Python project för Borås University course exam 2022

Bin Han

2025-07-23

Table of contents

1 Project Description

This project involves comprehensive data analysis and programming exercises using Python, focusing on foundational concepts and problem-solving. It includes three distinct tasks:

Basic Calculations & Logic – Calculating the volume of a pyramid and determining bus ticket affordability based on user input, which enhances understanding of arithmetic operations, user input handling, and control flow with conditions and validation.

Algorithm Development – Creating custom functions for statistical analysis (min, max, average), debugging a prime number generator, and implementing a basic Swedish-English translator. This section strengthens algorithmic thinking, debugging skills, and dictionary usage without relying on built-in functions.

Data Analysis & Visualization – Reading, cleaning, and analyzing real-world car data from a CSV file, followed by visualizing key metrics (transmission type, gear count, horsepower,

fuel efficiency) using matplotlib. This part improves skills in file handling, data structuring, manual aggregation, and data visualization.

Together, the tasks demonstrate logical reasoning, clean coding practices, and the ability to extract insights from structured data—key skills for technical roles.

2 Pyramid Volume Calculator and Age-Based Ticket Affordability Checker

This script includes two parts: (a) a function that calculates the volume of a pyramid based on given base area and height, rounding the result to one decimal place; and (b) a program that determines if a user can afford a bus ticket based on their age and available money, with ticket pricing by age group and input validation using datetime and conditional statements.

(a) Skriv en funktion volymPyramid(B, h) som beräknar volymen på en pyramid med godtycklig form på basytan. Parametern B är pyramidens basarea (dvs den måste vara känd redan) och h är pyramidens höjd. Formeln för volymberäkning av en pyramid är följande: Funktionen ska returnera volymen avrundat till 1 decimal. Vilken enhet som avses (m/m^3, cm/cm^3 osv) spelar ingen roll, se argumenten som skickas till funktionen och det returnerade resultatet som enhetslösa. Avsluta med att testa funktionen med användarinmatning, likt nedan:

```
def volymPyramid(B, h):
    V=B*h/3
    return V

print(f" -- Beräkna volymen på en pyramid -- ")
print(f"Baserea: 10")
print(f"Höjd: 5")
V=volymPyramid(10, 5)
print(f"-Volymen är {V:.1f}")
```

```
 -- Beräkna volymen på en pyramid --
Baserea: 10
Höjd: 5
-Volymen är 16.7
```

(b) Skriv ett program som efterliknar exempelkörningarna nedan som med hjälp av if-elif-else-satser kontrollerar huruvida en person har råd att köpa en bussbiljett. Beljetten kostar 100:- för vuxna (>=18 år), 50:- för ungdomar (<18 år och <=7 år) och 0:- för barn (< 7 år). Användaren får skriva in hur mycket pengar den har samt vilket år den är född. Beräkna ålder med hjälp av modulen datetime (för att hämta nuvarande år).

Sedan görs kontrollen och "resultatet" skrivs ut. Programmet ska även ha "inmatningsfelkontroller" - enkelt begränsat till att man med hjälp av while- och if- satser kollar om användaren har räkat mata in pengar mindre än 0, eller en ålder untanför rimliga 0-150 år spannet. Då får användaren mata in ingen till godkända värden är mottagna.

```
import datetime
year=datetime.date.today().year

birth_year=1989
money=1000


def ticket_cost(birth_year, money):
    age=year-birth_year
    print(f"Du är således {age} år gammal")
    if age>=18:
        print(f"Biljetten kostar 100: -för 18 år+")
        if money> 100:
            print(f"-Du har råd")
        else:
            print(f"-Du har inte råd")
    if 18> age >=7:
        print(f"Biljetten kostar 50: - för 7år")
        if money > 50:
            print(f"-Du har råd")
        else:
            print(f"-Du har inte råd")
    if age<7:
        print(f"Biljetten är gratis för barn under 7år")
        print(f"Du har råd")


ticket_cost(birth_year, money)

birth_year=1987
money=600
ticket_cost(birth_year, money)

birth_year=2010
money=35
ticket_cost(birth_year, money)

birth_year=2022
money=0
ticket_cost(birth_year, money)
```

```
Du är således 36 år gammal
Biljetten kostar 100: -för 18 år+
-Du har råd
Du är således 38 år gammal
Biljetten kostar 100: -för 18 år+
-Du har råd
Du är således 15 år gammal
Biljetten kostar 50: - för 7år
-Du har inte råd
Du är således 3 år gammal
Biljetten är gratis för barn under 7år
Du har råd
```

3 Custom Algorithms: Random Number Analysis, Prime Number Debugging, and Basic Swedish-English Translator

This script includes three tasks: (a) Generates a list of five random integers and computes the minimum, maximum, and average using custom functions without built-in helpers. (b) Identifies and fixes logical errors in a program that prints the first n prime numbers. (c) Implements a basic word-by-word translation tool between Swedish and English using a dictionary, without using external libraries. Each part reinforces programming logic, control structures, and dictionary usage in Python.

(a) Skriv ett program som skapar en lista kallad slumplista med 5st slumpsässgit genererade heltal mellan från >=1 och <= 10, och därefter hittar det minsta och det största av talen i listan, samt tar fram medelvärdet av alla talen i listan. Dessa 3st beräkningar ska läggas i egendefinierade funktioner kallade för minstaTal(lista), storstaTal(lista) och medelvarde(lista), som returnerar sina respektive resultat. Utskriften ska likna följande:

Använda modulen random för att generera talen. Du får inte på denna uppgift använda Pythons inbyggda funtioner min() max() eller liknande"helt färdiga" från importerade moduler, utan du ska tänka till och skriva egna versioner av alla dessa 3st beräkningsalgoritmer. Du får inte heller specifikt använda sort()/sorted() för att först sortera listan och på så sätt få ut minsta och största värdet i listans ändan.

```python
import random

def rmlist():
    lis=[]
    for i in range(5):
        a=random.randint(1, 10)
```

```
        lis.append(a)
    print(f"Den slumpsässgit list är {lis}")
    return lis


def mean(A):
    sum=0
    for item in A:
        sum=sum+item
    mean=sum/len(A)
    print(f"Medelvärde: {mean}")
def max(A):
    max=A[0]
    for item in A[1:]:
        if item>max:
            max=item
    print(f"Största talet: {max}")
def min(A):
    min=A[0]
    for item in A[1:]:
        if item<min:
            min=item
    print(f"Minsta talet: {min}")
L=rmlist()
mean(L)
max(L)
min(L)

H=rmlist()
mean(H)
max(H)
min(H)
```

```
Den slumpsässgit list är [1, 10, 10, 7, 2]
Medelvärde: 6.0
Största talet: 10
Minsta talet: 1
Den slumpsässgit list är [8, 3, 10, 8, 5]
Medelvärde: 6.8
Största talet: 10
Minsta talet: 3
```

(b) Följande program frågar användaren efter hur många av de första talen i primtalsserien(naturliga tal som är större än 1 och inte några positiva delare än 1 och talet självt) som önskas beräknas och skrivas ut. Men det finns 5st logiska “slarvfel” i programmet

som gör att det inte fungerar korrekt (dvs programmetr är redan körbart, men ger inte rätt resultat) Rätta felen. Skriv en kommentar på varje rad du gör en ändring, okommenterade ändringar ger inga poåang. Koden nedan finns inlagd i svarsfilen redo att rättas.

```
import random
ANTAL_KOLUMNER = 5
tal=2
raknare=0

antal_primtal= random.randint(1,50)
print(f"De första {antal_primtal} primtalen är: ")

while raknare < antal_primtal:
    ar_prim = True # if this is a prime number, then true
    divisor=2
    while divisor < tal: # divisor less than the number to check whether it is prime numbe
        if tal % divisor == 0:# difinition of non-prime number
            ar_prim = False
            break# stop the loop
        divisor +=1
    if ar_prim:
        raknare +=1 # the number will increase by one if we find a prime number
        print(f" {tal: 5d}", end ="")
        if raknare % ANTAL_KOLUMNER == 0:
            print()#
    tal+=1
```

```
De första 18 primtalen är:
     2     3     5     7    11
    13    17    19    23    29
    31    37    41    43    47
    53    59    61
```

(c) Skriv ett program som kan översätta en inläst mening på svenska till engelska, och vice versa. Någon importerad översättningsmodul ska inte användas, utan det ska göras från scratch med en enkel översättning “ord per ord” (även om användbarheten p.g.a grammatik m.m blir begränsad förstås). Använd följande ordbok:

Skapa en tvådimensionell lista för ordboken. Användaren ska mata in vilket språk som önskas översättas, följt av meningen som ska översättas. Dela upp meningen i enskilda ord och gå igeom dessa med hjälp av for-satser. Ord som återfinns i ordboken byts ut mot dess översättning, ord som inte hittas lämnas oöversatta. Skriv ut resultet.

T.ex. följande mening “jag tyket Python är roligt” ska ge “I think Python is fun” som resultat vid översättning svenska –> engelska (och även vice versa).

```
word_dict={
    "jag": "i",
    "tycker": "think",
    "är": "is",
    "lätt": "easy",
    "svärt": "hard",
    "roligt": "fun"
}

def SetoEn(Sentence):
    result=""
    for word in Sentence.lower().split():
        if word in word_dict:
            result+=word_dict[word]+" "
        else:
            result+=word+ " "
    return  result.strip()
def EntoSE(Sentence):
    reword_dict={v: k for k, v in word_dict.items()}
    result=""
    for word in Sentence.lower().split():
        if word in reword_dict:
            result+=reword_dict[word]+" "
        else:
            result+=word+ " "
    return  result.strip()

Sentence1="Jag tycker python är roligt"
S1=SetoEn(Sentence1)
print(f"Swedish to English: {S1}")
Sentence2="I think Python is fun"
S2=EntoSE(Sentence2)
print(f"English to Swedish: {S2}")
```

```
Swedish to English: i think python is fun
English to Swedish: jag tycker python är roligt
```

4 Exploratory Data Analysis of 1974 Car Specifications Using CSV and Matplotlib

This script analyzes vehicle data from a CSV file (cars.csv) containing specifications for 32 cars from the 1973–74 Motor Trend magazine. It includes: (a) reading and cleaning CSV data into a 2D list; (b) counting and visualizing transmission types and gear counts

using bar charts; (c) creating line, scatter, and bar plots for variables like horsepower, fuel efficiency, and cylinder count; and (d) computing and plotting average values of horsepower, weight, and displacement grouped by the number of carburetors. All statistical calculations and visualizations are done without advanced libraries beyond csv and matplotlib.pyplot.

Datan i cars.csv är extraherad från en upplaga anno 1974 av den amerikanska tidskriften Motor Trend, och består av bränsleförbrukningen samt 7st andra design- och prestanda-aspekter för 32st olika bilar (av årsmodeller 1973-74) skärmbild på den första raderna: beskrivning av innehållet: Funktion Beskrivning: mpg mpg-miles/gallon cyl antal cylindrar disp slagvolym (kubik-inch -in3) (eng. displacement) hp håstkrafter brutto wt wikt (uttryckt i per 1000 pounds - lbs) am växling (0= automat, 1 manuell) gear antal växlar carb antal förgasare (eng. carburator)

Inga importerade moduler med undantag för csv och matplotlib.pyplot får användas för följande uppgifter. Du får här inte heller använda Python's inbyggda min() och max()-funktioner.

Några generella tips på hur du bäst löser uppgiften:

Öppna cars.csv i Excel eller annat kalkylprogram och bekanta dig med innehållet. Läs igenom alla deluppgifterna innan du börjar så att du vet vad uppgiften i helhet gå ut på. änk igenom vilka datakolumner du behöver till för att läsa respektive deluppgifter, och överväg (för din egen skull) också att skissa ett enkelt flödesdiagram och hur du planerar att lösa dem. Det kan vara tidsbespaarande att ggöra sådant analyserade förhandsarbete innan man börjar med kodringen.

(a) Skapa en egendefinierad funktion read_file(file_name) som öppnar en csv-fil (separator=semikolon ;) med det angivna namnet till paraemetern file_name, och läser in dess innehåller och returnerar resultatet i en tvådimensionell lista.

Avsluta med att anropa funktionen och spara resultat i en lista du kallar cars_data (som du ska återanvända i eterkommande deluppgifter). Skriv sedan ut de fem första raderna för att verifiera att det funkar som det skall. Utskrifen ska setsom följande: OBS: innan du går vidare och börjat med resternade deluppgifter. Obesevera att vissa datakolumner i detrt dataset utgörs av decimatal, med komma(,) decimaltecken (vilket är vanligt förekommande i många språkreligioner i värden.) Detta lär komma utgöra ett problem senare vid typkoverteringen från sådana tal i din cars_data-lista till floot när de ska användas till beräkninnar, när man till exempel försöka

då Python och cen aktuella "locale-ininställningen kan förvänta sid decimmapunt."

Det ska nämnas att detta må inte vara det mest robusta sättet att hantera detta på -det finns t.ex funktioner importerade moduler som vi inte går in på denna kurs som har specifika lösningar för Dettaman kan utforska. Men detta tillvågagåmngssätt duger gott här för dess mindre anvgränsade program.

```
import csv
import matplotlib.pyplot as plt
def read_file(filename):
    data_list=[]
    with open(filename, "r") as file:
        csvfile= csv.reader(file, delimiter= ";")
        headers= next(csvfile)
        for lines in csvfile:
            cleaned_row=[cell.replace(",", ".") for cell in lines]
            print(cleaned_row)
            data_list.append(cleaned_row)
    return headers, data_list

headers, cars_data=read_file("cars.csv")
print(headers)
print(cars_data)
```

```
['Cadillac Fleetwood', '10.4', '8', '472', '205', '5.25', '0', '3', '4']
['Lincoln Continental', '10.4', '8', '460', '215', '5.424', '0', '3', '4']
['Camaro Z28', '13.3', '8', '350', '245', '3.84', '0', '3', '4']
['Duster 360', '14.3', '8', '360', '245', '3.57', '0', '3', '4']
['Chrysler Imperial', '14.7', '8', '440', '230', '5.345', '0', '3', '4']
['Maserati Bora', '15', '8', '301', '335', '3.57', '1', '5', '8']
['Merc 450SLC', '15.2', '8', '275.8', '180', '3.78', '0', '3', '3']
['AMC Javelin', '15.2', '8', '304', '150', '3.435', '0', '3', '2']
['Dodge Challenger', '15.5', '8', '318', '150', '3.52', '0', '3', '2']
['Ford Pantera L', '15.8', '8', '351', '264', '3.17', '1', '5', '4']
['Merc 450SE', '16.4', '8', '275.8', '180', '4.07', '0', '3', '3']
['Merc 450SL', '17.3', '8', '275.8', '180', '3.73', '0', '3', '3']
['Merc 280C', '17.8', '6', '167.6', '123', '3.44', '0', '4', '4']
['Valiant', '18.1', '6', '225', '105', '3.46', '0', '3', '1']
['Hornet Sportabout', '18.7', '8', '360', '175', '3.44', '0', '3', '2']
['Merc 280', '19.2', '6', '167.6', '123', '3.44', '0', '4', '4']
['Pontiac Firebird', '19.2', '8', '400', '175', '3.845', '0', '3', '2']
['Ferrari Dino', '19.7', '6', '145', '175', '2.77', '1', '5', '6']
['Mazda RX4', '21', '6', '160', '110', '2.62', '1', '4', '4']
['Mazda RX4 Wag', '21', '6', '160', '110', '2.875', '1', '4', '4']
['Hornet 4 Drive', '21.4', '6', '258', '110', '3.215', '0', '3', '1']
['Volvo 142E', '21.4', '4', '121', '109', '2.78', '1', '4', '2']
['Toyota Corona', '21.5', '4', '120.1', '97', '2.465', '0', '3', '1']
['Datsun 710', '22.8', '4', '108', '93', '2.32', '1', '4', '1']
['Merc 230', '22.8', '4', '140.8', '95', '3.15', '0', '4', '2']
['Merc 240D', '24.4', '4', '146.7', '62', '3.19', '0', '4', '2']
['Porsche 914-2', '26', '4', '120.3', '91', '2.14', '1', '5', '2']
['Fiat X1-9', '27.3', '4', '79', '66', '1.935', '1', '4', '1']
```

```
['Honda Civic', '30.4', '4', '75.7', '52', '1.615', '1', '4', '2']
['Lotus Europa', '30.4', '4', '95.1', '113', '1.513', '1', '5', '2']
['Fiat 128', '32.4', '4', '78.7', '66', '2.2', '1', '4', '1']
['Toyota Corolla', '33.9', '4', '71.1', '65', '1.835', '1', '4', '1']
['Cars', 'mpg', 'cyl', 'disp', 'hp', 'wt', 'am', 'gear', 'carb']
[['Cadillac Fleetwood', '10.4', '8', '472', '205', '5.25', '0', '3', '4'], ['Lincoln Conti
```

(b) Skriv kod som räknar

1. antal automat-respektive manuellt växlade bilar (am-kolumnen i datasetet)
2. antalet bilar som har 3, 4 respektive 5 växlar (gear-kolumnen i datasetet)

Skriv ut resultatet i en enkel tabell likt nedan. Presentera även resultatet liknande följande stapeldiagram. Använd suplot() för att kombinera ihop de två diagrammen.

```python
def analyze_data(headers, data_list):
    # Hitta index för kolumnerna
    am_index = headers.index("am")
    gear_index= headers.index("gear")
    am_counts={"automat": 0, "manuell": 0}
    gear_counts={3:0, 4:0, 5:0}

    for row in data_list:
        am=row[am_index].strip()
        gear=row[gear_index].strip()

        if am == "1":
            am_counts["automat"]+=1
        elif am == "0":
            am_counts["manuell"]+=1

        try:
            gear = int(gear)
            if gear in gear_counts:
                gear_counts[gear]+=1
        except ValueError:
            continue# if gear is not a intiger
    return am_counts, gear_counts

headers, cars_data = read_file("cars.csv")
am_counts, gear_counts=analyze_data(headers, cars_data)
```

```
['Cadillac Fleetwood', '10.4', '8', '472', '205', '5.25', '0', '3', '4']
['Lincoln Continental', '10.4', '8', '460', '215', '5.424', '0', '3', '4']
['Camaro Z28', '13.3', '8', '350', '245', '3.84', '0', '3', '4']
```

```
['Duster 360', '14.3', '8', '360', '245', '3.57', '0', '3', '4']
['Chrysler Imperial', '14.7', '8', '440', '230', '5.345', '0', '3', '4']
['Maserati Bora', '15', '8', '301', '335', '3.57', '1', '5', '8']
['Merc 450SLC', '15.2', '8', '275.8', '180', '3.78', '0', '3', '3']
['AMC Javelin', '15.2', '8', '304', '150', '3.435', '0', '3', '2']
['Dodge Challenger', '15.5', '8', '318', '150', '3.52', '0', '3', '2']
['Ford Pantera L', '15.8', '8', '351', '264', '3.17', '1', '5', '4']
['Merc 450SE', '16.4', '8', '275.8', '180', '4.07', '0', '3', '3']
['Merc 450SL', '17.3', '8', '275.8', '180', '3.73', '0', '3', '3']
['Merc 280C', '17.8', '6', '167.6', '123', '3.44', '0', '4', '4']
['Valiant', '18.1', '6', '225', '105', '3.46', '0', '3', '1']
['Hornet Sportabout', '18.7', '8', '360', '175', '3.44', '0', '3', '2']
['Merc 280', '19.2', '6', '167.6', '123', '3.44', '0', '4', '4']
['Pontiac Firebird', '19.2', '8', '400', '175', '3.845', '0', '3', '2']
['Ferrari Dino', '19.7', '6', '145', '175', '2.77', '1', '5', '6']
['Mazda RX4', '21', '6', '160', '110', '2.62', '1', '4', '4']
['Mazda RX4 Wag', '21', '6', '160', '110', '2.875', '1', '4', '4']
['Hornet 4 Drive', '21.4', '6', '258', '110', '3.215', '0', '3', '1']
['Volvo 142E', '21.4', '4', '121', '109', '2.78', '1', '4', '2']
['Toyota Corona', '21.5', '4', '120.1', '97', '2.465', '0', '3', '1']
['Datsun 710', '22.8', '4', '108', '93', '2.32', '1', '4', '1']
['Merc 230', '22.8', '4', '140.8', '95', '3.15', '0', '4', '2']
['Merc 240D', '24.4', '4', '146.7', '62', '3.19', '0', '4', '2']
['Porsche 914-2', '26', '4', '120.3', '91', '2.14', '1', '5', '2']
['Fiat X1-9', '27.3', '4', '79', '66', '1.935', '1', '4', '1']
['Honda Civic', '30.4', '4', '75.7', '52', '1.615', '1', '4', '2']
['Lotus Europa', '30.4', '4', '95.1', '113', '1.513', '1', '5', '2']
['Fiat 128', '32.4', '4', '78.7', '66', '2.2', '1', '4', '1']
['Toyota Corolla', '33.9', '4', '71.1', '65', '1.835', '1', '4', '1']
```

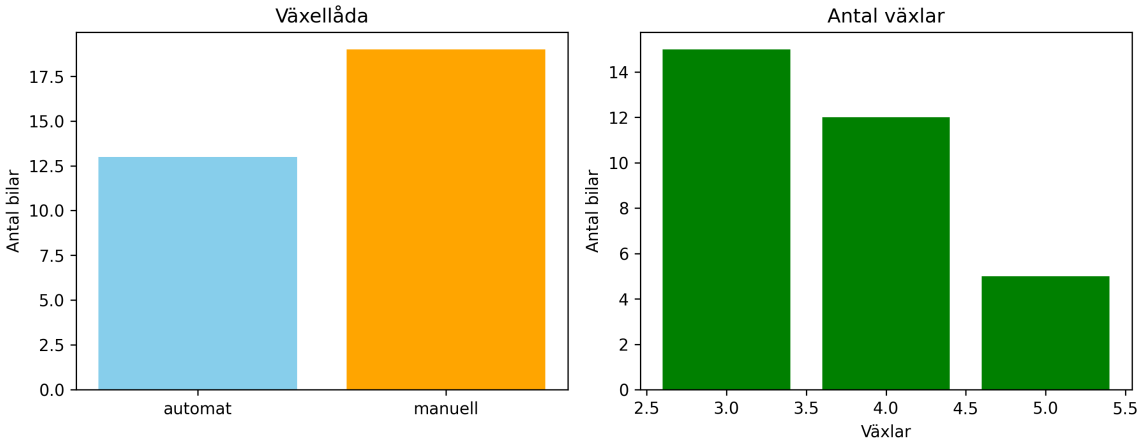
4.1 Ladda data och beräkning

```python
def print_tables(am_counts, gear_counts):
    print("\n Antal bilar per växellåda:")
    print("f{'växellåda':<10}| {'Antal':>5}")
    print("-" * 20)
    for key, value in am_counts.items():
        print(f"{key:<10} | {value:>5}")
    print("\nAntal bilar per antal växlar:")
    print(f"{'Växlar':<6}| {'Antal' :>5}")
    print("-" *20)
    for gear, count in sorted(gear_counts.items()):
        print(f"{gear:<6} | {count:>5}")
```

```
print_tables(am_counts, gear_counts)
```

```
 Antal bilar per växellåda:
f{'växellåda':<10}| {'Antal':>5}
--------------------
automat    |    13
manuell    |    19

Antal bilar per antal växlar:
Växlar| Antal
--------------------
3      |    15
4      |    12
5      |     5
```

```
def plot_bar_charts(am_counts, gear_counts):
    fig, axes= plt.subplots(1, 2, figsize=(10, 4))

    # Diagram 1
    axes[0].bar(am_counts.keys(), am_counts.values(), color=["skyblue", "orange"])
    axes[0].set_title("Växellåda")
    axes[0]. set_ylabel("Antal bilar")

    # Diagram 2
    axes[1].bar(gear_counts.keys(), gear_counts.values(), color="green")
    axes[1].set_title("Antal växlar")
    axes[1].set_xlabel("Växlar")
    axes[1].set_ylabel("Antal bilar")

    plt.tight_layout()
    plt.show()

plot_bar_charts(am_counts, gear_counts)
```



(c) Skriv kod som plott:ar följande 3st olika diagram

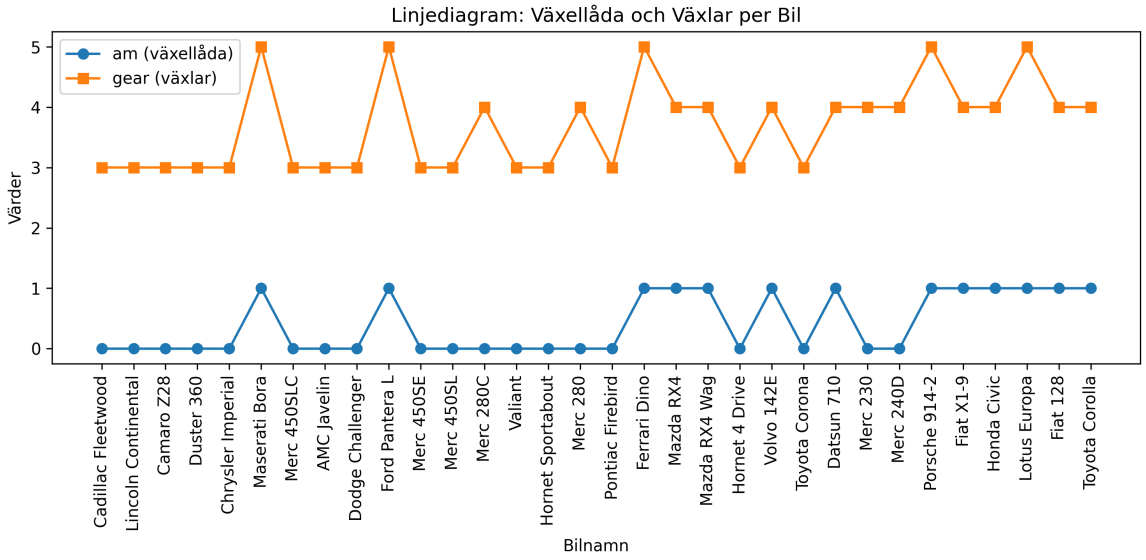
1. Linjediagrammet- skapa separata listor för kolumnerna Cars, am och gear. Låt x-axeln vara Cars (rotera bilnamnen 90-grader så att de får plats), och y-axeln vara am och gear. Lägg till en etikett för am and gear. Skriv diagramtiteln.
2. Spridningsdiagrammet (eng. scatterplot)- skapa separata listor för kolumnerna Cars, hp och mpg. Låt y-axeln vara hp och x-axeln mpg. Skriv ut bilnamnen tillhörande varje utskriven punkt ("annotate"). Skriv diagramtiteln.
3. Stapeldiagrammet- skapa separata listor för kolumnerna hp and cyl. Lår x-axeln vara cyl ("number of cylinders") och y-axeln hp. Skriv diagramtiteln.

```
def load_car_data(filename):
    cars=[]
    am_list=[]
    gear_list=[]
    hp_list=[]
    mpg_list=[]
    cyl_list=[]

    with open(filename, "r", encoding="utf-8") as file:
        reader= csv.DictReader(file, delimiter=";")
        for row in reader:
            cars.append(row["Cars"])
            am_list.append(int(row["am"]))
            gear_list.append(int(row["gear"]))
            hp_list.append(float(row["hp"].replace(",", ".")))
            mpg_list.append(float(row["mpg"].replace(",", ".")))
            cyl_list.append(int(row["cyl"]))
    return cars, am_list, gear_list, hp_list, mpg_list, cyl_list
```

```
# Läs data från fil
cars, am_list, gear_list, hp_list, mpg_list, cyc_list= load_car_data("cars.csv")
```

4.2 Line chart: Cars vs am & gear

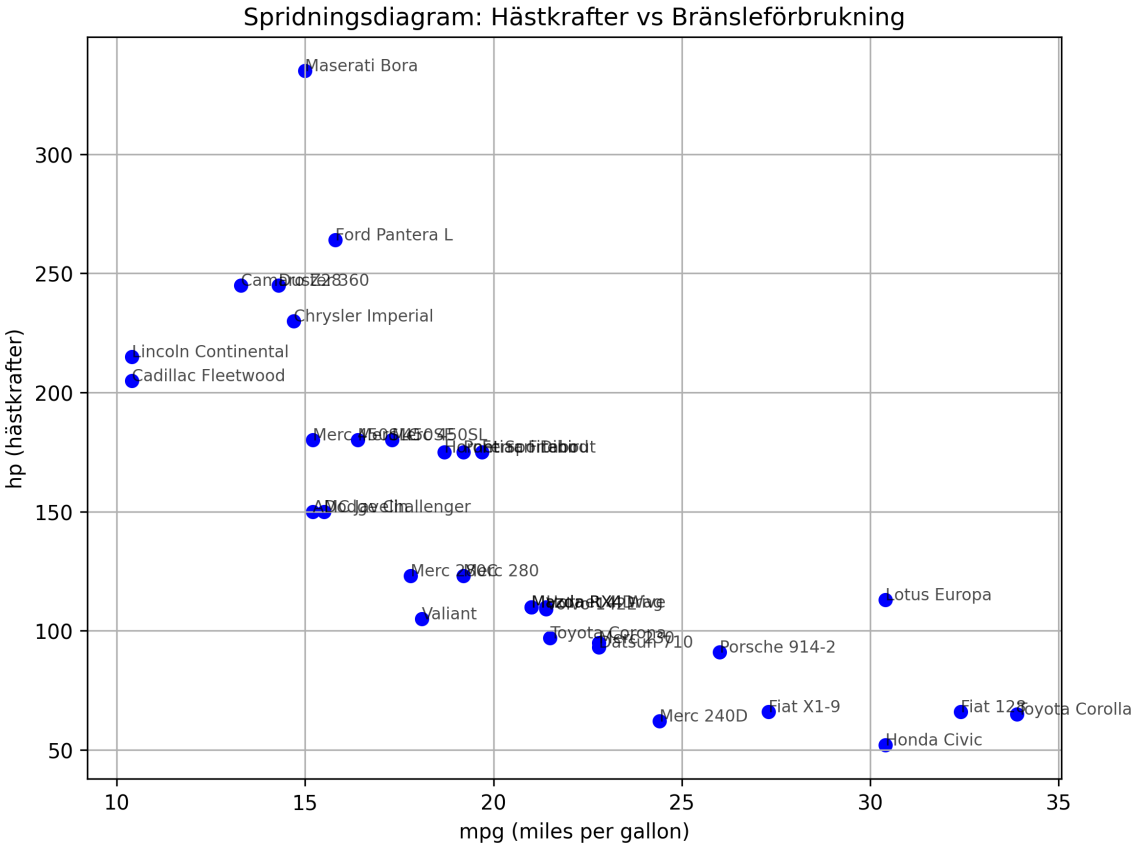
```
plt.figure(figsize=(10, 5))
plt.plot(cars, am_list, label="am (växellåda)", marker="o")
plt.plot(cars, gear_list, label="gear (växlar)", marker="s")
plt.xticks(rotation=90)
plt.xlabel("Bilnamn")
plt.ylabel("Värder")
plt.title("Linjediagram: Växellåda och Växlar per Bil")
plt.legend()
plt.tight_layout()
plt.show()
```



4.3 Scatter chart: mpg vs hp

```
plt.figure(figsize=(8, 6))
plt.scatter(mpg_list, hp_list, color="blue")

# Annotera varje punkt me bilnamne
for i in range(len(cars)):
    plt.annotate(cars[i], (mpg_list[i], hp_list[i]), fontsize=8, alpha=0.7)
```

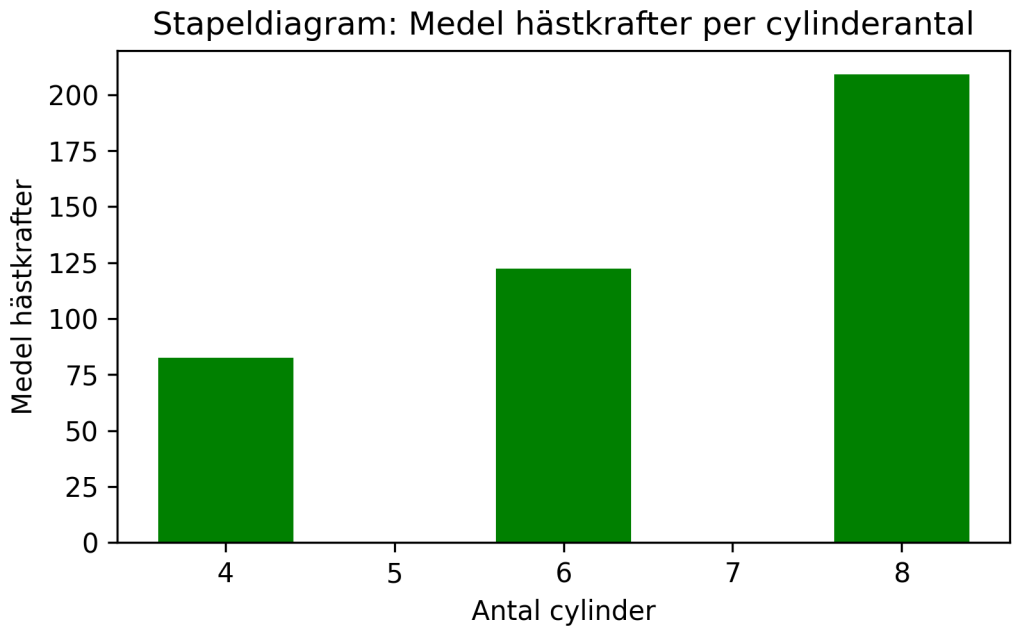
```
plt.xlabel("mpg (miles per gallon)")
plt.ylabel("hp (hästkrafter)")
plt.title("Spridningsdiagram: Hästkrafter vs Bränsleförbrukning")
plt.grid(True)
plt.tight_layout()
plt.show()
```



4.4 Bar chart: hp vs cyl

```
filename= "cars.csv"
cyl_list=[]
hp_list =[]

with open(filename, "r", encoding="utf-8") as file:
    reader= csv.DictReader(file, delimiter=";")
    for row in reader:
        try:
            cyl= int(row["cyl"])
```

```
            hp = float(row["hp"].replace(",", "."))
            cyl_list.append(cyl)
            hp_list.append(hp)
        except:
            continue # jump over the row with wrong data

# Gruppera hp efter cylinderantal (utan extra moduler)
unique_cylinders=[]
for c in cyl_list:
    if c not in unique_cylinders:
        unique_cylinders.append(c)

# Sotera manuellt
for i in range(len(unique_cylinders)):
    for j in range(i+1, len(unique_cylinders)):
        if unique_cylinders[i] > unique_cylinders[j]:
            unique_cylinders[i], unique_cylinders[j] = unique_cylinders[j], unique_cylinde

# Count average value manually
avg_hp_per_cyl=[]
for cyl in unique_cylinders:
    total=0
    count=0
    for i in range(len(cyl_list)):
        if cyl_list[i] ==cyl:
            total+=hp_list[i]
            count+=1
    avg=total/count
    avg_hp_per_cyl.append(avg)

# draw stapeldiagram

plt.bar(unique_cylinders, avg_hp_per_cyl, color="green")
plt.xlabel("Antal cylinder")
plt.ylabel("Medel hästkrafter")
plt.title("Stapeldiagram: Medel hästkrafter per cylinderantal")
plt.tight_layout()
plt.show()
```



(d) Skriv kod som beräknar mdelvärdet för hp, wt och disp för de olika förgasare-antalen(carb). Det finns i datasetet bilar som har antingen 1,2, 3, 4,6 samt 8st förgasare. Presentera resultaten i en tabell likt nedan, avrunda till 1 decimal på beräknade medelvärden. OBS, medelvärdena är ej utskrivna i denna konceptuellt visande tabell, utan du får kontrollera själv att dina beräkningar är rimliga.

```
# Förbereder datastrukturen
carb_values= [1, 2, 3, 4, 6, 8]
hp_data = {carb: [] for carb in carb_values}
wt_data= {carb: [] for carb in carb_values}
disp_data={carb: [] for carb in carb_values}

# Läser in CSV-filen and grouping the data according to carb.
with open("cars.csv", "r", encoding="utf-8") as file:
    reader = csv.DictReader(file, delimiter=";")
    for row in reader:# each row is dictionary
        try:
            carb=int(row["carb"])
            if carb in carb_values:
                hp= float(row["hp"].replace(",", "."))
                wt= float(row["wt"]. replace(",", "."))
                disp=float(row["disp"].replace(",", "."))

                hp_data[carb].append(hp)
                wt_data[carb].append(wt)
                disp_data[carb].append(disp)
```

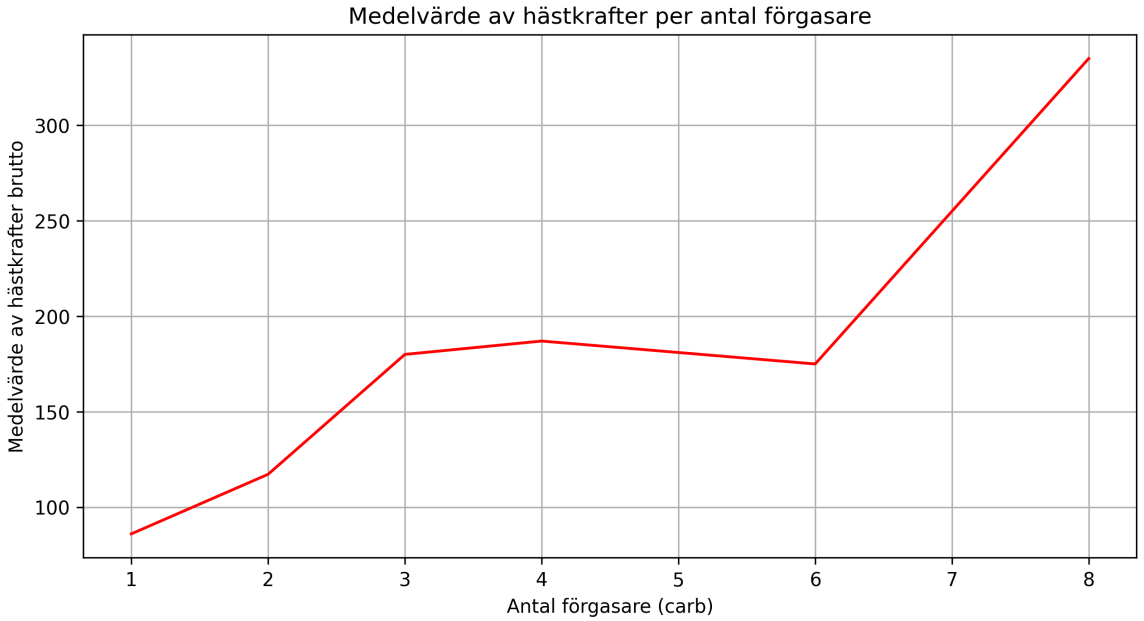
```
        except:
            continue #jump over bad rows.
```

```
#Räkna ut medelvärden (manuellt) och skriv ut tabell
print(f"{'Carb': <5} | {'HP':>6} | {'WT':>6} | {'DISP':>6}")
print("-"* 30)

avg_hp_list=[]
avg_wt_list=[]
avg_disp_list=[]

for carb in carb_values:
    def calc_avg(values):
        total=0
        count=0
        for v in values:
            total+=v
            count+=1
        return round(total/count, 1) if count >0 else 0

    avg_hp=calc_avg(hp_data[carb])
    avg_wt=calc_avg(wt_data[carb])
    avg_disp=calc_avg(disp_data[carb])

    print(f"{carb:<5} | {avg_hp:>6} | {avg_wt:>6} | {avg_disp:>6}")
    avg_hp_list.append(avg_hp)
    avg_wt_list.append(avg_wt)
    avg_disp_list.append(avg_disp)
```

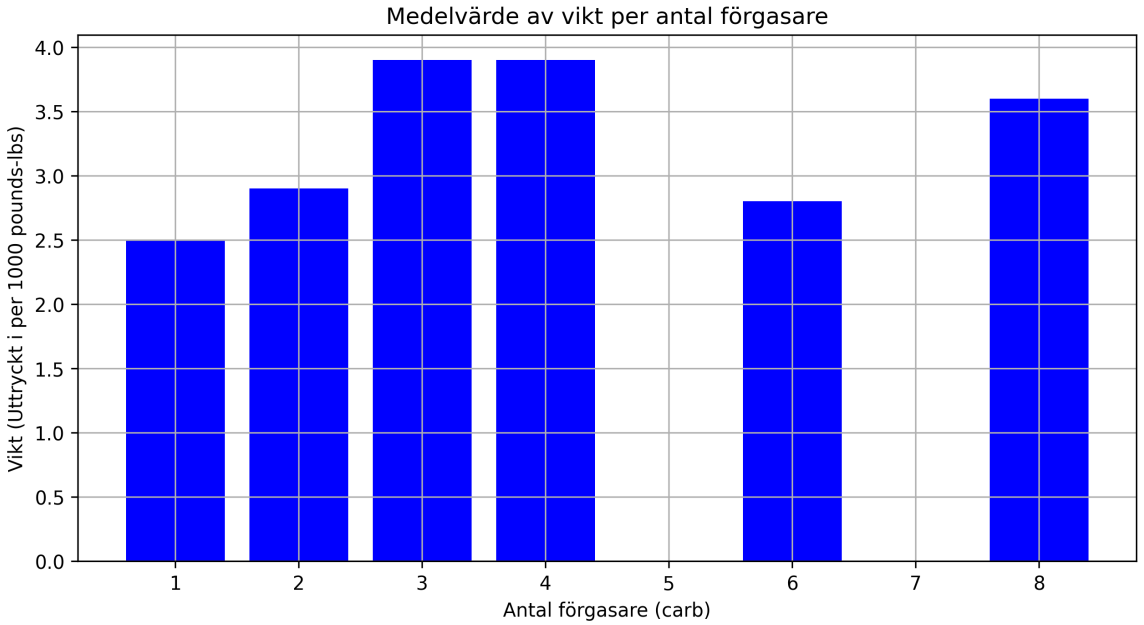
```
Carb  |     HP |     WT |   DISP
------------------------------
1     |   86.0 |    2.5 |  134.3
2     |  117.2 |    2.9 |  208.2
3     |  180.0 |    3.9 |  275.8
4     |  187.0 |    3.9 |  308.8
6     |  175.0 |    2.8 |  145.0
8     |  335.0 |    3.6 |  301.0
```

```
plt.figure(figsize=(10,5))
plt.plot(carb_values, avg_hp_list, color ="red")
plt.xlabel("Antal förgasare (carb)")
plt.ylabel("Medelvärde av hästkrafter brutto")
plt.title("Medelvärde av hästkrafter per antal förgasare")
#plt.ylim(0,400)
```

```
#plt.xlim(0,8.5)
plt.grid(True)
plt.show()
```




```
plt.figure(figsize=(10,5))
plt.bar(carb_values, avg_wt_list, color="blue")
plt.xlabel("Antal förgasare (carb)")
plt.ylabel("Vikt (Uttryckt i per 1000 pounds-lbs)")
plt.title("Medelvärde av vikt per antal förgasare")
plt.grid(True)
plt.show()
```




```
plt.figure(figsize=(10,5))
#plt.scatter(carb_values, avg_disp_list, color="green")
plt.bar(carb_values, avg_disp_list, color="red")
plt.xlabel("Antal förgasare (carb)")
plt.ylabel("Displacement")
plt.title("Medelvärde av displacement per antal förgasare")
plt.grid(True)
plt.show()
```