?course_id=_124256_1&wiki_id=_75492_1&page_guid=a195034190454ee4b1633497a463e36c