HW 2

This assignment covers several aspects of Linear Regresstion. **DO NOT ERASE MARKDOWN CELLS AND INSTRUCTIONS IN YOUR HW submission**

- **Q** - QUESTION
- **A** - Where to input your answer

Instructions

Keep the following in mind for all notebooks you develop:

- Structure your notebook.
- Use headings with meaningful levels in Markdown cells, and explain the questions each piece of code is to answer or the reason it is there.
- Make sure your notebook can always be rerun from top to bottom.
- Please start working on this assignment as soon as possible. If you are a beginner in Python this might take a long time. One of the objectives of this assignment is to help you learn python and scikit-learn package.
- Follow README.md for homework submission instructions

Tutorials

- scikit-learn linear model
- train-test-split
- least squares fitting
- Linear Regression
- Seaborn

REGRESSION TASK USING SKLEARN

In jupyter notebook environment, commands starting with the symbol % are magic commands or magic functions. `%%timeit` is one of such function. It basically gives you the speed of execution of certain statement or blocks of codes.

In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
```

Data Get the exploratory data and the folowwing files:
https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data

https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.names

or Use from our 2023Fall/data repository folder

- Link should automatically download the data
- copy them in your HW folder

- If you are using command line: `>> wget https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data`
  - If wget is not working
    - dowload it from link
    - follow steps

Q1 Read the data using pandas, and replace the ??? in the code cell below to accomplish this taks. Note that auto-mpg.data does not have the column headers. use auto-mpg.names file to provide column names to the dataframe.

A1

In [ ]:

```
# Replace ??? with code in the code cell below
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'accelerat
# df = pd.read_csv('../data/auto-mpg.data', names=column_names, na_values = '?', comme
df = pd.read_csv('../data/auto-mpg.data', delim_whitespace=True, names=column_names, r
```

In [ ]:

```
# View head of the data to confirm the correctness of your answer
df.head()
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

Data cleaning and manipulation

Use

Q2 Data cleaning and manipulation:

1. use `pandas.info()` method to find columns with large number of NaN values
2. remove the column with NaN values
3. Check if there are still NaN values in the dataframe using `isna()` method

A2 Replace ??? with code in the code cell below

In [ ]:
```
#1. use pandas.info() method to find columns with large number of NaN values
#???
df.info()

#2. remove the column with NaN values - replace ??? with code
#df.drop(???)
# Print head
#df.?
df.dropna(axis=0, inplace=True)
#3. Check if there are still NaN values in the dataframe using ```isna()``` method - r
#df.???
df.isna()
# drop if any left or replace Nan values
#???
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   year          398 non-null    int64  
 7   origin        398 non-null    int64  
 8   name          398 non-null    object 
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
393	False	False	False	False	False	False	False	False	False
394	False	False	False	False	False	False	False	False	False
395	False	False	False	False	False	False	False	False	False
396	False	False	False	False	False	False	False	False	False
397	False	False	False	False	False	False	False	False	False

392 rows × 9 columns

In [ ]:
```
#Print Tail
df.tail()
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
393	27.0	4	140.0	86.0	2790.0	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52.0	2130.0	24.6	82	2	vw pickup
395	32.0	4	135.0	84.0	2295.0	11.6	82	1	dodge rampage
396	28.0	4	120.0	79.0	2625.0	18.6	82	1	ford ranger
397	31.0	4	119.0	82.0	2720.0	19.4	82	1	chevy s-10

Q3:

1. Convert following columns 'cylinders', 'year', 'origin' to dummy variable using pandas get_dummies() function
2. Do data normalization on real value/continous columns
   - The formula for normalization is: (Col_value- Mean of the col)/ Standard Deviation of the col

A3 Replace ??? with code in the code cell below

In [ ]:

```
# 1. Convert following columns 'cylinders', 'year', 'origin'  to dummy variable using
cols = ['cylinders', 'year', 'origin']
df_dummies = pd.get_dummies(df, columns=cols, prefix=cols, prefix_sep='-')

#show the head
df_dummies.head()
# 2. Do data normalization on real value/continous columns
#realcols = ???
realcols = ['mpg', 'displacement', 'horsepower', 'weight', 'acceleration']
#for col in realcols:
  #mean = ??
  #std = ??
  #df[col] = ???
for col in realcols:
    mean = df[col].mean()
    std = df[col].std()
    df[col] = (df[col] - mean) / std
```

Regression Task

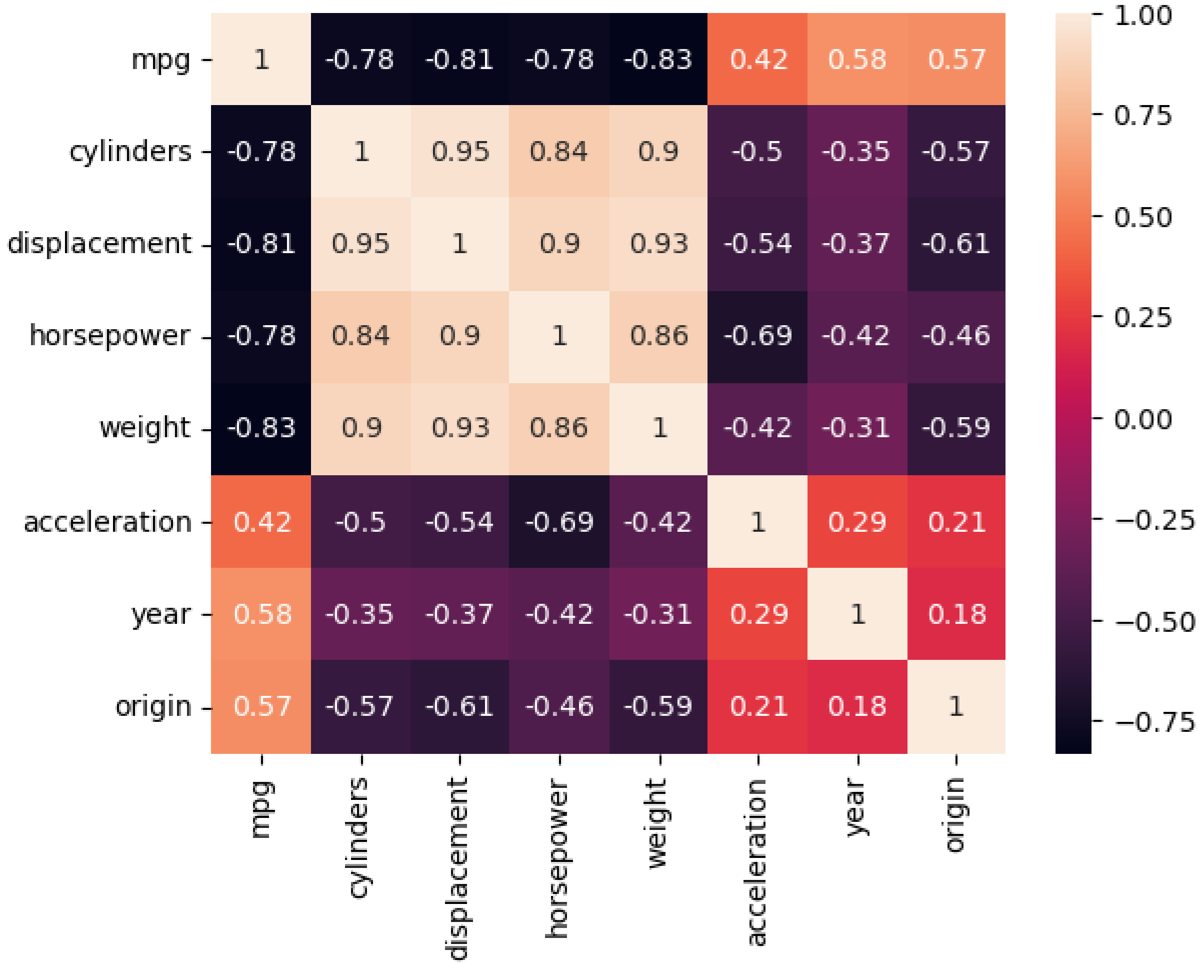
Given all the information we will try to predict mpg - miles per gallon. The First step toward predicting the mpg from the dataset is to find the correlation between the columns/features of the dataset.

Q4

1. Use heatmap chart from seaborn library to findout the correlation between the columns.
2. Which of the columns is mostly related to mpg column and why?

In [ ]:
```python
# A4 code goes below
#sns.heatmap(???, ???, ???,)
#sns.heatmap(df.drop(columns=['name']))
df_no_name = df.drop(columns=['name'])
bool_cols = df_no_name.select_dtypes(include='bool').columns
df_no_name[bool_cols] = df_no_name[bool_cols].astype(int)
sns.heatmap(df_no_name.corr(), annot=True)
```

Out[ ]:
```
<Axes: >
```



A4

I argue that weight is mostly related to mpg. As weight decreased, mpg increased. This would make sense, as cars became less heavy, they become more fuel efficient as there is less weight to move.

Q5

1. Draw a lineplot or scattered plot between mpg and your answer from the above cell.
2. Use pairplot from sns to plot our data frame df for better understanding of your selection
   - NOTE: 2. should inform 3.
3. Choose a set of columns/ features based on pairplot and heatmap for the mpg prediction.

- Justify your answer using some explanation from the heatmap and pairplot graph formulated from the dataset.

A5 For 1. and 2. replace ??? with code in the code cell below.

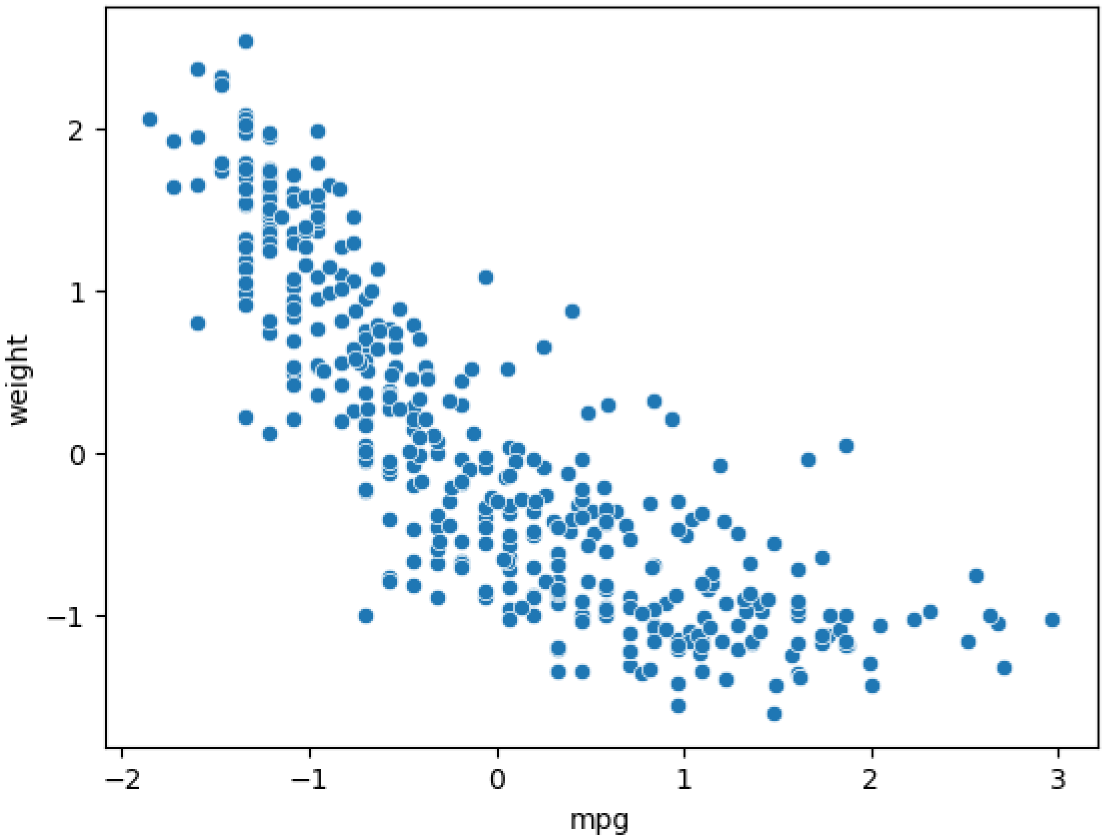
In [ ]:

```
# 1. Draw a lineplot or scattered plot between mpg and your answer from the above cell
#sns.scatterplot(???)
sns.scatterplot(data=df, x='mpg', y='weight')
# 2. Use pairplot from sns to plot our data frame df for better understanding of your
#sns.pairplot(???)
sns.pairplot(data=df)
```

```
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hunter\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x28bc3f55050>
```


weight
mpg
2
1
0
−1
−2
−1
0
1
2
3

A5

I would choose the weight, horsepower, displacement, and cylinders columns for the prediction. On the heat map, each of these have a very strong relationship with how mpg is affected. On the heatmap, each have a strong negative correlation. On the pairplot, it can be observed that as each of these features decrease, mpg increases.

Q6 Data Visualization:

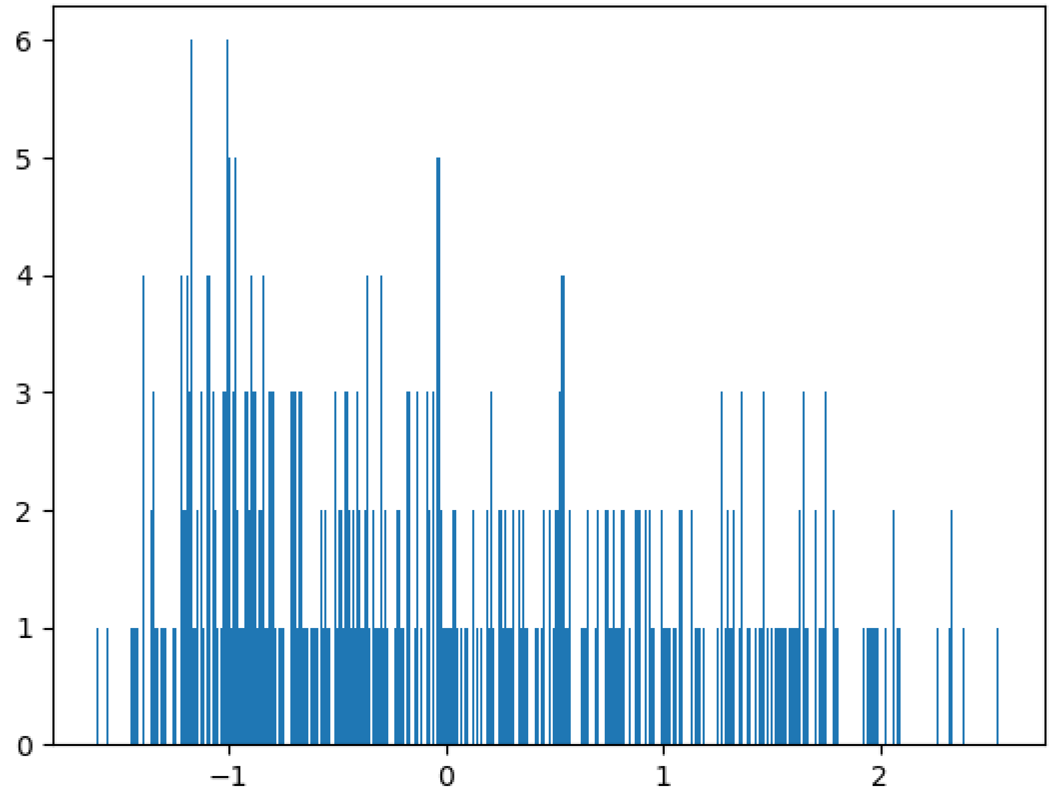
1. Now, create a histogram which represents number items with per cylinder class

A6 Replace ??? with code in the code cell below

In [ ]:

```
import matplotlib.pyplot as plt
# 3 to 8 cylinders
#plt.?
#plt.show()
```

```
plt.hist(df['weight'],bins=len(df))
plt.show()
```



Data Preparation

Q7 Assign mpg column value to y and rest columns to x, remember x shouldn't have mpg

A7 Replace ??? with code in the code cell below

In [ ]:
```
#y = df.???
#df.drop(???)
#x = df.???

y = df['mpg'].values
#print(y)
df.drop(columns=['mpg', 'name'], inplace=True)
x = df.values
```

Q8 Use train_test_split to split the data set as train:test=(80%:20%) ratio.

A8 Replace ??? with code in the code cell below

In [ ]:
```
from sklearn.model_selection import train_test_split

#xtrain, xtest, ytrain, ytest = train_test_split(?????)
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.20, random_state=42)
# View the shape of your data set
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape
```

Out[ ]: ((313, 7), (79, 7), (313,), (79,))

Q9 Follow examples from references given in the top of this notebook

- Note:Use linear model to fit regression line and plot
- Our linear model will be of following type
- Y = b + coef0*x0* + *coef1*x1+coef2*x2+...........

A9: Replace ??? with code in the code cell below

In [ ]:
```
from sklearn import linear_model

#reg = linear_model.???
#reg.fit(??,??)
reg = linear_model.LinearRegression()
reg.fit(xtrain, ytrain)
#Now view the coefficient use .coef_ and shape of .coef_
print(reg.coef_)
print(reg.intercept_)
print(reg.coef_.shape[0])
```

```
[-0.04430346  0.20256687 -0.10505166 -0.66838015  0.01341433  0.09836539
  0.20672076]
-7.550378129756388
7
```

Q10 Relates to the code in the cell below. Why the printed values the same?

In [ ]:
```
# Now if you view
print(f'{reg.coef_.shape[0]},{xtrain.shape[1]}, ', f'are equal? {reg.coef_.shape[0]==x
```

```
7,7,  are equal? True
```

A10 The number of coefficients represent the number of features in the model. xtrain.shape[1] shows the number of columns, which is the total number of features.

Model Scoring

In [ ]:
```
# Model Score
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(xtrain, ytrain)
reg.score(xtest,ytest)

# Calculate the score on train and test sets
# Your code goes below
reg.score(xtrain,ytrain), reg.score(xtest,ytest)
```

Out[ ]: (0.826001578671067, 0.7901500386760347)

Q11 Each of the sklearn models have different model evaluations core value.

- LinearRegression documentation
- More on model_evaluation

Explain what's the meaning of reg.score return value in this notebook.

A11 After having fit the data, the reg.score() method returns the r2score of the input data set. It returns the R2 score, or "coefficient of determination" defined as (1 - u/v), where u is the residual sum of squares, and v is the total sum of squares.

In [ ]:
```
# A custom function to calculate r2 score
# Details on the custom scorers: https://scikit-learn.org/stable/modules/model_evaluat

def r2score_(ytrue, ypred):
  rss = ((ytrue - ypred)**2).sum()
  tss = ((ytrue - ytrue.mean()) ** 2).sum()
  r2 = 1 - rss/tss
  return r2

# Now do prediction on xtrain and xtest and check your r2 score by printing score valu
trainpredict = reg.predict(xtrain)
testpredict = reg.predict(xtest)

print(r2score_(ytrain, trainpredict), r2score_(ytest, testpredict))
```

```
0.826001578671067 0.7901500386760347
```

One way of achieving linear regression is by minimizing the error between actual y and predicted y. The method is known as least square method. We will make our custom least square optimize to calculate model parameters that minimizes output error.

Q12 Write a function which takes weights(or params), x and y and do following

- 1. calculate dot product between x and params , which is ypredicted
- 1. calculate difference between actual y and ypredicted
- 1. return the difference

A12 complete the code below

In [ ]:
```
import scipy.optimize as optimization
from sklearn.metrics import r2_score

def constraint(params, x, y):
  ypred = x@params
  return y-ypred

# Our initial params is a vector of size equal to dimension of x, or you can say numbe
# You can create zeros vector using np.zeros(size)

# complete code
params = np.zeros(x.shape[1])


# Now study the documentation and complete following code
#params, _ = optimization.leastsq(???, ????, ????)
params, _ = optimization.leastsq(constraint, params, args=(xtrain, ytrain))

# Now we have parameter or weight we can now create our model
model = lambda x:np.dot(x,params)
```

```python
# Now predict ytrain using model and see first 5 predicted and actual values
#ypred_train = model(????)
ypred_train = model(xtrain)
# see first 5 predicted values
#print(????)
print(ypred_train[0:5])
# see first 5 actual values
#print(????)
print(ytrain[0:5])

# Now predict ytest using model and see first 5 predicted and actual values
ypred_test = model(xtest)
print(ypred_test[0:5])
print(ytest[0:5])

# Now use custom made r2score calculator to calculate r2 score on both train and test
print(r2score_(ytest, ypred_test), r2score_(ytrain, ypred_train))

# Now use sklearn build-in r2score calculator to calculate r2 score on both train and
#print(??), print(??)
print(r2_score(ytest, ypred_test), r2_score(ytrain, ypred_train))
```

```
[-0.49639183  0.44953015 -0.2601641  -1.32019369  0.92056451]
[-0.62087299  0.19911341 -0.69774672 -1.08211534  1.99283366]
[0.74935878 0.25614133 1.31487668 0.64477985 0.46189283]
[ 0.32723628 -0.23650437  1.62127732  0.32723628  0.45535916]
0.69959492762219 0.7462842501753547
0.69959492762219 0.7462842501753547
```