Homework 2 (HW2)

By the end of this homework, we expect you to be able to:

- Preprocess data and make it amenable to statistical analysis and machine learning models;
- Train and test out-of-the-box machine learning models in `sklearn`;
- Carry out simple logistic regression and decision tree analysis;
- Use propensity score matching to estimate treatment effects;

Important Dates

- Homework release: Fri 15 Nov 2024
- **Homework due**: Fri 29 Nov 2024, 23:59
- Grade release: Mon 09 Dec 2024

Some rules

1. We have provided `TODO` comments in the code cells that you need to fill out with your solutions. For some questions, we have also provided `Your response` comments, where you should provide a textual answer.
2. You are allowed to use any built-in Python library that is included in the `requirements.txt` for this homework. If you use any additional library, this may complicate the grading process, and we reserve the right to penalize your grade for unnecessary complexity of the solution. All the questions can be solved with the libraries in `requirements.txt`.
3. Please write all your comments in English, and use meaningful variable names in your code. Your repo should have a single notebook (plus the required data files) in the *master/main* branch. If there are multiple notebooks present, we will **not grade** anything.
4. We will **not run your notebook for you**! Rather, we will grade it as is, which means that only the results contained in your evaluated code cells will be considered, and we will not see the results in unevaluated code cells. Thus, be sure to hand in a **fully-run and evaluated notebook**. In order to check whether everything looks as intended, you can check the rendered notebook on the GitHub website once you have pushed your solution there.
5. In continuation to the previous point on additional library, interactive plots, such as those generated using `plotly`, should be **strictly avoided**!

A Note on using Language Models (LMs)

If you try hard enough, you will likely get away with cheating. Fortunately, our job is not to police, but rather to educate! So, please consider the following:

Presumably, you are taking this course to learn something! LMs are not always right (they often fail in silly ways). This course should prepare you to detect when they are wrong! Some of the TAs on this course literally published many works on detecting machine-generated text.

Here LLM includes but not limited to chatbots like ChatGPT, coding assistants like Copilot. Do not even use them to prettify your code or correct English. If you are caught using LLMs, you will be reported to the instructor and subject to the consequences.

Grading

- The homework has a total of 100 points, distributed as follows:
  - Part 1: Data Preprocessing (20 points)
  - Part 2: Linear Regression (30 points)
  - Part 3: Supervised Learning (40 points)
  - Part 4: Propensity Score Matching (10 points)

Context

Within EPFL's master program, you are excited to start an internship as a data scientist. After rounds of interviews, you have been selected to work with the biggest car dealership in Switzerland !

Your mentor at the company Tim, has explained to you that the company is interested in a pricing model for used cars.

- Tim: "We have a lot of used cars in our inventory, and we need to determine the price at which we should sell these cars. We have some ideas about the factors that influence the price of a used car, but so far we have just been using our experience and intuition to determine the price of a used car. Sometimes it works, but probably we can do better and a more data-driven approach would also help our new employees in the sales team as they have less experience."
- You: "That sounds like a great project! What kind of data do we have?"
- Tim: "We sell all kinds of cars here, but maybe we can start with a specific brand and model. For example, the Toyota Corolla is the best-selling car worldwide in 2023, and we have a lot of data on it. We can start by analyzing the data on used Toyota Corolla cars. If it works well, we can extend the analysis to other brands."

The dataset contains the following columns:

- `Age`: Age of the car in months.
- `Mileage`: Number of distance the car has been driven. (km or miles)
- `FuelType`: Fuel type of the car (Petrol, Diesel, or CNG)
- `HP`: Horsepower
- `MetColor`: Is the color of the car metallic? (Yes=1, No=0)
- `Automatic`: Is the car automatic? (Yes=1, No=0)
- `CC`: Cylinder volume in cubic centimeters
- `Doors`: Number of doors
- `Weight`: Weight of the car in kilograms
- `Price`: Price of the car in euros

Data

The data is provided in the `data` folder and it contains the following 3 csv files:

- `Task1-2.ToyotaCorolla.csv` for Part 1 and Part 2
- `Task3.ToyotaCorolla_sales_3months.csv` for Part 3
- `Task4.ToyotaCorolla_discount_sales` for Part 4

You should not use any other data source for this homework.

For some questions, you might need to slightly modify the data. But overall, you should avoid making any major changes to the data, which may affect your analysis.

References:

The data is based on the ToyotaCorolla dataset from the UCI Machine Learning Repository here. We have made some modifications to the original dataset, so please use the data provided in the `data` folder in the course repo.

Task 1 (20 pts) - Get to know the data

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import random
import sklearn

# fix random seed for reproducibility
np.random.seed(42)
random.seed(42)
```

1.1 (2 pts): Load the data from the file `Task1-2.ToyotaCorolla-raw.csv` into a pandas DataFrame. Display the first 5 rows of the DataFrame. Hint: A naive loading of the data will raise an error. You will need to figure out how to load the data correctly. (Hint: localise which row is causing the error)

In [2]:
```python
# Load the data

# data_df = pd.read_csv('data/Task1-2.ToyotaCorolla.csv') this will fail because the file is not in

# The best way is to fix the line 33 in the file by either manually editing it or writing a functio

try:
    data_df = pd.read_csv('data/Task1-2.ToyotaCorolla-raw.csv')
except Exception as e:
    print(e)

# If the students skip the bad line with the following or similar way, they will only get 1/2 point

data_df = pd.read_csv('data/Task1-2.ToyotaCorolla-raw.csv', on_bad_lines='skip')


# Display the first 5 rows of the data
print(data_df.head())
```

```
Error tokenizing data. C error: Expected 11 fields in line 33, saw 12

     Price   Age  Mileage FuelType    HP  MetColor  Automatic      CC  Doors  \
0  13500.0  23.0  46986.0   Diesel  90.0       1.0        0.0  2000.0    3.0
1  11878.0  23.0  72937.0   Diesel  90.0       1.0        0.0  2000.0    3.0
2  12050.0  24.0  41711.0   Diesel  90.0       1.0        0.0  2000.0    3.0
3  12914.0  26.0  48000.0   Diesel  90.0       0.0        0.0  2000.0    3.0
4  11878.0  30.0  38500.0   Diesel  90.0       0.0        0.0  2000.0    3.0

   Weight Currency
0  1165.0     EURO
1  1165.0      CHF
2  1165.0      CHF
3  1165.0      CHF
4  1170.0      CHF
```

1.2 (2 pts): Check if there are nan values in the Dataframe. If there are, try to find out which row is problematic and fix it. If you can't fix it, drop the row.

In [3]:
```python
# get nan-index
nan_index = data_df.isnull().any(axis=1)
print(data_df[nan_index])

# it seems that on that row the delimiter is not correct, it uses a semicolon instead of a comma, i
# fix it by replacing the semicolon with a comma manually or by writing a function to do it

# drop the nan row after we manually fixed it
data_df = data_df.dropna()
```

```
                                             Price  Age Mileage FuelType  \
905  8423.0;68;58860.0;Petrol;110;1;0;1600;3;1055;CHF  NaN     NaN      NaN

     HP  MetColor  Automatic  CC  Doors  Weight Currency
905 NaN       NaN        NaN NaN    NaN     NaN      NaN
```

In [4]:
```python
nan_index = data_df.isnull().any(axis=1)
print(data_df[nan_index])
```

```
Empty DataFrame
Columns: [Price, Age, Mileage, FuelType, HP, MetColor, Automatic, CC, Doors, Weight, Currency]
Index: []
```

1.3 (4 pts): Compute the mean, median of the `Price` column.

- Compute the mean and median of the `Price` column. If you encounter error, try to understand why this error is happening and propose a solution.

Hint: Is all values in the `Price` column numerical?

In [5]:
```python
# directly apply mean will not work because the column is not numeric due to the £ symbol
try:
    data_df['Price'].mean()
except Exception as e:
    print(e)

# list the unique values in the column to see what is wrong
# print(data_df['Price'].unique())

# 2 issues:
# 1. there is some values with £ symbol (-1 point if not found this issue)
#  they use comma as decimal separator instead of dot (-1 point if not found this issue)
```


```
Could not convert string '13500.011878.012050.012914.011878.011187.014599.016068.018572.011187.0180
97.017233.016931.018572.019436.019005.019652.015506.016750.014642.013778.014642.015950.016950.01403
7.013778.015112.013606.014642.015506.011187.013778.012914.015,398,49999999999813606.013778.012914.0
13606.012742.012050.014470.012050.014642.014642.016413.015506.013649.015506.018961.015506.013606.01
7709.018961.013390.011446.013173.013173.016370.013821.012914.014253.016197.015506.015506.016950.016
370.012914.019220.015950.013778.011226.016370.013606.017233.014642.016197.015938.014594.012871.0163
70.017250.013347.015506.014383.015074.012871.015506.013778.018961.014210.019220.017233.013778.01890
0.017233.013778.015950.016197.015074.016404.014037.018500.015981.016802.014642.016240.015074.01550
6.028074.026779.027016.021552.021552.019825.021588.021950.015463.016629.019220.016370.017233.01412
4.016370.014642.018789.013778.014253.015506.013691.014037.013778.014037.013778.014253.014253.01593
8.014037.019869.017190.014210.020689.017233.015981.016370.014210.017709.021164.019450.018097.01720
0.017233.018450.016845.018789.014571.016845.016327.017061.017061.016370.017925.016931.016845.01765
0.017233.017233.018097.020500.015372.015761.020516.016845.016370.018961.019,817.94999999999716370.0
17233.018961.022500.015981.016153.018249.018572.017795.015761.06004.08207.010323.06694.011950.0375
8.04103.010150.011446.010323.010280.012742.08595.010323.09930.09718.010500.09027.011187.09934.01079
8.010950.09890.010323.011446.012742.010185.09890.013500.09459.011662.09459.09459.011187.010323.0107
55.010323.014,851.09999999999912450.011187.010323.011690.010755.011014.010301.011187.011950.011144.
010280.010064.010950.013950.012050.010323.09459.012450.010323.011662.010098.011662.010323.011144.01
1662.010150.010150.09372.010150.012914.08587.011144.011662.010150.010323.011619.010323.010793.01166
2.012750.010366.010323.010793.012450.012742.09459.011662.012950.013500.013450.011662.09917.011619.0
9930.012750.012949.011187.011187.011100.012050.010323.011187.010107.09950.011895.011187.010798.0119
86.09070.010621.012050.010877.84999999999911187.011100.012090.011878.011014.010798.012050.09934.013
950.010275.08595.011662.09890.010755.011187.012090.010150.010064.08595.012050.011187.09459.08552.01
0323.011990.09287.012050.09718.011187.010323.09459.011187.011187.010103.09502.012050.010323.010150.
010711.010798.011144.010539.011014.010323.010280.010323.014,851.09999999999910323.011187.012914.011
619.011878.011187.011014.011895.08595.012450.010798.012914.011878.010967.012949.011014.012396.01118
7.09934.010323.013450.011144.09070.09459.010323.09890.011446.08854.012090.010323.011446.011187.0101
50.011,424.09999999999911662.05615.05529.06047.06694.07689.07343.07731.08552.08854.07990.06694.0816
3.06694.07127.08595.03844.08595.07774.08595.010755.09070.010750.08950.09070.04449.09459.08163.0859
5.09459.010280.08595.010323.06825.010,877.8499999999997731.010323.010900.08595.09950.010950.07731.0
7731.09070.07731.07990.09934.08423.08595.09890.010798.09070.011187.010539.09459.09459.09752.09287.0
9411.09934.08423.08854.010798.010323.010750.09890.011,871.44999999999910150.09459.010323.09498.0989
0.07689.09070.011750.09890.010366.010323.08509.09459.09459.07511.09493.08207.010323.09287.07559.075
59.09890.08595.011,871.44999999999912050.011250.09416.08423.08595.010323.012,017.4999999999987731.0
10250.09,864£9070.010323.09934.09934.09890.09,834£8207.09070.09287.08950.010366.08587.09459.09287.0
8465.010150.010323.09718.010150.010950.011250.08595.09700.010323.09900.08630.08185.09934.010323.099
34.09070.010900.011700.010280.012050.09459.010500.09287.010323.08638.09066.09890.08120.011950.0833
6.016370.09890.08854.09890.08595.09070.013,659.69999999999998595.08854.09372.011,816.249999999998111
87.010323.09287.08250.09459.010150.09459.010755.09070.09416.011187.09459.010798.08595.09287.010798.
09027.09287.011187.09498.010323.09718.010323.011878.09502.013500.09459.010750.07731.011187.08423.09
416.09498.09287.09459.011230.011187.09934.09459.010115.08621.010582.09934.010323.09934.011900.01030
6.010500.07731.09027.09070.011187.09950.011144.08595.08595.09459.09459.08595.09459.09330.09070.0902
7.010600.09027.012950.09718.06479.07731.06004.06825.05140.06479.06479.05961.04968.06950.06868.0775
0.06868.07127.05399.08207.05961.07299.06350.07731.05961.07731.07559.06868.07731.07731.07731.06868.0
6694.06479.07731.08466.08163.07731.08750.09070.06907.09027.08595.08595.07731.06868.010950.08595.074
29.06263.06004.08000.08595.08163.07,898.2£8163.08595.06004.08250.08595.07127.08595.09070.06868.0842
3.07990.08207.08595.06694.08207.08595.08423.08423.05140.08500.07338.07990.05961.07731.08207.07990.0
8548.08595.06868.07559.07127.07731.07731.07731.07731.08163.08950.07990.07559.08595.08595.08552.0773
1.07731.010582.07990.08854.07731.06910.09900.07127.09070.07299.08552.09070.07343.08207.08163.08552.
08250.08750.08207.07299.07334.07559.08423.08595.07343.07299.06825.07559.06868.05961.07040.09498.076
89.08207.08950.09934.09027.09070.07990.09459.09027.07990.07559.07559.08466.07127.09027.09750.07731.
06435.08450.07127.09459.08950.07559.09027.09459.07299.08595.08595.08750.08595.08207.07127.09459.068
68.07127.08548.07887.09459.09070.07689.08207.09459.06868.06868.06868.08207.09459.08423.08423.08163.
08950.07731.07689.08595.07559.07731.08990.07731.07731.08595.08552.07731.07731.08207.06868.06694.077
31.010323.07731.010323.07731.08595.07127.07770.08634.07990.09459.07689.08207.09950.06868.07127.0816
3.07950.08207.07731.09459.07731.08163.06910.08552.07731.09416.08163.07299.08207.09070.06004.08423.0
6004.08336.08595.07990.07731.09900.09498.09070.08950.08423.09459.08423.08595.09070.08984.07602.0105
00.07990.06738.09950.09287.09459.08595.08595.08100.07990.06738.07774.07731.07731.08595.07343.06907.
07559.08423.09459.08423.07084.07731.08207.09459.08423.08595.09027.08509.08163.08893.09750.07731.085
52.07559.08595.07343.08595.09459.08461.07559.07127.08595.08595.09070.08423.09718.07689.07950.09934.
08163.06907.09498.07731.08250.07731.07731.08207.06479.07731.08595.08423.08163.08595.07731.07127.066
94.07731.08595.08163.011187.08595.09066.08868.09070.07731.09900.08634.08595.09,884.257127.08595.077
31.09459.07990.08595.08634.07559.09459.010350.08595.07731.09459.09459.08950.07689.08854.08163.0799
0.09070.06350.08854.08854.09934.07731.06694.08207.07299.08950.07256.07990.07689.07559.09459.07731.0
7731.07684.08111.08750.08423.08595.09459.08552.08207.08595.08595.09070.08595.09066.08207.07343.0104
50.07731.08552.07689.07554.07559.09287.09750.07645.06868.08163.08595.07127.08595.08634.08595.09066.
06868.07731.09695.06694.08595.08595.08595.08552.09459.07731.07731.07127.08854.07559.08423.010950.09
459.08423.07689.08207.08595.08595.08250.010450.09459.07990.08,842.3499999999999416.08423.010323.068
25.09416.09027.09459.08595.09287.08552.09287.09027.08163.09287.09459.08638.09070.010798.07731.0907
0.07986.010950.08207.09416.09459.08163.05096.06004.05183.04535.03801.05830.08,449.9995312.06004.049
67.07559.05615.05140.06500.09070.08500.06868.05010.05830.07,992.4999999999994958.05658.07731.05830.
06868.06868.06435.06694.05571.06825.05961.04837.05140.06004.06868.08,890.656868.09500.07429.05140.0
6694.06868.06004.06263.05615.06263.07990.07127.06263.04535.06825.05961.06825.06263.07299.06868.0686
8.06,406.656650.06868.06263.06435.06868.06263.07127.07731.06694.07343.06694.04967.05961.05615.0561
5.07600.07731.06435.07,302.4999999999996694.06444.07990.06263.05615.05874.07515.06479.06694.06868.0
8595.05736.07559.06694.05140.05830.05615.08750.06868.05830.06868.06868.08950.06694.05571.06900.0729
9.05830.06954.08207.06694.07343.06733.05607.06868.05550.07731.06004.07559.07299.06868.06868.07689.0
7731.07689.06868.05,610.06263.07990.05745.06038.07750.06004.06263.08595.07990.07200.07127.07731.071
27.05399.08552.07300.07731.05615.06868.06435.06868.07343.06004.06868.05830.07299.06479.07559.07559.
07731.06435.07948.07731.07850.06004.06220.06435.05830.06479.06435.06868.07250.06868.07127.07127.058
30.08750.07731.07343.07127.08163.06868.06825.08500.07127.07127.07731.06868.06004.06263.06694.07515.
07731.05830.06479.06868.07731.06868.06435.05140.06694.06950.07559.07731.06474.06694.06950.06902.062
63.06900.07750.06263.07731.07343.06868.06004.06910.06868.07343.06470.08250.07990.06868.08207.07731.
08450.06907.07559.05140.07343.08950.07559.06565.06172.07299.05140.07774.06263.06392.07602.06694.064
79.07299.06392.06479.07731.07731.06868.06868.06868.08950.06694.07127.06868.06694.07343.06004.06479.
06694.05140.08,625.07731.06868.05961.06435.07343.06479.07559.06479.08595.07950.08595.07990.07990.08
595.05140.06907.07731.06435.07127.07343.07343.08120.06263.07731.08163.07343.05830.06392.07731.06,85
5.156694.08950.06004.07338.08638.06046.07731.07343.07559.06478.07774.07731.06004.08500.07299.07731.
07127.06868.06435.07774.06176.06694.05050.08500.07127.09,387.4499999999995830.07256.07900.07950.075
59.08552.05610.07127.05961.07500.06868.07127.07731.09,686.4499999999994,089.99997127.07689.07731.05
830.07950.07429.06694.06738.07559.08207.06694.08595.06694.05140.09459.08163.07127.08423.07450.0755
9.07559.07343.07731.06479.06263.06435.07559.08466.07731.08950.06435.07731.09070.06047.07343.0
7750.07731.07127.07990.06825.08500.06868.08595.07559.06479.06004.07731.07559.06694.07299.07040.0734
3.06565.06868.06694.06868.08595.07731.07299.07731.07299.06479.09369.07343.06263.06004.0' to numeric
```


In [6]:
```python
def convert_raw_price_to_price(raw_price:str)->float:
    if raw_price.endswith('£'):
        raw_price = raw_price.replace('£', '')
    # some numbers have comma as thousand separator
    raw_price = raw_price.replace(',', '')
    return float(raw_price)

# Convert the price column to float
data_df['Price'] = data_df['Price'].apply(convert_raw_price_to_price)

mean = data_df['Price'].mean()
median = data_df['Price'].median()

print(f'Mean: {mean}, Median: {median}')
```

```
Mean: 10730662030331.08, Median: 8595.0
```

In [7]:
```python
# The huge difference implies that there may be an outlier that is skewing the mean
# Let's sort the prices and see if there are any outliers
sorted_prices = data_df['Price'].sort_values()
print(sorted_prices)
```

```
190     3.758000e+03
1047    3.801000e+03
392     3.844000e+03
1368    4.090000e+03
191     4.103000e+03
            ...
178     2.250000e+04
109     2.677900e+04
110     2.701600e+04
108     2.807400e+04
33      1.539850e+16
Name: Price, Length: 1435, dtype: float64
```

In [8]:
```python
# remove the last value as it is an outlier
data_df = data_df[data_df['Price'] != sorted_prices.iloc[-1]]
new_mean = data_df['Price'].mean()
new_median = data_df['Price'].median()
print(f'New Mean: {new_mean}, New Median: {new_median}') # if the student didn't compute the new me
```

```
New Mean: 9431.730438563458, New Median: 8595.0
```

1.4 (4 pts): Convert Units

You notice that some prices are in CHF (Swiss Francs), while others are in EUR (Euros) or GBP (British Pounds). Additionally, for cars priced in GBP, the mileage is in miles rather than kilometers.

For consistency, convert all prices to CHF and all distances to kilometers.

- Exchange rates:
  - 1 CHF = 1.05 EUR
  - 1 GBP = 1.15 CHF
  - 1 mile = 1.61 km

Make the following conversions:

1. Convert prices in EUR or GBP to CHF, rounding to the nearest integer.
2. Convert distances in miles (for GBP cars) to kilometers, rounding to the nearest integer.
3. Drop the 'Currency' column.
4. Calculate the min, mean, median and max of the 'Price' and 'Distance' columns after the conversion.

In [9]:
```python
# load the cleaned data

data_df = pd.read_csv('data/Task1-2.ToyotaCorolla-clean.csv')

# shuffle the data and save it

data_df = data_df.sample(frac=1)

data_df.to_csv('data/Task1-2.ToyotaCorolla-clean-shuffled.csv', index=False)

# Convert the prices to CHF, round to the nearest integer
data_df['Price'] = np.where(data_df['Currency'] == 'EURO', data_df['Price']/1.05, data_df['Price'])
data_df['Price'] = np.where(data_df['Currency'] == 'GBP', data_df['Price']*1.15, data_df['Price'])
data_df['Price'] = np.round(data_df['Price'])

# Convert the distance to kilometers
data_df['Mileage'] = np.where(data_df['Currency'] == 'GBP', data_df['Mileage']*1.61, data_df['Milea
data_df['Mileage'] = np.round(data_df['Mileage'])

# Drop the 'Currency' column
data_df = data_df.drop(columns=['Currency'])

# Calculate the min, mean, median and max of the 'Price' and 'Distance' columns after the conversio
print(data_df[['Price', 'Mileage']].agg(['min', 'mean', 'median', 'max']))


# the correct values should be below, if Milege/Price column does not have the correct values, the
```

```
               Price        Mileage
min      3758.000000       1.000000
mean     9423.536212   68533.259749
median   8595.000000   63389.500000
max     28074.000000  243000.000000
```

1.5 (2 pts): Analyze Average Price

A. Print the average price for each fuel type. Determine which fuel type has the highest average price.

B. Print the average price for different numbers of doors. Determine which number of doors has the highest average price.

In [10]:
```python
print(data_df.groupby('FuelType')['Price'].mean())

print(data_df.groupby('Doors')['Price'].mean())

# 1 point should be deducted if the student does not use groupby function, even though the result i
```

```
FuelType
CNG       8383.235294
Diesel    9878.619355
Petrol    9381.722310
Name: Price, dtype: float64
Doors
2     6997.500000
3     8898.673633
4     8604.384058
5    10082.823442
Name: Price, dtype: float64
```

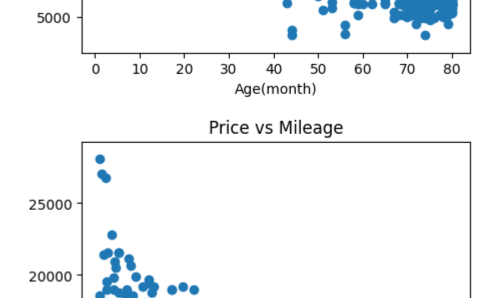
1.6 (2 pts): Relationship Between Car Age and Price

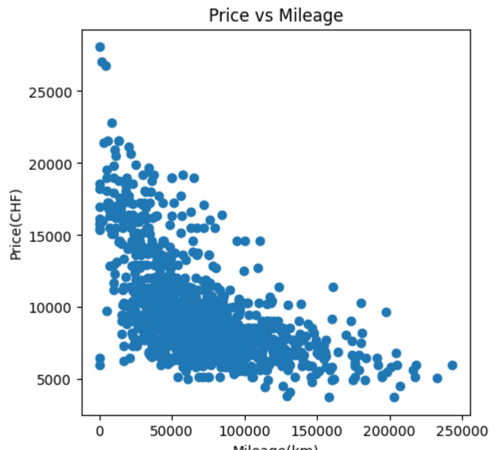
It is intuitive that an older car tends to be cheaper, and a car with more mileage might also be less expensive.

To explore this intuition, create two scatter plots:

1. Car Age vs Price
2. Mileage vs Price

In [11]:
```python
plt.figure(figsize=(5,5))
plt.scatter(data_df['Age'], data_df['Price'])
plt.title('Price vs Age')
plt.xlabel('Age(month)')
plt.ylabel('Price(CHF)')
plt.show()



plt.figure(figsize=(5,5))
plt.scatter(data_df['Mileage'], data_df['Price'])
plt.title('Price vs Mileage')
plt.xlabel('Mileage(km)')
plt.ylabel('Price(CHF)')
plt.show()

# we don't grade based on the style of the image but if there is missing labels, title, or not unit
```





1.7 (4 pts): Correlation Between Price and Mileage

The relationship between car price and mileage appears non-linear, with a steeper price drop initially followed by a flatter curve.

A.(2 pts) Calculate both the Pearson and Spearman correlations between the price of the car and the distance driven.

In [12]:
```python
# Calculate Pearson and Spearman correlations between Price and Mileage
pearson_corr = data_df[['Price', 'Mileage']].corr(method='pearson')['Price']['Mileage']
spearman_corr = data_df[['Price', 'Mileage']].corr(method='spearman')['Price']['Mileage']

print("Pearson Correlation:\n", pearson_corr)
print("Spearman Correlation:\n", spearman_corr)

# if the correlation values are not correct, 1 point should be deducted
```

```
Pearson Correlation:
 -0.5634156654606507
Spearman Correlation:
 -0.6053559019304244
```

B.(2 pts) Which correlation value is higher? Does this result align with your expectations?

Your Response: Spearman correlation is higher than Pearson correlation, which means that the relationship between Price and Mileage is non-linear. This is aligned with our observation from the scatter plot.

Part 2 Linear Regression (30 pts)

You want to build a linear regression model to predict the price of a car based on the features you have.

2.0 (8 pts) Helper functions

Before building the linear regression model, you need to implement some helper functions.

Implement the `accuracy`, `precision`, `recall` and `f1_score` functions.

1. These functions should take in the true labels(`np.array`) and the predicted labels(`np.array`) and return the corresponding metric.
2. They should follow the convention that the positive class is 1 and the negative class is 0.
3. Apply the functions to the following data:

```python
true_labels = np.array([1, 0, 1, 1, 0, 1, 0, 0, 1, 0])
predicted_labels = np.array([1, 1, 1, 1, 0, 0, 1, 0, 1, 0])
```

- Compare the results with the implementation in `sklearn` and see if they match.

In [13]:
```python
# implement the accuracy, precision, recall, and F1-score functions using vectorized operations wit

def accuracy_fn(y_true, y_pred):
    return np.mean(y_true == y_pred)

def precision_fn(y_true, y_pred):
    tp = np.sum(np.logical_and(y_true == 1, y_pred == 1))
    fp = np.sum(np.logical_and(y_true == 0, y_pred == 1))
    if tp + fp == 0:
        return 0.0  # Define precision as 0 when no positive predictions
    return tp / (tp + fp)

def recall_fn(y_true, y_pred):
    tp = np.sum(np.logical_and(y_true == 1, y_pred == 1))
    fn = np.sum(np.logical_and(y_true == 1, y_pred == 0))
    if tp + fn == 0:
        return 0.0  # Define recall as 0 when no positive true labels
    return tp / (tp + fn)

def f1_score_fn(y_true, y_pred):
    prec = precision_fn(y_true, y_pred)
    rec = recall_fn(y_true, y_pred)
    if prec + rec == 0:
        return 0.0  # Define F1-score as 0 when both precision and recall are 0
    return 2 * prec * rec / (prec + rec)

#  Comments:
# if not using vectorized operations, 4 pts will be deducted
# if not using safe division, 2 pts will be deducted

# Test the functions
true_labels = np.array([1, 0, 1, 1, 0, 1, 0, 0, 1, 0])
predicted_labels = np.array([1, 1, 1, 1, 0, 0, 1, 0, 1, 0])

print("Accuracy:", accuracy_fn(true_labels, predicted_labels))
print("Precision:", precision_fn(true_labels, predicted_labels))
print("Recall:", recall_fn(true_labels, predicted_labels))
print("F1-Score:", f1_score_fn(true_labels, predicted_labels))


# see if the results are correct with sklearn
import sklearn.metrics

print("Accuracy:", sklearn.metrics.accuracy_score(true_labels, predicted_labels))
print("Precision:", sklearn.metrics.precision_score(true_labels, predicted_labels))
print("Recall:", sklearn.metrics.recall_score(true_labels, predicted_labels))
print("F1-Score:", sklearn.metrics.f1_score(true_labels, predicted_labels))


# 4 points should be removed if the student does not consider the case where the denominator is 0
```

```
Accuracy: 0.7
Precision: 0.6666666666666666
Recall: 0.8
F1-Score: 0.7272727272727272
Accuracy: 0.7
Precision: 0.6666666666666666
Recall: 0.8
F1-Score: 0.7272727272727272
```

2.1 (6 pts) Preprocess the Data

To prepare your data for building a linear regression model, complete the following steps:

A.(1 pts) Convert the categorical variables to one-hot encoding using the `pd.get_dummies()` function, how many columns do you have after the one-hot encoding? (P.S. You may want to avoid introducing multicollinearity with one-hot encoding, what should you do to avoid this?)

In [14]:
```python
data_df = data_df.sort_index()
```

In [15]:
```python
data_df_processed = pd.get_dummies(data_df, columns=['FuelType', 'MetColor', 'Automatic', 'Doors'],

# if they condiered the 'Doors' column not as categorical, it's also correct.
data_df_processed_door_non_categorical = pd.get_dummies(data_df, drop_first=True)

# if drop_first not set, they will lose 1 point

print(f"Columns in the processed data: len={len(data_df_processed.columns)} or len={len(data_df_pro
```

```
Columns in the processed data: len=13 or len=11
```

In [16]:
```python
print(data_df_processed.columns)

print(data_df_processed_door_non_categorical.columns)
```

```
Index(['Price', 'Age', 'Mileage', 'HP', 'CC', 'Weight', 'FuelType_Diesel',
       'FuelType_Petrol', 'MetColor_1', 'Automatic_1', 'Doors_3', 'Doors_4',
       'Doors_5'],
      dtype='object')
Index(['Price', 'Age', 'Mileage', 'HP', 'MetColor', 'Automatic', 'CC', 'Doors',
       'Weight', 'FuelType_Diesel', 'FuelType_Petrol'],
      dtype='object')
```

B.(1 pts) Split the data into features (X) and target (y) variables. The target variable is the 'Price' column. Then split the data into train test sets using a 80-20 split. Use `random_state=42` for reproducibility. How many samples are in the training set and how many samples are in the test set?

In [17]:
```python
X = data_df_processed.drop(columns=['Price'])
y = data_df_processed['Price']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"No. of training samples: {len(X_train)} No. of testing samples: {len(X_test)}")
```

```
No. of training samples: 1148 No. of testing samples: 288
```

C.(1 pts) Why do we split the data into only train-test sets but not train-validation-test sets?

Your Response: We split the data into train-test sets to train the model on the training set and evaluate it on the test set. The validation set is used to tune the hyperparameters of the model, and we don't need it here because we are not tuning any hyperparameters.

D.(1 pts) **Standardize the Features**: Use `StandardScaler` from `sklearn.preprocessing` and then add a constant column using `sm.add_constant()`. Print the average and standard deviation of the training set after standardization.

In [18]:
```python
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm

scaler_train = StandardScaler()
scaler_train.fit(X_train)

X_train_scaled = pd.DataFrame(scaler_train.transform(X_train), index=X_train.index, columns=X_trair
X_test_scaled = pd.DataFrame(scaler_train.transform(X_test), index=X_test.index, columns=X_test.col


# the student must use the scaler from the training set to scale the test set
# if they use a new scaler for the test set, 1 point should be deducted

# add constant column to both train and test sets

X_train_scaled_const = sm.add_constant(X_train_scaled)


X_test_scaled_const = sm.add_constant(X_test_scaled)

# print the avg and std of the training set
print("Training set mean: ", np.mean(X_train_scaled_const, axis=0))
print("Training set std: ", np.std(X_train_scaled_const, axis=0))
```

```
Training set mean:  const             1.000000e+00
Age                1.485455e-16
Mileage           -1.485455e-16
HP                 3.868373e-17
CC                -2.963174e-16
Weight            -6.854757e-16
FuelType_Diesel    1.005777e-17
FuelType_Petrol    2.321024e-17
MetColor_1        -7.736746e-17
Automatic_1        4.642047e-18
Doors_3           -5.570457e-17
Doors_4           -1.315247e-17
Doors_5           -8.665155e-17
dtype: float64
Training set std:  const              0.0
Age                1.0
Mileage            1.0
HP                 1.0
CC                 1.0
Weight             1.0
FuelType_Diesel    1.0
FuelType_Petrol    1.0
MetColor_1         1.0
Automatic_1        1.0
Doors_3            1.0
Doors_4            1.0
Doors_5            1.0
dtype: float64
```

E.(2 pts) Should we first standardize the data and then split it into train and test sets or vice versa? why?

Your Response: We should first split the data into train and test sets and then standardize the features. This is because we want to avoid data leakage from the test set to the training set. If we standardize the data before splitting, the mean and standard deviation of the entire dataset will be used to standardize the data, which is incorrect.

2.2 (10 pts) Train and Evaluate the Linear Regression Model

To train a linear regression model using and evaluate its performance, follow these steps:

1. (2 pts) Train a linear regression model on the training dataset using `sm.OLS` from `statsmodels`, print the summary of the model using `model.summary()`.

In [19]:
```python
logistic_regression_model = sm.OLS(y_train, X_train_scaled_const).fit()

print(logistic_regression_model.summary())
# normally the student should get R-squared = 0.850 and Adj. R-squared = 0.848, but it's also possi
# as long as the values are between 0.82-0.88, it's acceptable
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.850
Model:                            OLS   Adj. R-squared:                  0.848
Method:                 Least Squares   F-statistic:                     536.0
Date:                Tue, 10 Dec 2024   Prob (F-statistic):               0.00
Time:                        21:30:25   Log-Likelihood:                -9803.5
No. Observations:                1148   AIC:                         1.963e+04
Df Residuals:                    1135   BIC:                         1.970e+04
Df Model:                          12
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const            9441.4983     36.725    257.086      0.000    9369.442    9513.555
Age             -1971.9732     51.805    -38.065      0.000   -2073.618   -1870.328
Mileage          -503.5581     52.696     -9.556      0.000    -606.951    -400.165
HP                476.6196     94.331      5.053      0.000     291.538     661.701
CC               -477.0432    111.211     -4.290      0.000    -695.246    -258.841
Weight           1152.9564     75.770     15.217      0.000    1004.292    1301.621
FuelType_Diesel   444.7077    170.847      2.603      0.009     109.496     779.920
FuelType_Petrol   373.5415    106.631      3.503      0.000     164.325     582.758
MetColor_1         12.8252     37.119      0.346      0.730     -60.005      85.655
Automatic_1        -9.4423     39.024     -0.242      0.809     -86.010      67.125
Doors_3          -148.9949    437.519     -0.341      0.734   -1007.431     709.441
Doors_4           -20.1629    262.086     -0.077      0.939    -534.391     494.065
Doors_5          -214.8473    441.735     -0.486      0.627   -1081.557     651.862
==============================================================================
Omnibus:                      171.243   Durbin-Watson:                   2.076
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2370.654
Skew:                          -0.046   Prob(JB):                         0.00
Kurtosis:                      10.039   Cond. No.                         34.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

2. (2 pts) Evaluate the model on the test dataset using the square root of the mean squared error (RMSE) metric.
   - A. Report the RMSE value.
   - B. Your boss wants to know how far off the model's predictions are from the actual price of the car. What would you tell him? Given a number and explain how you got it.

In [20]:
```python
# Evaluate the model on the test dataset using the square root of the mean squared error (RMSE) met
from sklearn.metrics import mean_squared_error

y_pred = logistic_regression_model.predict(X_test_scaled_const)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("RMSE:", rmse)
```
```
RMSE: 1434.3390492274823
```

Your Response: We may be off by 1434 CHF on average becaue the RMSE is 1434. This means that the model's predictions are off by 1434 CHF on average.

3. (2 pts) Report the R² score on the test dataset and interpret it.

In [21]:
```python
from sklearn.metrics import r2_score

r2 = r2_score(y_test, y_pred)
print("R2 score on test dataset: ", r2)
```
```
R2 score on test dataset:  0.8122727616953431
```

Your Response: The R² score on test set is 0.812, which means that 81.2% of the variance in the target variable (Price) can be explained by the features in the model.

Comments: If the student reports the R² score on the training set, 3 points will be deducted. The remaining 1 point will be deducted if the interpretation is incorrect.

4. (2 pts) Which features are statistically significant at a 5% significance level?

In [22]:
```python
p_values = logistic_regression_model.pvalues

significant_features = p_values[p_values < 0.05].index

# comment:
# Technically, the constant term is not a feature, but we don't deduct points for this.
# If the student report the X1, X2 etc instead of the actual column names, 1 point should be deduct

significant_features = significant_features.drop('const')

print("Significant features: ", significant_features)
```
```
Significant features:  Index(['Age', 'Mileage', 'HP', 'CC', 'Weight', 'FuelType_Diesel',
       'FuelType_Petrol'],
      dtype='object')
```

5. (2 pts) Determine which two feature have the highest coefficient? What does it imply?

Your Response:

- Age and Weight.
- These two features have the most significant impact on the price of the car.

comments:

- constant term is not a feature, 1 point should be deducted if the student includes it in the answer.

2.3 (2 pts): Improvement Discussion

- Suggest a few additional features that could potentially explain this remaining variance in the data (at least 2 features).

Your Response:

possible answers: damage condition, optional upgrade, maintenance history, Previous owner, etc.

But car brand is a **wrong** answer because this dataset only contains data for Toyota Corolla cars ! Car model is fine because Toyota Corolla has different models. If the student suggests car brand, 1 points will be deducted. The remaining 1 point depends on if there is another valid feature suggestion.

2.4 (2 pts): Identifying Confounding Variables

The feature "Weight" shows a very low p-value and a high coefficient, but it doesn't seem to be a major factor for customers buying a second-hand car. You go to your mentor Tim to discuss this issue. Indeed, Tim suggests that never in his career has he seen a customer who asked for the weight of a car before buying it. You suspect that there might be a confounding variable that is correlated with the car's weight and significantly influences its price.

- Suggest a possible confounding variable that may be correlated with the car's weight and significantly influence its price (it doesn't need to be a variable in the dataset). Explain why this variable could be a confounding variable.

Your Response:

possible answers: The Toyota Corolla has different versions, such as the Corolla Hatchback, Corolla Sedan, and Corolla Touring Sports. The weight of the car might be correlated with the version of the car. A heavier version is usually a bigger car with more features, which could explain the higher price.

2.5 (2 pts): Adding an Inverse Mileage Term

From the previous scatter plot, the relationship between car price and mileage appears non-linear, with a steep price drop initially and then a flattening. A suitable approach to model this behavior is by incorporating an inverse term of mileage.

- Add the inverse mileage term to the model and retrain it using the code provided. Print the model summary and interpret the effect of the inverse mileage term.

In [23]:
```python
X['Mileage_inverse'] = 1/X['Mileage']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

X_train = sm.add_constant(X_train)

X_test = sm.add_constant(X_test)

logistic_regression_model = sm.OLS(y_train, X_train).fit()

print(logistic_regression_model.summary())

y_pred = logistic_regression_model.predict(X_test)

r2 = r2_score(y_test, y_pred)

print("R2 score on test dataset: ", r2)

# comment:
# The student can report either the R2 score showed in the summary or the one calculated using the
# The R2 score should be pretty much the same as the one withou the inverse feature. This means tha
# is not bringing any additional information to the model.
```
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.850
Model:                            OLS   Adj. R-squared:                  0.849
Method:                 Least Squares   F-statistic:                     496.1
Date:                Tue, 10 Dec 2024   Prob (F-statistic):               0.00
Time:                        21:30:26   Log-Likelihood:                -9801.7
No. Observations:                1148   AIC:                         1.963e+04
Df Residuals:                    1134   BIC:                         1.970e+04
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       9441.4983     36.685    257.367      0.000    9369.520    9513.476
x1         -1974.6675     51.769    -38.144      0.000   -2076.241   -1873.094
x2          -511.6765     52.819     -9.687      0.000    -615.310    -408.043
x3           472.3240     94.256      5.011      0.000     287.389     657.259
x4          -477.8449    111.091     -4.301      0.000    -695.811    -259.878
x5          1166.8577     76.053     15.343      0.000    1017.636    1316.079
x6           441.3814    170.670      2.586      0.010     106.516     776.247
x7           373.8717    106.515      3.510      0.000     164.883     582.860
x8            13.5550     37.081      0.366      0.715     -59.200      86.310
x9           -12.8931     39.026     -0.330      0.741     -89.464      63.677
x10         -150.5418    437.042     -0.344      0.731   -1008.044     706.961
x11          -17.8593    261.804     -0.068      0.946    -531.533     495.815
x12         -219.9402    441.262     -0.498      0.618   -1085.722     645.842
x13          -70.1493     37.611     -1.865      0.062    -143.943       3.645
==============================================================================
Omnibus:                      167.538   Durbin-Watson:                   2.073
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2236.151
Skew:                          -0.001   Prob(JB):                         0.00
Kurtosis:                       9.837   Cond. No.                         34.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
R2 score on test dataset:  0.8132778753638819
```

Part 3 Supervised Learning (40 pts)

After completing your analysis, you're satisfied with the results. You handed the Jupyter notebook over to your mentor.

(Fun fact: The name "Jupyter" is derived from Julia, Python, and R—three programming languages that the platform was originally designed for.)

Your mentor Tim is very impressed with your work and asks you the following question:

" This looks great! It will be very useful for our sales team. While looking at the results, I realized that there might be one thing that we can improve. For companies like us, it is important to sell the cars quickly. If we are patient, we might be able to sell the car for a higher price, but that's not always the best strategy. We need to consider the maintenance costs for the car, the cash flow and the fact that the price of the car decreases over time."

He then continues: "Three months is a sweet spot for us. If we can sell the car within the first three months, it is great. If not, it is worth considering lowering the price to sell it faster and increase our cash flow. I can ask Ivan from Sales to collect data in the last few months on whether the car was sold within the first three months or not. This would be great if you could have a model that tells us if the car will be sold in the first three months or not. "

This sparks your interest, and soon Ivan has provided you with the new data containing an additional column `sold_within_3_months` which is a binary variable indicating whether the car was sold within the first three months or not.

Note: The data for this part is in the file `Task3.ToyotaCorolla_sales_3months.csv` and it has already unified the currency and distance units.

In [24]:
```python
data_df = pd.read_csv('data/Task3.ToyotaCorolla_sales_3months.csv', index_col=0)
print(data_df.head())
```
```
   Price  Age     KM FuelType  HP  MetColor  Automatic    CC  Doors  Weight  \
0  13500   23  46986   Diesel  90         1          0  2000      3    1165
1  13750   23  72937   Diesel  90         1          0  2000      3    1165
2  13950   24  41711   Diesel  90         1          0  2000      3    1165
3  14950   26  48000   Diesel  90         0          0  2000      3    1165
4  13750   30  38500   Diesel  90         0          0  2000      3    1170

   sold_within_3_months
0                     0
1                     0
2                     0
3                     0
4                     0
```

3.1 (2 pts): Preprocess the Data

- (1 pts) How many cars in the dataset were sold in the first three months, and how many were not?

In [25]:
```python
print(data_df['sold_within_3_months'].value_counts())
```
```
sold_within_3_months
1    880
0    556
Name: count, dtype: int64
```

- (1 pts) Preprocess the categorical variables to one-hot encoding using the `pd.get_dummies()` function.

In [26]:
```python
# TODO

data_df = pd.get_dummies(data_df, columns=['FuelType', 'MetColor', 'Automatic', 'Doors'], drop_firs
# comment:
# deduct 1 point if not using drop_first=True
```

3.2 (20 pts): Logistic Regression Model

1. (2 pts) Split the data into features (X) and target (y) variables. The target variable is the 'sold_within_3_months' column. The `Price` column should be included as a feature.

In [27]:
```python
y_df = data_df['sold_within_3_months']
X_df = data_df.drop(columns=['sold_within_3_months'])
```

2. (2 pts) Then split the data into train test sets using a 80-20 split. Use `random_state=42` for reproducibility.

In [28]:
```python
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.2, random_state=42)
print(f"No. of training samples: {len(X_train)} No. of testing samples: {len(X_test)}")
```
```
No. of training samples: 1148 No. of testing samples: 288
```

3. (2 pts) Standardize the features using `StandardScaler` from `sklearn.preprocessing` and then add a constant column using `sm.add_constant()`.

In [29]:
```python
#

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

X_train = sm.add_constant(X_train)

X_test = sm.add_constant(X_test)
```

4. (2 pts) Fit a logistic regression model on the training dataset.

In [30]:
```python
logistic_regression_model = sm.Logit(y_train, X_train).fit()
print(logistic_regression_model.summary())
```
```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.165914
         Iterations: 35
                           Logit Regression Results
================================================================================
Dep. Variable:     sold_within_3_months   No. Observations:                 1148
Model:                            Logit   Df Residuals:                     1134
Method:                             MLE   Df Model:                           13
Date:                  Tue, 10 Dec 2024   Pseudo R-squ.:                  0.7511
Time:                          21:30:27   Log-Likelihood:                -190.47
converged:                        False   LL-Null:                       -765.10
Covariance Type:              nonrobust   LLR p-value:                 1.448e-237
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.3529     19.015     -0.019      0.985     -37.621      36.915
x1            -9.9849      0.882    -11.325      0.000     -11.713      -8.257
x2            -0.6112      0.319     -1.918      0.055      -1.236       0.013
x3            -0.2607      0.211     -1.235      0.217      -0.674       0.153
x4            -1.5443      0.654     -2.360      0.018      -2.827      -0.262
x5             1.5921      0.683      2.329      0.020       0.252       2.932
x6             0.1231      0.541      0.228      0.820      -0.937       1.184
x7            -2.0212      1.026     -1.970      0.049      -4.033      -0.010
x8             0.0738      0.589      0.125      0.900      -1.081       1.229
x9             0.1427      0.129      1.105      0.269      -0.110       0.396
x10           -0.1535      0.135     -1.134      0.257      -0.419       0.112
x11           -6.2822   5403.705     -0.001      0.999   -1.06e+04    1.06e+04
x12           -3.7054   3212.435     -0.001      0.999   -6299.962    6292.551
x13           -6.3805   5448.479     -0.001      0.999   -1.07e+04    1.07e+04
==============================================================================

Possibly complete quasi-separation: A fraction 0.20 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
/Users/saibo/miniconda3/envs/ada_hw2_dryrun/lib/python3.10/site-packages/statsmodels/base/model.py:
607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

5. (2 pts) Evaluate the model on the test dataset using the accuracy score metric. Report the accuracy score.

In [31]:
```python
y_pred_p_logistic = logistic_regression_model.predict(X_test)
y_test_pred_logistic = np.where(y_pred_p_logistic > 0.5, 1, 0)


print(accuracy_fn(y_test, y_test_pred_logistic))
# Accuracy between 0.91-0.95 gives full points
```
```
0.9305555555555556
```

6. (2 pts) Calculate the precision, recall, and F1-score.

In [32]:
```python
precision = precision_fn(y_test, y_test_pred_logistic)
recall = recall_fn(y_test, y_test_pred_logistic)
f1 = f1_score_fn(y_test, y_test_pred_logistic)

print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)

# Comment:
# Any difference within 1% is considered correct. Otherwise, 1 point should be deducted for each me
```
```
Precision: 0.9375
Recall: 0.9482758620689655
F1-Score: 0.9428571428571428
```

7. (2 pts) Suppose that your company is running short on cash flow and needs to sell the cars quickly. How should you adjust the threshold for the logistic regression model to ensure that the company can sell the cars as quickly as possible?
   - A. Increase the threshold
   - B. Decrease the threshold

In a more general sense, how does the choice of threshold affect the precision and recall of the model?

Your Response:

- Increase the threshold will make the model predict fewer positive cases(cars will be sold within 3 months). For all the positive cases that the model predicts, the sales team will apply a discount to sell the cars quickly. So we should decrease the threshold to ensure that the company can sell the cars as quickly as possible.

However, this question has been asked in a confusing way. The student may answer the question in a different way. So we give full points for both increasing and decreasing the threshold as long as the explanation is reasonable.

8. (6 pts) Use binary search to find the optimal threshold that maximizes the F1-score. Implement a binary search algorithm to find the threshold that maximizes the f1-score of the logistic regression model on the training set. The search interval should be between 0 and 1, and the stopping criterion is 10 iterations. What is the optimal threshold and what difference does the optimal threshold make in the F1-score?

In [33]:
```python
def get_f1_score(y_test, y_pred_p, threshold):
    y_pred = np.where(y_pred_p > threshold, 1, 0)
    return f1_score_fn(y_test, y_pred)

def binary_search_threshold(y_test, y_pred_p, low, high, iterations):
    for i in range(iterations):
        mid = (low + high) / 2
        f1_score_mid = get_f1_score(y_test, y_pred_p, mid)
        f1_score_high = get_f1_score(y_test, y_pred_p, high)
        if f1_score_mid > f1_score_high:
            high = mid
        else:
            low = mid
    return mid

threshold = binary_search_threshold(y_test, y_pred_p_logistic, 0, 1, 10)
print("Threshold:", threshold)

y_test_pred_logistic = np.where(y_pred_p_logistic > threshold, 1, 0)

optimal_f1 = f1_score_fn(y_test, y_test_pred_logistic)
print("Optimal F1-Score:", optimal_f1- f1)

# comment:
# This question is flawed. Any attempt to solve this question should be given full points.
```
```
Threshold: 0.4677734375
Optimal F1-Score: 0.00032467532467539417
```

3.3(18 pts) Decision Tree Model

Use a Decision Tree model from `sklearn` to predict whether a car will be sold within the first three months.

Follow these steps to complete the task:

1. (2 pts) Train a Decision Tree Classifier to predict the target variable (`sold_within_3_months`).You can reuse the train and test sets from the previous section. Set `random_state=42` for reproducibility in `DecisionTreeClassifier`.

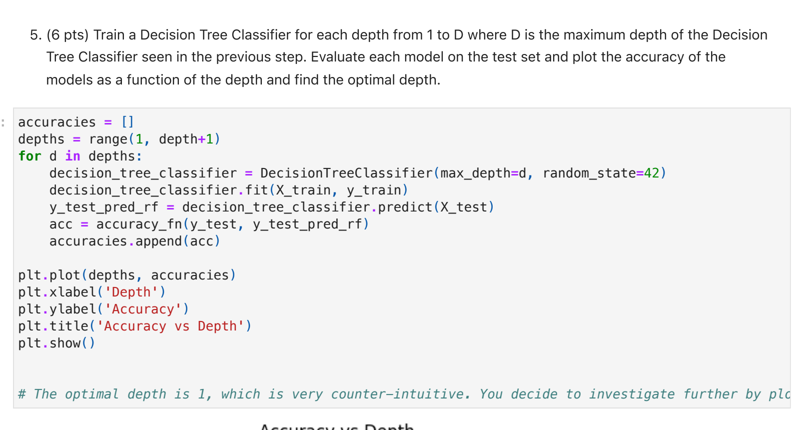
In [34]:
```python
from sklearn.tree import DecisionTreeClassifier

decision_tree_classifier = DecisionTreeClassifier(random_state=42)

decision_tree_classifier.fit(X_train, y_train)
```
Out[34]:
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

2. (2 pts) Evaluate the model on the test set and report the depth of the tree.

In [35]:
```python
# This recall looks not very promising, you wonder if you had a bug in your case.
# you decide to calculate a baseline which all predictions are randomly 0 or 1 with the same probab
y_test_pred_rf = decision_tree_classifier.predict(X_test)

acc = accuracy_fn(y_test, y_test_pred_rf)
depth = decision_tree_classifier.get_depth()
# comments:
# Any difference within 1% is considered correct. Otherwise, 1 point should be deducted for each me

print(f"Accuracy: {acc}, Depth: {depth}")

# 1% difference is acceptable, otherwise 1 point should be deducted for each metric, max 2 points d
# depth should be 16, if not, check what could be wrong, if issue comes from incorrect application
```
```
Accuracy: 0.9201388888888888, Depth: 16
```

3. (2 pts) Visualize the Decision Tree

In [36]:
```python
# Visualize the decision tree
from sklearn.tree import plot_tree
plt.figure(figsize=(20, 10))
plot_tree(decision_tree_classifier, filled=True)
plt.show()
```

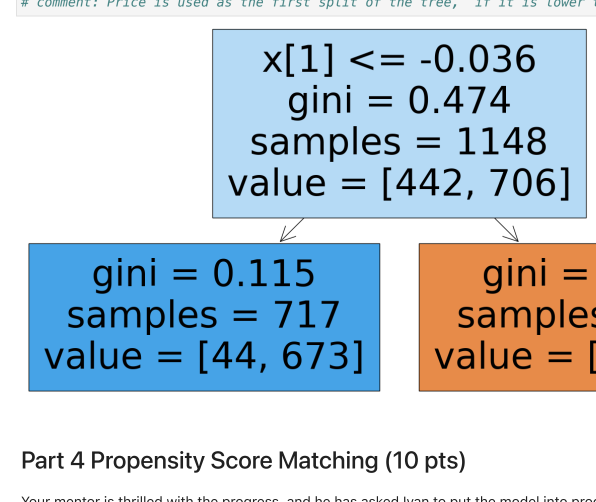


4. (2 pts) Retrain the Decision Tree Classifier with a maximum depth of 8 and evaluate it on the test set. Compare and explain the results.

In [37]:
```python
# train a decision tree with max_depth=8
decision_tree_classifier_depth_5 = DecisionTreeClassifier(max_depth=8, random_state=42)
decision_tree_classifier_depth_5.fit(X_train, y_train)

y_test_pred_rf_depth_5 = decision_tree_classifier_depth_5.predict(X_test)

print(accuracy_fn(y_test, y_test_pred_rf_depth_5))

# comments:
# the tree with max_depth=8 should have a better performance than the one with the default depth=16
# this is because the tree with max_depth=8 is less likely to overfit the training data.
```
```
0.9236111111111112
```

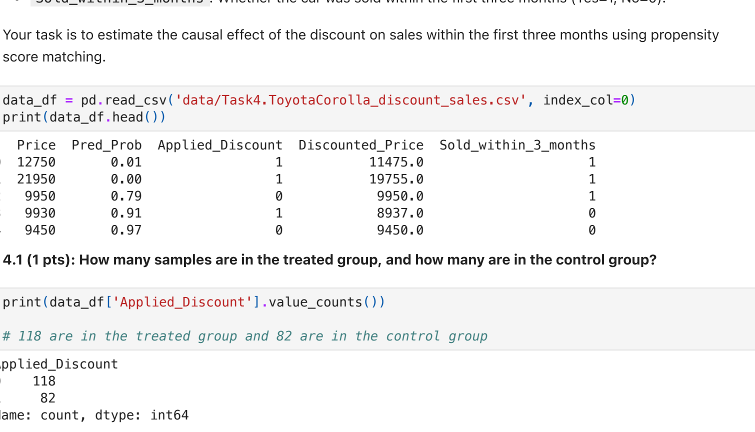
5. (6 pts) Train a Decision Tree Classifier for each depth from 1 to D where D is the maximum depth of the Decision Tree Classifier seen in the previous step. Evaluate each model on the test set and plot the accuracy of the models as a function of the depth and find the optimal depth.

In [38]:
```python
accuracies = []
depths = range(1, depth+1)
for d in depths:
    decision_tree_classifier = DecisionTreeClassifier(max_depth=d, random_state=42)
    decision_tree_classifier.fit(X_train, y_train)
    y_test_pred_rf = decision_tree_classifier.predict(X_test)
    acc = accuracy_fn(y_test, y_test_pred_rf)
    accuracies.append(acc)

plt.plot(depths, accuracies)
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Depth')
plt.show()


# The optimal depth is 1, which is very counter-intuitive. You decide to investigate further by plo
```



6. (4 pts) Train a decision tree of depth = 1 , visualize the tree and explain what is the decision rule at the root node.

In [39]:
```python
decision_tree_classifier_depth_1 = DecisionTreeClassifier(max_depth=1, random_state=42)
decision_tree_classifier_depth_1.fit(X_train, y_train)

plt.figure(figsize=(20, 10))
plot_tree(decision_tree_classifier_depth_1, filled=True)
plt.show()

# comment: Price is used as the first split of the tree,  if it is lower than -0.036, the car is cl
```



Part 4 Propensity Score Matching (10 pts)

Your mentor is thrilled with the progress, and he has asked Ivan to put the model into production. Based on the model's prediction, the sales manager Ivan will decide whether to lower the car's price by 5%.

A new quarter has passed, and Ivan has collected updated sales data, which includes the following columns:

- `Price` : The initial price of the car.
- `Pred_Prob` : The predicted probability of the car being sold within the first three months.
- `Applied_Discount` : Whether the discount was applied (Yes=1, No=0).
- `Discounted_Price` : The car's final price, calculated as `Price * 95%` if the discount was applied; otherwise, it's equal to `Price` .
- `Sold_within_3_months` : Whether the car was sold within the first three months (Yes=1, No=0).

Your task is to estimate the causal effect of the discount on sales within the first three months using propensity score matching.

In [40]:
```python
data_df = pd.read_csv('data/Task4.ToyotaCorolla_discount_sales.csv', index_col=0)
print(data_df.head())
```
```
   Price  Pred_Prob  Applied_Discount  Discounted_Price  Sold_within_3_months
0  12750       0.01                 1           11475.0                     1
1  21950       0.00                 1           19755.0                     1
2   9950       0.79                 0            9950.0                     1
3   9930       0.91                 1            8937.0                     0
4   9450       0.97                 0            9450.0                     0
```

4.1 (1 pts): How many samples are in the treated group, and how many are in the control group?

In [41]:
```python
print(data_df['Applied_Discount'].value_counts())

# 118 are in the treated group and 82 are in the control group
```
```
Applied_Discount
0    118
1     82
Name: count, dtype: int64
```

4.2 (5 pts): Propensity Score Matching

- The propensity score is the predicted probability of the car being sold within the first three months from the logistic regression model, i.e. `Pred_Prob` column in the `Task4.ToyotaCorolla_discount_sales.csv` file. Create pairs of matched samples as follows:
  - For each treated sample (discount applied), find a control sample (discount not applied) with a difference in propensity score of less than 0.05.
  - If there is more than one control sample for a treated sample, choose the control sample with the smallest difference in propensity score.
  - If there is no control sample satisfying the condition, discard the treated sample.
  - How many successful matches did you get?

Notice that your output should be 1-to-1 matching, meaning that each treated sample should be matched with at most one control sample. So each sample can only appear once in the pairs.

In [42]:
```python
# TODO
treatment_df = data_df[data_df['Applied_Discount'] == 1]
control_df = data_df[data_df['Applied_Discount'] == 0]

print(f"There are {treatment_df.shape[0]} samples in the treated group.")
print(f"There are {control_df.shape[0]} samples in the control group.")

import networkx as nx
def get_similarity(propensity_score1, propensity_score2):
    '''Calculate similarity for instances with given propensity scores'''
    return 1 - np.abs(propensity_score1 - propensity_score2)

G = nx.Graph()

for treatment_id, treatment_row in treatment_df.iterrows():
    for control_id, control_row in control_df.iterrows():
        similarity = get_similarity(control_row['Pred_Prob'], treatment_row['Pred_Prob'])
        if similarity>0.95:
            G.add_weighted_edges_from([(control_id, treatment_id, similarity)])

matching = nx.max_weight_matching(G, maxcardinality=True)
print(f"We have {len(matching)} successful matches!")

# comment:
# the networkx library is the correct library to use for this task and it was used in the lab sessi
# Technically, the student can use other libraries to solve this task, but most often they use a gr
# We don't deduct points for the argument "max_cardinality=True" in the nx.max_weight_matching func
```
```
There are 82 samples in the treated group.
There are 118 samples in the control group.
We have 49 successful matches!
```

4.3 (4 pts): Average Treatment Effect (ATE)

Now let's estimate the effect of the discount on sales.

For each matched pair, there is one treated sample and one control sample. They may have different outcomes and we can calculate the average treatment effect (ATE) as

$$ATE = \frac{1}{N}\sum_{i}^{N} y_{treat}^{(i)} - y_{control}^{(i)}$$

where $y_{treat}^{(i)}$ and $y_{control}^{(i)}$ are the outcomes for the treated and control samples, respectively.

Notice that here the outcome is a simple binary variable, which is whether the car was sold within the first three months or not.

- (3 pts) Calculate the ATE based on the matched pairs and report the result

In [43]:
```python
ordered_matching = []

for control_id, treatment_id in matching:
    if control_id in control_df.index and treatment_id in treatment_df.index:
        ordered_matching.append((control_id, treatment_id))
    elif treatment_id in control_df.index and control_id in treatment_df.index:
        ordered_matching.append((treatment_id, control_id))

ATE = 0
for control_id, treatment_id in ordered_matching:
    treatment_value = treatment_df.loc[treatment_id, "Sold_within_3_months"]
    control_value = control_df.loc[control_id, "Sold_within_3_months"]

    ATE += treatment_value - control_value

ATE /= len(ordered_matching)

print(f"Value of ATE: {ATE:.4f}")

# comment:
# In the matching pairs, sometimes the treatment is the first element, sometimes the control is the
# if the student didn't consider this, they will get a value lower than 0.429, like 0.388.  3 pts s
```
```
Value of ATE: 0.4286
```

- (1 pts) What is your conclusion about the effect of the discount on sales within the first three months?

Your Response:

- The discount has a clear positive effect on sales within the first three months.