Exercises for Introduction to Data Science in Python

Prof. Dr. Markus Löcher

05/28/2023

Table of contents

Preface

This is a collection of exercises that accompany the python workshop.

1 Lists, Loops, Conditions

In this lab we will get to know and become experts in:

1. Lists
    - DataCamp, Introduction to Python, Chap 2
2. Loops
    - DataCamp, Intermediate Python, Chap 4
3. Conditions
    - DataCamp, Intermediate Python, Chap 3

1.0.1 Manipulating lists of lists

The following list of lists contains names of sections in a house and their area.

1. Extract the area corresponding to kitchen
2. String Tasks:
    - Extract the first letters of each string
    - Capitalize all strings
    - Replace all occurrences of "room" with "rm"
    - count the number of "l" in "hallway"
3. Insert a "home office" with area 10.75 after living room
4. Append the total area to the end of the list
5. **Boolean** operations:
    - Generate one True and one False by comparing areas
    - Generate one True and one False by comparing names

```
house = [['hallway', 11.25],
 ['kitchen', 18.0],
 ['living room', 20.0],
 ['bedroom', 10.75],
 ['bathroom', 9.5]]
```

1.0.2 Automation by iterating

for loops are a powerful way of automating MANY otherwise tedious tasks that repeat.

1. Repeat the tasks 2 and 4 from above by using a for loop
    - using `enumerate`
    - using `range`
2. Create two separates new lists which contain only the names and areas separately
3. Clever Carl: Compute

$$\sum_{i=1}^{100} i$$

```
list(range(5))
```

```
[0, 1, 2, 3, 4]
```

1.0.3 Conditions

1. Find the **max** of the areas by using `if` inside a for loop
2. Print those elements of the list with
    - area > 15
    - strings that contain "room" (or "rm" after your substitution)

2 Functions, Dictionaries

In this lab we will get to know and become experts in:

1. Functions
    - DataCamp, Introduction to Python, Chap 3
2. Dictionaries
    - DataCamp, Intermediate Python, Chap 2
3. Introduction to numpy
    - DataCamp, Introduction to Python, Chap 4

2.0.1 Functions

Functions are essential building blocks to **reuse code** and to **modularize code**.

We have already seen and used many **built-in functions/methods** such as `print()`, `len()`, `max()`, `round()`, `index()`, `capitalize()`, etc..

```
areas = [11.25, 18.0, 20.0, 10.75, 10.75, 9.5]
print(max(areas))
print(len(areas))
print(round(10.75,1))
print(areas.index(18.0))
```

```
20.0
6
10.8
1
```

But of course we want to define our own functions as well ! As a rule of thumb, if you anticipate needing to repeat the same or very similar code more than once, it may be worth writing a reusable function. Functions can also help make your code more readable by giving a name to a group of Python statements.

For example, we computed the BMI previously as follows:

```
height = 1.79
weight = 68.7
bmi = weight/height**2
print(bmi)
```

```
21.44127836209856
```

Functions are declared with the `def` keyword. A function contains a block of code with an optional use of the `return` keyword:

```
def compute_bmi(height, weight):
    return weight/height**2

compute_bmi(1.79, 68.7)
```

```
21.44127836209856
```

Each function can have *positional* arguments and *keyword* arguments. Keyword arguments are most commonly used to specify default values or optional arguments. For example:

```
def compute_bmi(height, weight, ndigits=2):
    return round(weight/height**2, ndigits)

print(compute_bmi(1.79, 68.7))
print(compute_bmi(1.79, 68.7,4))
```

```
21.44
21.4413
```

2.0.1.1 Multiple Return Values

are easily possible in python:

```
def compute_bmi(height, weight, ndigits=2):
    bmi = round(weight/height**2, ndigits)
    #https://www.cdc.gov/healthyweight/assessing/index.html#:~:text=If%20your%20BMI%20is%2
    if bmi < 18.5:
```

```
        status="underweight"
    elif bmi <= 24.9:
        status="healthy"
    elif bmi <= 29.9:
        status="underweight"
    elif bmi >= 30:#note that a simple else would suffice here!
        status="obese"
    return bmi, status

print(compute_bmi(1.79, 68.7))
print(compute_bmi(1.79, 55))
```

```
(21.44, 'healthy')
(17.17, 'underweight')
```

Recall from the previous lab how we

1. found the largest room,
2. computed the sum of integers from 1 to 100

```
#find the maximum area:
areas = [11.25, 18.0, 20.0, 10.75, 10.75, 9.5]
currentMax = areas[0] # initialize to the first area seen

for a in areas:
  if a > currentMax:
    currentMax = a

print("The max is:", currentMax)
```

```
The max is: 20.0
```

```
#Clever IDB students: Compute the sum from 1 to 100:
Total =0

for i in range(101):#strictly speaking we are adding the first  0
  Total = Total + i
  #Total += i

print(Total)
```

2.0.1.2 Tasks

Write your own function

1. to find the min and max of a list
2. to compute the Gauss sum with defaukt values $m = 1, n = 100$

$$\sum_{i=m}^{n} i$$

2.0.1.3 Namespaces and Scope

Functions seem straightforward. But one of the more confusing aspects in the beginning is the concept that we can have **multiple instances** of the same variable!

Functions can access variables created inside the function as well as those outside the function in higher (or even global) scopes. An alternative and more descriptive name describing a variable scope in Python is a *namespace*. Any variables that are assigned within a function by default are assigned to the local namespace. The local namespace is created when the function is called and is immediately populated by the function's arguments. After the function is finished, the local namespace is destroyed.

Examples:

```
height = 1.79
weight = 68.7
bmi = weight/height**2
#print("height, weight, bmi OUTSIDE the function:",height, weight,bmi)

def compute_bmi(h, w):
    height = h
    weight = w
    bmi = round(weight/height**2,2)
    status="healthy"
    print("height, weight, bmi INSIDE the function:",height, weight,bmi)
    print("status:", status)
    return bmi

compute_bmi(1.55, 50)

print("height, weight, bmi OUTSIDE the function:",height, weight,bmi)
#print(status)
```

```
height, weight, bmi INSIDE the function: 1.55 50 20.81
status: healthy
height, weight, bmi OUTSIDE the function: 1.79 68.7 21.44127836209856
```

2.0.2 Dictionaries

A dictionary is basically a **lookup table**. It stores a collection of key-value pairs, where key and value are Python objects. Each key is associated with a value so that a value can be conveniently retrieved, inserted, modified, or deleted given a particular key.

The dictionary or `dict` may be the most important built-in Python data structure. In other programming languages, dictionaries are sometimes called *hash maps* or *associative arrays*.

```
#This was the house defined as a list of lists:
house = [['hallway', 11.25],
 ['kitchen', 18.0],
 ['living room', 20.0],
 ['bedroom', 10.75],
 ['bathroom', 9.5]]

#Remember all the disadvantages of accessing elements

#Better as a lookup table:
house = {'hallway': 11.25,
    'kitchen': 18.0,
    'living room': 20.0,
    'bedroom': 10.75,
    'bathroom': 9.5}
```

```
europe = {'spain':'madrid', 'france' : 'paris'}
print(europe["spain"])
print("france" in europe)
print("paris" in europe)#only checks the keys!
europe["germany"] = "berlin"
print(europe.keys())
print(europe.values())
```

```
madrid
True
False
dict_keys(['spain', 'france', 'germany'])
dict_values(['madrid', 'paris', 'berlin'])
```

If you need to iterate over both the keys and values, you can use the `items` method to iterate over the keys and values as 2-tuples:

```
#print(list(europe.items()))

for country, capital in europe.items():
    print(capital, "is the capital of", country)
```

```
madrid is the capital of spain
paris is the capital of france
berlin is the capital of germany
```

Note: You can use integers as keys as well. However -unlike in lists- one should not think of them as positional indices!

```
#Assume you have a basement:
house[0] = 21.5
house
```

```
{'hallway': 11.25,
 'kitchen': 18.0,
 'living room': 20.0,
 'bedroom': 10.75,
 'bathroom': 9.5,
 0: 21.5}
```

```
#And there is a difference between the string and the integer index!
house["0"] = 30.5
house
```

```
{'hallway': 11.25,
 'kitchen': 18.0,
 'living room': 20.0,
 'bedroom': 10.75,
 'bathroom': 9.5,
 0: 21.5}
```

Categorize a list of words by their first letters as a dictionary of lists:

```
words = ["apple", "bat", "bar", "atom", "book"]

by_letter = {}

for word in words:
     letter = word[0]
     if letter not in by_letter:
        by_letter[letter] = [word]
     else:
         by_letter[letter].append(word)
```

```
{'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

2.0.2.1 Tasks

1. Find the maximum of the areas of the houses
2. Remove the two last entries.
3. Write a function named word_count that takes a string as input and returns a dictionary with each word in the string as a key and the number of times it appears as the value.

2.0.3 Introduction to numpy

NumPy, short for Numerical Python, is one of the most important foundational packages for numerical computing in Python.

1. Vectorized, fast mathematical operations.
2. Key features of NumPy is its N-dimensional array object, or ndarray

```
height = [1.79, 1.85, 1.95, 1.55]
weight = [70, 80, 85, 65]

#bmi = weight/height**2
```

```
import numpy as np

height = np.array([1.79, 1.85, 1.95, 1.55])
weight = np.array([70, 80, 85, 65])

bmi = weight/height**2
np.round(bmi,2)
```

```
array([21.84700852, 23.37472608, 22.35371466, 27.05515088])
```

3 Numpy Arrays, Randomness

In this lab we will get to know and become experts in:

1. Numpy Arrays
    - Slicing and Accessing
    - Properly using axis
2. Random Data Generation
    - Random integers, permutations and sampling
3. Simulating Probabilistic Events
    -

3.0.1 Numpy Arrays

3.0.1.1 Tasks

1. Generate a sequence from 1 to 64
2. Print every other element
3. Using Boolean indexing: print only those numbers that are greater than 10
4. Reshape into a $8x8$ matrix and print its "shape"
5. Compute the colum and row sums

3.0.2 Random Data Generation

1. "Flip a fair coin" 20 times and save into an array. Note that instead of using "heads/tails" you should "code" the outcome as 0/1.
2. Randomly "draw" 2 integers without replacement from the sequence 1-5. Repeat this process 30 times and store the results in an array.
3. Compute the counts

3.0.3 Simulating Probabilistic Events

1. **Overbooking flights**: airlines
2. **Home Office** days: planning office capacities and minimizing social isolation

4 Probabilistic Events

More Simulations of Probabilistic Events

```
import numpy as np
from numpy.random import default_rng
```

4.0.1 Simulating Probabilistic Events

1. **Biased Coin**: Simulate 365 days with a $p = \frac{1}{4}$ chance of being sunny (=1). Hint: exploit the fact that p is a fraction!
2. **Birthday problem** Change the "birthday code" into a function with "n = number of people in a room" as an argument. (What other arguments might be useful?) Execute this function for $n = 10, 25, 50$.
3. **Overbooking flights**: Imagine an airline sold 105 tickets on a flight with 100 seats. Assuming there is a 10% no-show probability per passenger, "compute" (simulate) the probability that the airline will need to pay someone to not board.

5 Pandas

All about pandas

```
import numpy as np
import pandas as pd
!pip install gapminder
from gapminder import gapminder
```

```
height = np.array([1.79, 1.85, 1.95, 1.55])
weight = np.array([70, 80, 85, 65])
hw = np.array([height, weight]).transpose()

hw
```

```
array([[ 1.79, 70.  ],
       [ 1.85, 80.  ],
       [ 1.95, 85.  ],
       [ 1.55, 65.  ]])
```

```
df = pd.DataFrame(hw , columns = ["height", "weight"])
print(df)
```

```
   height  weight
0    1.79    70.0
1    1.85    80.0
2    1.95    85.0
3    1.55    65.0
```

```
df = pd.DataFrame(hw , columns = ["height", "weight"],
                  index = ["Peter", "Matilda", "Bee", "Bee"])
print(df)
```

```
         height  weight
Peter      1.79    70.0
Matilda    1.85    80.0
Bee        1.95    85.0
Bee        1.55    65.0
```

Can you extract:

0. All weights
1. Peter's height
2. Bee's full info
3. the average height
4. get all persons with height greater than 180cm

5.0.1 Gapminder

```
gapminder.head()
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

Tasks

- find the unique years
- get all rows with year 1952
- get all rows from 1952:1962
- get all rows from Afghanistan to Albania

6 Titanic

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#pd.options.mode.chained_assignment = None # disable chained assignment warning
import seaborn as sns
from scipy.stats import norm
```

```
from datetime import datetime
datetime.today().timetuple().tm_yday
```

285

6.1 Explore the Titanic Data

```
titanic = sns.load_dataset('titanic')
titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	dec
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Na
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Na
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Na

```
titanic[['survived']].mean()
```

```
survived    0.383838
dtype: float64
```

6.1.1 Task

We have seen a strong dependence of the *outcome* on the two "variables"/"features"/"regressors" *pclass* and *sex*. The natural question is whether there could be more factors "correlated with"/"influencing"/"affecting" Survival.

1. Does the port of embarkment matter ?

   - (**MC**) What is the distribution (counts) of embarkment? (Hint: look at `pd.value_counts`)
     - **A** 168, 77, 644
     - **B** 158, 80, 636
     - **C** 170, 75, 639
     - **D** 164, 79, 667
   - (**MC**) What are the survival rates for *Southampton* as a function of `pclass`?
     - **A** 0.54, 0.42, 0.17
     - **B** 0.62, 0.39, 0.15
     - **C** 0.58, 0.46, 0.19
     - **D** 0.56, 0.37, 0.21
   - Do the survival rates "look" different from *Cherbourg* ?
   - How would you make sure that the observed differences are not due to chance ?

2. Does the *fare* paid matter ?

   - How would you quantify/visualize this ?
   - What is the fundamental difference between the previous relationship of two variables ?
   - Have you heard of the terms *confounding* or *confounders* or *marginal dependence* versus *conditional dependence* ?
   - Discuss dependencies among the features. Revisit the port of embarkment question in this light !

3. Does *age* matter ?

   - (**MC**) What is the survival rate for passengers below the age of 18?
     - **A** 0.47
     - **B** 0.74
     - **C** 0.54
     - **D** 0.45
   - (**MC**) What are the survival rates for passengers below the age of 18 stratified by pclass?
     - **A** 0.91, 0.87, 0.36
     - **B** 0.93, 0.88, 0.38
     - **C** 0.95, 0.93, 0.37

  - **D** 0.92, 0.91, 0.37

- How would you make sure that the observed differences are not due to chance ?

7 Missing Data

More pandas but this time on a real data set, namely the kaggle Housing Data which you can read directly from Google Drive

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#!pip install gapminder
#from gapminder import gapminder
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#read in the data
rootPath = "/content/drive/MyDrive/"#same for all of you
loecherPath = "Teaching/SS2023/IntroCoding/datasets/"
df = pd.read_csv(rootPath + loecherPath + "train.csv")
#df = pd.read_csv('/content/drive/MyDrive/Teaching/SS2023/IntroCoding/datasets/train.csv')
```

```
#or
url ="https://drive.google.com/file/d/1hzvcubf2B8PKtjG4OAcytQKwOlESkBvW/view?usp=sharing"
url='https://drive.google.com/uc?id=' + url.split('/')[-2]
df = pd.read_csv(url)
df.head()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
```

```
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
```

```
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

```
df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilit
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPu
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPu
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPu
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPu
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPu

7.0.1 Tasks:

1. Identify the columns with missing values (Hint: use the `any` function) and read up their description on the kaggle site
2. Replace missing values with "appropriate" values, as follows:

- for "categorical" data (e.g. strings) use the most frequent value (`mode`)
- for numerical data: plot a histogram and look at the distribution. For rather symmetric looking data, choose the mean, otherwise the median.
- for "time" variables such as year: find another year variable as a proxy (Hint: read up on the combine_first function)

3. Find those columns with fewer than 8 unique values (Hint: use the pandas method `nunique()`)

- Create 2 insightful boxplots: SalePrice versus YrSold or MSZoning. Decide if a log scale would be more discerning.
- Use `groupby`to compute the boxes, i.e. the lower and upper quartiles. (Hint: use the numpy or pandas method `quantile`)
- Then compute the whiskers
- And find the outliers

8 Regression, seaborn

Correlation and Regression as well as a quick exploration of the seaborn visualization capabilities

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from numpy.random import default_rng
#!pip install gapminder
#from gapminder import gapminder
```

```
#new library
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

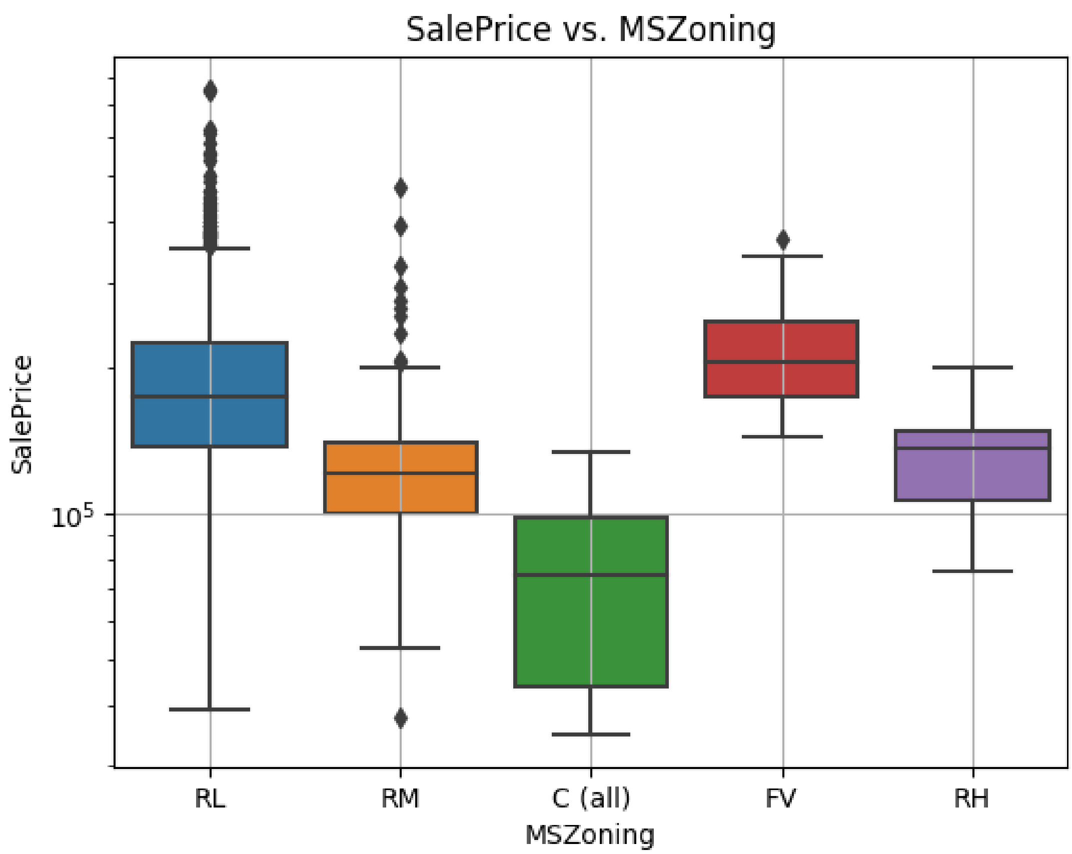
kaggle Housing Data

```
#or
url ="https://drive.google.com/file/d/1hzvcubf2B8PKtjG40AcytQKw0lESkBvW/view?usp=sharing"
url='https://drive.google.com/uc?id=' + url.split('/')[-2]
df = pd.read_csv(url)
df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilit
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPu
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPu
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPu
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPu
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPu

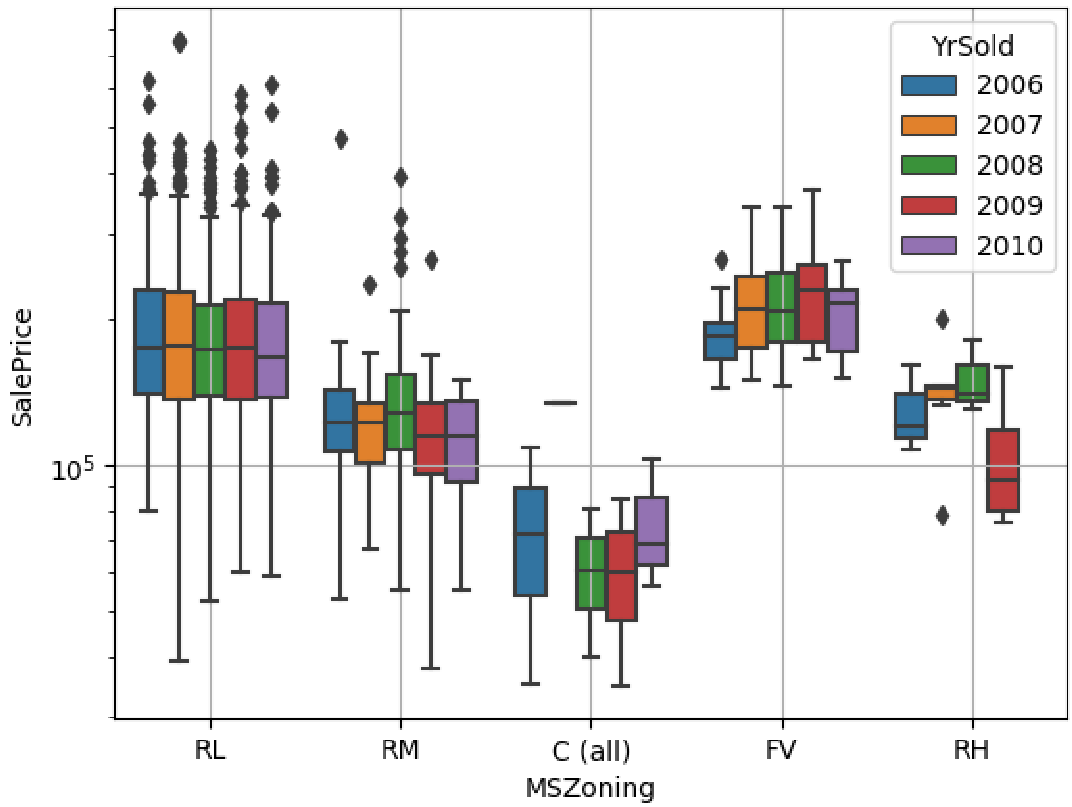
8.1 Seaborn Graphs

```
sns.boxplot(df, y = "SalePrice", x = "MSZoning");
plt.yscale("log");plt.grid();
plt.title("SalePrice vs. MSZoning");
```



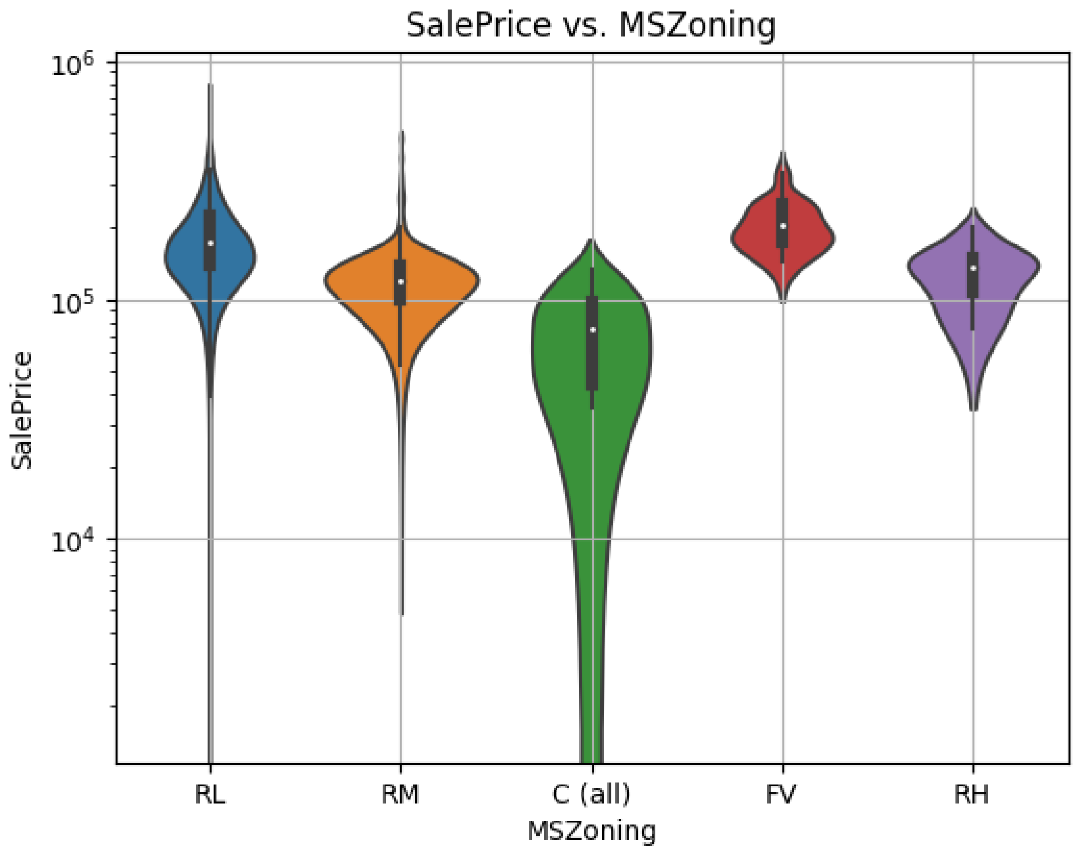
8.1.0.1 Multiple Groups

```
sns.boxplot(df, y = "SalePrice", x = "MSZoning", hue = "YrSold");
plt.yscale("log");
plt.grid();
plt.title("SalePrice vs. MSZoning");
```



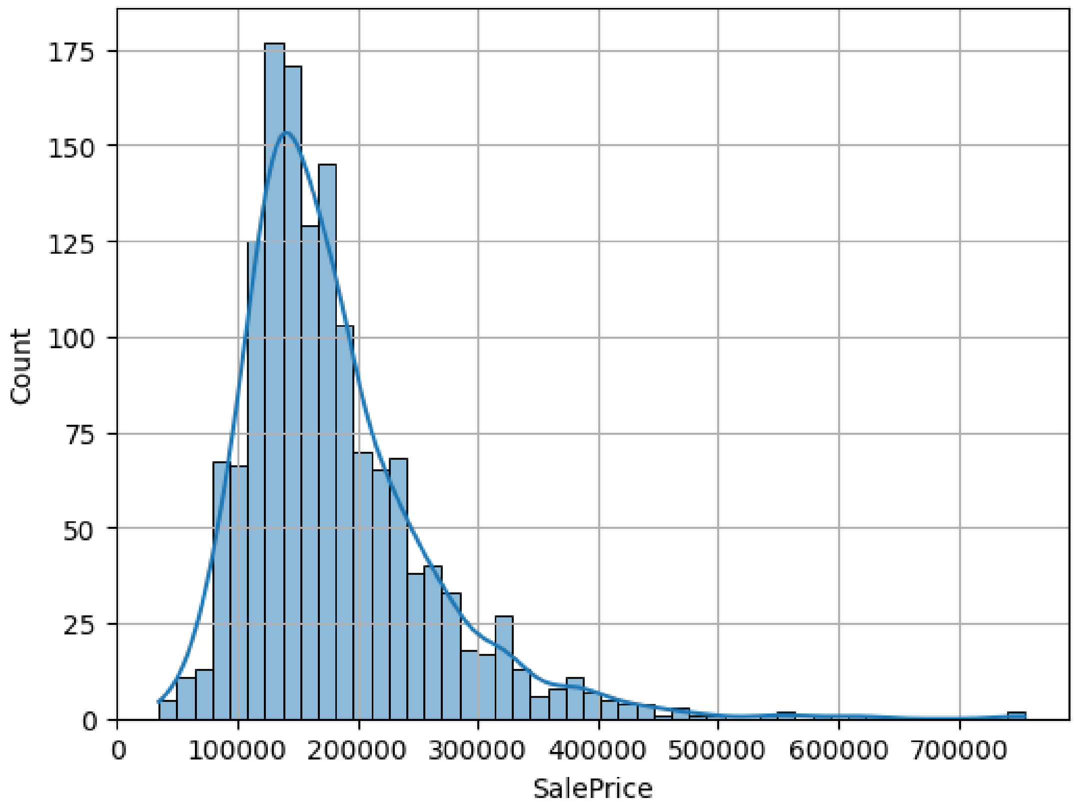
8.2 Violin Plots

```
sns.violinplot(df, y = "SalePrice", x = "MSZoning");
plt.yscale("log");plt.grid();
plt.title("SalePrice vs. MSZoning");
```

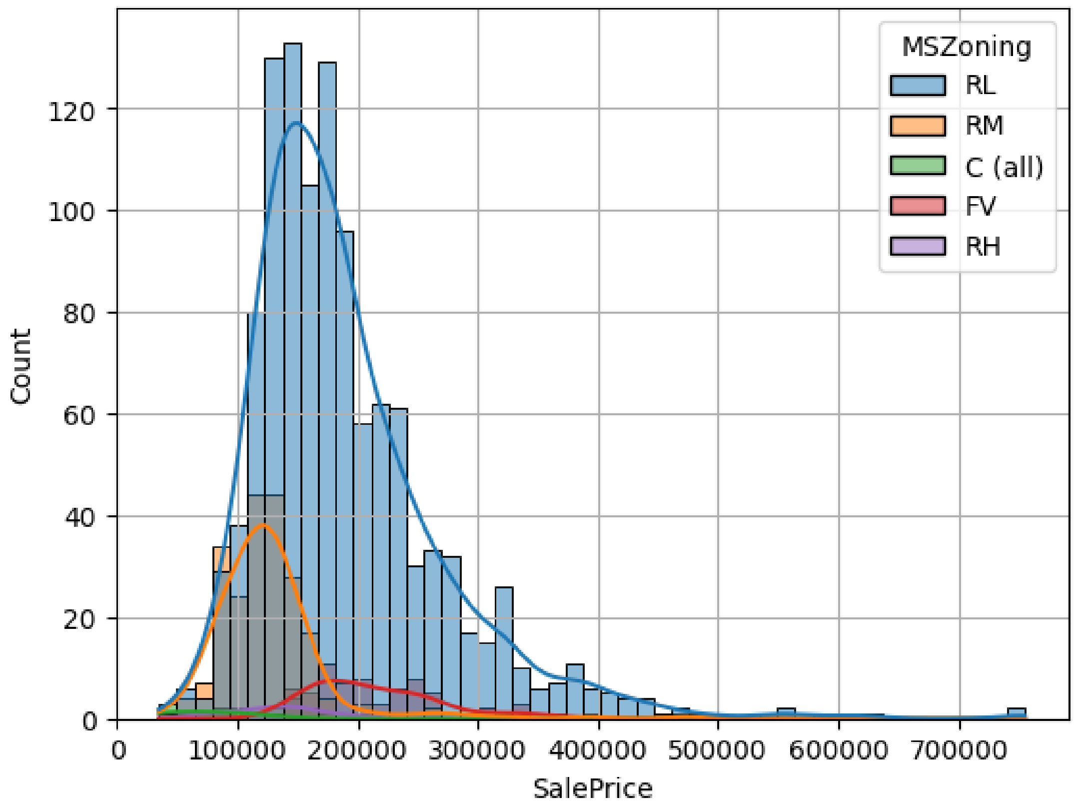


8.2.1 Histograms

```
sns.histplot(data=df, x="SalePrice", kde=True);plt.grid();
```




```
sns.histplot(data=df, x="SalePrice", kde=True, hue = "MSZoning");plt.grid();
```



8.2.2 Tasks:

8.2.2.1 Regression/Correlation (Housing Data)

1. Look up the `pairplot` function and create pairwise scatter plots of

- 5-7 hand-picked numerical features, one of them being `SalePrice`
- Hint: look at `dtypes`

2. Choose the row with `SalePrice` and pick two reasonably strong correlations.

- Compute the correlation coefficients
- Fit a simple regression line (with `statsmodels`) for each and visualize them using `regplot`
- Fit a **multiple regression** by including both *explanatory variables* and compare the coefficients

```
df.dtypes != "object"
```

```
Id                True
MSSubClass        True
MSZoning         False
LotFrontage       True
LotArea           True
                 ...
MoSold            True
YrSold            True
SaleType         False
SaleCondition    False
SalePrice         True
Length: 81, dtype: bool
```

```
df.columns[df.dtypes != "object"][1:]
```

```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

8.3 Extra Credit

8.3.0.1 Modeling Missing Values Titanic Data

1. detect the missing values
2. replace the NAs in survived with the estimate grouped by sex

```
#titanic
titanic = sns. load_dataset('titanic')
titanic["3rdClass"] = titanic["pclass"]==3
titanic["male"] = titanic["sex"]=="male"
#titanic.head()

#Introduce some missing values
rng = default_rng()
```

```
missingRows = rng.integers(0,890,20)
print(missingRows)
#introduce missing values
titanic.iloc[missingRows] = np.nan
```

9 Boxplots/Correlation

1. Boxplots
    - Quantiles
    - Whiskers
2. Histograms and Standard Deviation

Task 0

1. Read chapters 1 and 2 in the ThinkStats book in the cloud folder

9.1 Explore the Titanic Data

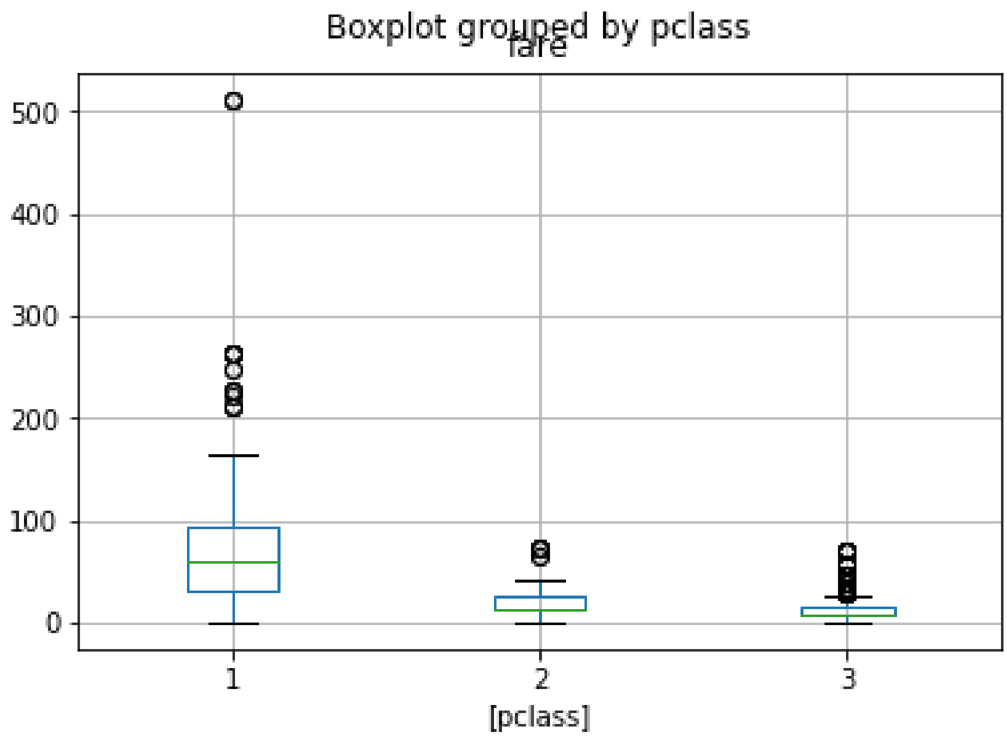
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
titanic = sns.load_dataset('titanic')
titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	dec
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Na
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Na
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Na

9.1.1 Boxplots

```
boxplot = titanic[['fare','pclass']].boxplot(by='pclass',return_type='dict' )
```



9.2 Task 1

1. Read up the basics of boxplots: https://en.wikipedia.org/wiki/Box_plot, in particular the paragraph explaining the **whiskers**.
2. Read up the definition of **Quartiles** and **Quantiles** and **IQR**. A good source would be the *ThinkStats* book (in the cloud folder).
3. (**MC**) What are the exact values of the lower and upper whiskers (of fare) for the pclass2 passengers?

   - **A** $[0, 41.6]$
   - **B** $[0, 45.5]$
   - **C** $[-6.5, 45.5]$
   - **D** $[0, 46.1]$

Recall the Wikipedia definition:*From above the upper quartile, a distance of 1.5 times the IQR is measured out and a whisker is drawn up to the largest observed point from the dataset that falls within this distance. Similarly, a distance of 1.5 times the IQR is measured out below the lower quartile and a whisker is drawn up to the lower observed point from the dataset that falls within this distance.*

9.2.1 Task 1.3

- **(MC) What are the exact values of the lower and upper whiskers (of fare) for the pclass2 passengers?**

```
titanic[['fare','pclass']].groupby('pclass').describe()
```

	fare							
	count	mean	std	min	25%	50%	75%	max
pclass								
1	216.0	84.154687	78.380373	0.0	30.92395	60.2875	93.5	512.3292
2	184.0	20.662183	13.417399	0.0	13.00000	14.2500	26.0	73.5000
3	491.0	13.675550	11.778142	0.0	7.75000	8.0500	15.5	69.5500

9.3 Explore the Auto Data

```
df = pd.read_csv('../data/Auto.csv')
df.head()
#df.info()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle mal
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

```
# global mean
df.mean()
```

```
mpg                23.445918
cylinders           5.471939
displacement      194.411990
horsepower        104.469388
weight           2977.584184
acceleration       15.541327
year               75.979592
origin              1.576531
dtype: float64
```

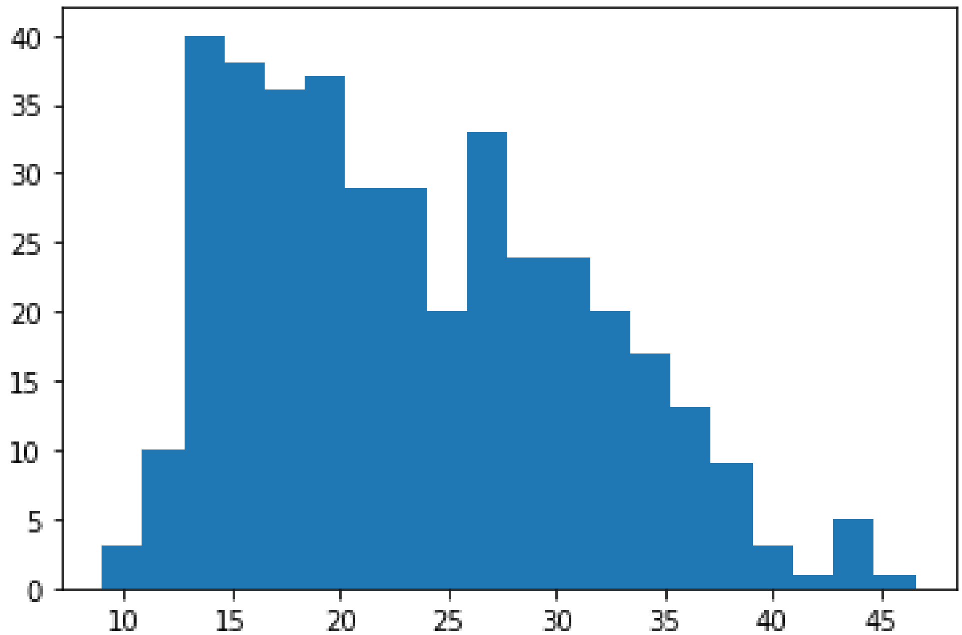
```
# mpg mean
a = df["mpg"].mean()
b = df.iloc[:,0].mean()
c = np.mean(df["mpg"])

print(f'mpg mean:\na = {a}\nb = {b}\nc = {c}')
```

```
mpg mean:
a = 23.44591836734694
b = 23.44591836734694
c = 23.44591836734694
```

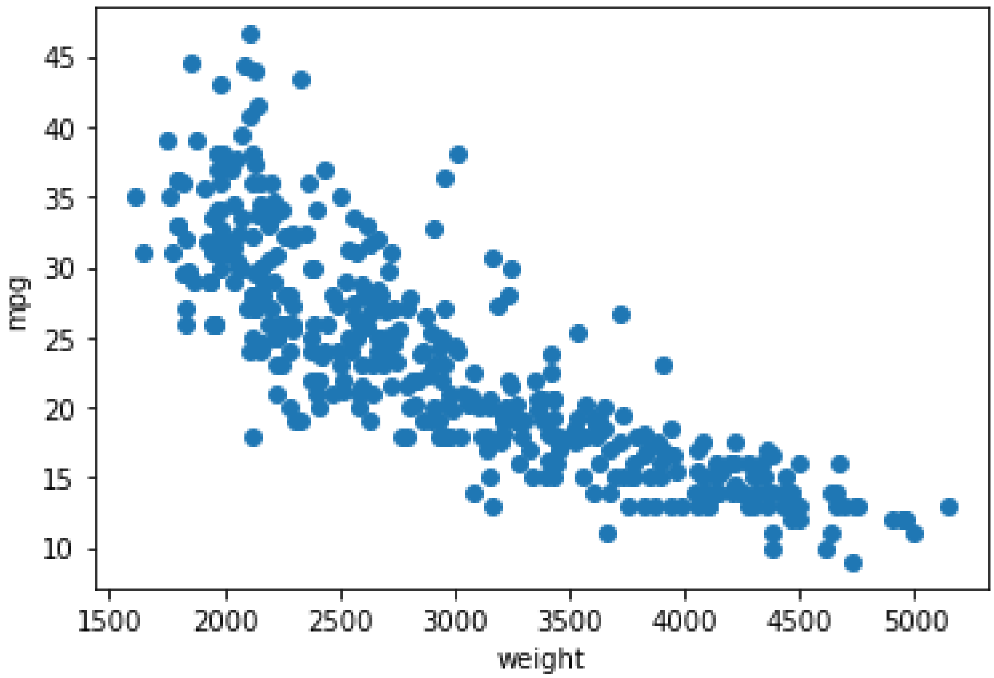
```
#. Plot a histogram of mpg
plt.hist(df["mpg"], 20)
```

```
(array([ 3., 10., 40., 38., 36., 37., 29., 29., 20., 33., 24., 24., 20.,
        17., 13.,  9.,  3.,  1.,  5.,  1.]),
 array([ 9.  , 10.88, 12.76, 14.64, 16.52, 18.4 , 20.28, 22.16, 24.04,
        25.92, 27.8 , 29.68, 31.56, 33.44, 35.32, 37.2 , 39.08, 40.96,
        42.84, 44.72, 46.6 ]),
 <BarContainer object of 20 artists>)
```




```
#scatterplot
plt.scatter("weight", "mpg",data=df)
plt.xlabel("weight")
plt.ylabel("mpg")
```

```
Text(0, 0.5, 'mpg')
```



9.4 Task 2

1. Compute the mean mpg grouped by cylinder.
2. Create a boxplot of mpg vs. cylinder
3. Find the median and lower/upper quartiles
4. Read up the definition of **correlation**. (*ThinkStats* book in the cloud folder). Compute the correlation coefficient between *mpg* and *weight*.
5. Compute the correlation coefficient ρ between *mpg* and *origin*. Discuss whether (i) there is a conceptual difference between the previous task, and (ii) whether it even makes sense to compute ρ for this pair of variables. In that context, learn about categorical data types in pandas.
6. What is the correlation coefficient "good for" ? Can you e.g. use it to make predictions, like in our previous simple probability model ? If not, what is missing ? Think about a *loss function* which would make sense for such a prediction task.