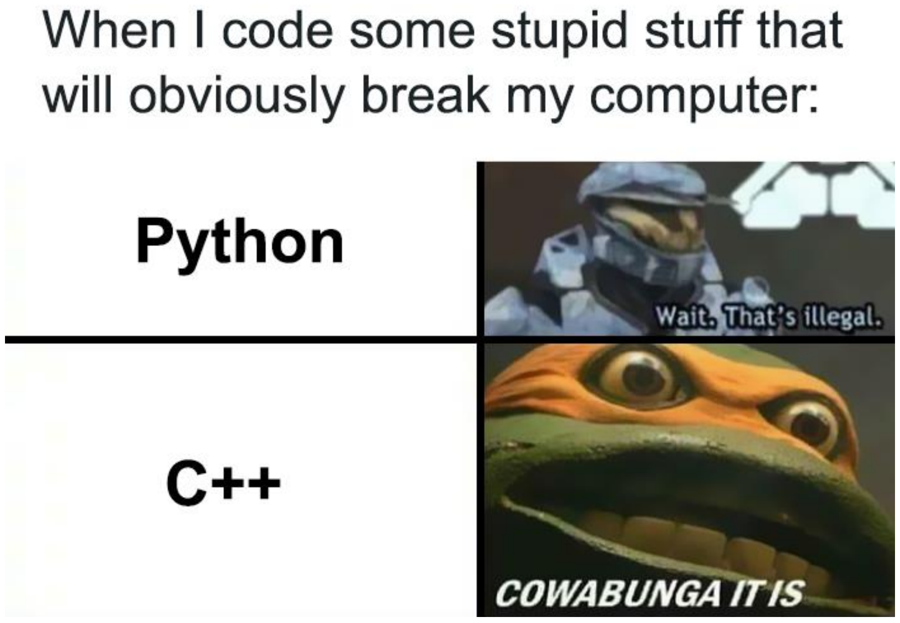
O1

SWMAL-01 Group 13


When I code some stupid stuff that will obviously break my computer:
Python
Wait. That's illegal.
C++
COWABUNGA IT IS

Group members
Christoffer Regnar Mølck - 202009347
Martin Teit Bruun Andersen - 202109592
Morten Kierkegaard Erichsen - 201807571

13. februar 2024

1 Intro L01

1.0.1 Qa) the Φ and the R^2 score

Code:

```

model.score(X,y), model.coef_, model.intercept_
```

Which outputs:

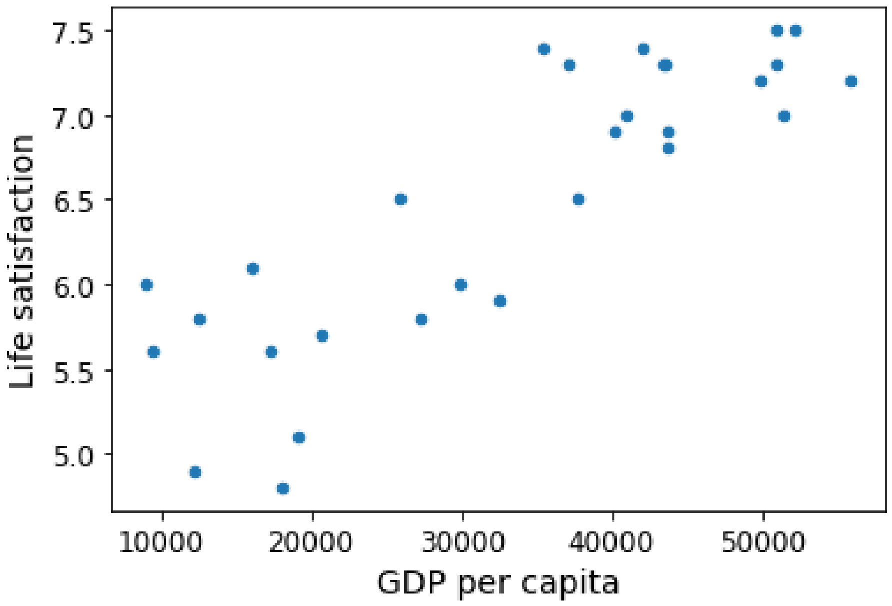
```
    R^2: 0.734
    Coefficient: 4.91e-5
    Intercept: 4.85
```

The R^2 value describes how well the data is fitted, 1 being best (albeit unrealistic) The higher the better R^2, measuring goodness of fit Coefficient parameter is the slope, and intercept is, well, the intercept with the y-axis

1.0.2 Qb) Using k-Nearest neighbors

```

# Select and train a model
# TODO: add your code here..
import sklearn.neighbors

#assert False, "TODO: add you instatiation and training of the knn model here.."
k_model = sklearn.neighbors.KNeighborsRegressor(3)

k_model.fit(X,y)

pred = [[22587]]
k_y_pred = k_model.predict(pred)

print(f"\nKnn value = {k_y_pred}")
k_model.score(X,y)
```

Which outputs:



Figur 1.1: Life satisfaction as a function of GDP pr capita

And:

```
    Knn value: 5.766
    k score: 0.85
```

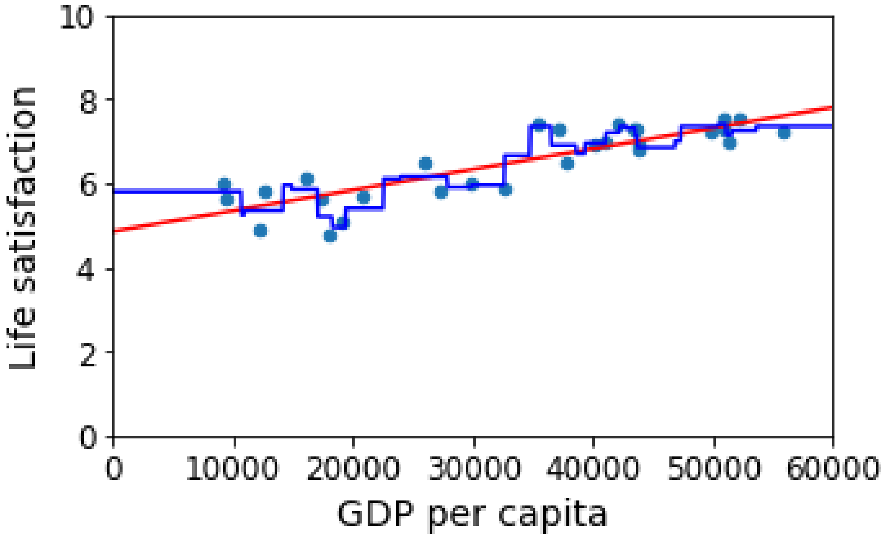
K model: R^2 Linear model: R^2 Can be compared

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

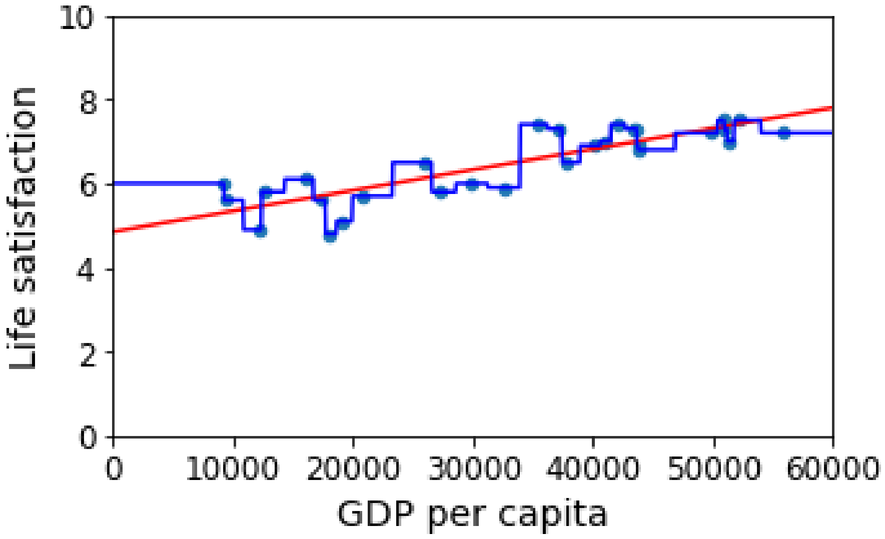
Code:

```
# TODO: add your code here..
# assert False, "TODO: try knn with different k_neighbor params, that is
    ↪ re-instantiate knn, refit and replot.."
k_model = sklearn.neighbors.KNeighborsRegressor(2)

k_model.fit(X,y)

pred = [[22587]]
k_y_pred = k_model.predict(pred)


# from this test M data, predict the y values via the lin.reg. and k-nearest
    ↪ models
y_pred_lin = model.predict(M)
y_pred_knn2 = k_model.predict(M) # ASSUMING the variable name 'knn' of your
    ↪ KNeighborsRegressor

# use plt.plot to plot x-y into the sample_data plot..
plt.plot(m, y_pred_lin, "r")
plt.plot(m, y_pred_knn, "b")

k_model.score(X,y)
```

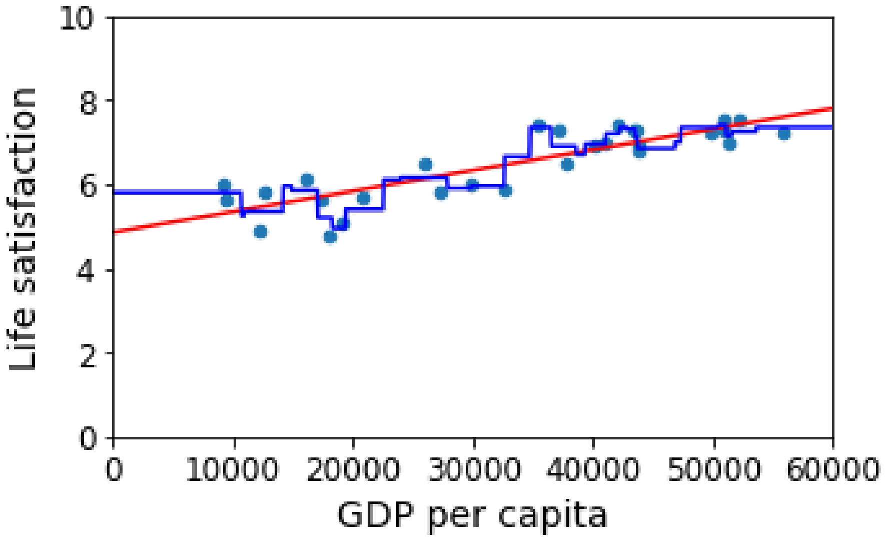
Which outputs for kneighbors: 1,2,3 and 15:



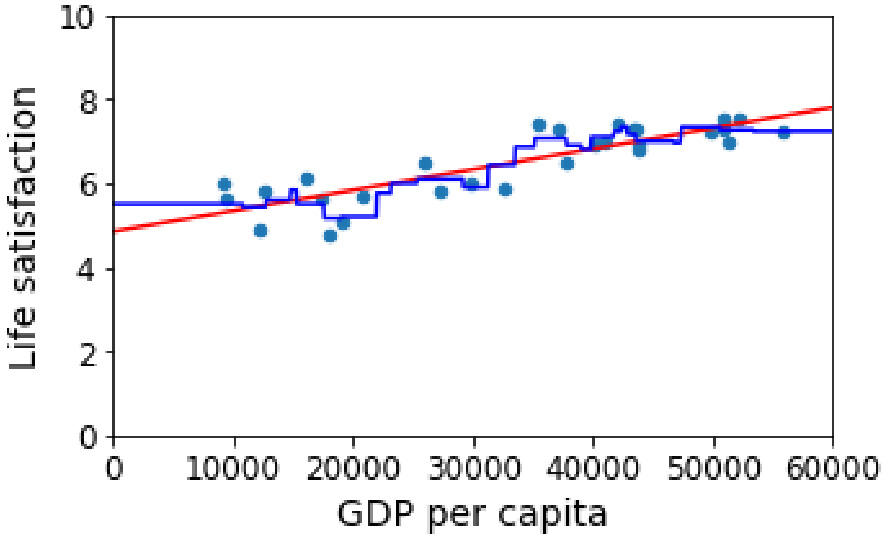
Figur 1.2: k neighbors = 1



Figur 1.3: k neighbors = 2



Figur 1.4: k neighbors = 3



Figur 1.5: k neighbors = 15

```
    k model score: 1 (for k neighbors = 1)
    k model score: 0.909 (for k neighbors = 2)
    k model score: 0.852 (for k neighbors = 3)
    k model score: 0.715 (for k neighbors = 15)
```

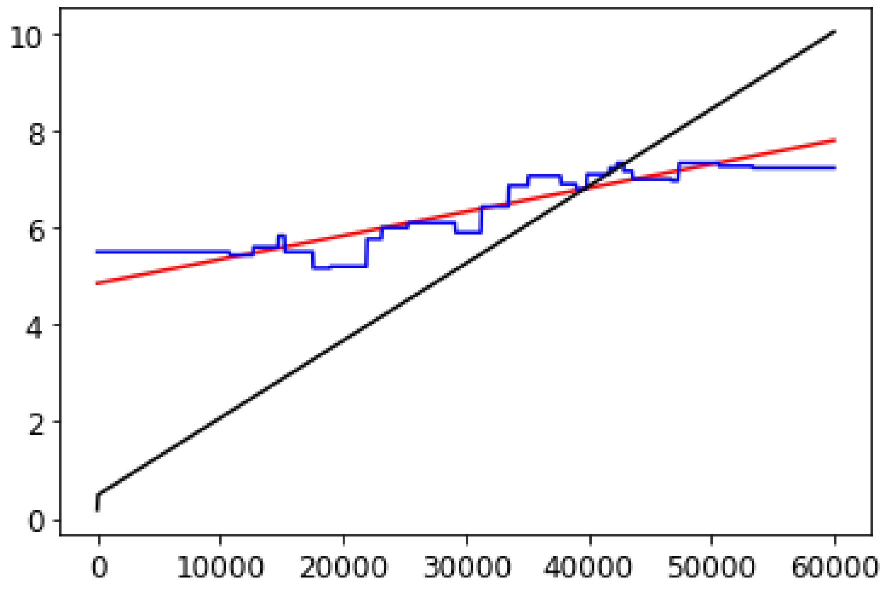
In conclusion, the performance of sklearn.neighbors.KNeighborsRegressor(n) varies significantly depending on the choice of parameter n. For n=1, the model tends to overfit the data, resulting in high variance and poor generalization to unseen data. When n=2 or n=3, the model's performance improves as it begins to capture more general trends in the data, but there might still be some overfitting. However, with n=15, the model achieves a good balance between bias and variance, leading to better generalization and more reliable predictions. Therefore, the selection of n should be carefully considered based on the specific dataset and the desired trade-off between bias and variance.

1.0.4 Qd) Trying out a neural network

Code:

```
    from sklearn.neural_network import MLPRegressor

# Setup MLPRegressor
mlp = MLPRegressor( hidden_layer_sizes=(10,), solver='adam', activation='relu',
    ↪ tol=1E-5, max_iter=100000, verbose=True)
mlp.fit(X, y.ravel())

# lets make a MLP regressor prediction and redo the plots
y_pred_mlp = mlp.predict(M)

plt.plot(m, y_pred_lin, "r")
plt.plot(m, y_pred_knn, "b")
plt.plot(m, y_pred_mlp, "k")

# TODO: add your code here..
#assert False, "TODO: predict value for Cyprus and fetch the score() from the
    ↪ fitting."
pred = [[22875]]
y_pred = mlp.predict(pred)

y_pred, mlp.score(X, y)
```

Output:



Figur 1.6: MLPRegressor comparison with linear and KNN-scores

Throughout 718 iterations the loss was reduced from J = 290434295 to J = 1.590, resulting in a predicted life satisfaction score for Cyprus, and score of:

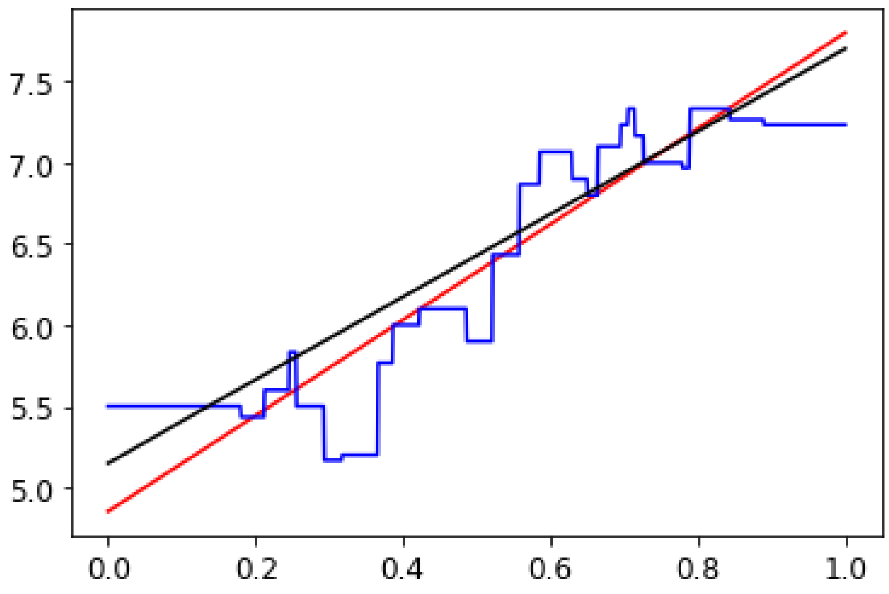
```
Life satisfaction prediction: 4.1
MLP score: -3.67
```

With a GDP predicting value of 22875

1.0.5 [OPTIONAL] Qe) Neural network with pre-scaling

Code:

```
    # TODO: add your code here..
#assert False, "TODO: try prescale data for the MPL...any better?"

from sklearn import preprocessing
import numpy as np

# X transformation
min_max_scaler = preprocessing.MinMaxScaler([0,1])
X_new = min_max_scaler.fit_transform(X)

mlp = MLPRegressor( hidden_layer_sizes=(10,), solver='adam', activation='relu',
    ↪ tol=1E-5, max_iter=100000, verbose=True)
mlp.fit(X_new, y.ravel())

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

# lets make a MLP regressor prediction and redo the plots
y_pred_mlp = mlp.predict(M)

plt.plot(m, y_pred_lin, "r")
plt.plot(m, y_pred_knn, "b")
plt.plot(m, y_pred_mlp, "k")


# TODO: add your code here..
#assert False, "TODO: predict value for Cyprus and fetch the score() from the
    ↪ fitting."
pred = [[0.22587]]
y_pred = mlp.predict(pred)

y_pred, mlp.score(X_new, y)

# With the scaled input, it is much better
```



Figur 1.7: MLPRegressor with [0:1] scaling

After 2792 iterations the loss was reduced from J = 16.53 to J = 0.00973 (much better than the nonscaled version). Since the neurons in the neural net expects/better handles values between [0:1], meaning it doesnt saturate as often/easily as the unscaled version.

This gives a more accurate prediction and score, of:

Output:

```
    Life satisfaction prediction: 5.73
    MLP score: 0.725
```

1.0.6 Intro L01 conclusion

ChatGPT conclusion:

"LinearRegression() is good for simple, straight relationships between features and targets. KNeighborsRegressor() is like a nosy neighbor, it's good for picking up on patterns but can get confused easily with too much noise. MLPRegressor() is like a multi-tool, it can handle complex relationships but might need a lot of tinkering to work just right."

The conclusion that ChatGPT wrote makes a lot of sense, especially MLPregressor with unscaled training input. After scaling, the predictions were much better. The same with KNN, it can easily underfit/overfit, meaning that some tinkering with the kneighbors value is necessary. Linear regression is linear regression - good for data that has a somewhat apparent linear relationship.

2 Module and Classes

2.1 Qa Load and test the 'libitmal' module

The group uses a mixture of IDE's. Here the choosen IDE's are Visual Studio Code and JetBrains DataSpell. With the use of IDE's the setting of the PATH is not needed.

For Question A the provided code snippet was inserted into the Jupyter code-cell and executed a cording to 'libitmal' directory PATH. Here the group have decided to create a shared Git-Depository for the course SWMAL-01 called MLKursus.

```
import sys,os
sys.path.append(R'../ML/MLKursus/UndervisningsGit/GITMAL/')

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

The print statement output was the following:

```
['AssertInRange', 'CheckFloat', 'DToXy', 'GenerateConfusionMatrix', 'GenerateResults',
    ↪ 'InRange', 'Iterable', 'ListToMatrix', 'ListToVector', 'PrintMatrix', 'ResetRandom',
    ↪ 'ShowResult', 'TEST', 'TestAll', 'TestCheckFloat', 'TestPrintMatrix', 'TestVarName',
    ↪ 'VarName', 'XyToD', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
    ↪ '__name__', '__package__', '__spec__', 'ctxlib', 'inf', 'inspect', 'isFloat', 'isList',
    ↪ 'isNumpyArray', 'nan', 'np', 'random', 're', 'sklearn']
C:\Users\45201\OneDrive\Uni\6_Semester\ML\MLKursus\UndervisningsGit\GITMAL\libitmal\utils.py
```

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

For Question B a module was created containing two functions. The module was created within a sub-folder for easier access. Here the sub-folder got called O1 and module O2. The module contains a Python file called 'Test_Module'.

'Test_Module' contains the following code:

```
def test():
   print("Hello this is a test, and you passed")

def AOR(Lenght, Width):
   return Lenght * Width

```

```
def HappyDog():
   print("""
  / \\__
 ( @\\___
 / O
/ (_____/
/_____/ U
   """)
```

Here the executing code for the inclusion of the module looks like the following:

```
import sys,os
sys.path.append(R'C:\Users\45201\OneDrive\Uni\6_Semester\ML\MLKursus\Martin\01')
from 02_Qb import Test_Module as QB

print(dir(QB))
print(QB.__file__)

print()

# Testing function AOR in module
## Function should return area of a square or rectangle
A = 12
B = 8
print(QB.AOR(A,B))

print()

# Tesing function HappyDog
## Function should return a good boi
print(QB.HappyDog())
```

Running the code-cell from the Jupyter Notebook returns the following output:

```
['AOR', 'HappyDog', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
    ↪ '__name__', '__package__', '__spec__', 'test']
C:\Users\45201\OneDrive\Uni\6_Semester\ML\MLKursus\Martin\01\02_Qb\Test_Module.py

96


  / \__
 ( @\___
 / O
/ (_____/
/_____/ U

None
```

Since the output matches the wanted function setup in the 'Test_Module' the inclusion of our module is working.

2.3 Qc How do you 'recompile' a module?

There is a built-in reload function in the "importlib-package. Caling the importlib.reload(#ModuleName) will recall the module and recall the lastest version.

```
## For our created modules it could be called like so:
import importlib

importlib.reload(QB)
```

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

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

Python does not like other programming languages have public and private member. In a function or class the naming can be made more private by highlighting the function names with a double underscore at the beginning and end of the variable name. Example:

```
    def __measurement__(self):
    etc...

# Another way could be:

def __my_Variable():
    etc...
```

Here the underscore is suppose to indicate the "private"instance of the function member.

2.4.2 What is the meaning of ‘self‘ in python classes?

The introduced code snippet will initiate it ’self’.

```
    class MyClass:

    def myfun(self):
        self.myvar = "blah" # NOTE: a per class-instance variable.
        print(f"This is a message inside the class, myvar={self.myvar}.")

myobjectx = MyClass()

myobjectx.myfun()
```

With output:

```
   This is a message inside the class, myvar=blah.
```

The variable is used to access the objects (in this case the class) attributes and methods depending on the setup of the code.

When the object does not inherit the self attribute/the self call making the call:
- myobjectx.myfun()
Will return a TypeError saying the object wanted zero attribute where one was given.
Removing the brackets at the call like so:
myobjectx.myfun
will initiate the defined function myfun.

2.5 Qf Extend the class with a Constructor

2.5.1 Figure a way to declare/define a constructor (CTOR) in a python class

A constructor is can be created to initialize the objects of a class when created. This can be done the following way:

```
    class university:

    # Function init creates the constructor with variables:
    # self.#VariableName = Variable
   def __init__(self, student, major):
        self.student = student
        self.major = major

    def reg(self)
        print(f"{self.student} is currently really well. However it seems he is
            ↪ failing {self.major}")

x = university("Alex","SWMAL-01")
x.reg()
```

With output:

```
    Alex is currently really well. However it seems he is failing SWMAL
```

In the above code snippet the constructor is created within the 'init' function so the rest of the functions can inherit and be used. However, when calling the class, the call needs to include the variables as shown above.

2.5.2 Is there a class destructor in python (DTOR)?

Python does not contain a class destructor as so. If for example a file-handler class was made, and within the class a function was made that closes the file, then a deconstructor would have been made.

2.6 Qg Extend the class with a to-string function

We are asked to serialize a class, to make it printable and have some kind of meaning behind the return value. In python we can use the __str__ built in function to define a return string for a class when called with print, like so:

```
class myclass:
    value1 = None

    def __init__(self, val1):
        self.value1 = val1

    def __str__(self):
        return f"I am a class of type {self.__class__.__name__}, i have the
            ↪ varaible 1 of value: {self.value1}"

```

```python

myclass1 = myclass(10)
myclass2 = myclass(20)

print(myclass1)
print(myclass2)
```

this outputs:

```
I am a class of type myclass, i have the varaible 1 of value: 10
I am a class of type myclass, i have the varaible 1 of value: 20
```

Note that the self.__class__.__name__ is kinda dirty and ugly but functional.

3 Cost Function

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

```python
# TODO..create and print the full matrix

X = np.array([[1,2,3],[4,2,1],[3,8,5],[-9,-1,0]])

print(f"Matrix x:\n {X}")
```

Output:

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

3.0.2 Qb Implement the L^1 and L^2 norms for vectors in python.

Code:

```python
    # TODO: solve Qb...implement the L1, L2 and L2Dot functions...
def L1(vec):
    if len(vec) == 1:
        raise AssertionError("Must be a vector, not a scalar!")

    v0 = 0
    for v in vec:
        if v < 0:
            v = -v
        v0 += v

    # return sum(abs(vec))
    return v0

def L2(vec):
    if vec.ndim != 1:
        raise AssertionError("Vector must be 1-dimensional!")
    v0 = 0
    for v in vec:
        v0 += (v**2)

    v0 = v0**0.5
    # return (sum(vec**2))**0.5
    return v0

def L2Dot(vec):
    if not isinstance(vec, np.ndarray):
        raise ValueError("Wrong input type, must be numpy.ndarray")
    return np.linalg.norm(vec)
```

Output:

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

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

Code:

```

def RMSE(X,y):
    if not X.shape == y.shape:
        raise ValueError(f"The dimensions of inputs are incorrect.\nX has shape:
            ↪ {X.shape}, while y has shape: {y.shape}")

    m = len(X)

    dvar = (1/m)**0.5 * L2(X-y)

    RMSE = dvar
    return RMSE
```

Output:

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

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

```python
def MAE(X,y):
    for i in range(len(X)):
        if isinstance(X[i],np.complex128):
            raise ValueError("Complex number spotted! \nOnly real numbers
             ↪ please!")

    if X.shape != y.shape:
        raise ValueError("X and y must have the same shape!")

    m = len(X)
    print(type(X))

    MAE = 1/m * L1(X-y)

    return MAE
```

Output:

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

3.0.5 Qe Robust Code

This has been implemented in all functions, after having made them without the error-checking

3.0.6 Qf Conclusion

Given the design matrix X at the beginning, we learned that it was important to fully understand the nomenclature behind how it is designed. Meaning that a row vector is "disguised"as a column vector in the design matrix. We had trouble in the beginning, thinking that we actually has to transpose the x(1), x(2)... and ended with a 3x4 matrix, instead of the correct 4x3 matrix.

The first exercise was good at testing how to write simple functions in a low-level way, better understanding the coding aspect.

The MSE/RMSE function gave us a good understanding of what it means, when we want to reduce the loss function, since relating it to linear algebra in terms of relating it to minimizing a distance makes great sense. I cant say that we feel as if we grasp the concept fully, but it makes more and more sense.

Comparing the MAE and RMSE it seems that the RMSE is less robust than MAE, since its value is more influenced to squares and roots.

The errorhandling exercise was fine, since it gave a better understanding of the functions, making sure that the inputs correspond to what we want the function to do. For example, we got to the exercise with the incorrect 3x4 design matrix, which the function used just fine, but gave an incorrect result. But when raising the check for equal dimensionality between h(X) and y, it became clear that the design matrix was incorrect.

All in all, good exercises, though the errorhandling was a bit tedious (but good to learn and use in the future nonetheless).

4 Dummy classifier

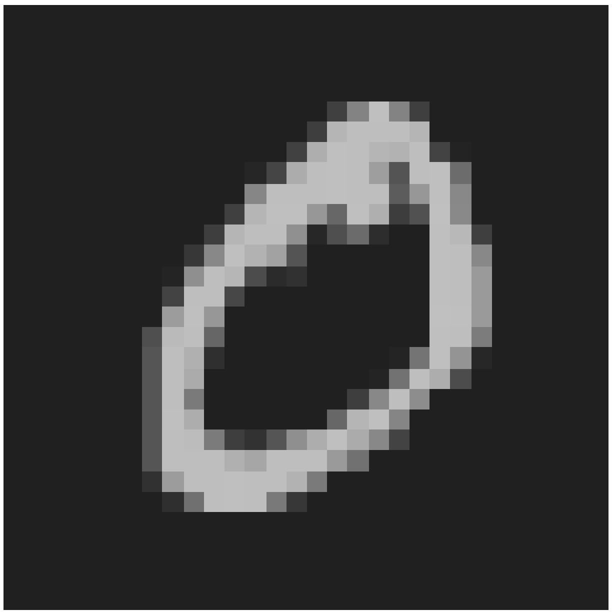
4.1 Qa Load and display the MNIST data

we are asked to build a function that gets the data, and then to plot a single digit of the data:

```
def MNIST_GetDataSet():
    Data, Target = fetch_openml(name="mnist_784",return_X_y=True,
        ↪ as_frame=False)
    Data = Data / 255 #Converts to binary
    return Data,Target
```

then we can create an image from the binary data using the provided MNIST_PlotDigit function called with the data[i] at some index i.

given i = 1 we get an image of a zero:



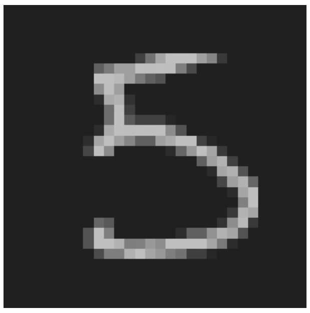
4.2 Qb Add a Stochastic Gradient Decent [SGD] Classifier

we are asked to use the SGD classifier from sklearn our data, and then display a correct number using the plotDigit function.

```
X,y = MNIST_GetDataSet()
```

```python

#y = y.astype(np.uint8)

"""
Below our training and test data is trained on the first 10000 entries and
    ↪ tested on the last 10000.
"""
X_train, X_Test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

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

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train,y_train_5)
Arr = sgd_clf.predict(X_Test)


# Defines a function to read through a boolean array and indicate which
    ↪ location returns a "True" value.
def True_seeker(arr):
  transitions = []
  for i, val in enumerate(arr):
    if val:
      transitions.append(i)
  return transitions
```

here our truth seeker function simply prints the indexes of the predictions that the classifier got correct, just to find a good index to display. As 15 was one of the true numbers we can then display this number.



4.3 Qc Implement a dummy binary classifier

we are asked to build a dummy clacifier that simply returns zeros, in our case, that would mean that the classifies everything as not 5.

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

# Instantiates the DummyClassifier
Dummy_clf = DummyClassifier()

pred = Dummy_clf.predict(X_Test)

accuracy = accuracy_score(Arr,pred)

print("Accuracy of DummyClassifier:", accuracy)
```

this prints:

```
Accuracy of DummyClassifier: 0.9259
```

this is worse then the one from HOLM as expected, but its still quite high as most numbers in our dataset isn't 5. Or in other words, 92% of our dataset is not 5.

4.4 Qd Conclusion

The first assignment was a useful introduction to the openml.org. This exercise showed us that there exist with-in reach training data, should we try and train our machine learning knowledge in our spare time. In the second exercise we used the SGDClassifier to predict either or not the number/picture in our test sample were a five or not. The exercise gave us an hands on experience with setting up and visualising the output of a binary SGDClassifier. In the finale exercise we used the accuracy_score to see what the prediction-score would be if tested on our boolean prediction from the SGDClassifier. Here the prediction was that 92.59% of boolean array was zeroes! In correlation to HOML "Measurring Accuracy Using Cross-Validation"the procentage seemed reasonable due to the way each accuracy is calculated differently.

5 L02 Performance metrics

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

The purpose of this assignment was to create a custom accuracy function and run it on both a "real"classifier and a dummy classifier that only returns zeros.

```python
def MyAccuracy(y_true, y_pred):
    if len(y_true) == 0: #Safeguard against division by zero
        # return an error message if the length of y_true is 0
        print("Error: y_true is empty")
        return 0
    TrueValues = 0
    for y, x, in zip(y_true, y_pred): #zip is just syntactic sugar for pairing
        ↪ the two arrays together
        if y == x:
            TrueValues += 1
    return TrueValues / len(y_true) #This is the accuracy formula as TrueValues
        ↪ contain both TP and TN
```

we then run the function on our test data vs our predictions made by our two classifiers from earlier.

```python
print("MyAccuracy: ", MyAccuracy(y_test_5, clf_Predict))
print("Sklearn accuracy: ", accuracy_score(y_test_5, clf_Predict))

print("MyAccuracy: ", MyAccuracy(y_test_5, Dummy_pred))
print("Sklearn accuracy: ", accuracy_score(y_test_5, Dummy_pred)) # Cirka 9%
    ↪ numbers are 5's
```

which outputs:

```
MyAccuracy: 0.9769
Sklearn accuracy: 0.9769
MyAccuracy: 0.9108
Sklearn accuracy: 0.9108
```

As we can see, the accuracy is the same for both my custom function and the sklearn function. Meaning that my custom function works as intended.

5.2 Qb Implement Precision, Recall and F_1-score and test it on the MNIST data for both the SGD and Dummy classifier models

We have been asked to implement the different types of "scoring". In our case this will be Recall, sensitivity, Precision and F1 score.

```python
def MyPrecision(y_true, y_pred):
    TP = sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp == True) #let's
        ↪ use list comprehension to make the code more readable
    FP = sum(1 for yt, yp in zip(y_true, y_pred) if yt == False and yp == True)

    if TP + FP == 0:
        return 0 #Safeguard against division by zero

    return TP / (TP + FP) #Precision formula



def MyRecall(y_true, y_pred):

    TP = sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp == True)
    FN = sum(1 for yt, yp in zip(y_true, y_pred) if yt == True and yp == False)

    if TP + FN == 0:
        return 0 #Safeguard against division by zero

    return TP / (TP + FN) #Recall formula


def MyF1Score(y_true, y_pred):

    precision = MyPrecision(y_true, y_pred)
    recall = MyRecall(y_true, y_pred)

    if precision + recall == 0:
        return 0 #Safeguard against division by zero

    return 2/((1/recall) + (1/precision))#F1 score formula
```

we also build in safety for when the length is zero by simply returning zero. We could also interrupt the code at this point or use assert.

we then test the different scorings using the same datasets and classifiers as before:

```python
#Test the functions with the SGD classifier

print("MyPrecision: ", MyPrecision(y_test_5, clf_Predict))
print("Sklearn Precision: ", precision_score(y_test_5, clf_Predict))

print("MyRecall: ", MyRecall(y_test_5, clf_Predict))
print("Sklearn Recall: ", recall_score(y_test_5, clf_Predict))

print("MyF1Score: ", MyF1Score(y_test_5, clf_Predict))
print("Sklearn F1Score: ", f1_score(y_test_5, clf_Predict))

#Test the functions with the Dummy classifier
print("\n\n")

print("MyPrecision: ", MyPrecision(y_test_5, Dummy_pred))
print("Sklearn Precision: ", precision_score(y_test_5, Dummy_pred))

print("MyRecall: ", MyRecall(y_test_5, Dummy_pred))
print("Sklearn Recall: ", recall_score(y_test_5, Dummy_pred))

print("MyF1Score: ", MyF1Score(y_test_5, Dummy_pred))
print("Sklearn F1Score: ", f1_score(y_test_5, Dummy_pred))
```

this prints:

```
MyPrecision: 0.9460188933873145
Sklearn Precision: 0.9460188933873145
MyRecall: 0.7858744394618834
Sklearn Recall: 0.7858744394618834
MyF1Score: 0.8585425597060624
Sklearn F1Score: 0.8585425597060624



MyPrecision: 0
Sklearn Precision: 0.0
MyRecall: 0.0
Sklearn Recall: 0.0
MyF1Score: 0
Sklearn F1Score: 0.0
```

As we can see, the precision, recall and F1 score is the same for both my custom function and the sklearn function. Meaning that my custom function works as intended. Precision, recall and F1 score are all 0 for the dummy classifier, as it always predicts 0.

5.3 Qc The Confusion Matrix

Here we are asked to construct a confusion matrix using sklearns own functions using our data. We were also asked what would happen if we input the data in the wrong order.

its quite simply done like so:

```python
from sklearn.metrics import confusion_matrix

M_dummy = confusion_matrix(y_test_5, Dummy_pred)
M_SGD = confusion_matrix(y_test_5, clf_Predict)

print("M_dummy: ", M_dummy)
print("M_SGD: ", M_SGD)
```

this prints:

```
M_dummy: [[9108 0]
          [ 892 0]]
M_SGD: [[9068 40]
        [ 191 701]]
```

The confusion matrix is organized as follows:

[[TN FP]

[FN TP]]

If you mess up the parameters, you will get a transposed version of the confusion matrix. This is because the first parameter is the predicted values and the second parameter is the actual values. If you switch the two parameters, you will get the predicted values as the rows and the actual values as the columns.

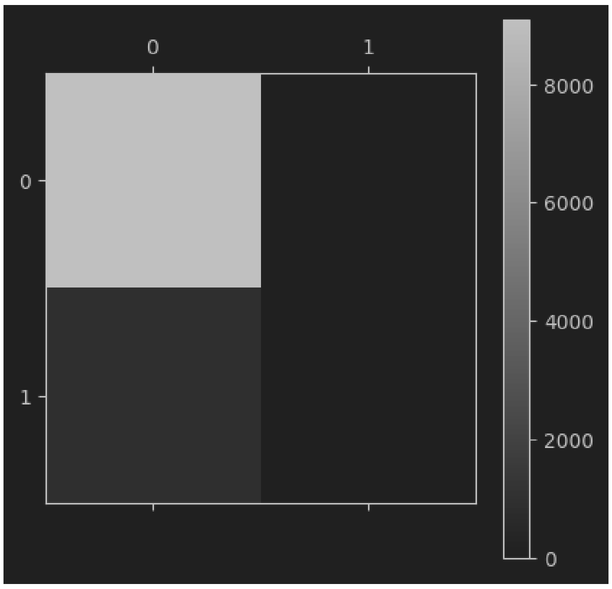
5.4 Qd A Confusion Matrix Heat-map

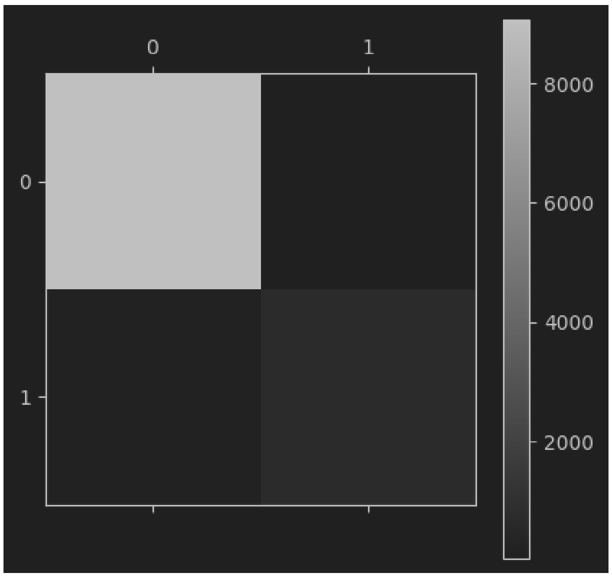
We were asked to create a heatmap of our confusion matrix. We can use matplot lib for this.

```python
import matplotlib.pyplot as plt

plt.matshow(M_dummy, cmap=plt.cm.gray_r) # gray_r is a gray colormap, r stands
    ↪ for reverse, meaning that 0 is white and 1 is black
```

```
plt.colorbar()
plt.show()

plt.matshow(M_SGD, cmap=plt.cm.gray_r)
plt.colorbar()
plt.show()
```

we get the following plots:





as we can see, we more values that are TN then TP as expected since we are only predicting 5s

5.5 Qe Conclusion

Its really important to fully understand why we use certain metrics and how they are calculated. This is important because it allows us to understand the performance of our models (kinda) and how they can be improved. The overall learning outcome of the exercises is that we now have a better understanding of how to calculate and use the accuracy, precision, recall and F1 score metrics. We also have a better understanding of how to use the confusion matrix and how to interpret it. This is important because it allows us to understand the performance of our models and how they can be improved. One note is that all these metrics perform very poorly on more complex datasets, as they are not robust against outliers or rapid change in data.