Arbeitspaket (AP) 3: Machine Learning (ML)

Persönliche Angaben (bitte ergänzen)

Vorname:	
Nachname:	
Immatrikulationsnummer:	
Modul:	Data Science
Prüfungsdatum / Raum / Zeit:	11.11.2024 / Raum: SF O3.54 / 8:00 – 11:45
Erlaubte Hilfsmittel:	w.MA.XX.DS.24HS (Data Science) Open Book, Eigener Computer, Internet-Zugang
Nicht erlaubt:	Nicht erlaubt ist der Einsatz beliebiger Formen von generativer KI (z.B. Copilot, ChatGPT) sowie beliebige Formen von Kommunikation oder Kollaboration mit anderen Menschen.

Bewertungskriterien

(max. erreichbare Punkte: 48)

Kategorie	Beschreibung	Punkteverteilung
Code nicht lauffähig oder Ergebnisse nicht sinnvoll	Der Code enthält Fehler, die verhindern, dass er ausgeführt werden kann (z.B. Syntaxfehler) oder es werden Ergebnisse ausgegeben, welche nicht zur Fragestellung passen.	0 Punkte
Code lauffähig, aber mit gravierenden Mängeln	Der Code läuft, aber die Ergebnisse sind aufgrund wesentlicher Fehler unvollständig (z.B. die Erstellung der Train- und Test Daten ist fehlerhaft, die gewählte Modellierungsmethode entspricht nicht den Anforderungen gemäss Fragestellung). Nur geringer Fortschritt erkennbar.	25% der max. erreichbaren Punkte
Code lauffähig, aber mit mittleren Mängeln	Der Code läuft und liefert teilweise korrekte Ergebnisse, aber es gibt grössere Fehler (z.B. Auswahl der Features entspricht nicht den Anforderungen gemäss Fragestellung). Die Ergebnisse sind nachvollziehbar, aber unvollständig oder ungenau.	50% der max. erreichbaren Punkte

Kategorie	Beschreibung	Punkteverteilung
Code lauffähig, aber mit minimalen Mängeln	Der Code läuft und liefert ein weitgehend korrektes Ergebnis, aber kleinere Fehler (z.B. Formatierung der Modellergebnisse entspricht nicht den Anforderungen gemäss Fragestellung) beeinträchtigen die Vollständigkeit des Ergebnisses.	75% der max. erreichbaren Punkte
Code lauffähig und korrekt	Der Code läuft einwandfrei und liefert das korrekte Ergebnis ohne Mängel.	100% der max. erreichbaren Punkte

Python Libraries und Settings

In [1]:
```
# Libraries
import os
import pickle
import sqlite3
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt
from IPython.display import Image

from sklearn import tree
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.metrics import RocCurveDisplay
from sklearn.datasets import make_regression
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\AP03
```

Vorbereitung (Dieser Teil wird nicht bewertet!)

1.) Starten Sie eine GitHub Codespaces Instanz auf Basis Ihres Forks des folgenden GitHub Repositories:

GitHub-Repository: https://github.com/mario-gellrich-zhaw/python_machine_learning_basics

Einlesen des vorbereiteten Smartphone Datensatzes

In [2]:
```
# Verbindung zur SQLite-Datenbank herstellen
conn = sqlite3.connect('./Data/smartphone_data_prepared.db')

# Cursor erstellen (der Cursor führt SQL-Abfragen aus)
cursor = conn.cursor()

# SQL-Abfrage erstellen und ausführen
df = pd.read_sql_query("""SELECT * FROM smartphone_data_prepared""", con=conn)

# Verbindung schliessen
conn.close()

# Anzeigen des DataFrames (erste 5 Zeilen)
df.head()
```

Out[2]:

	offer_id	dealer_plz	dealer_city	dealer_street_house_nr	make	model	price_raw
0	83197857	6394	Volketswil	Blaserstrasse 19	Apple	iPhone 13	CHF 2518.-
1	36687537	9532	Oberwil	Furrerstrasse 1	Vivo	Vivo V21	CHF 266.-
2	66722344	2227	Rüti	Maierstrasse 31	Vivo	Vivo V21	CHF 424.-
3	56164955	5766	Bülach	Sidlerstrasse 6	Google	Pixel 6a	CHF 597.-
4	94374605	8304	Belp	Steinerstrasse 8	Google	Pixel 6	CHF 1934.-

Aufgaben (Dieser Teil wird bewertet!)

Hinweise zu den folgenden Aufgabenstellungen:

- Beachten Sie auch die zu jeder Aufgabenstellung zugehörenden Details zu Aufgabenstellung.
- Lösen Sie jede Aufgabe mit Hilfe von Python Code. Integrieren Sie den Python Code in die Code-Zellen der jeweiligen Aufgabe.
- Stellen Sie sämtliche von Ihnen erstellte Ergebnisse inklusive Graphiken im Jupyter Notebook dar.

Aufgabe (1): Erstellen Sie Samples für das Trainieren und Testen eines Regressions-Modells auf Basis des Smartphone Datensatzes.

Details zur Aufgabenstellung:

- Verwenden Sie die Variable 'price' als Zielvariable.
- Verwenden Sie folgende erklärenden Variablen: 'camera_resolution', 'battery_life', 'storage_size', 'screen_size'.
- Es werden folgende Samples (Subsets der Dateb) für das Trainieren und Testen

benötigt: X_train, X_test, y_train, y_test.

- Im Sample für das Trainieren müssen 80% der Daten, im Sample und für das Testen müssen 20% der Daten enthalten sein.
- Stellen Sie Ihre X_train und y_train Daten im Jupyter Notebook dar (z.B. erste 5 Zeilen/Werte).

(max. erreichbare Punkte: 4)

In [3]:
```
# Erstellen der Trainings- und Testdaten
X_train, X_test, y_train, y_test = train_test_split(df[['camera_resolution',
                                                        'battery_life',
                                                        'storage_size',
                                                        'screen_size']],
                                                    df['price'],
                                                    test_size=0.20,
                                                    random_state=42)

# Anzeigen von X_train
print('X_train:')
print(X_train.head(), '\n')

# Anzeigen y_train
print('y_train:')
print(y_train.head())
```

```
X_train:
      camera_resolution  battery_life  storage_size  screen_size
4227                 50            34            64          6.1
4676                 16            28           256          5.8
800                  48            29            64          6.1
3671                 12            17            64          5.8
4193                 12            19           512          6.5

y_train:
4227    1218
4676    1295
800      991
3671     266
4193    2520
Name: price, dtype: int64
```

Aufgabe (2): Fitten Sie ein Multiples Lineares Regressionsmodell für die Vorhersage der Smartphone Preise.

Details zur Aufgabenstellung:

- Ergänzen Sie den Python Code unten und Fitten Sie mit Hilfe von y_train und X_train_const ein Multiples Lineares Regressionsmodell.
- Zeigen Sie den Modell-Output im Jupyter Notebook an.
- Beantworten Sie in der Codezelle die Frage: Wieviel Prozent der Variablitiät (engl.: proportion of the variation) in y_train wird mit dem Modell erklärt?
- Schreiben Sie Ihre Antwort zu der Frage per Hand in die vorbereitete print() Funktion (Ergänzen des Teils '... %').

(max. erreichbare Punkte: 4)

In [4]:
```
# Einlesen der Trainings- und Testdaten (damit alle die gleichen Daten haben)
X_train = pd.read_csv('./Data/X_train.csv')
X_test = pd.read_csv('./Data/X_test.csv')
y_train = pd.read_csv('./Data/y_train.csv')
y_test = pd.read_csv('./Data/y_test.csv')

# Hinzufügen einer Konstante zu den Trainingsdaten
X_train_const = sm.add_constant(X_train)

# Fitten des Multiplen Linearen Regressionsmodells
olsmod = sm.OLS(y_train, X_train_const)
olsres = olsmod.fit()

# Anzeigen des Modell-Outputs
print(olsres.summary())

# Beantworten der Frage: Wieviel Prozent der Variablitiät in y_train wird mit de
print(f'\nVon dem Modell wird rund 31 % der Variabilität in y_train erklärt.')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.310
Model:                            OLS   Adj. R-squared:                  0.309
Method:                 Least Squares   F-statistic:                     448.7
Date:                Sun, 09 Nov 2025   Prob (F-statistic):          7.89e-320
Time:                        16:15:00   Log-Likelihood:                -33710.
No. Observations:                4000   AIC:                         6.743e+04
Df Residuals:                    3995   BIC:                         6.746e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -5867.0691    315.464    -18.598      0.000   -6485.555   -5248.584
camera_resolution    21.2946      0.801     26.574      0.000      19.724      22.866
battery_life         42.9607      3.275     13.116      0.000      36.539      49.382
storage_size          2.6148      0.106     24.706      0.000       2.407       2.822
screen_size         852.4294     49.302     17.290      0.000     755.770     949.089
==============================================================================
Omnibus:                     1979.972   Durbin-Watson:                   1.968
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            18319.731
Skew:                           2.159   Prob(JB):                         0.00
Kurtosis:                      12.554   Cond. No.                     4.91e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.91e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Von dem Modell wird rund 31 % der Variabilität in y_train erklärt.
```

Aufgabe (3): Visualisieren Sie die Residuen eines Multiplen Linearen Regressionsmodells.

Details zur Aufgabenstellung:

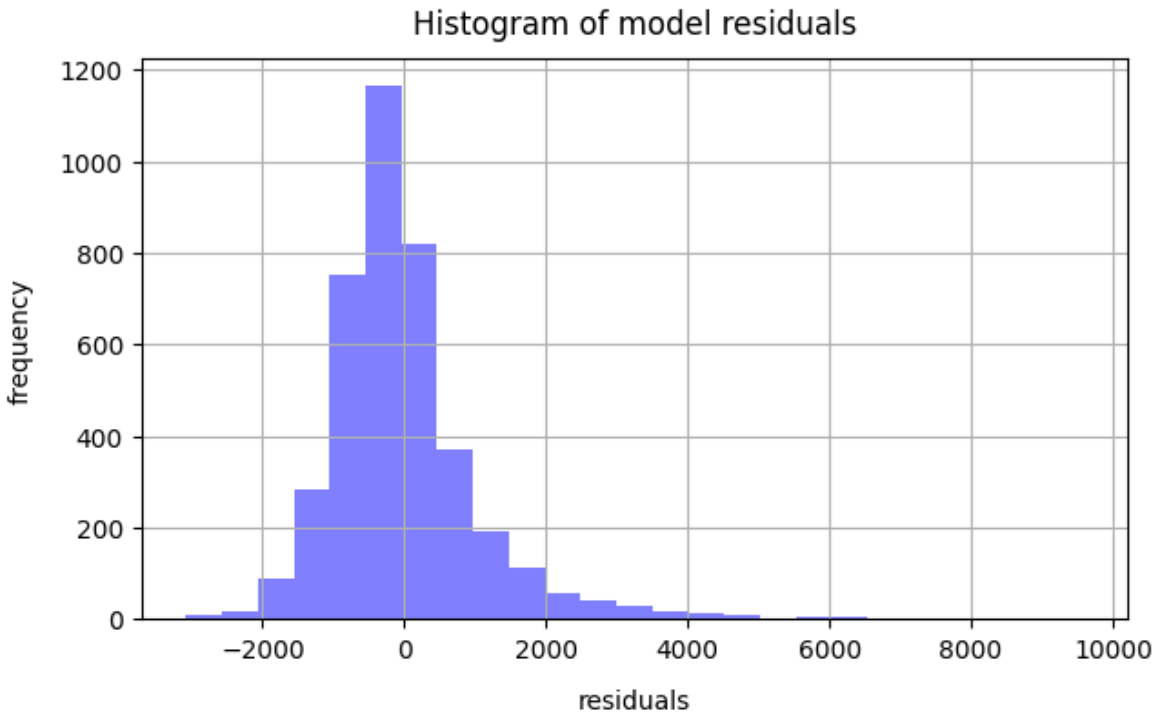
- Verwenden Sie den Python Code unten, um ein vorher abgespeichertes Multiples Lineares Regressionsmodell zu laden.
- Zeigen Sie nach dem Laden des Modells die ersten 5 Modellresiduen an.
- Erstellen Sie ein Histogramm der Modellresiduen.
- Beschriften Sie das Histogramm (Titel, Achsen) und fügen sie der Graphik ein Gitternetz (grid) hinzu.
- Stellen Sie das Histogram im Jupyter Notebook dar.

- Tipp: Die Modelresiduen sind in ols_model.resid gespeichert.

(max. erreichbare Punkte: 4)

In [5]:
```
# Laden eines gespeicherten Multiplen Linearen Regressionsmodells (damit alle da
with open('./Data/ols_model.pkl', 'rb') as f:
    ols_model = pickle.load(f)

# Anzeigen der ersten 5 Modellresiduen
print(ols_model.resid.head())

# Erstellen eines Histogramms der Modellresiduen
fig = plt.figure( figsize=(7,4))
n, bins, patches = plt.hist(x=ols_model.resid,
                            bins=25,
                            color='blue',
                            alpha=0.5
                   )

# Anpassen der Plot-Achsen, -Titel und Hinzufügen eines Grids
plt.xlabel('residuals', fontsize=10, labelpad=10)
plt.ylabel('frequency', fontsize=10, labelpad=10)
plt.title('Histogram of model residuals', fontsize=12, pad=10)
plt.grid()

# Anzeigen des Histogramms
plt.show()
```

```
0   -807.488711
1      4.985419
2   -777.095860
3     35.766430
4    435.730396
dtype: float64
```



Aufgabe (4): Erstellen Sie eine Preisvorhersage für ein Smartphone auf der Basis der geschätzten Modellkoeffizienten.

Details zur Aufgabenstellung:

- Verwenden Sie den Python Code unten, um ein abgespeichertes Multiples Lineares Regressionsmodell zu laden.
- Im Python Code werden die geschätzen Koeffizienten der erklärenden Variablen sowie der Achsenabschnitt (engl.: intercept oder const) angezeigt.
- Verwenden Sie die geschätzen Koeffizienten inkl. const, und erstellen Sie per Hand eine Preisschätzung für ein Smartphone mit den unten angegebenen Eigenschaften.
- Fügen Sie Ihre Formel für die Schätzung in die vorbereitete print() Funktion ein (Ersetzen Sie die '...').

(max. erreichbare Punkte: 4)

In [6]:
```
# Laden eines gespeicherten Multiplen Linearen Regressionsmodells (damit alle da
with open('./Data/ols_model.pkl', 'rb') as f:
    ols_model = pickle.load(f)

# Anzeigen der geschätzen Koeffizienten (gerundet auf 2 Dezimalstellen)
print(ols_model.params.round(2), '\n')

# Schätzen Sie den Preis eines Smartphones mit folgenden Eigenschaften:
# camera_resolution = 12
# battery_life = 16
# storage_size = 128
# screen_size = 6.5

try:
    # Geben Sie hier Ihre Lösung ein (ersetzen Sie jeweils die drei Punkte durch
    print(f'Lösung: {-5867.07 + (12*21.29) + (16*42.96) + (128*2.61) + (6.5*852.
except:
    pass
```

```
const               -5867.07
camera_resolution      21.29
battery_life           42.96
storage_size            2.61
screen_size           852.43
dtype: float64

Lösung: 950.64
```

Aufgabe (5): Erstellen Sie ein Regression Tree Model für die Vorhersage der Smartphone Preise.

Details zur Aufgabenstellung:

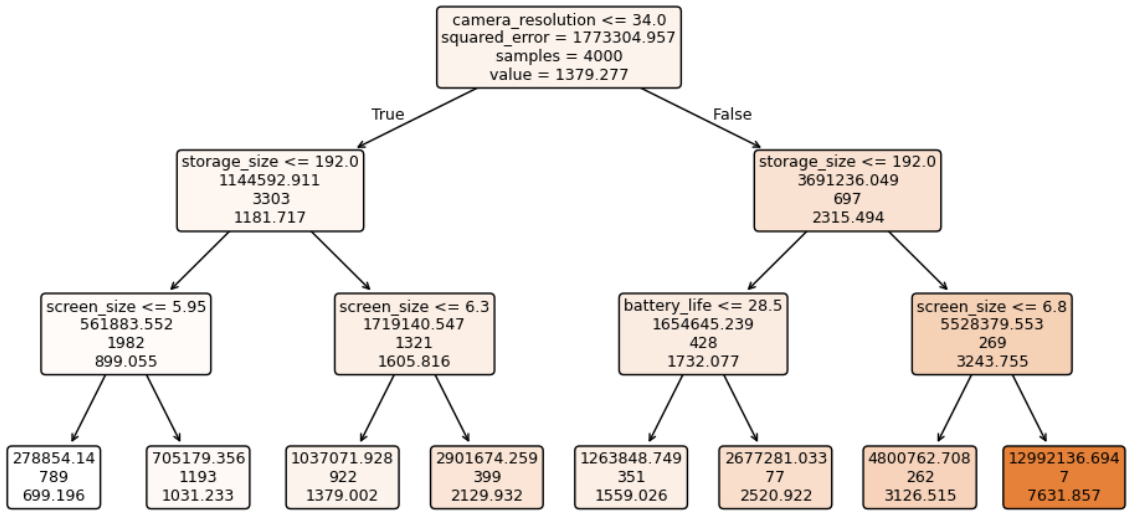
- Verwenden Sie den Python Code in der folgenden Codezelle, um die Trainings- und Testdaten einzulesen.
- Erstellen und Trainieren Sie ein Regression Tree Model mit einer maximalen Tiefe von 3 (max_depth=3).
- Erstellen Sie eine Vorhersage der Preise für die Testdaten (X_test).
- Berechnen Sie die Vorhersagegüte mit Hilfe des r-squared (coefficient of determination).
- Stellen Sie das Regression Tree Model im Jupyter Notebook graphisch dar.

(max. erreichbare Punkte: 8)

In [7]:
```
# Einlesen der Trainings- und Testdaten (damit alle die gleichen Daten haben)
X_train = pd.read_csv('./Data/X_train.csv')
X_test = pd.read_csv('./Data/X_test.csv')
y_train = pd.read_csv('./Data/y_train.csv')
y_test = pd.read_csv('./Data/y_test.csv')

# Erstellen eines 'decision tree regressor' Objekts
reg_tree = DecisionTreeRegressor(random_state=42, max_depth=3)

# Trainieren des 'decision tree regressor' Objekts
reg_tree.fit(X_train, y_train)

# Vorhersage der Preise für die Testdaten
```

```
y_pred = reg_tree.predict(X_test)

# Berechnen des r-squared (coefficient of determination) Wertes
print('R-squared:', round(r2_score(y_test, y_pred), 4))

# Darstellen des Regression Trees
fig = plt.figure(figsize=(12,6))
reg_tree_plot = tree.plot_tree(reg_tree,
                               feature_names=list(X_train.columns),
                               class_names=['price'],
                               filled=True,
                               fontsize=9,
                               label='root',
                               rounded=True)
```

```
R-squared: 0.2251
```



Aufgabe (6): Erstellen Sie ein Random Forest Regression Model für die Vorhersage der Smartphone Preise.

Details zur Aufgabenstellung:

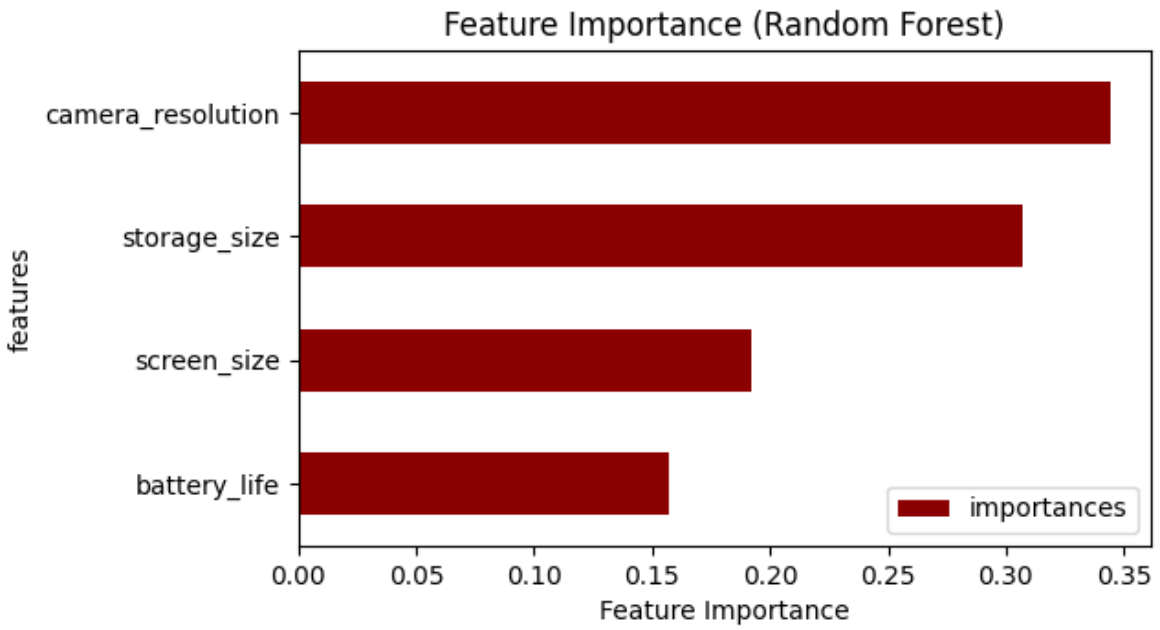
- Verwenden Sie den Python Code in der folgenden Codezelle, um die Trainings- und Testdaten einzulesen.
- Erstellen und Trainieren Sie ein Random Forest Regression Model mit 500 einzelnen Trees und einer maximalen Tiefe von 5.
- Erstellen Sie eine Vorhersage der Preise für die Testdaten (X_test).
- Berechnen Sie die Vorhersagegüte mit Hilfe des r-squared (coefficient of determination).
- Stellen Sie die Feature Importance mit Hilfe eines Barcharts graphisch dar.

(max. erreichbare Punkte: 8)

In [8]:
```
# Einlesen der Trainings- und Testdaten (damit alle die gleichen Daten haben)
X_train = pd.read_csv('./Data/X_train.csv')
X_test = pd.read_csv('./Data/X_test.csv')
y_train = pd.read_csv('./Data/y_train.csv')
y_test = pd.read_csv('./Data/y_test.csv')
```

```
# Erstellen/Initialisieren eines 'random forest regressor' Objekts
reg_rf = RandomForestRegressor(random_state=42, n_estimators=250, max_depth=5)

# Trainieren des 'random forest regressor' Objekts
reg_rf.fit(X_train, y_train)

# Vorhersage der Preise für die Testdaten X_test
y_pred = reg_rf.predict(X_test)

# Berechnen des r-squared (coefficient of determination)
print('R-squared:', round(r2_score(y_test, y_pred), 4))

# Graphische Darstellung der Feature Importance
cols = X_train.columns
importances = reg_rf.feature_importances_
std         = np.std([tree.feature_importances_ for tree in reg_rf.estimators_],
indices     = np.argsort(importances)[::-1]

df_fi = pd.DataFrame({'features':cols,'importances': importances})
df_fi.sort_values('importances', inplace=True)
df_fi.plot(kind='barh',
           y='importances',
           x='features',
           color='darkred',
           xlabel='Feature Importance',
           title='Feature Importance (Random Forest)',
           figsize=(6,3.5))
plt.show()
```

```
R-squared: 0.3005
```



Aufgabe (7): Erstellen Sie ein Random Forest Classification Model für die Vorhersage der Hersteller.

Details zur Aufgabenstellung:

- Verwenden Sie den Python Code in der folgenden Codezelle, um die Trainings- und Testdaten für die Klassifikation einzulesen.
- Die Zielvariable in diesem Modell ist die Variable make (Hersteller).

- Die erklärenden Variablen in diesem Modell sind: price, camera_resolution, battery_life, storage_size, screen_size.
- Erstellen und Trainieren Sie ein Random Forest Classification Model.
- Erstellen Sie eine Vorhersage der Zielvariable auf der Basis der Testdaten (X_test).
- Erstellen Sie eine Confusion Matrix, und stellen Sie diese im Jupyter Notebook dar.
- Erstellen Sie einen Classification report, und stellen Sie diesen im Jupyter Notebook dar.
- Beantworten Sie die Frage: Wie hoch ist der Anteil der Hersteller welche vom Modell korrekt klassifiziert wurden?
- Verwenden Sie für die Beantwortung der Frage die vorgefertigte print() Funktion (Ersetzen Sie '... %').

(max. erreichbare Punkte: 8)

In [9]:
```
# Einlesen der Trainings- und Testdaten (damit alle die gleichen Daten haben)
X_train = pd.read_csv('./Data/X_train_class.csv')
X_test = pd.read_csv('./Data/X_test_class.csv')
y_train = pd.read_csv('./Data/y_train_class.csv')
y_test = pd.read_csv('./Data/y_test_class.csv')

# Umformen der y_train and y_test zu 1-dimensional arrays
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

# Erstellen/Initialisieren des 'random forest classifier' Objekts
rfc = RandomForestClassifier(n_estimators=500, random_state=42, max_depth=5)

# Trainieren des Random Forest Classification Modells
rfc = rfc.fit(X_train, y_train)

# Vorhersage der Zielvariable für die Testdaten X_test
y_pred_rf = rfc.predict(X_test)

# Confusion matrix
print('Confusion matrix')
print(confusion_matrix(y_test, y_pred_rf), '\n')

# Classification report
print('Classification report')
print(classification_report(y_test, y_pred_rf))

# Beantworten der Frage: Wieviel % der Smartphones wurden korrekt klassifiziert?
print(f'\nDer Anteil der vom Modell korrekt klassifizierten Smartphones beträgt
```

```
Confusion matrix
[[135  13   6  42  11]
 [ 86  25  22  48  25]
 [ 64  27  19  77  30]
 [ 20  23  12 108  14]
 [ 40  17  18  94  24]]

Classification report
              precision    recall  f1-score   support

       Apple       0.39      0.65      0.49       207
      Google       0.24      0.12      0.16       206
     Samsung       0.25      0.09      0.13       217
        Vivo       0.29      0.61      0.40       177
      Xiaomi       0.23      0.12      0.16       193

    accuracy                           0.31      1000
   macro avg       0.28      0.32      0.27      1000
weighted avg       0.28      0.31      0.26      1000


Der Anteil der vom Modell korrekt klassifizierten Smartphones beträgt 31%.
```

Aufgabe (8): Erstellen Sie ein Random Forest Classification Model für die Vorhersage der beiden Hersteller 'Apple' und 'Vivo'.

Details zur Aufgabenstellung:

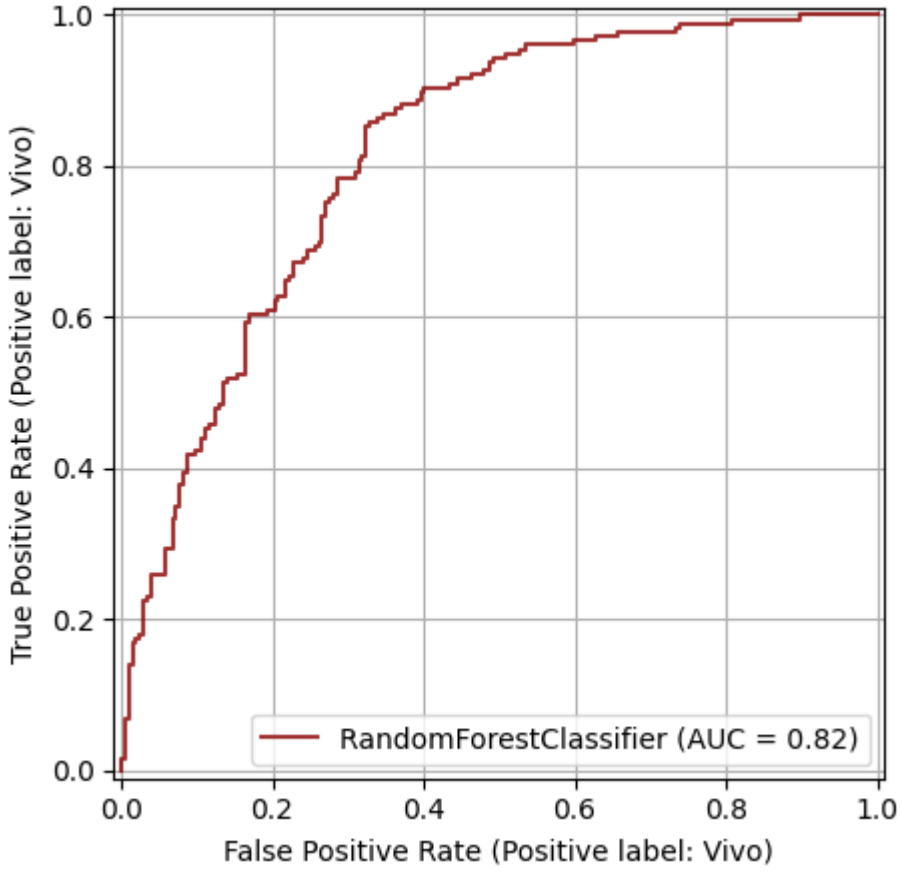
- Es handelt sich bei diesem Beispiel um ein Klassifikationsproblem mit zwei Klassen ('Apple' und 'Vivo').
- Ergänzen Sie den Python Code dort wo notwendig.
- Beantworten Sie mit eigenen Worten die Frage: Wie gut ist das Modell darin, Smartphones von Apple und Vivo zu unterscheiden?
- Verwenden Sie für die Beantwortung der Frage die vorgefertigte print() Funktion (Ersetzen Sie die '...').

(max. erreichbare Punkte: 8)

In [10]:
```python
# Einlesen der Trainings- und Testdaten (damit alle die gleichen Daten haben)
X_train = pd.read_csv('./Data/X_train_class.csv')
X_test = pd.read_csv('./Data/X_test_class.csv')
y_train = pd.read_csv('./Data/y_train_class.csv')
y_test = pd.read_csv('./Data/y_test_class.csv')

# Subset der Trainings- und Testdaten erstellen (nur zwei Hersteller)
X_train_subset = X_train[y_train['make'].isin(['Apple', 'Vivo'])]
y_train_subset = y_train[y_train['make'].isin(['Apple', 'Vivo'])]
X_test_subset = X_test[y_test['make'].isin(['Apple', 'Vivo'])]
y_test_subset = y_test[y_test['make'].isin(['Apple', 'Vivo'])]

# Umformen der y_train and y_test zu 1-dimensional arrays
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

# Erstellen/Initialisieren des 'random forest classifier' Objekts
rfc = RandomForestClassifier(n_estimators=500, random_state=42, max_depth=5)
```

```python
# Trainieren des Random Forest Classification Modells
rfc = rfc.fit(X_train_subset, y_train_subset)

# Vorhersage der Zielvariable für die Testdaten
y_pred_rf = rfc.predict(X_test_subset)

# Confusion matrix
print('Confusion matrix')
cm = confusion_matrix(y_test_subset, y_pred_rf)
print(cm, '\n')

# Classification report
print('Classification report')
print(classification_report(y_test_subset, y_pred_rf))

# Erstellen der Graphik mit ROC curve und AUC
plt.figure(figsize=(5,5))
ax = plt.gca()
rfc_disp = RocCurveDisplay.from_estimator(rfc,
                                          X_test_subset,
                                          y_test_subset,
                                          ax=ax,
                                          alpha=0.8,
                                          c="darkred")
plt.grid()
plt.show()

# Beantworten Sie die Frage: Wie gut ist das Modell darin, Smartphones von Apple
print(f'\nDas Modell ist gut darin, Smartphones von Apple und Vivo zu unterschei
```

```
Confusion matrix
[[152  55]
 [ 47 130]]

Classification report
              precision    recall  f1-score   support

       Apple       0.76      0.73      0.75       207
        Vivo       0.70      0.73      0.72       177

    accuracy                           0.73       384
   macro avg       0.73      0.73      0.73       384
weighted avg       0.74      0.73      0.73       384
```



Das Modell ist gut darin, Smartphones von Apple und Vivo zu unterscheiden, da die AUC bei 0.82 liegt.

Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [11]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('IP Address:', socket.gethostbyname(socket.gethostname()))
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:15:02
Python Version: 3.13.9
IP Address: 192.168.178.34
-----------------------------------
```

Linear regression to predict prices of rental apartments

Libraries and settings

In [1]:
```
# Libraries
import os
import numpy as np
import pandas as pd
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.model_selection import train_test_split

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Linear_Regression
```

Import the apartment data

In [2]:
```
# Define columns for import
columns = [ 'web-scraper-order',
            'address_raw',
            'rooms',
            'area',
            'luxurious',
            'price',
            'price_per_m2',
            'lat',
            'lon',
            'bfs_number',
            'bfs_name',
            'pop',
            'pop_dens',
            'frg_pct',
            'emp',
            'mean_taxable_income',
            'dist_supermarket']

# Read and select variables
df_orig = pd.read_csv("./Data/apartments_data_enriched_cleaned.csv",
                      sep=";",
                      encoding='utf-8')[columns]

# Rename variable 'web-scraper-order' to 'apmt_id'
df_orig = df_orig.rename(columns={'web-scraper-order': 'id'})
```

```
# Remove missing values
df = df_orig.dropna()
df.head(5)

# Remove duplicates
df = df.drop_duplicates()

# Remove some 'extreme' values
df = df.loc[(df['price'] >= 1000) &
            (df['price'] <= 5000)]

print(df.shape)
df.head(5)
```

```
(722, 17)
```

Out[2]:

	id	address_raw	rooms	area	luxurious	price	price_per_m2	
0	1693998201-1	Neuhusstrasse 6, 8630 Rüti ZH, ZH	3.0	49	0	1441	29.41	47.252
1	1693998233-172	Widacherstrasse 5, 8630 Rüti ZH, ZH	3.0	111	0	2600	23.42	47.252
2	1693998256-331	Widenweg 14, 8630 Rüti ZH, ZH	3.0	58	0	1490	25.69	47.253
3	1693998265-381	Rain 1, 8630 Rüti ZH, ZH	4.0	118	0	3240	27.46	47.259
4	1693998276-419	Bachtelstrasse 24b, 8630 Rüti ZH, ZH	3.0	66	0	1450	21.97	47.266

Simple linear regression (only one explanatory variable in the model)

For details see: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

Create train and test samples (train = 80%, test = 20% of the data)

In [3]:

```
# Create train and test samples
X_train, X_test, y_train, y_test = train_test_split(df['area'],
                                                    df['price'],
                                                    test_size=0.20,
                                                    random_state=42)

# Show X_train
print('X_train:')
print(X_train.head(), '\n')

# Show y_train
print('y_train:')
```

```
print(y_train.head())
```

```
X_train:
503    120
6       65
400     90
644     63
484     91
Name: area, dtype: int64

y_train:
503    1900
6      1850
400    2090
644    2102
484    1800
Name: price, dtype: int64
```

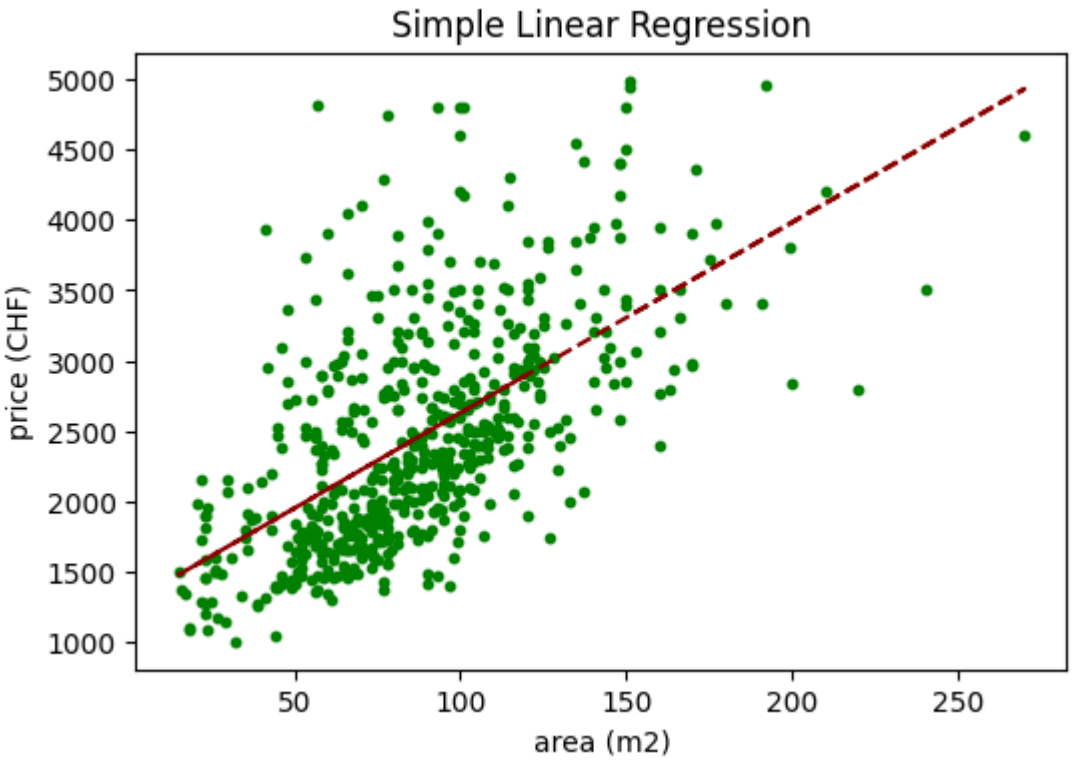
Fit the simple linear regression model

In [4]:
```
# Fit the regression model
slope, intercept, r, p, std_err = stats.linregress(X_train, y_train)

# Print results of the regression model
print('Linear regression result:')
print(f'Intercept with y-axis (alpha):            {intercept:.2f}')
print(f'Slope of regression line (beta):          {slope:.3f}')
print(f'p-value:                                  {p:.4f}')
print(f'R-squared (coefficient of determination): {r**2:.4f}')
```

```
Linear regression result:
Intercept with y-axis (alpha):            1272.94
Slope of regression line (beta):          13.548
p-value:                                  0.0000
R-squared (coefficient of determination): 0.3678
```

Plot regression line

In [5]:
```
# Function to calculate model predictions
def myfunc(x):
    return slope * x + intercept

# Apply myfunc() to x, i.e. make predictions
mymodel = pd.Series(map(myfunc, X_train))

# Scatterplot with regression line
plt.figure(figsize=(6,4))
plt.scatter(X_train, y_train, s=10, color='green')
plt.plot(X_train, mymodel, color='darkred', linestyle='dashed')
plt.title('Simple Linear Regression')
plt.xlabel('area (m2)')
plt.ylabel('price (CHF)')

plt.show()
```



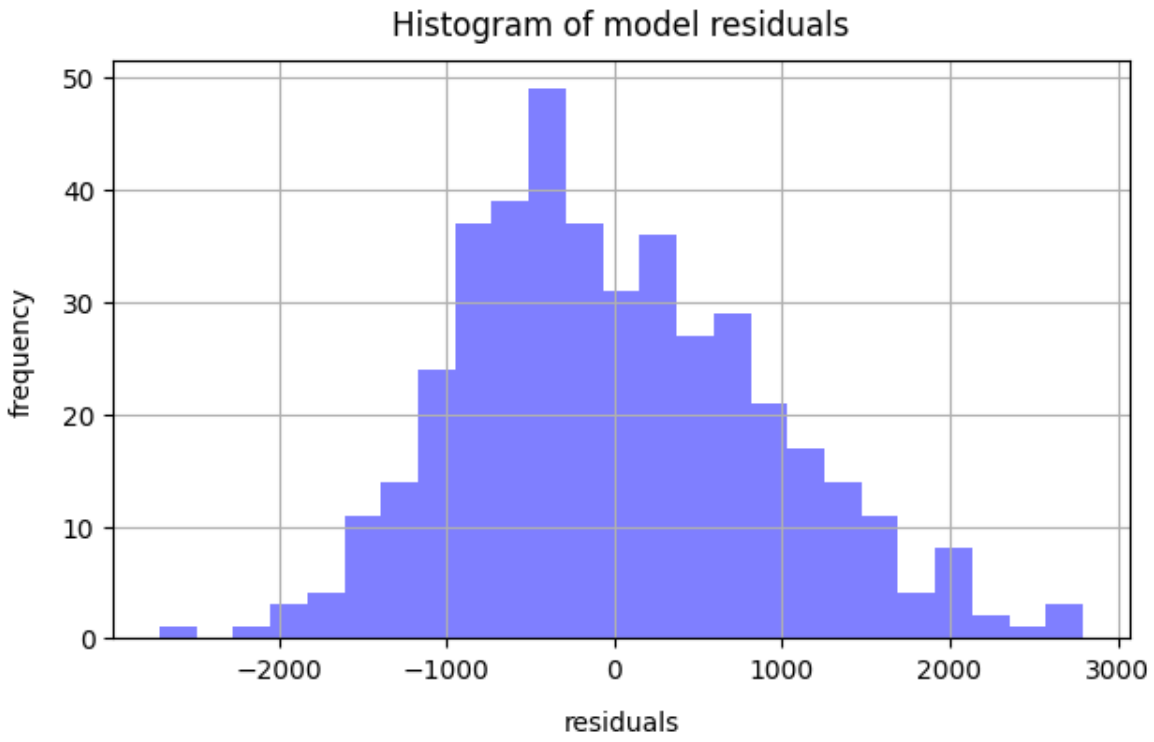
Check model residuals (residuals = observed prices minus predicted prices)

In [6]:
```
# Calculate model residuals for train data
residuals = y_train - mymodel

# Check the first residual value in our data set
print(f'1st Predicted price in dataset: {mymodel[0]:.2f}')
print(f'1st Observed price in dataset: {y_train[0]:.2f}')
print(f'1st Residual price in dataset: {residuals[0]:.2f}')
```

```
1st Predicted price in dataset: 2898.72
1st Observed price in dataset: 1441.00
1st Residual price in dataset: -1457.72
```

Plot histogram of residuals

In [7]:
```
# Plot histogram of residuals
fig = plt.figure( figsize=(7,4))
n, bins, patches = plt.hist(x=residuals,
                            bins=25,
                            color='blue',
                            alpha=0.5
                    )

# Set title and labels
plt.xlabel('residuals', fontsize=10, labelpad=10)
plt.ylabel('frequency', fontsize=10, labelpad=10)
plt.title('Histogram of model residuals', fontsize=12, pad=10)
plt.grid()

# Show plot
plt.show()
```



Compare the observed prices with the predicted prices

In [8]:
```
# Create model predictions for test data
predicted = myfunc(X_test)
predicted.round(1)

# Compare the observed prices with the predicted prices
for i in range(0,10):
    print(f'Observed price: {y_test.iloc[i]:.1f}, Predicted price: {predicted.il
```

```
Observed price: 1360.0, Predicted price: 2031.6
Observed price: 1640.0, Predicted price: 2004.5
Observed price: 1568.0, Predicted price: 2167.1
Observed price: 2290.0, Predicted price: 2207.8
Observed price: 4950.0, Predicted price: 2627.8
Observed price: 1787.0, Predicted price: 2248.4
Observed price: 3495.0, Predicted price: 2519.4
Observed price: 2450.0, Predicted price: 2031.6
Observed price: 3390.0, Predicted price: 2871.6
Observed price: 2836.0, Predicted price: 2573.6
```

Multiple linear regression (more than one explanatory variable in the model)

For details see: https://www.statsmodels.org/dev/examples/notebooks/generated/predict.html

Create train and test samples (train = 80%, test = 20% of the data)

In [9]:
```
# Create train and test samples (we name it X2_ and y_2 because we already used
X2_train, X2_test, y2_train, y2_test = train_test_split(df[['area',
```

```
                                                    'pop_dens']],
                                                df['price'],
                                                test_size=0.20,
                                                random_state=42)

# Show X2_train
print('X2_train:')
print(X2_train.head(), '\n')

# Show y2_train
print('y2_train:')
print(y2_train.head())
```

```
X2_train:
     area     pop_dens
503   120   165.018625
6      65   525.662252
400    90   424.146342
644    63  1044.628957
484    91   399.525129

y2_train:
503    1900
6      1850
400    2090
644    2102
484    1800
Name: price, dtype: int64
```

Fit the multiple regression model (yes, the output is rich :-), but we need only part of it for interpretation!)

In [10]:

```
# Add constant to the model
X2_train_const = sm.add_constant(X2_train)

# Create the multiple regression model
olsmod = sm.OLS(y_train, X2_train_const)
olsres = olsmod.fit()

# Print full model output
print(olsres.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.588
Model:                            OLS   Adj. R-squared:                  0.586
Method:                 Least Squares   F-statistic:                     409.2
Date:                Sun, 09 Nov 2025   Prob (F-statistic):          3.50e-111
Time:                        16:33:49   Log-Likelihood:                -4420.2
No. Observations:                 577   AIC:                             8846.
Df Residuals:                     574   BIC:                             8860.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        331.1861     78.086      4.241      0.000     177.818     484.554
area          17.9213      0.649     27.625      0.000      16.647      19.195
pop_dens       0.2379      0.014     17.502      0.000       0.211       0.265
==============================================================================
Omnibus:                       79.849   Durbin-Watson:                   1.970
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              140.633
Skew:                           0.843   Prob(JB):                     2.90e-31
Kurtosis:                       4.734   Cond. No.                     1.06e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.06e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation of the relevant (in this course) statistics in the table above

R-squared: This is the coefficient of determination (see slides of lessons). A value of 0.522 means, that the explanatory variables explain 52% of the variaton of our target variable (rental prices) - not bad, but could be improved.

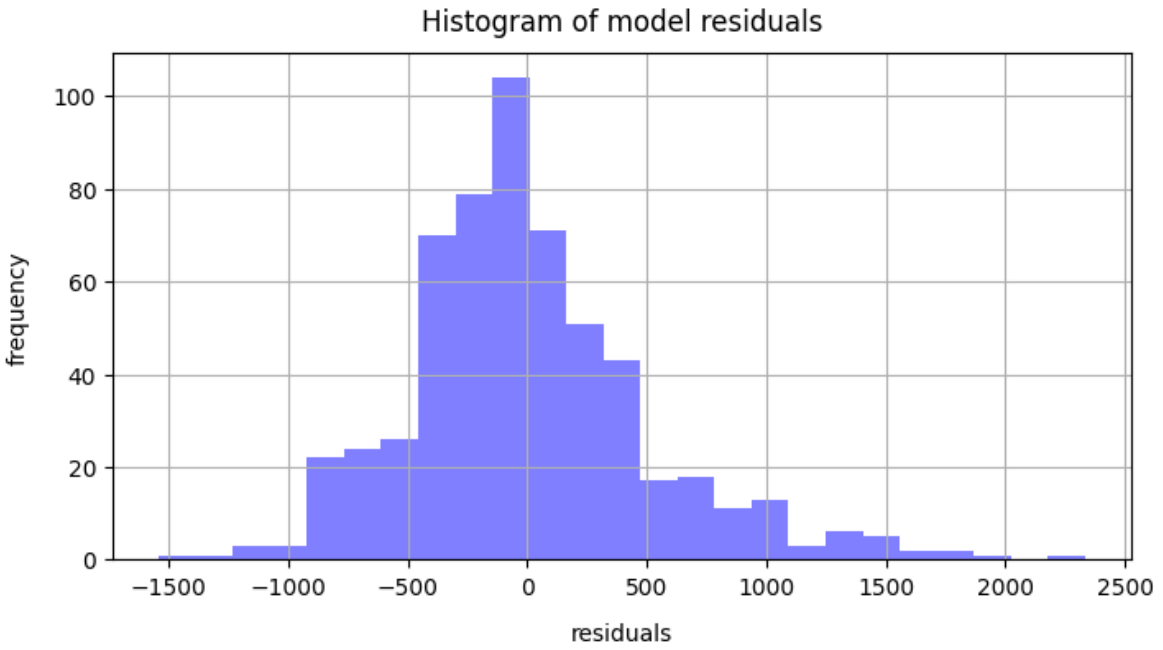
coef: These are the estimated coefficients of the explanatory variables ('slopes of the regression line' of each variable). These are nedded for the price predictions in our model.

P>|t|: These are the p-values. If < 0.05, the explanatory variables shows a statistically siginificant (5% significance level) contribution in explaining the target variable. Except for the distiance to the nearest supermarkt, all variables are significant here.

Plot histogram of residuals

In [11]:
```
# Plot histogram of residuals
fig = plt.figure( figsize=(8,4))
n, bins, patches = plt.hist(x=olsres.resid,
                            bins=25,
                            color='blue',
                            alpha=0.5
                   )
```

```
# Set labels
plt.xlabel('residuals', fontsize=10, labelpad=10)
plt.ylabel('frequency', fontsize=10, labelpad=10)
plt.title('Histogram of model residuals', fontsize=12, pad=10)
plt.grid()

plt.show()
```



Compare observed prices with predicted prices

In [12]:
```
# Add constant to X2_test
X2_test_const = sm.add_constant(X2_test)
predicted_new = olsres.predict(X2_test_const)

# Compare the observed prices with the predicted prices
for i in range(0,10):
    print(f'Observed price: {y_test.iloc[i]:.1f}, Predicted price: {predicted_ne
```

```
Observed price: 1360.0, Predicted price: 1730.3
Observed price: 1640.0, Predicted price: 1600.6
Observed price: 1568.0, Predicted price: 1806.3
Observed price: 2290.0, Predicted price: 2704.6
Observed price: 4950.0, Predicted price: 2519.3
Observed price: 1787.0, Predicted price: 2201.5
Observed price: 3495.0, Predicted price: 3116.8
Observed price: 2450.0, Predicted price: 2471.6
Observed price: 3390.0, Predicted price: 2685.2
Observed price: 2836.0, Predicted price: 2405.3
```

Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [13]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime
```

```python
print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:33:49
Python Version: 3.13.9
-----------------------------------
```

Linear Regression using Batch Gradient Descent

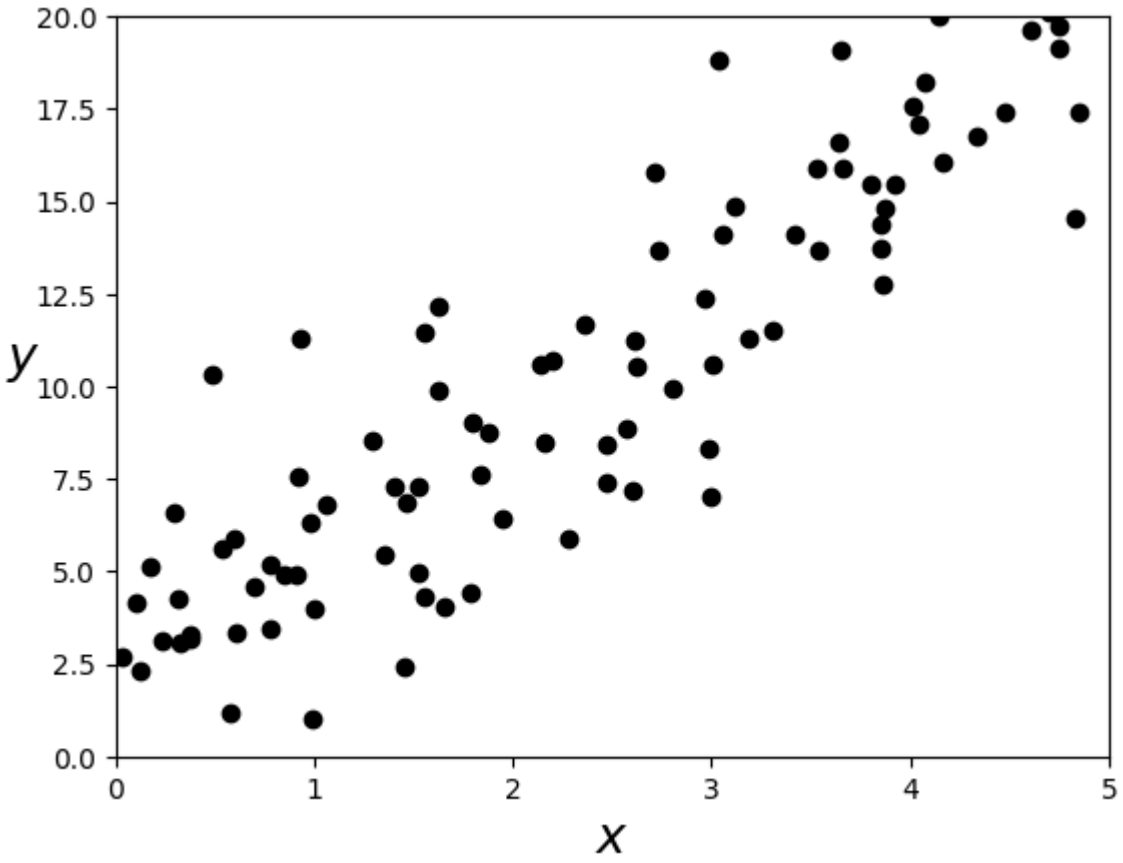
Libraries and Settings

In [1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt

import sklearn
from sklearn.linear_model import LinearRegression
assert sklearn.__version__ >= "0.20"

# To make this notebook's output stable across runs
np.random.seed(42)

# Ignore warnings
import warnings
warnings.filterwarnings(action="ignore")

# Show current working directory
print(os.getcwd())
```

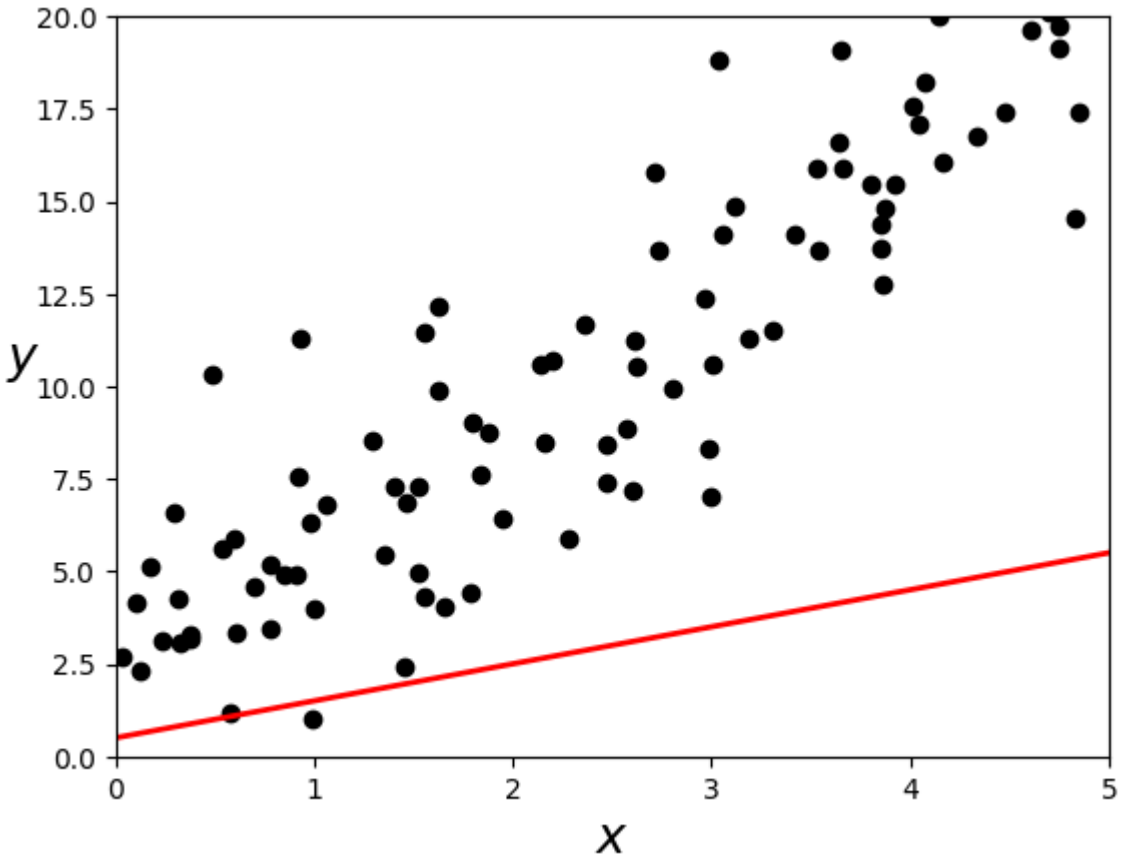
```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Linear_Regression
```

Linear regression Do-It-Yourself

In [2]:
```
X = 5 * np.random.rand(100, 1)
y = 1 + 4 * X + 3*np.random.randn(100, 1)

plt.scatter(X, y, color = 'black')
plt.xlabel("$x$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 5, 0, 20])
plt.show()
```



In [3]:
```
# Sample data
x_new = np.linspace(0, 5, 100)

# Make a guess for theta_0 and theta_1
theta_0 = 0.5 # Please make a better guess for theta_0
theta_1 = 1.0 # Please make a better guess for theta_1

y_hyp =  + theta_0 + theta_1 * x_new

# Plot data an guess of theta_1 & theta_0
plt.scatter(X, y, color = 'black' )
plt.plot(x_new, y_hyp, color = 'red', lw = 2)
plt.xlabel("$x$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 5, 0, 20])
plt.show()
```



Search for optimal values of theta_0 und theta_1 using batch gradient descent

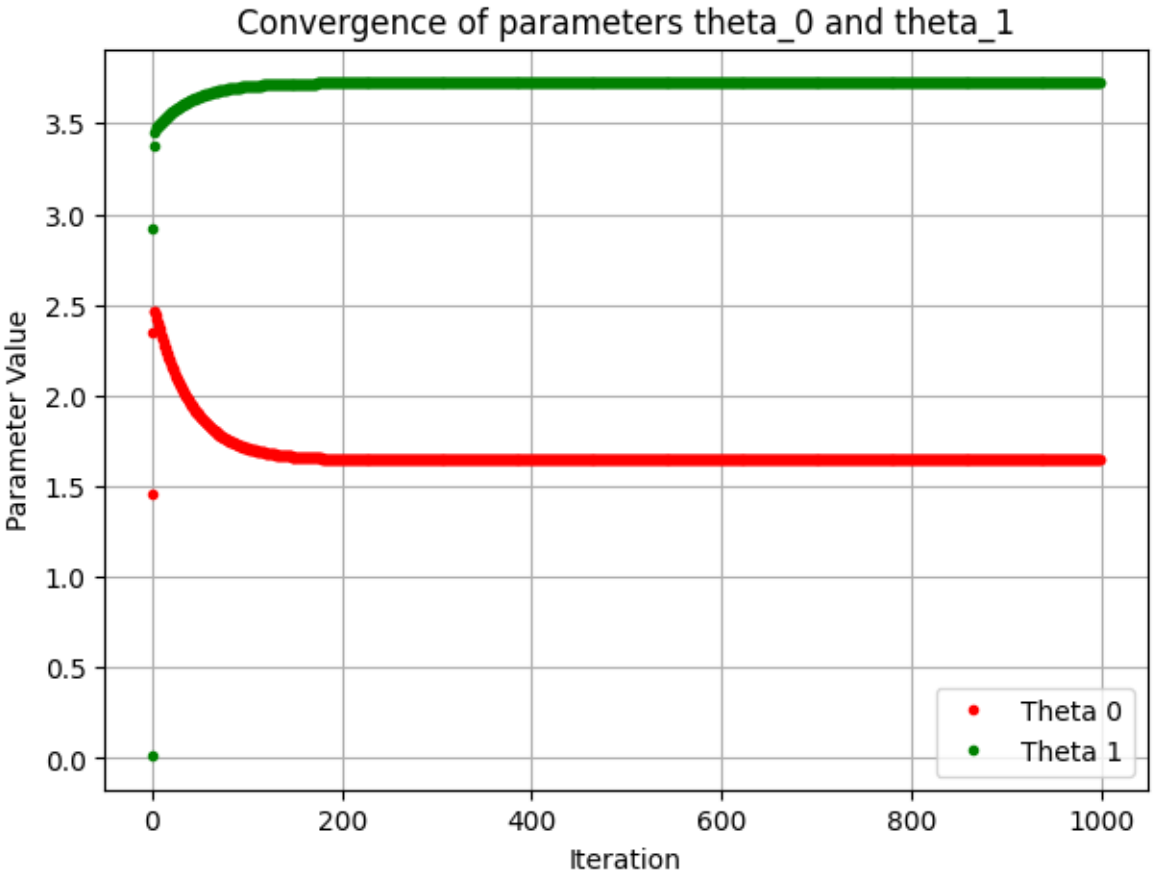
In [4]:
```
# Learning rate
eta = 0.1

# Number of iterations
n_iterations = 1000

# Number of samples
N = 100

# Random initialization of theta_1 and theta_0
theta_1 = np.random.randn(1)
theta_0 = np.random.randn(1)

# Lists to store the values of theta_1 and theta_0 during the iterations
cum_theta1=[]
cum_theta0=[]

# Gradient Descent
for iteration in range(n_iterations):
    cum_theta0.append(theta_0)
    cum_theta1.append(theta_1)

    # Compute the gradients
    gradient_theta_0 = 1/N * np.sum((theta_1*X + theta_0 - y))
    gradient_theta_1 = 1/N * np.sum((theta_1*X + theta_0 - y)*X)

    # Update theta_0 and theta_1
```

```
    theta_0 = theta_0 - eta * gradient_theta_0
    theta_1 = theta_1 - eta * gradient_theta_1

# Plot the values of theta_1 and theta_0
print("theta_0: ", theta_0)
print("theta_1: ", theta_1)
```

```
theta_0:  [1.64528847]
theta_1:  [3.72413606]
```

Plot the values of theta_1 and theta_0 for each iteration

In [5]:
```
# Plot the values of theta_1 and theta_0
plt.figure(figsize=(7, 5))
plt.plot(cum_theta0, 'r.', label='Theta 0')
plt.plot(cum_theta1, 'g.', label='Theta 1')
plt.xlabel('Iteration')
plt.ylabel('Parameter Value')
plt.title('Convergence of parameters theta_0 and theta_1')
plt.legend()
plt.grid()
plt.show()
```



Calculate values for theta_0 und theta_1 using linear regression

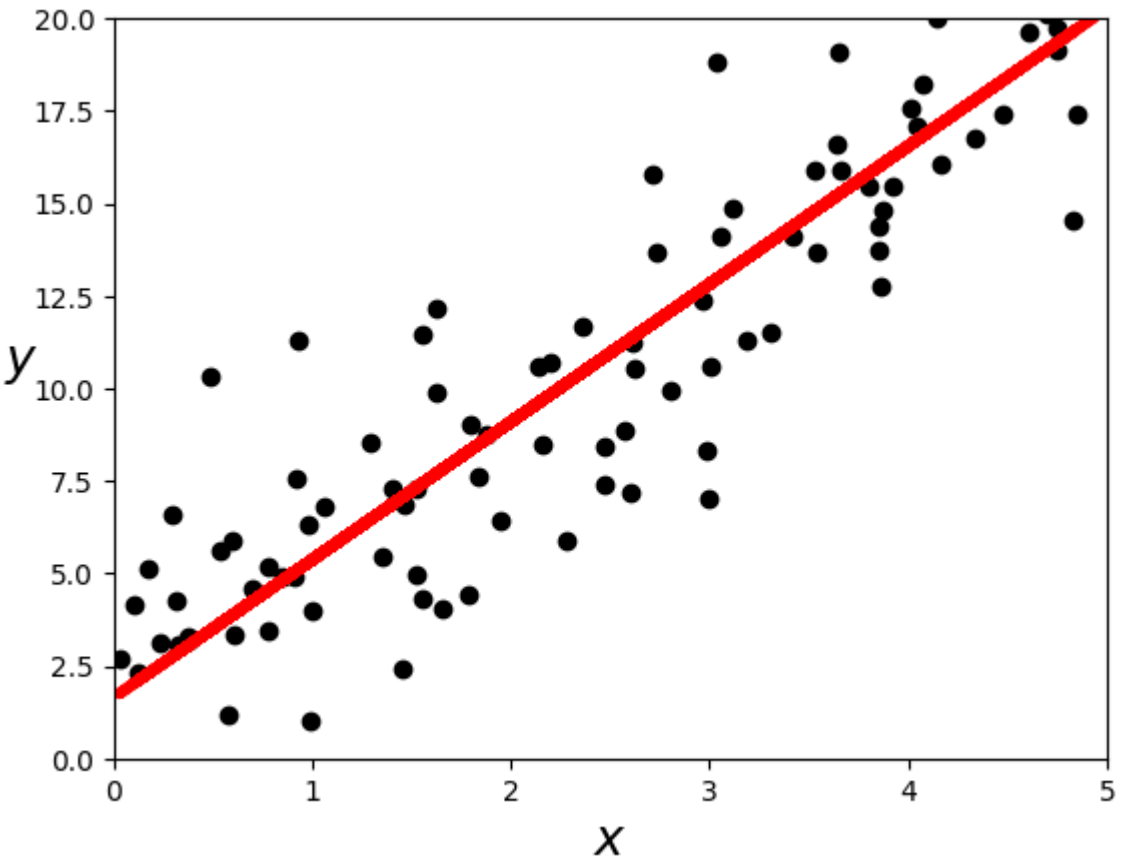
In [6]:
```
# Calculate values for theta_0 und theta_1 using linear regression
```

```
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Print values for theta_0 und theta_1
print("theta_0: ", lin_reg.intercept_)
print("theta_1: ", lin_reg.coef_)
```

```
theta_0:  [1.64528847]
theta_1:  [[3.72413606]]
```

Performing a prediction using a linear regression model

In [7]:
```
# Performing a prediction using a linear regression model
ypred_exact = lin_reg.coef_ * X + lin_reg.intercept_

# Plot the data and the prediction
plt.scatter(X, y, color = 'black' )
plt.plot(X, ypred_exact, color = 'red', lw = 4)
plt.xlabel("$x$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 5, 0, 20])
plt.show()
```



Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [8]:
```
import os
import platform
import socket
```

```python
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:34:43
Python Version: 3.13.9
-----------------------------------
```

Simple Linear Regression: Confidence- and Prediction-Intervals

This notebook demonstrates how to create confidence and prediction intervals for a linear regression model using Python, based on rental price data (Y) and living area (X).

See also: https://lmc2179.github.io/posts/confidence_prediction.html

Libraries and Settings

In [1]:
```
# Import libraries
import os
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import statsmodels.formula.api as smf

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Linear_Regression
```

Generate Apartment Data

We will use the provided data representing rental prices (`Y`) and living area (`X`).

In [2]:
```
# Set the random seed for reproducibility
np.random.seed(42)

# Generate values for rental_price and living_area
x = np.linspace(45, 160, 25)
y = np.interp(x, [45, 160], [1500, 4900]) + np.random.normal(0, 400, 25)

# Create a new DataFrame with 25 data points
data = pd.DataFrame({
    'rental_price': y,
    'living_area': x
})

df = pd.DataFrame(data)
df.columns = ['rental_price', 'living_area']
df.head()
```

Out[2]:

	rental_price	living_area
0	1698.685661	45.000000
1	1586.360946	49.791667
2	2042.408749	54.583333
3	2534.211943	59.375000
4	1973.005317	64.166667

Fit an OLS Regression Model

We will use `statsmodels` to fit an ordinary least squares (OLS) regression model for rental price as a function of living area.

In [3]:

```
# Fit OLS regression
model = smf.ols('rental_price ~ living_area', data=df)

# Get and show results
results = model.fit()
print(results.summary())

# Plot data and regression line
plt.scatter(df['living_area'], df['rental_price'])
plt.plot(df['living_area'],
         results.predict(df['living_area']),
         color='darkred',
         linewidth=2)
plt.xlabel('Living Area')
plt.ylabel('Rental Price')
plt.title('Living Area vs Rental Price')

# Add annotation (regression equation with f-string)
plt.text(50, 4500, f"y = {results.params[0]:.2f} + {results.params[1]:.2f}x",
         fontsize=12, color='black')

plt.show()
```

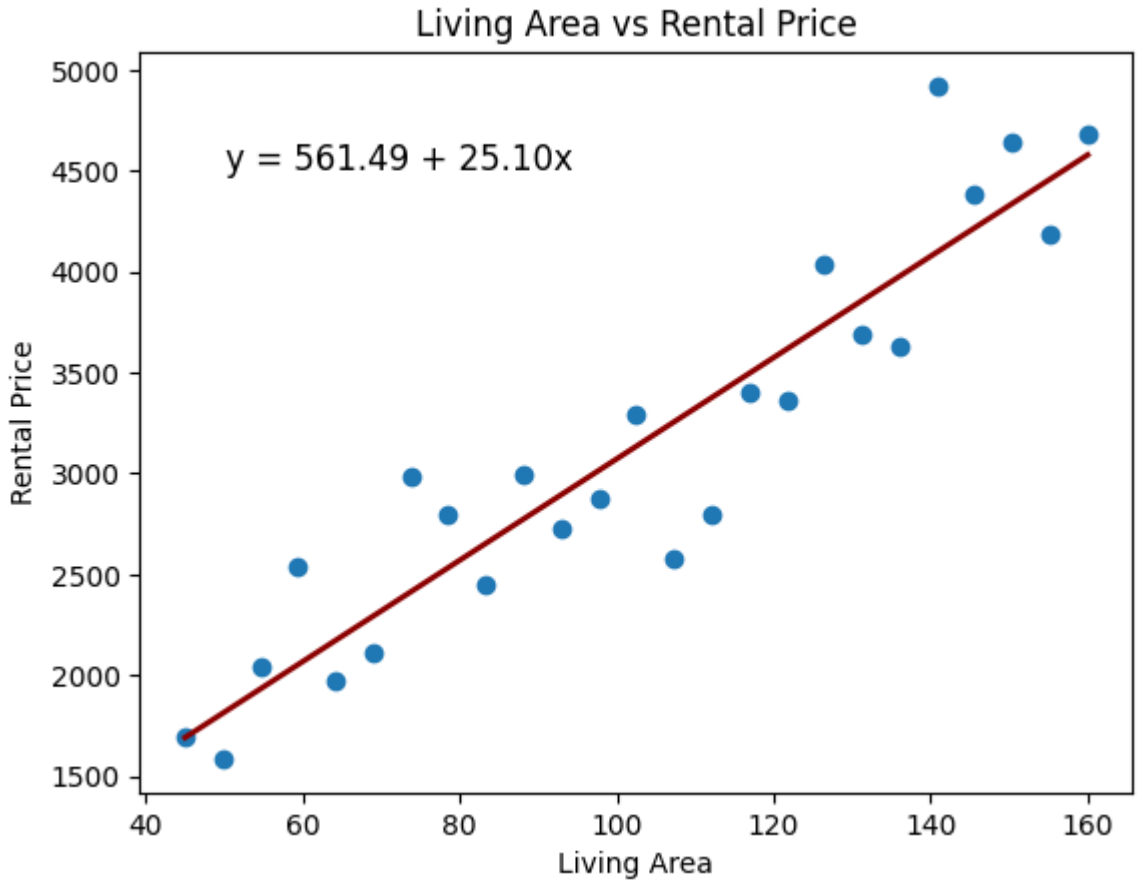
OLS Regression Results

Dep. Variable:	rental_price	R-squared:	0.866
Model:	OLS	Adj. R-squared:	0.860
Method:	Least Squares	F-statistic:	148.2
Date:	Sun, 09 Nov 2025	Prob (F-statistic):	1.66e-11
Time:	16:34:25	Log-Likelihood:	-181.32
No. Observations:	25	AIC:	366.6
Df Residuals:	23	BIC:	369.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	561.4882	223.059	2.517	0.019	100.055	1022.921
living_area	25.1035	2.062	12.173	0.000	20.838	29.369

Omnibus:	0.802	Durbin-Watson:	2.018
Prob(Omnibus):	0.670	Jarque-Bera (JB):	0.416
Skew:	0.314	Prob(JB):	0.812
Kurtosis:	2.941	Cond. No.	339.

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



Generate Prediction- and Confidence Intervals

We calculate the prediction- and confidence intervals for the model.

In [4]:
```
# Generate predictions and confidence intervals
alpha = 0.05
predictions = results.get_prediction(df).summary_frame(alpha)
predictions.head()
```

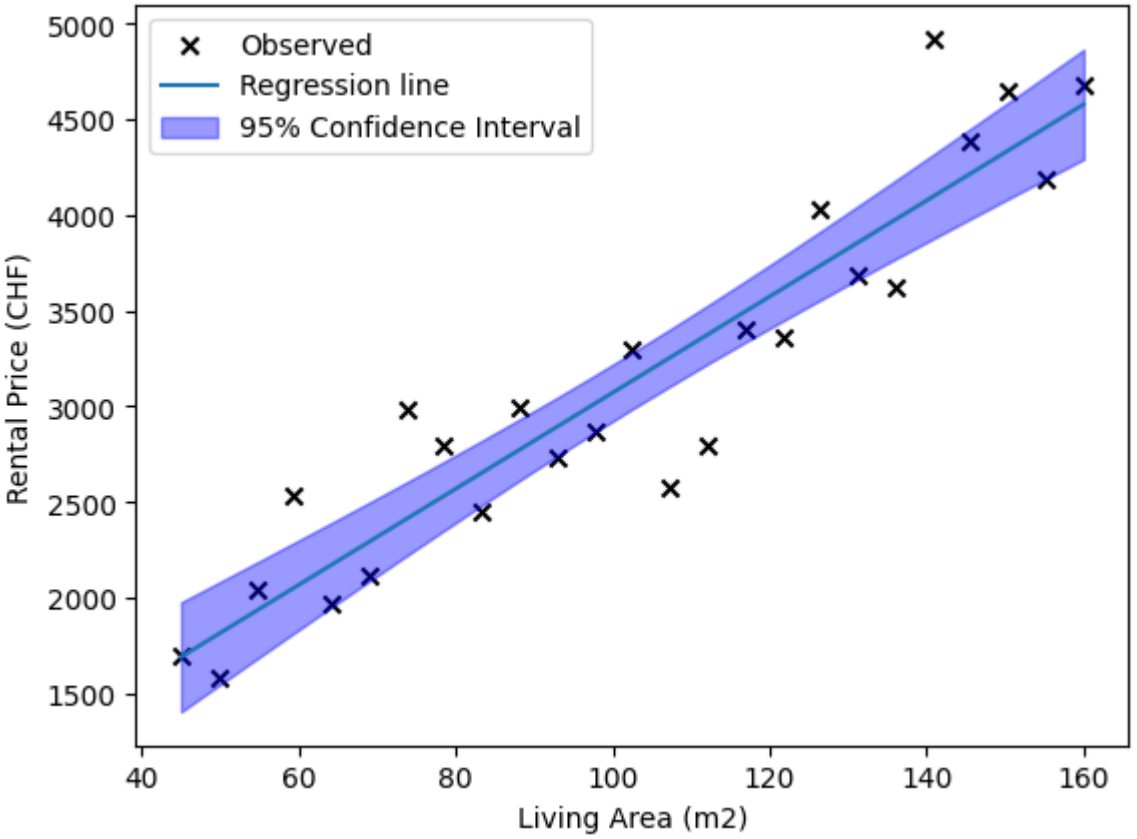
Out[4]:

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	1691.145599	138.337095	1404.973515	1977.317683	900.530640	2481.760559
1	1811.433197	129.967171	1542.575619	2080.290776	1026.919367	2595.947028
2	1931.720795	121.823875	1679.708909	2183.732682	1152.819298	2710.622293
3	2052.008394	113.955800	1816.272860	2287.743927	1278.219796	2825.796992
4	2172.295992	106.424008	1952.141158	2392.450825	1403.110901	2941.481082

Visualizing the Regression Line and Confidence Interval

We now plot the observed data along with the regression line and confidence intervals.

In [5]:
```
# Plot observed data and regression line
plt.scatter(df['living_area'],
            df['rental_price'],
            label='Observed',
            marker='x',
            color='black')

# Plot regression line
plt.plot(df['living_area'],
         predictions['mean'],
         label='Regression line')

# Plot confidence intervals
plt.fill_between(df['living_area'],
                 predictions['mean_ci_lower'],
                 predictions['mean_ci_upper'],
                 color='blue',
                 alpha=0.4,
                 label='95% Confidence Interval')

# Add legend and labels
plt.xlabel('Living Area (m2)')
plt.ylabel('Rental Price (CHF)')
plt.legend()
plt.show()
```

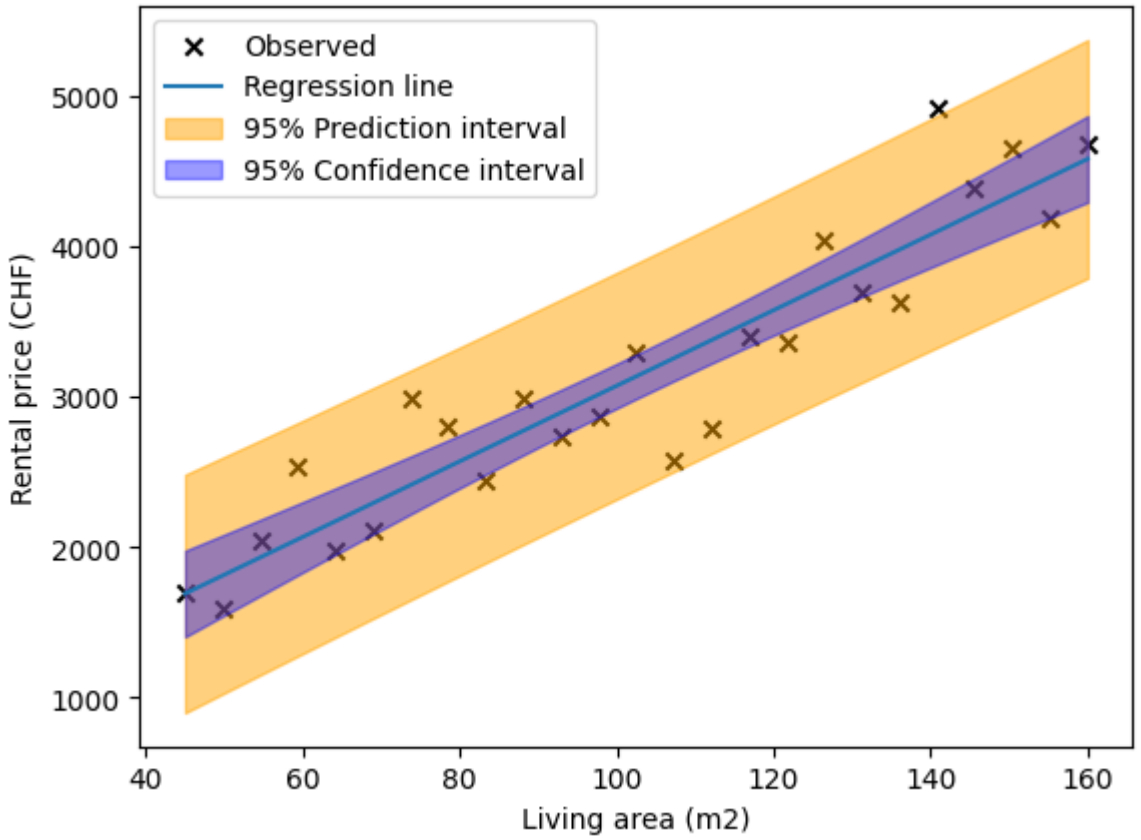


Visualizing the Regression Line and Prediction Interval

We now plot the observed data along with the regression line, confidence- and prediction intervals.

In [6]:
```
# Plot observed data and regression line
plt.scatter(df['living_area'],
            df['rental_price'],
            label='Observed',
            marker='x',
            color='black')

# Plot regression line
plt.plot(df['living_area'],
         predictions['mean'],
         label='Regression line')

# Plot prediction interval
plt.fill_between(df['living_area'],
                 predictions['obs_ci_lower'],
                 predictions['obs_ci_upper'],
                 alpha=.5,
                 label='95% Prediction interval',
                 color='orange')

# Plot confidence interval
plt.fill_between(df['living_area'],
                 predictions['mean_ci_lower'],
                 predictions['mean_ci_upper'],
                 alpha=.4,
```

```
                label='95% Confidence interval',
                color='blue')

# Add labels
plt.xlabel('Living area (m2)')
plt.ylabel('Rental price (CHF)')
plt.legend()

# Show plot
plt.show()
```



Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [7]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:34:25
Python Version: 3.13.9
-----------------------------------
```

Regression Trees & Random Forest Regression (apartment data)

In this notebook, we will use Regression Tree and Random Forest Regressors to predict rental apartment prices based on various features like living area, number of rooms and more. We will evaluate the models using goodness-of-fit measures like R-squared.

Libraries and settings

In [1]:
```
# Libraries
import os
import numpy as np
import pandas as pd
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.datasets import make_regression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\CART_RandomForest
```

Import the apartment data

In [2]:
```
# Define columns for import
columns = [ 'web-scraper-order',
            'address_raw',
            'rooms',
            'area',
            'luxurious',
            'price',
            'price_per_m2',
            'lat',
            'lon',
            'bfs_number',
            'bfs_name',
            'pop',
            'pop_dens',
            'frg_pct',
            'emp',
            'mean_taxable_income',
            'dist_supermarket']

# Read and select variables
```

```
df_orig = pd.read_csv("./Data/apartments_data_enriched_cleaned.csv", sep=";", en

# Rename variable 'web-scraper-order' to 'apmt_id'
df_orig = df_orig.rename(columns={'web-scraper-order': 'id'})

# Remove missing values
df = df_orig.dropna()
df.head(5)

# Remove duplicates
df = df.drop_duplicates()

# Remove some 'extreme' values
df = df.loc[(df['price'] >= 1000) &
            (df['price'] <= 5000)]

# Reset index
df = df.reset_index(drop=True)

print(df.shape)
df.head(5)
```

```
(722, 17)
```

Out[2]:

	id	address_raw	rooms	area	luxurious	price	price_per_m2	
0	1693998201-1	Neuhusstrasse 6, 8630 Rüti ZH, ZH	3.0	49	0	1441	29.41	47.252
1	1693998233-172	Widacherstrasse 5, 8630 Rüti ZH, ZH	3.0	111	0	2600	23.42	47.252
2	1693998256-331	Widenweg 14, 8630 Rüti ZH, ZH	3.0	58	0	1490	25.69	47.253
3	1693998265-381	Rain 1, 8630 Rüti ZH, ZH	4.0	118	0	3240	27.46	47.259
4	1693998276-419	Bachtelstrasse 24b, 8630 Rüti ZH, ZH	3.0	66	0	1450	21.97	47.266

Regression Tree

See also: https://data36.com/regression-tree-python-scikit-learn

Create train and test samples for the regression tree (train = 80%, test = 20% of the data)

In [3]:

```
# Create train and test samples
X_train, X_test, y_train, y_test = train_test_split(df[['area',
                                                        'rooms',
                                                        'pop_dens',
                                                        'mean_taxable_income',
                                                        'dist_supermarket']],
```

```
                                                    df['price'],
                                                    test_size=0.20,
                                                    random_state=42)

# Show X_train
print('X_train:')
print(X_train.head(), '\n')

# Show y_train
print('y_train:')
print(y_train.head())
```

```
X_train:
     area  rooms     pop_dens  mean_taxable_income  dist_supermarket
456   120    6.5   165.018625          65380.98802       1958.318650
6      65    3.5   525.662252         111422.35870        282.095910
362    90    3.5   424.146342          72133.75058       2606.231069
594    63    2.5  1044.628957          70964.08794        564.607066
439    91    3.5   399.525129          62000.54187        646.823905

y_train:
456    1900
6      1850
362    2090
594    2102
439    1800
Name: price, dtype: int64
```

Fit the regression tree model

In [4]:
```
# Create decision tree regressor object
reg = DecisionTreeRegressor(random_state=20, max_depth=3)

# Train decision tree regressor
reg = reg.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = reg.predict(X_test)
```

Calculate coefficient of determination (R-squared)

In [5]:
```
# Calculate coefficient of determination
print('R-squared:', round(r2_score(y_test, y_pred), 4))
```

```
R-squared: 0.4502
```

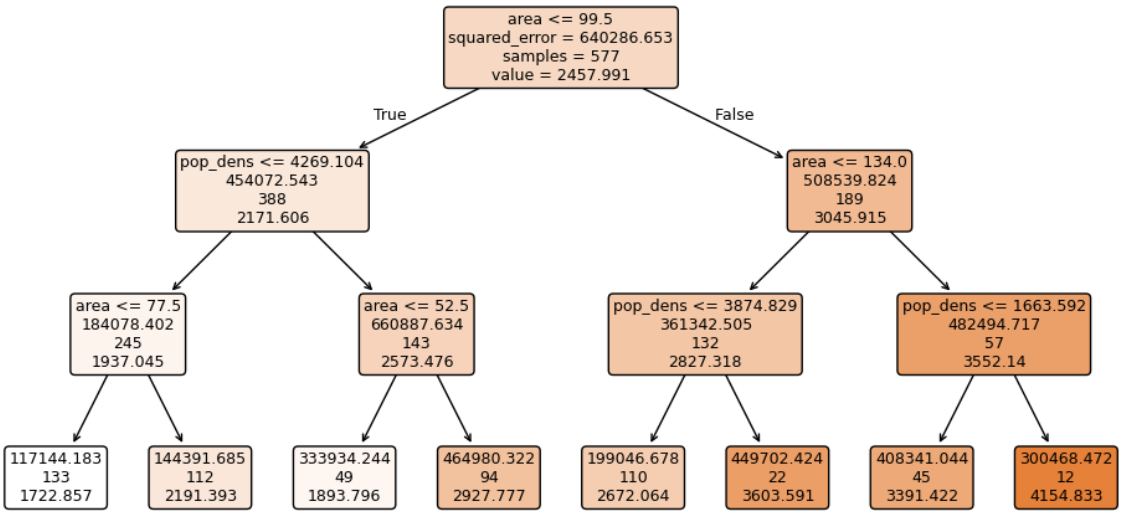
Print text representation of the regression tree

In [6]:
```
# Text representation of the regression tree
text_representation = tree.export_text(reg,
                                       feature_names=list(X_train.columns))

# Print text_representation
print(text_representation)
```

```
|--- area <= 99.50
|   |--- pop_dens <= 4269.10
|   |   |--- area <= 77.50
|   |   |   |--- value: [1722.86]
|   |   |--- area >  77.50
|   |   |   |--- value: [2191.39]
|   |--- pop_dens >  4269.10
|   |   |--- area <= 52.50
|   |   |   |--- value: [1893.80]
|   |   |--- area >  52.50
|   |   |   |--- value: [2927.78]
|--- area >  99.50
|   |--- area <= 134.00
|   |   |--- pop_dens <= 3874.83
|   |   |   |--- value: [2672.06]
|   |   |--- pop_dens >  3874.83
|   |   |   |--- value: [3603.59]
|   |--- area >  134.00
|   |   |--- pop_dens <= 1663.59
|   |   |   |--- value: [3391.42]
|   |   |--- pop_dens >  1663.59
|   |   |   |--- value: [4154.83]
```

Vizualizing the regression tree

In [7]:
```
fig = plt.figure(figsize=(12,6))
_ = tree.plot_tree(reg,
                   feature_names=list(X_train.columns),
                   class_names=['price'],
                   filled=True,
                   fontsize=9,
                   label='root',
                   rounded=True)
```



Random Forest Regression

For details see: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Create train and test samples for the random forest (train = 80%, test = 20% of the data)

In [8]:
```
# Create train and test samples (the names X2_ and y2_ were used because X_ and
X2_train, X2_test, y2_train, y2_test = train_test_split(df[['area',
                                                            'rooms',
                                                            'pop_dens',
                                                            'mean_taxable_income
                                                            'dist_supermarket']]
                                                           df['price'],
                                                           test_size=0.20,
                                                           random_state=42)


# Show X2_train
print('X2_train:')
print(X2_train.head(), '\n')

# Show y2_train
print('y2_train:')
print(y2_train.head())
```

```
X2_train:
     area  rooms     pop_dens  mean_taxable_income  dist_supermarket
456   120    6.5   165.018625          65380.98802       1958.318650
6      65    3.5   525.662252         111422.35870        282.095910
362    90    3.5   424.146342          72133.75058       2606.231069
594    63    2.5  1044.628957          70964.08794        564.607066
439    91    3.5   399.525129          62000.54187        646.823905

y2_train:
456    1900
6      1850
362    2090
594    2102
439    1800
Name: price, dtype: int64
```

Fit the Random Forest Regression

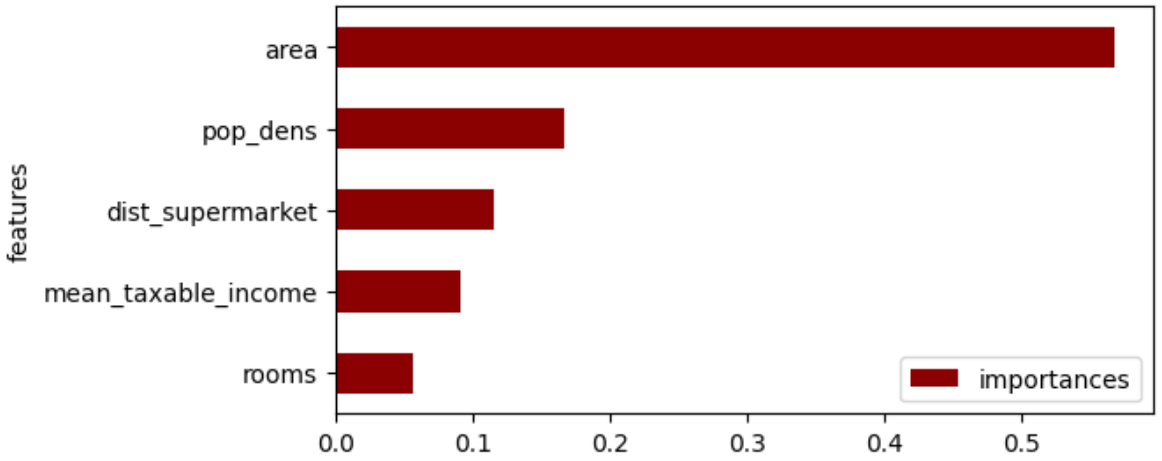
In [9]:
```
X, y = make_regression(n_features=4, n_informative=2,
                       random_state=5, shuffle=False)


reg_rf = RandomForestRegressor(n_estimators=500,
                               max_depth=10,
                               random_state=5)
reg_rf.fit(X2_train, y2_train)

# Calculate coefficient of determination (R-squared)
print('R-squared:', round(reg_rf.score(X2_test, y2_test), 4))
```

```
R-squared: 0.5481
```

Show feature importance

In [10]:
```
cols = X2_train.columns

# Derive feature importance from random forest
importances = reg_rf.feature_importances_
std         = np.std([tree.feature_importances_ for tree in reg_rf.estimators_],
indices     = np.argsort(importances)[::-1]

# Print col-names and importances-values
print( cols[indices] )
print( importances[indices] )

# Barplot with feature importance
df_fi = pd.DataFrame({'features':cols,'importances': importances})
df_fi.sort_values('importances', inplace=True)
df_fi.plot(kind='barh',
           y='importances',
           x='features',
           color='darkred',
           figsize=(6,3))

plt.show()
```

```
Index(['area', 'pop_dens', 'dist_supermarket', 'mean_taxable_income', 'rooms'], d
type='object')
[0.56749536 0.16751781 0.11611526 0.09147853 0.05739305]
```



Jupyter notebook --footer info-- (please always provide this at the end of each submitted notebook)

In [11]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
```

```
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:29:56
Python Version: 3.13.9
-----------------------------------
```

Classification Trees and Random Forest Classification (supermarket data)

In this notebook, we will use Classification Tree and Random Forest classifiers to predict supermarket brands based on various features like location, population density, brand, and more. We will evaluate the models using feature importance, confusion matrices, and ROC curves.

Libraries and settings

In [1]:
```
# Libraries
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import Image

from sklearn import tree
from sklearn.metrics import RocCurveDisplay
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\CART_RandomForest
```

Import supermarkets data

In [2]:
```
# Read and select variables
columns = ['id',
           'bfs_name',
           'bfs_number',
           'lat',
           'lon',
           'brand',
           'pop',
           'pop_dens',
           'frg_pct',
           'emp',
          ]
df_supermarkets_orig = pd.read_csv("./Data/supermarkets_data_enriched.csv",
                                   sep=",",
                                   encoding="utf-8")[columns]
```

```
# Number of rows and columns
print(df_supermarkets_orig.shape)

# Brand to upper case letters
df_supermarkets_orig['brand'] = df_supermarkets_orig['brand'].str.upper()

# First records
df_supermarkets_orig.head(5)
```

```
(3242, 10)
```

Out[2]:

	id	bfs_name	bfs_number	lat	lon	brand	pop	pop_dens
0	33126515	Schänis	3315	47.155616	9.037915	SPAR	3876	97.142857
1	280130028	Schänis	3315	47.155492	9.039666	ALDI	3876	97.142857
2	6122906632	Schänis	3315	47.158959	9.044477	DENNER	3876	97.142857
3	9019298862	Schänis	3315	47.155185	9.038472	LIDL	3876	97.142857
4	36726161	Uznach	3339	47.226191	8.980329	MIGROS	6489	860.610080

Count and remove missing values

In [3]:

```
# Count missing values
print(df_supermarkets_orig.isna().sum())

# Remove missing values
df_supermarkets = df_supermarkets_orig.dropna()
df_supermarkets
```

```
id              0
bfs_name        0
bfs_number      0
lat             0
lon             0
brand        1233
pop             0
pop_dens        0
frg_pct         0
emp            45
dtype: int64
```

Out[3]:

	id	bfs_name	bfs_number	lat	lon	brand	pop	pop_de
0	33126515	Schänis	3315	47.155616	9.037915	SPAR	3876	97.1428
1	280130028	Schänis	3315	47.155492	9.039666	ALDI	3876	97.1428
2	6122906632	Schänis	3315	47.158959	9.044477	DENNER	3876	97.1428
3	9019298862	Schänis	3315	47.155185	9.038472	LIDL	3876	97.1428
4	36726161	Uznach	3339	47.226191	8.980329	MIGROS	6489	860.6100
...	...	...	...	...	...	...	...	
3230	9584570723	Seedorf (UR)	1214	46.881529	8.615975	VOLG	2051	106.3796
3231	9593770082	Bäretswil	111	47.339296	8.839173	VOLG	5053	227.7151
3233	9624205242	Buch am Irchel	24	47.549645	8.618709	VOLG	979	95.8863
3238	9950926547	Marbach (SG)	3253	47.392404	9.569855	VOLG	2110	481.7351
3239	9963973121	Rüte	3103	47.321761	9.426943	VOLG	3692	90.4458

1979 rows × 10 columns

Subset with selected brands

In [4]:
```
df_sub = df_supermarkets.loc[df_supermarkets['brand'].isin(['MIGROS', 'VOLG'])]
print(df_sub.shape)
df_sub.head()
```

```
(696, 10)
```

Out[4]:

	id	bfs_name	bfs_number	lat	lon	brand	pop	pop_de
4	36726161	Uznach	3339	47.226191	8.980329	MIGROS	6489	860.6100
8	48932835	Zürich	261	47.375020	8.522895	MIGROS	420217	4778.9946
11	83330862	Zürich	261	47.344749	8.529981	MIGROS	420217	4778.9946
12	119249170	Zürich	261	47.375255	8.536107	MIGROS	420217	4778.9946
15	262400822	Zürich	261	47.364072	8.530945	MIGROS	420217	4778.9946

Pivot table

In [5]:
```
# Using pivot_table to reshape the data and calculate means
pd.pivot_table(df_sub[['lat',
                       'lon',
                       'brand',
                       'pop',
                       'pop_dens',
                       'frg_pct',
                       'emp']],
```

```
                index=['brand'],
                values=['lat', 'lon', 'pop', 'pop_dens', 'frg_pct', 'emp'],
                aggfunc=(np.mean, 'count')).round(0)
```

Out[5]:

	emp		frg_pct		lat		lon		pop	
	count	mean	count	mean	count	mean	count	mean	count	mean
brand										
MIGROS	486	60264.0	486	28.0	486	47.0	486	8.0	486	58452.0
VOLG	210	2382.0	210	16.0	210	47.0	210	9.0	210	4811.0

Classification Tree

For details see: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Create train and test samples (train = 80%, test = 20% of the data)

In [6]:
```
# Create train and test samples
X_train, X_test, y_train, y_test = train_test_split(df_sub[['lat',
                                                            'lon',
                                                            'pop',
                                                            'pop_dens',
                                                            'frg_pct',
                                                            'emp']],
                                                    df_sub['brand'],
                                                    test_size=0.20,
                                                    random_state=42)

# Show X_train
print('X_train:')
print(X_train.head(), '\n')

# Show y_train
print('y_train:')
print(y_train.head())
```

```
X_train:
            lat       lon     pop      pop_dens    frg_pct       emp
427   46.767777  9.062487    1728     17.891903   8.217593     725.0
271   47.552670  7.592713  173232   7263.396226  37.955459  185432.0
1149  46.203871  6.146234  203951  12810.992462  47.954656  186620.0
2655  46.774599  9.207659    4757     35.638298  16.312802    3229.0
2369  47.126317  7.246363    6872   4521.052632  26.062282    3185.0

y_train:
427       VOLG
271     MIGROS
1149    MIGROS
2655    MIGROS
2369    MIGROS
Name: brand, dtype: object
```

Fit the classification tree model and make predictions

In [7]:
```
# Initialize the classification tree model
clf = DecisionTreeClassifier(random_state=42,
                             max_depth=5)

# Train the classification tree model
clf = clf.fit(X_train, y_train)

# Make model predictions
y_pred = clf.predict(X_test)
y_pred
```

Out[7]:
```
array(['MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'VOLG', 'VOLG', 'VOLG', 'VOLG', 'VOLG',
       'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS', 'VOLG', 'VOLG',
       'MIGROS', 'VOLG', 'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'VOLG', 'VOLG', 'VOLG', 'MIGROS', 'VOLG',
       'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'VOLG', 'VOLG',
       'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'VOLG',
       'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'VOLG', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS', 'VOLG',
       'VOLG', 'MIGROS', 'VOLG', 'VOLG', 'VOLG', 'MIGROS', 'VOLG', 'VOLG',
       'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'VOLG', 'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS',
       'MIGROS', 'MIGROS', 'MIGROS', 'MIGROS'], dtype=object)
```

Show confusion matrix and classification report

In [8]:
```
# Confusion matrix
print('Confusion matrix')
print(confusion_matrix(y_test, y_pred), '\n')

# Classification report
print('Classification report')
print(classification_report(y_test, y_pred))
```

```
Confusion matrix
[[97 11]
 [ 7 25]]

Classification report
              precision    recall  f1-score   support

      MIGROS       0.93      0.90      0.92       108
        VOLG       0.69      0.78      0.74        32

    accuracy                           0.87       140
   macro avg       0.81      0.84      0.83       140
weighted avg       0.88      0.87      0.87       140
```

Print text representation of the classification tree

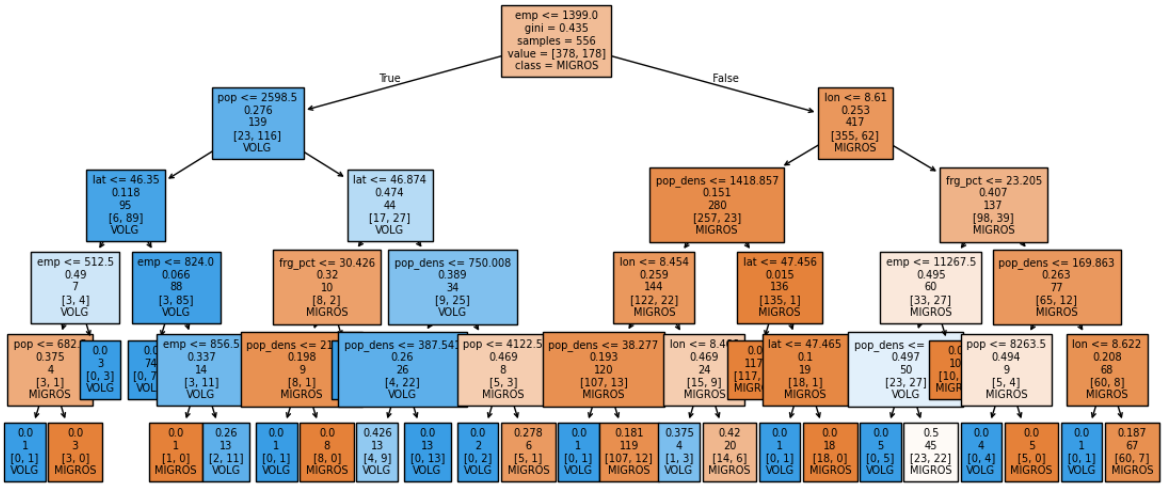
In [9]:
```
# Text representation of the classification tree
text_rep = tree.export_text(clf,
                            feature_names=list(X_train.columns))

# Print text_representation
print(text_rep)
```

```
|--- emp <= 1399.00
|   |--- pop <= 2598.50
|   |   |--- lat <= 46.35
|   |   |   |--- emp <= 512.50
|   |   |   |   |--- pop <= 682.50
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- pop >  682.50
|   |   |   |   |   |--- class: MIGROS
|   |   |   |--- emp >  512.50
|   |   |   |   |--- class: VOLG
|   |   |--- lat >  46.35
|   |   |   |--- emp <= 824.00
|   |   |   |   |--- class: VOLG
|   |   |   |--- emp >  824.00
|   |   |   |   |--- emp <= 856.50
|   |   |   |   |   |--- class: MIGROS
|   |   |   |   |--- emp >  856.50
|   |   |   |   |   |--- class: VOLG
|   |--- pop >  2598.50
|   |   |--- lat <= 46.87
|   |   |   |--- frg_pct <= 30.43
|   |   |   |   |--- pop_dens <= 21.35
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- pop_dens >  21.35
|   |   |   |   |   |--- class: MIGROS
|   |   |   |--- frg_pct >  30.43
|   |   |   |   |--- class: VOLG
|   |   |--- lat >  46.87
|   |   |   |--- pop_dens <= 750.01
|   |   |   |   |--- pop_dens <= 387.54
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- pop_dens >  387.54
|   |   |   |   |   |--- class: VOLG
|   |   |   |--- pop_dens >  750.01
|   |   |   |   |--- pop <= 4122.50
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- pop >  4122.50
|   |   |   |   |   |--- class: MIGROS
|--- emp >  1399.00
|   |--- lon <= 8.61
|   |   |--- pop_dens <= 1418.86
|   |   |   |--- lon <= 8.45
|   |   |   |   |--- pop_dens <= 38.28
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- pop_dens >  38.28
|   |   |   |   |   |--- class: MIGROS
|   |   |   |--- lon >  8.45
|   |   |   |   |--- lon <= 8.47
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- lon >  8.47
|   |   |   |   |   |--- class: MIGROS
|   |   |--- pop_dens >  1418.86
|   |   |   |--- lat <= 47.46
|   |   |   |   |--- class: MIGROS
|   |   |   |--- lat >  47.46
|   |   |   |   |--- lat <= 47.46
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- lat >  47.46
|   |   |   |   |   |--- class: MIGROS
|   |--- lon >  8.61
```

```
|   |   |--- frg_pct <= 23.20
|   |   |   |--- emp <= 11267.50
|   |   |   |   |--- pop_dens <= 28.81
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- pop_dens >  28.81
|   |   |   |   |   |--- class: MIGROS
|   |   |   |--- emp >  11267.50
|   |   |   |   |--- class: MIGROS
|   |   |--- frg_pct >  23.20
|   |   |   |--- pop_dens <= 169.86
|   |   |   |   |--- pop <= 8263.50
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- pop >  8263.50
|   |   |   |   |   |--- class: MIGROS
|   |   |   |--- pop_dens >  169.86
|   |   |   |   |--- lon <= 8.62
|   |   |   |   |   |--- class: VOLG
|   |   |   |   |--- lon >  8.62
|   |   |   |   |   |--- class: MIGROS
```

Visualize the classification tree

In [10]:
```
# For the meaning of numbers in boxes see root node
fig = plt.figure(figsize=(14,6))
tree_plot = tree.plot_tree(clf,
                           feature_names=list(X_train.columns),
                           class_names=['MIGROS', 'VOLG'],
                           filled=True,
                           fontsize=7,
                           label='root')
```



Random Forest Classifier

For details see: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Create train and test samples (train = 80%, test = 20% of the data)

In [11]:
```
# Create train and test samples
X2_train, X2_test, y2_train, y2_test = train_test_split(df_sub[['lat',
                                                                'lon',
                                                                'pop',
                                                                'pop_dens',
                                                                'frg_pct',
                                                                'emp']],
                                                        df_sub['brand'],
                                                        test_size=0.20,
                                                        random_state=5)

# Show X2_train
print('X2_train:')
print(X2_train.head(), '\n')

# Show y2_train
print('y2_train:')
print(y2_train.head())
```

```
X2_train:
            lat       lon    pop      pop_dens    frg_pct     emp
2814  47.388170  8.253929   2445   315.483871  14.683027   858.0
2425  46.259128  7.902847   1337    25.895797   4.936425   539.0
2473  47.169307  7.524045   2844   634.821429   8.368495  1037.0
2705  46.527225  6.602452  12413  5642.272727  41.843229  6748.0
2839  47.450087  9.650465   5960   867.540029  49.580537  3158.0

y2_train:
2814      VOLG
2425      VOLG
2473      VOLG
2705    MIGROS
2839    MIGROS
Name: brand, dtype: object
```

Fit the Random Forest Classifier

In [12]:
```
# Initialize the random forest classifier
rfc = RandomForestClassifier(n_estimators=500, random_state=42, max_depth=5)

# Train the classification tree model
rfc = rfc.fit(X2_train, y2_train)

# Predict the target variable
y_pred_rf = rfc.predict(X2_test)

# Crosstab
print('Brand versus predicted brand:', '\n')
df_pred = pd.DataFrame(data={'brand': y2_test,
                             'brand_predicted': y_pred_rf}).reset_index(drop=Tru
df_pred
```

```
Brand versus predicted brand:
```

Out[12]:

	brand	brand_predicted
0	MIGROS	MIGROS
1	MIGROS	MIGROS
2	MIGROS	MIGROS
3	VOLG	MIGROS
4	MIGROS	MIGROS
...	...	...
135	MIGROS	MIGROS
136	MIGROS	MIGROS
137	MIGROS	MIGROS
138	VOLG	MIGROS
139	MIGROS	MIGROS

140 rows × 2 columns

Show confusion matrix and classification report

In [13]:
```
# Confusion matrix
print('Confusion matrix')
print(confusion_matrix(y2_test, y_pred_rf), '\n')

# Classification report
print('Classification report')
print(classification_report(y2_test, y_pred_rf))
```

```
Confusion matrix
[[98  6]
 [ 8 28]]

Classification report
              precision    recall  f1-score   support

      MIGROS       0.92      0.94      0.93       104
        VOLG       0.82      0.78      0.80        36

    accuracy                           0.90       140
   macro avg       0.87      0.86      0.87       140
weighted avg       0.90      0.90      0.90       140
```

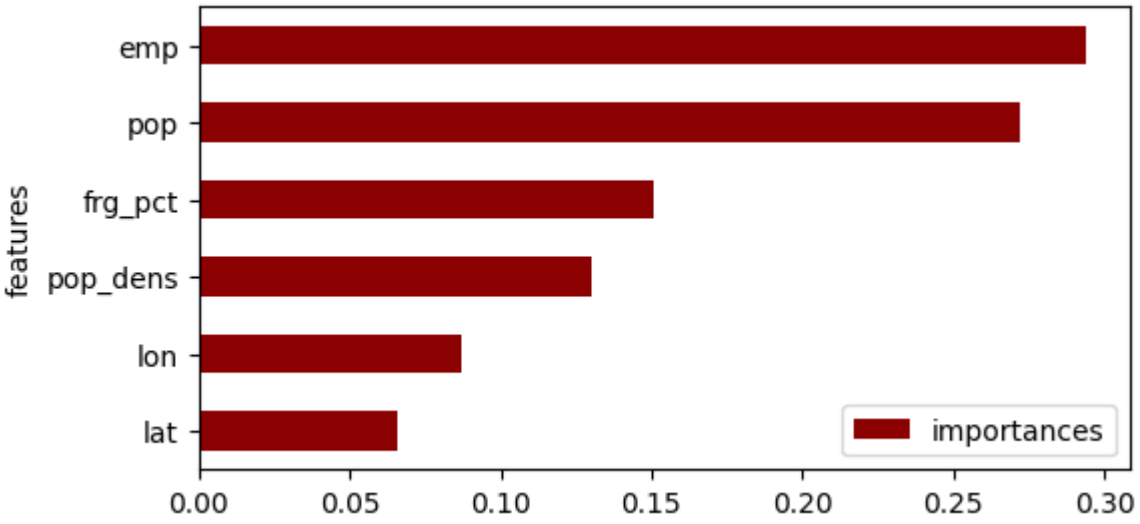
Show feature importance

In [14]:
```
cols = X2_train.columns

# Derive feature importance from the random forest classifier
importances = rfc.feature_importances_
std = np.std([tree.feature_importances_ for tree in rfc.estimators_], axis=0)
indices = np.argsort(importances)[::-1]
```

```
# Print col-names and importances-values
print( cols[indices] )
print( importances[indices] )

# Barplot with feature importance
df_fi = pd.DataFrame({'features':cols,'importances': importances})
df_fi.sort_values('importances', inplace=True)
df_fi.plot(kind='barh',
           y='importances',
           x='features',
           color='darkred',
           figsize=(6,3))
```

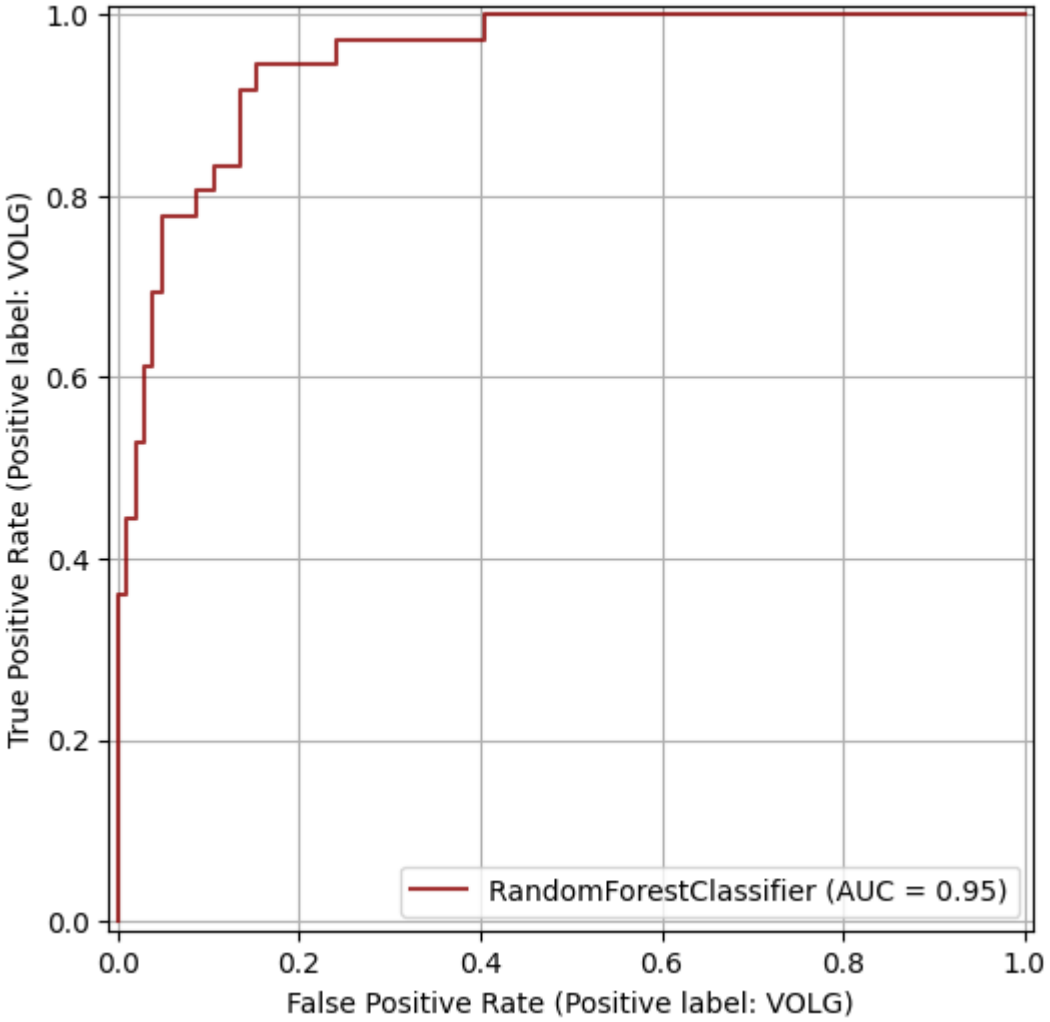
```
Index(['emp', 'pop', 'frg_pct', 'pop_dens', 'lon', 'lat'], dtype='object')
[0.29380007 0.27210589 0.15087431 0.13026407 0.08730287 0.06565278]
```

Out[14]: <Axes: ylabel='features'>



ROC curve and AUC

In [15]:
```
# Plot ROC curve and calculate AUC
plt.figure(figsize=(6,6))
ax = plt.gca()
rfc_disp = RocCurveDisplay.from_estimator(rfc,
                                          X2_test,
                                          y2_test,
                                          ax=ax,
                                          alpha=0.8,
                                          c="darkred")
plt.grid()
plt.show()
```



Jupyter notebook --footer info-- (please always provide this at the end of each submitted notebook)

In [16]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:29:30
Python Version: 3.13.9
-----------------------------------
```

K-Means Clustering

K-Means Clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a set of distinct, non-overlapping groups (clusters). The goal is to divide the data points into (K) clusters where each data point belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

Table of contents

Libraries and settings

In [1]:
```
# Libraries
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Current working directory
print('Current working directory:', os.getcwd())
```

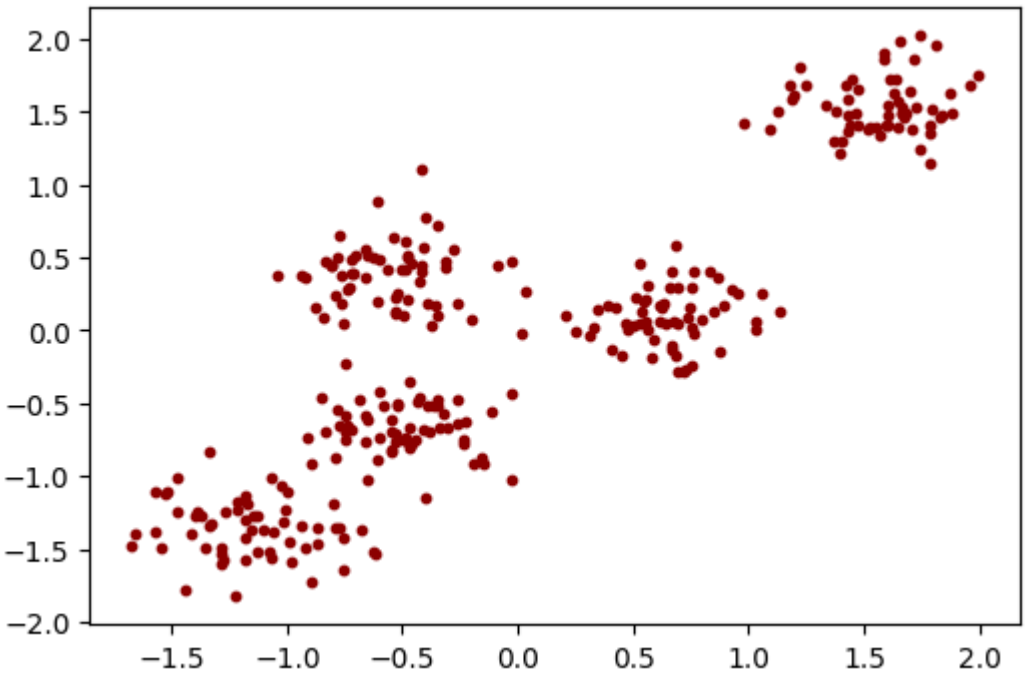
```
Current working directory: c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\K-Means
```

Example (1): Simple k-means clustering example

Create the dataset

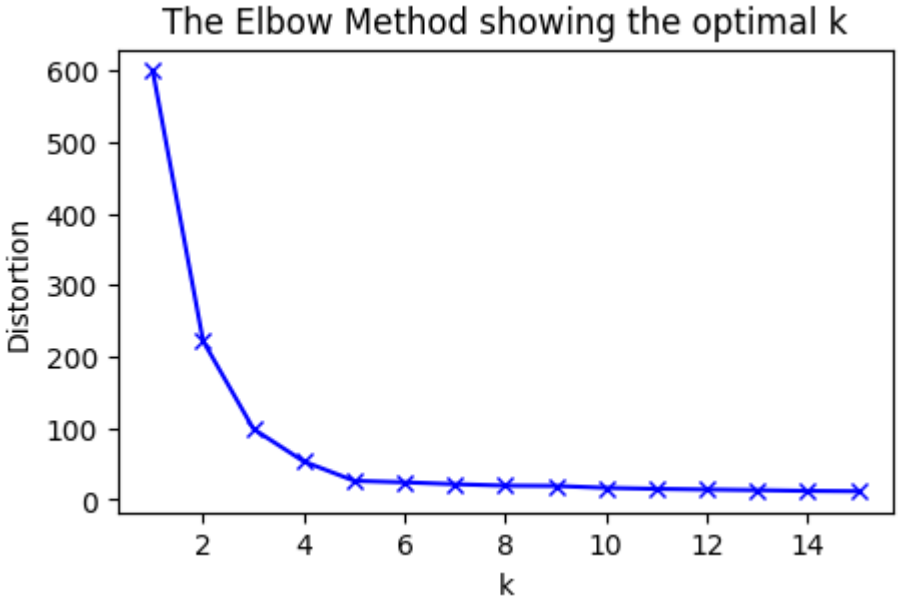
In [2]:
```
# Create data
centers = [[2,1], [-2,2], [-2,-2], [-4,-5], [5,7]]
X, y = make_blobs(n_samples=300,
                  centers=centers,
                  cluster_std=0.8,
                  random_state=42)

# Normalization of the values
```

```
X = StandardScaler().fit_transform(X)

# Plot the data
plt.figure(figsize=(6,4))
plt.scatter(X[:,0], X[:,1], s=10, color='darkred')
plt.show()
```



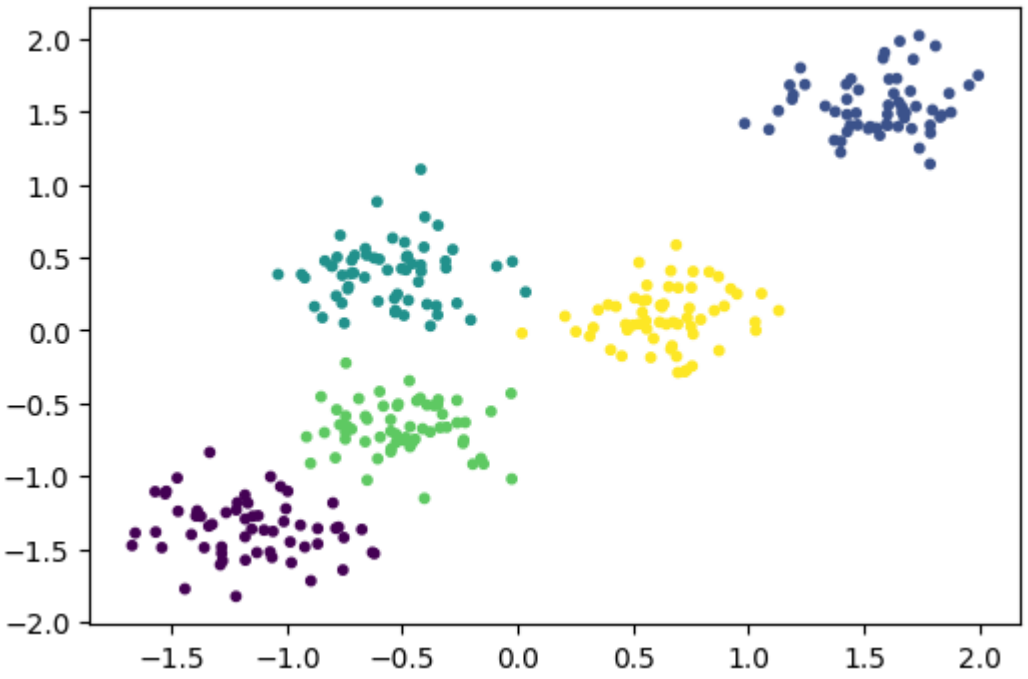
Elbow Method showing the optimal k

In [3]:
```
# Sum of squared distances of samples to their closest cluster center
distortions = []

# Range of k's
K = range(1,16,1)

# Loop to find the optimal k
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X)
    distortions.append(kmeanModel.inertia_)

# Elbow plot
plt.figure(figsize=(5,3))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')

plt.show()
```



Perform k-means clustering

In [4]:
```
# Number of clusters
k = 5

# k-means clustering
kmeans = KMeans(n_clusters=k, random_state=42).fit(X)

# Predict the values
y2 = kmeans.predict(X)

# Plot the clusters
plt.figure(figsize=(6,4))
plt.scatter(X[:, 0], X[:, 1], c=y2, s=10)
plt.show()
```



Get and check the converged cluster centroids

In [5]:
```
# Print centroids
centroids = kmeans.cluster_centers_
print('Cluster centroids:')
print(centroids, '\n')

# Check the 1st cluster's centroid 'by hand'
clust_00 = X[y2 == 0]
print('The 1st cluster\'s centroid:')
print(f'x = {sum(clust_00[:,0]/len(clust_00[:,0])):.8f}')
print(f'y = {sum(clust_00[:,1]/len(clust_00[:,1])):.8f}')
```

```
Cluster centroids:
[[-1.16214999 -1.36217282]
 [ 1.56276666  1.54186443]
 [-0.55467879  0.3889481 ]
 [-0.49867406 -0.67040574]
 [ 0.64167824  0.09023659]]

The 1st cluster's centroid:
x = -1.16214999
y = -1.36217282
```

Get the inertia or 'within-cluster sum-of-squares (WCSS)' of the k-means model

In [6]:
```
print(f'Within-cluster sum-of-squares: {kmeans.inertia_:.4f}')
```

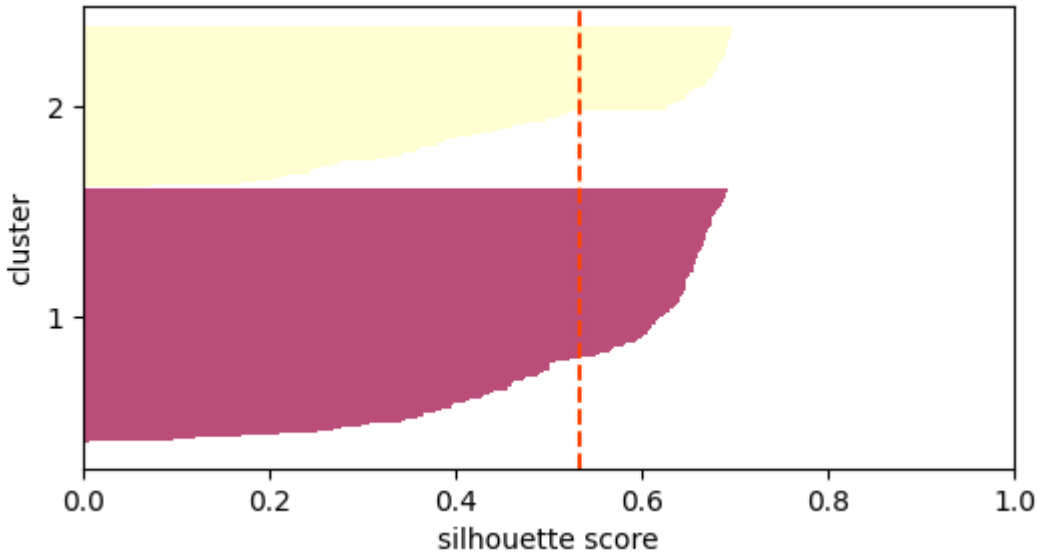
```
Within-cluster sum-of-squares: 26.3487
```

Perform Silhouette Analysis

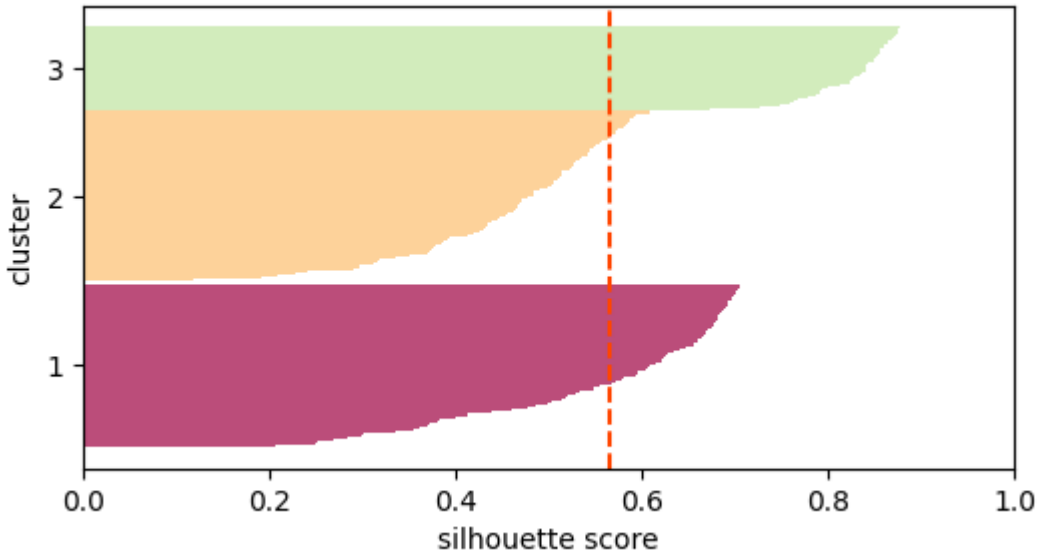
- For examples see:
- https://laid-back-scientist.com/en/k-means
- https://machinelearninggeek.com/evaluating-clustering-methods
- https://medium.com/@favourphilic/how-to-interpret-silhouette-plot-for-k-means-clustering-414e144a17fe

In [7]:
```
# Import own module for Silhouette plots
from silhouette import *

# Create Silhouette plots for different k's
# Note: range(2,12,1) provides: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
for i in [2,3]:
    model = KMeans(n_clusters=i,
                   random_state=42,
                   init='random')
    model.fit(X)
    print(f'k={i}, Silhouette Score: {silhouette_score(X, model.labels_):.4f}')
    plt.figure(figsize=(6,3))
    show_silhouette(X=X, fitted_model=model)
```

```
k=2, Silhouette Score: 0.5325
```




```
k=3, Silhouette Score: 0.5659
```

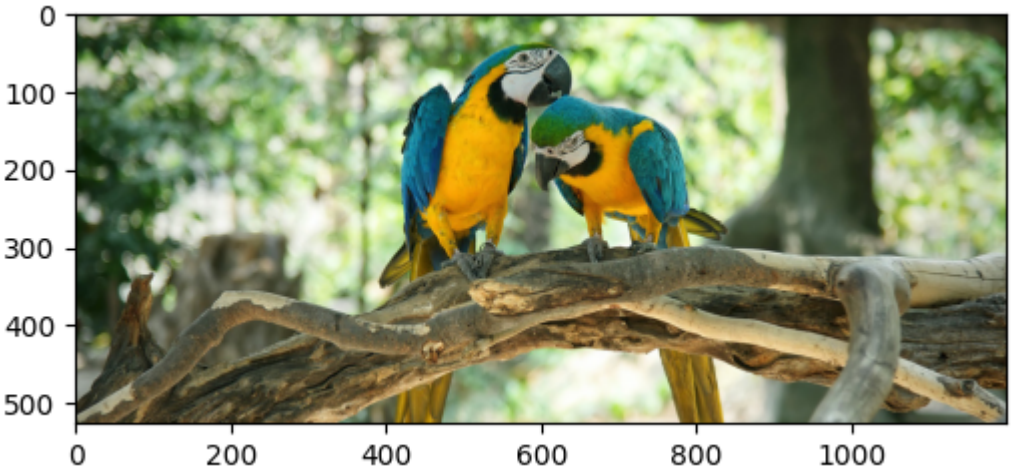


Example (2): Image segmentation

Read the image

In [8]:
```
# Read the image
image = cv2.imread('./Data/parrot.jpg')

# Change the color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Plot the image
plt.figure(figsize=(6,8))
plt.imshow(image)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x2887b3b7e00>
```



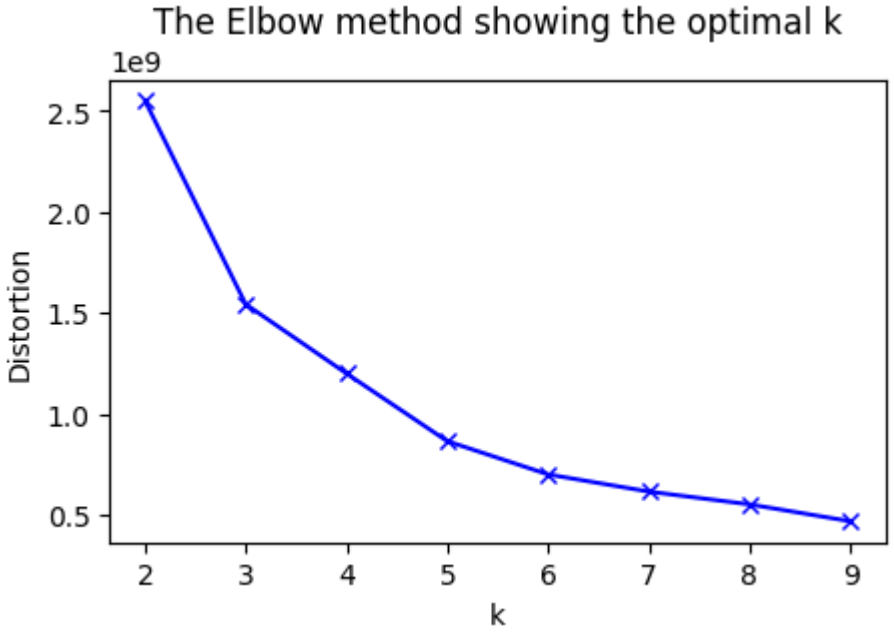
Reshape the image

In [9]:
```
# Reshaping the image into a 2D array of pixels and RGB colors
pixel_vals = image.reshape((-1,3))

# Convert to float
pixel_vals = np.float32(pixel_vals)
pixel_vals
```

Out[9]:
```
array([[ 37.,  68.,  37.],
       [ 37.,  70.,  39.],
       [ 43.,  76.,  45.],
       ...,
       [102., 112., 113.],
       [102., 114., 114.],
       [102., 114., 114.]], shape=(632400, 3), dtype=float32)
```

Elbow method showing the optimal k

In [10]:
```
# Sum of squared distances of samples to their closest cluster center
distortions = []

# Range of k's
K = range(2,10,1)

# Loop to find the optimal k
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(pixel_vals)
    distortions.append(kmeanModel.inertia_)

# Elbow plot
plt.figure(figsize=(5,3))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow method showing the optimal k')

plt.show()
```



Perform image segmentation

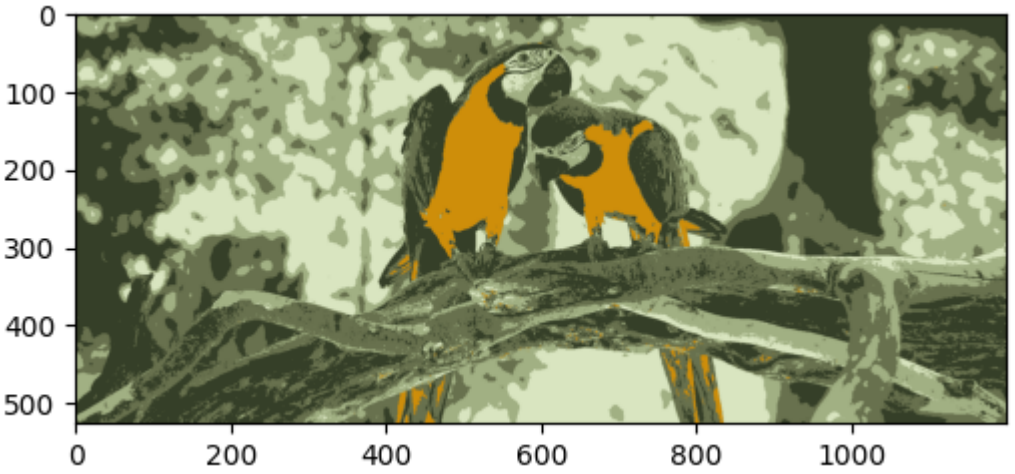
In [11]:
```
# Number of clusters
k = 5

# Criteria for the segmentation algorithm to stop running
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)

# Perform k-means clustering
retval, labels, centers = cv2.kmeans(pixel_vals,
                                     k,
                                     None,
                                     criteria,
                                     10,
                                     cv2.KMEANS_RANDOM_CENTERS)

# Print cluster labels
print('Cluster labels:')
print(labels, '\n')

# Print cluster centroids
print(f'Centroids of {k} clusters')
print(centers)
```

```
Cluster labels:
[[2]
 [2]
 [2]
 ...
 [4]
 [4]
 [4]]

Centroids of 5 clusters
[[161.7978   176.42497  131.57068 ]
 [206.1085   142.65912   10.698247]
 [ 53.6455    63.024746  40.379005]
 [217.82266  229.63536  192.77307 ]
 [104.76609  113.03692   78.56786 ]]
```

Change data types and reshape the segmented data for visualization

In [12]:
```
# Convert data into 8-bit values
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]

# Reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))

# Show result
plt.figure(figsize=(6,8))
plt.imshow(segmented_image)
```

Out[12]: <matplotlib.image.AxesImage at 0x2887fb951d0>



Example (3): Finding clusters in the apartment data

Import the apartment data

In [13]:
```
# Define columns for import
columns = [ 'web-scraper-order',
            'address_raw',
            'rooms',
            'area',
            'luxurious',
            'price',
            'price_per_m2',
            'lat',
            'lon',
            'bfs_number',
            'bfs_name',
            'pop',
            'pop_dens',
            'frg_pct',
            'emp',
            'mean_taxable_income',
            'dist_supermarket']

# Read and select variables
df_orig = pd.read_csv("./Data/apartments_data_enriched_cleaned.csv", sep=";", en

# Rename variable 'web-scraper-order' to 'apmt_id'
df_orig = df_orig.rename(columns={'web-scraper-order': 'id'})

# Remove missing values
df = df_orig.dropna()
df.head(5)

# Remove duplicates
df = df.drop_duplicates()

# Remove some 'extreme' values
df = df.loc[(df['price'] >= 1000) &
            (df['price'] <= 5000)]

print(df.shape)
```

```
df.head(5)
```

```
(722, 17)
```

Out[13]:

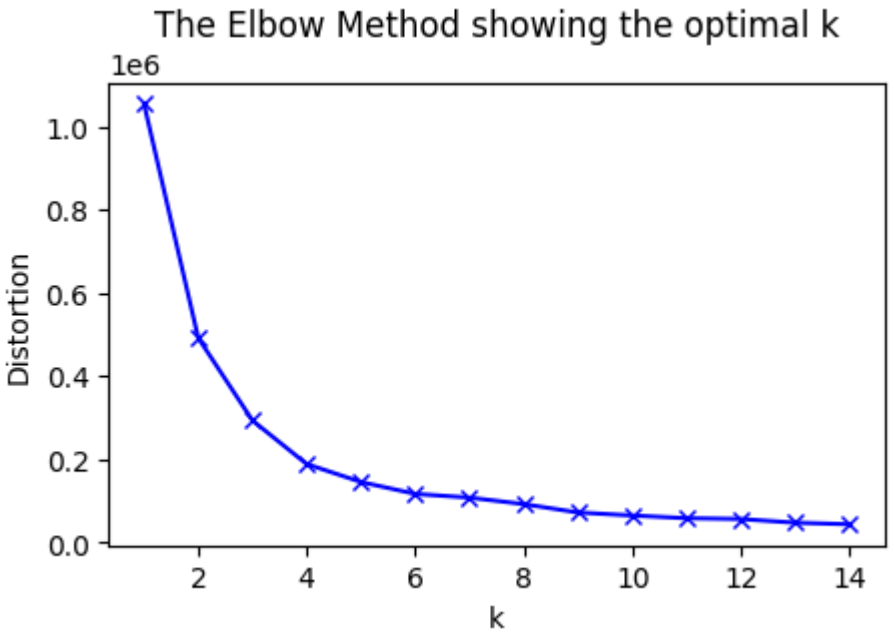
	id	address_raw	rooms	area	luxurious	price	price_per_m2	
0	1693998201-1	Neuhusstrasse 6, 8630 Rüti ZH, ZH	3.0	49	0	1441	29.41	47.252
1	1693998233-172	Widacherstrasse 5, 8630 Rüti ZH, ZH	3.0	111	0	2600	23.42	47.252
2	1693998256-331	Widenweg 14, 8630 Rüti ZH, ZH	3.0	58	0	1490	25.69	47.253
3	1693998265-381	Rain 1, 8630 Rüti ZH, ZH	4.0	118	0	3240	27.46	47.259
4	1693998276-419	Bachtelstrasse 24b, 8630 Rüti ZH, ZH	3.0	66	0	1450	21.97	47.266

Subset of the apartment data frame for k-means clustering

In [14]:
```
# Define a subset of the data frame for k-means clustering
X3 = df[['rooms',
         'area',
         'price_per_m2']]
```

Elbow method showing the optimal k

In [15]:
```
# Sum of squared distances of samples to their closest cluster center
distortions = []

# Range of k's
K = range(1,15)

# Loop to find the optimal k
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X3)
    distortions.append(kmeanModel.inertia_)

# Elbow plot
plt.figure(figsize=(5,3))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')

plt.show()
```



Perform k-means clustering on the apartment data

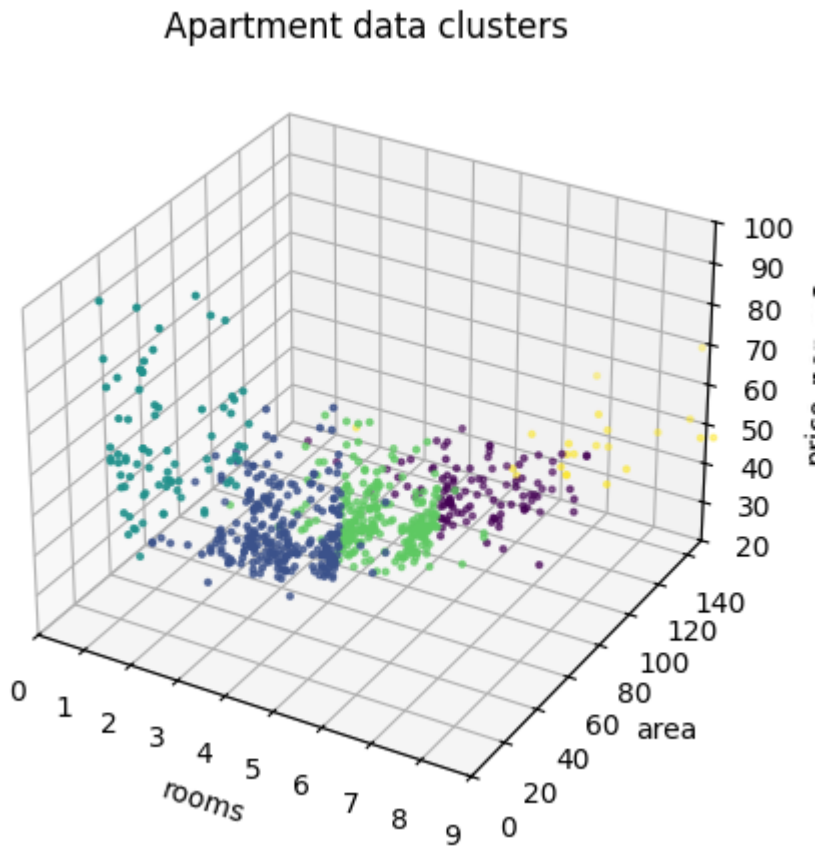
In [16]:
```
# Number of clusters
k = 5

# Perform k-means clustering
kmeans_apmts = KMeans(n_clusters=k, random_state=42).fit(X3)

# Add the clusters to data frame
X3['cluster'] = kmeans_apmts.predict(X3)

# Show number of apartments per cluster
X3['cluster'].value_counts().sort_values(ascending=False)
```

Out[16]:
```
cluster
1    261
3    252
0    108
2     79
4     22
Name: count, dtype: int64
```

Plot the apartment clusters

In [17]:
```
fig = plt.figure(figsize=(6,5))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['rooms'], df['area'], df['price_per_m2'], c=kmeans_apmts.labels_,

# Set title and axes labels
ax.set_title('Apartment data clusters', fontsize=12)
ax.set_xlabel('rooms', fontsize=10)
ax.set_ylabel('area', fontsize=10)
ax.set_zlabel('price_per_m2', fontsize=10)

# Set axes range
ax.set_xlim([0,9])
ax.set_ylim([0,150])
ax.set_zlim([20,100])
```

```
plt.show()
```



Calculate the Silhouette Score

In [18]:
```
print(f'Silhouette Score: {silhouette_score(X3, kmeans_apmts.labels_):.4f}')
```

```
Silhouette Score: 0.4547
```

Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [19]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:31:36
Python Version: 3.13.9
-----------------------------------
```

Cluster the Olivetti Faces Dataset with K-Means

(Notebook adapted from: A.Geron, Hands On ML with Scikit-Learn, Keras und Tensorflow, O'Reilly)

In this exercise we will try to cluster human pictures, in order to identify which picture belong to the same subject/person. We will use an unsupervised learning approach, which means we will train without knowing the correct assignment of the pictures (in other words the person have no labels). As usual, answer the questions/comments and add your code where indicated. Otherwise just try to understand what's happening :-).

About the database: The classic Olivetti faces dataset contains 400 grayscale 64 × 64–pixel images of faces. Each image is flattened to a 1D vector of size 4,096. 40 different people were photographed (10 times each). Load the dataset using the `sklearn.datasets.fetch_olivetti_faces()` function. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

Libraries and settings

In [1]:
```
# Libraries
import os
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.datasets import fetch_olivetti_faces

# Settings
import warnings
warnings.filterwarnings("ignore")

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\K-Means
```

Import data

In [2]:
```
# Import data
olivetti = fetch_olivetti_faces()

# Show data description
```

```
# print(olivetti.DESCR)

# Data matrix
X = olivetti.data

# Data shape (nrows, ncols)
X.shape
```

Out[2]: (400, 4096)

Reduce dimensionality

Since the images are 64x64pixels, the number of features (=number of total pixels) is 4096. This a huge number of variables to check out for clustering. So, for convenience we should first try to see if we can reduce the number of features we feed to k-means. This operation is called "Dimensionality Reduction".

In [3]:
```
# Use PCA to reduce the dimensionality of the data
pca = PCA(0.99)

# Fit the PCA model to the data
X_pca = pca.fit_transform(olivetti.data)

# Check the number of components
pca.n_components_
```

Out[3]: np.int64(260)

Perform k-means clustering

Well done :) we reduced dimensionality quite a lot!

Let's now perform k-means on this subset of features: X_pca

Use the function https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html and the advices given in the class.

Remember that you have to define the number of clusters k.

How to set k? Start with a "reasonable" guess and qualitatively look how good the clustering is.

Try to change k manually and see if it improves or get worse.

In [4]:
```
# Perform k-means clustering
kmeans = KMeans(n_clusters=5, random_state=42) ### ADD YOUR CODE HERE ###
kmeans.fit(X_pca)

def plot_faces(faces, n_cols=5):
    n_rows = (len(faces) - 1) // n_cols + 1
    plt.figure(figsize=(n_cols, n_rows * 1.1))
    for index, face in enumerate(faces):
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(face.reshape(64, 64), cmap="gray")
```

```
        plt.axis("off")
    plt.show()

for cluster_id in np.unique(kmeans.labels_):
    print("Cluster", cluster_id)
    in_cluster = kmeans.labels_==cluster_id
    faces = X[in_cluster].reshape(-1, 64, 64)
    plot_faces(faces)
```

```
Cluster 0
```

Cluster 1

Cluster 2

Cluster 3

Cluster 4

What was your best k?

Use a loop to identify the best k

For a serious approach we should try to identify K through a more robust approach. Let's run the k-means algorithm in a loop, every time with a different number of clusters, K.

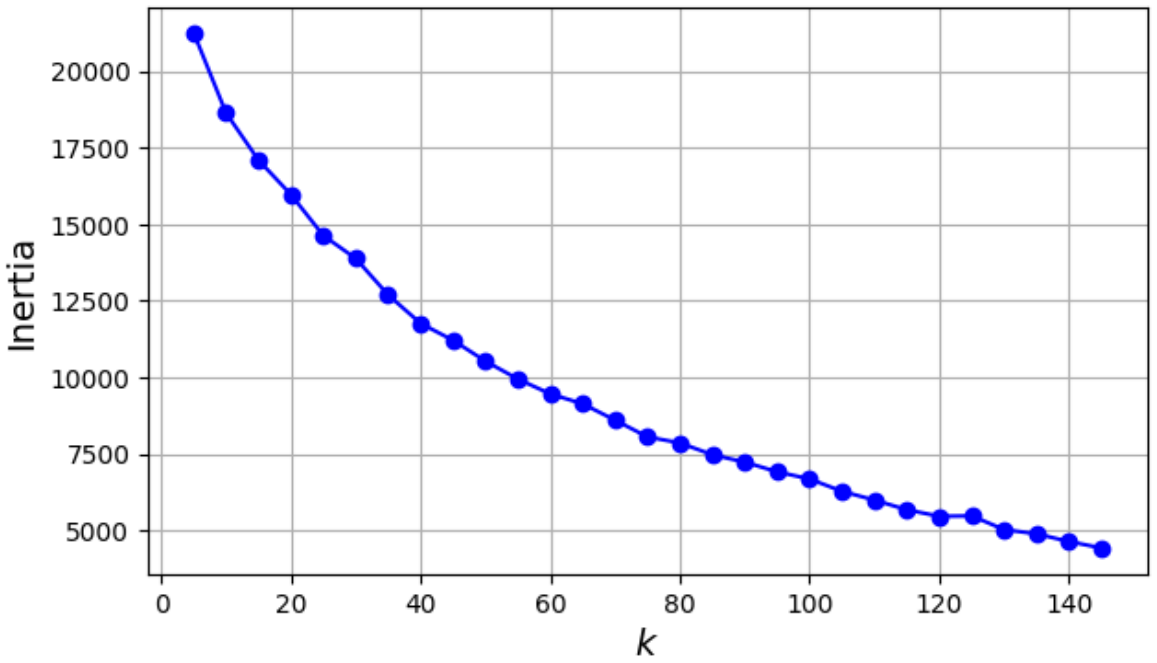
In [5]:
```
 # Feel free to change the range limits or step
k_range = range(5, 150, 5)
kmeans_per_k = []
for k in k_range:
    print("k={}".format(k))
    kmeans = KMeans(n_clusters=k)  ### ADD YOUR CODE HERE ###
    kmeans.fit(X_pca)
    kmeans_per_k.append(kmeans)
```

```
k=5
k=10
k=15
k=20
k=25
k=30
k=35
k=40
k=45
k=50
k=55
k=60
k=65
k=70
k=75
k=80
k=85
k=90
k=95
k=100
k=105
k=110
k=115
k=120
k=125
k=130
k=135
k=140
k=145
```

Calculate Intertia (Within-Cluster Sum of Squares, WCSS) for each run

Let's now evaluate for each of this run the inertia of the model. Luckily, this is part of the Sklearn KMeans function's output (Z.B. kmeans.inertia_)

In [6]:
```
# Calculate inertia for each k
inertias = [model.inertia_ for model in kmeans_per_k]

plt.figure(figsize=(7, 4))
plt.plot(k_range, inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.grid()
plt.show()
```



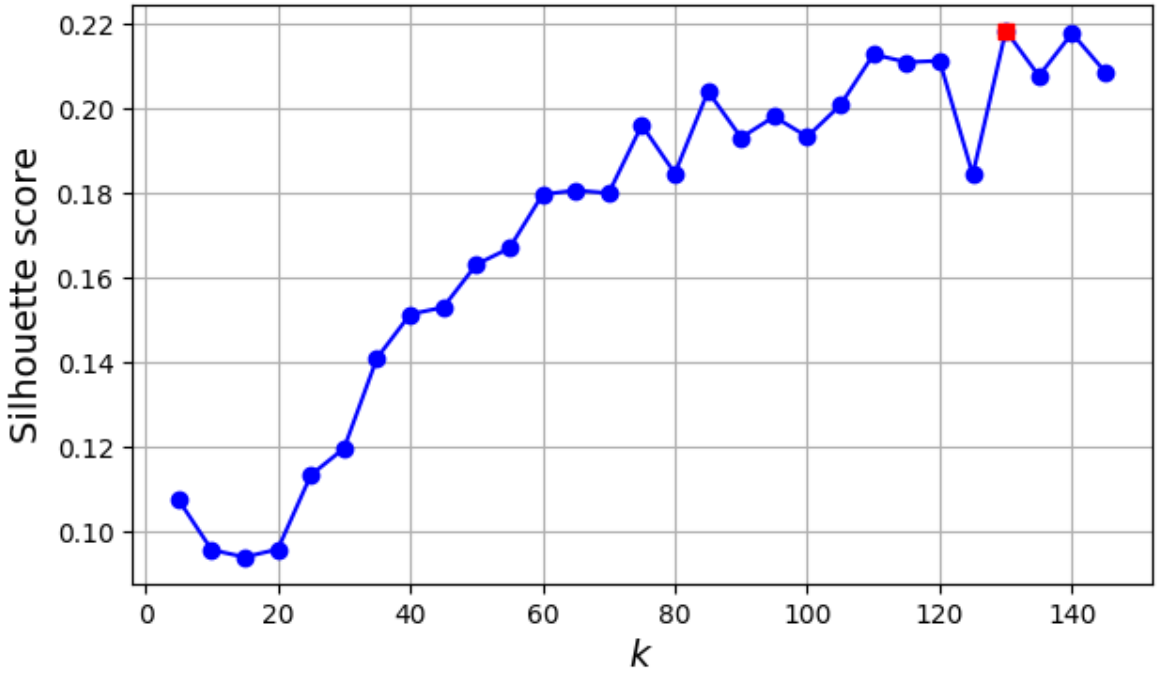
Can you get from this plot a clear indication on what could be a good value for k?

Do you see an "elbow"?

Calculate silhouette score

Let's try with the more sophisticated method called silhouette analysis.

In [7]:
```
# Calculate silhouette score for each k
silhouette_scores = [silhouette_score(X_pca, model.labels_) for model in kmeans_
best_index = np.argmax(silhouette_scores)
best_k = k_range[best_index]
best_score = silhouette_scores[best_index]

plt.figure(figsize=(7, 4))
plt.plot(k_range, silhouette_scores, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
plt.plot(best_k, best_score, "rs")
plt.grid()
plt.show()
```



What is the best value for k according to this method?

In [8]:
```
print(f"The best k is: {best_k}")
```

```
The best k is: 130
```

It looks like the best number of clusters is quite high. How does it compare with your first reasonable guess?

Final result

Let's have a direct look of how pictures are grouped according to the best k

In [9]:
```
# Fit the best k-means model
kmeans = KMeans(n_clusters = 140) ### ADD VALUE OF BEST K HERE ###
kmeans.fit(X_pca)
```

Out[9]:

KMeans

▶ Parameters

n_clusters	140
init	'k-means++'
n_init	'auto'
max_iter	300
tol	0.0001
verbose	0
random_state	None
copy_x	True
algorithm	'lloyd'

In [10]:

```
# Plot the faces in each cluster
def plot_faces(faces, n_cols=5):
    n_rows = (len(faces) - 1) // n_cols + 1
    plt.figure(figsize=(n_cols, n_rows * 1.1))
    for index, face in enumerate(faces):
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(face.reshape(64, 64), cmap="gray")
        plt.axis("off")
    plt.show()

for cluster_id in np.unique(kmeans.labels_):
    print("Cluster", cluster_id)
    in_cluster = kmeans.labels_==cluster_id
    faces = X[in_cluster].reshape(-1, 64, 64)
    plot_faces(faces)
```

Cluster 0

Cluster 1

Cluster 2

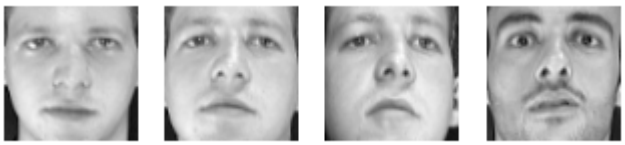
Cluster 3

Cluster 4

Cluster 5

Cluster 6

Cluster 7



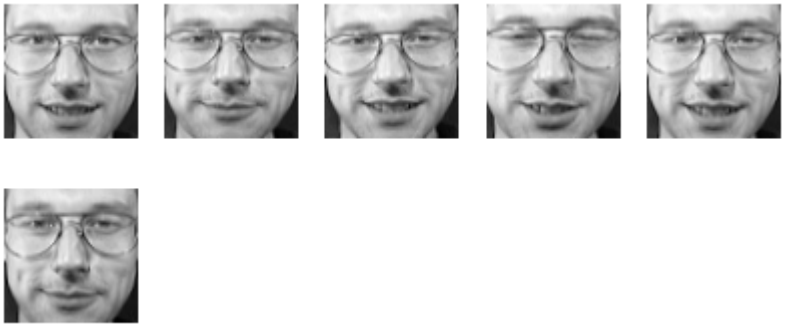
Cluster 8

Cluster 9

Cluster 10

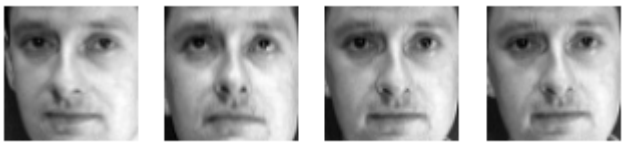
Cluster 11

Cluster 12

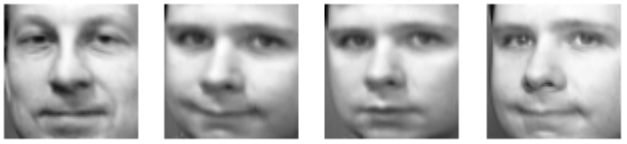


Cluster 13

Cluster 14



Cluster 15



Cluster 16

Cluster 17

Cluster 18

Cluster 19

Cluster 20

Cluster 21

Cluster 22

Cluster 23

Cluster 24

Cluster 25

Cluster 26

Cluster 27

Cluster 28

Cluster 29

Cluster 30

Cluster 31

Cluster 32

Cluster 33

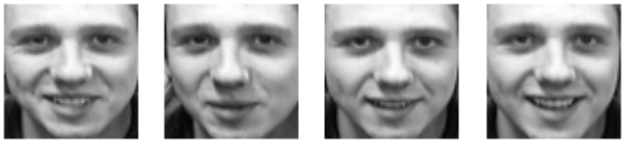
Cluster 34

Cluster 35

Cluster 36

Cluster 37

Cluster 38



Cluster 39

Cluster 40

Cluster 41

Cluster 42

Cluster 43

Cluster 44

Cluster 45

Cluster 46

Cluster 47

Cluster 48

Cluster 49

Cluster 50

Cluster 51

Cluster 52

Cluster 53

Cluster 54

Cluster 55

Cluster 56

Cluster 57

Cluster 58

Cluster 59

Cluster 60

Cluster 61

Cluster 62

Cluster 63

Cluster 64

Cluster 65

Cluster 66

Cluster 67

Cluster 68

Cluster 69

Cluster 70

Cluster 71

Cluster 72

Cluster 73

Cluster 74

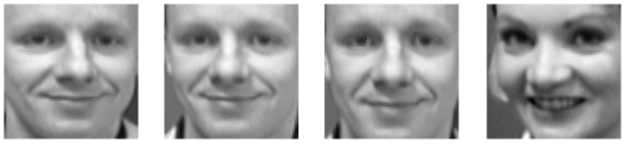
Cluster 75

Cluster 76

Cluster 77

Cluster 78

Cluster 79



Cluster 80

Cluster 81

Cluster 82

Cluster 83

Cluster 84

Cluster 85

Cluster 86

Cluster 87


Cluster 88

Cluster 89



Cluster 90



Cluster 91



Cluster 92


Cluster 93




Cluster 94

Cluster 95

Cluster 96

Cluster 97

Cluster 98

Cluster 99

Cluster 100

Cluster 101

Cluster 102

Cluster 103

Cluster 104

Cluster 105

Cluster 106

Cluster 107

Cluster 108

Cluster 109

Cluster 110

Cluster 111

Cluster 112

Cluster 113

Cluster 114

Cluster 115

Cluster 116

Cluster 117

Cluster 118

Cluster 119

Cluster 120

Cluster 121

Cluster 122

Cluster 123

Cluster 124


Cluster 125

Cluster 126



Cluster 127

Cluster 128



Cluster 129

Cluster 130



Cluster 131

Cluster 132

Cluster 133

Cluster 134

Cluster 135

Cluster 136

Cluster 137

Cluster 138

Cluster 139

Well done :-)

Jupyter notebook --footer info-- (please always provide this

at the end of each notebook)

In [11]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:32:08
Python Version: 3.13.9
-----------------------------------
```

K-Nearest Neighbors: Classifying Supermarkets by Brand

In this notebook, we will use **K-Nearest Neighbors (KNN)** to classify supermarkets based on their brand. The goal is to predict the supermarket brand (e.g., Spar, ALDI, Lidl, Migros) using features like population density, percentage of foreigners, employment rate, and latitude and longitude.

KNN is a simple supervised learning algorithm used for classification, where an object is classified by the majority label of its neighbors.

Libraries and Settings

In [1]:
```
# Libraries
import os
import re
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\KNN
```

Import Data

In [2]:
```
# Load the supermarket dataset
df = pd.read_csv('./Data/supermarkets_data_enriched.csv',
                 sep=',',
                 encoding='utf-8',
                 index_col=0)

# Show dimension (rows, columns)
print(df.shape)

# Show the first 5 rows
df.head()
```

```
(3242, 17)
```

Out[2]:

	type	id	lat	lon	brand	shop	addr:city	addr:housenu
0	node	33126515	47.155616	9.037915	Spar	supermarket	NaN	
1	node	280130028	47.155492	9.039666	ALDI	supermarket	Schänis	
2	node	6122906632	47.158959	9.044477	Denner	supermarket	Schänis	
3	node	9019298862	47.155185	9.038472	Lidl	supermarket	NaN	
4	node	36726161	47.226191	8.980329	Migros	supermarket	Uznach	

Data Preprocessing

Remove rows with missing or duplicated values

In [3]:
```
# Remove rows with missing values
df = df.dropna()

# Remove rows with duplicated values
df = df.drop_duplicates()
```

Change brand names to upper

In [4]:
```
# Change brand names to uppercase
df['brand'] = df['brand'].str.upper()
```

Create a subset of the data with defined brands

In [5]:
```
# Select subset of supermarket data (reset the index in the subset)
df_sub = df[df['brand'].isin(['SPAR', 'LANDI', 'MIGROS', 'ALDI'])].reset_index(d
df_sub.head()
```

Out[5]:

	type	id	lat	lon	brand	shop	addr:city	addr:housenu
0	node	280130028	47.155492	9.039666	ALDI	supermarket	Schänis	
1	node	36726161	47.226191	8.980329	MIGROS	supermarket	Uznach	
2	node	48932835	47.375020	8.522895	MIGROS	supermarket	Zürich	
3	node	75749133	47.340967	8.530601	ALDI	supermarket	Zürich	
4	node	83330862	47.344749	8.529981	MIGROS	supermarket	Zürich	

Select relevant features for classification

In [6]:
```
# Select relevant features for classification
features = ['pop_dens', 'frg_pct', 'emp', 'lat', 'lon']
X = df_sub[features]
y = df_sub['brand']
```

Split the data into training and testing sets

In [7]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Standardize the features (removes the mean and scales to unit variance)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model Training: K-Nearest Neighbors

In [8]:
```
# Initialize the KNN model with 5 neighbors
knn = KNeighborsClassifier(n_neighbors=5)

# Train the model
knn.fit(X_train, y_train)
```

Out[8]:

▾ KNeighborsClassifier

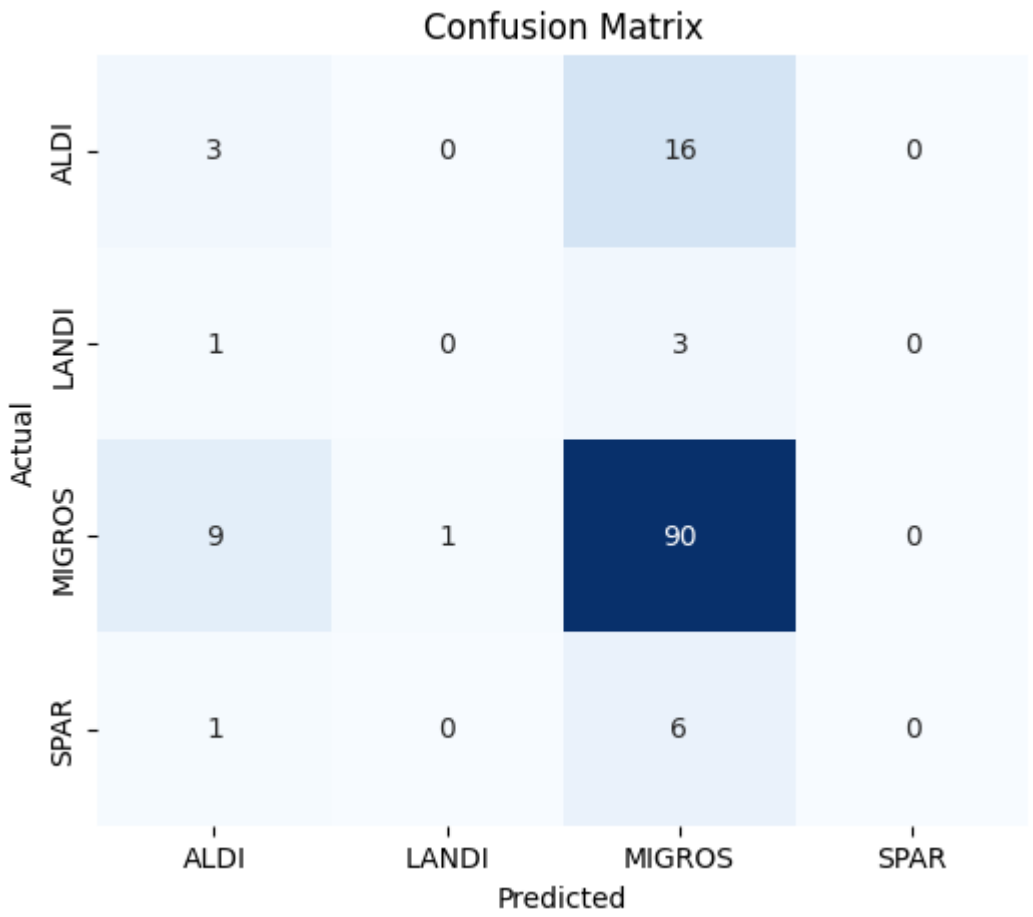
▸ Parameters

	n_neighbors	5
	weights	'uniform'
	algorithm	'auto'
	leaf_size	30
	p	2
	metric	'minkowski'
	metric_params	None
	n_jobs	None

Model Evaluation

In [9]:

```
# Predict on the test set
y_pred = knn.predict(X_test)

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='Blues',
            xticklabels=knn.classes_,
            yticklabels=knn.classes_,
            cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print('\nClassification report:\n', classification_report(y_test, y_pred))
```




```
Classification report:
              precision    recall  f1-score   support

        ALDI       0.21      0.16      0.18        19
       LANDI       0.00      0.00      0.00         4
      MIGROS       0.78      0.90      0.84       100
        SPAR       0.00      0.00      0.00         7

    accuracy                           0.72       130
   macro avg       0.25      0.26      0.25       130
weighted avg       0.63      0.72      0.67       130
```

Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [10]:

```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:32:58
Python Version: 3.13.9
-----------------------------------
```

Logistic Regression: Classifying Apartments as Expensive or Inexpensive

In this notebook, we will use **Logistic Regression** to classify apartments based on their price per square meter. The goal is to predict whether an apartment is "expensive" or "inexpensive" using available features such as the number of rooms, area, luxurious status, and more.

Logistic Regression is a supervised machine learning algorithm used for classification problems.

Libraries and settings

In [1]:
```
# Libraries
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_curve,
from sklearn.preprocessing import StandardScaler

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Logistic_Regression
```

Import Data

In [2]:
```
# Load dataset
df = pd.read_csv('./Data/apartments_data_enriched_cleaned.csv',
                 delimiter=';',
                 index_col=0)

# Display dataset
df.head()
```

Out[2]:

field_1	web-scraper-order	address_raw	lat	lon	bfs_number	bfs_name	rc
0	1693998201-1	Neuhusstrasse 6, 8630 Rüti ZH, ZH	47.252171	8.845797	118	Rüti (ZH)	
1	1693998233-172	Widacherstrasse 5, 8630 Rüti ZH, ZH	47.252087	8.854919	118	Rüti (ZH)	
2	1693998256-331	Widenweg 14, 8630 Rüti ZH, ZH	47.253670	8.853993	118	Rüti (ZH)	
3	1693998265-381	Rain 1, 8630 Rüti ZH, ZH	47.259834	8.851705	118	Rüti (ZH)	
4	1693998276-419	Bachtelstrasse 24b, 8630 Rüti ZH, ZH	47.266113	8.866872	118	Rüti (ZH)	

Data Preprocessing

Before building the model, we need to preprocess the data. This includes:

- Handling missing values (if any).
- Creating a binary target variable for classification.

In [3]:
```
# Check for missing values
df.isna().sum()

# For simplicity, let's define an 'expensive' apartment as one with a price_per_
median_price_per_m2 = df['price_per_m2'].median()

# Create a new column 'expensive' that is 1 if the price_per_m2 is greater than
df['expensive'] = (df['price_per_m2'] > median_price_per_m2).astype(int)

# Display the updated dataframe
df[['price_per_m2', 'expensive']].head()
```

Out[3]:

field_1	price_per_m2	expensive
0	29.41	1
1	23.42	0
2	25.69	0
3	27.46	0
4	21.97	0

Splitting the Data

We'll split the dataset into a training set and a test set (80% training, 20% testing).

In [4]:
```
# Select features for the model
features = ['rooms',
            'area',
            'luxurious',
            'pop_dens',
            'mean_taxable_income',
            'dist_supermarket']
X = df[features]
y = df['expensive']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Building the Logistic Regression Model

Now we'll train a logistic regression model on the training set.

In [5]:
```
# Train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Display the model coefficients
coefficients = pd.DataFrame({
    'Feature': features,
    'Coefficient': model.coef_[0]
})
coefficients
```

Out[5]:

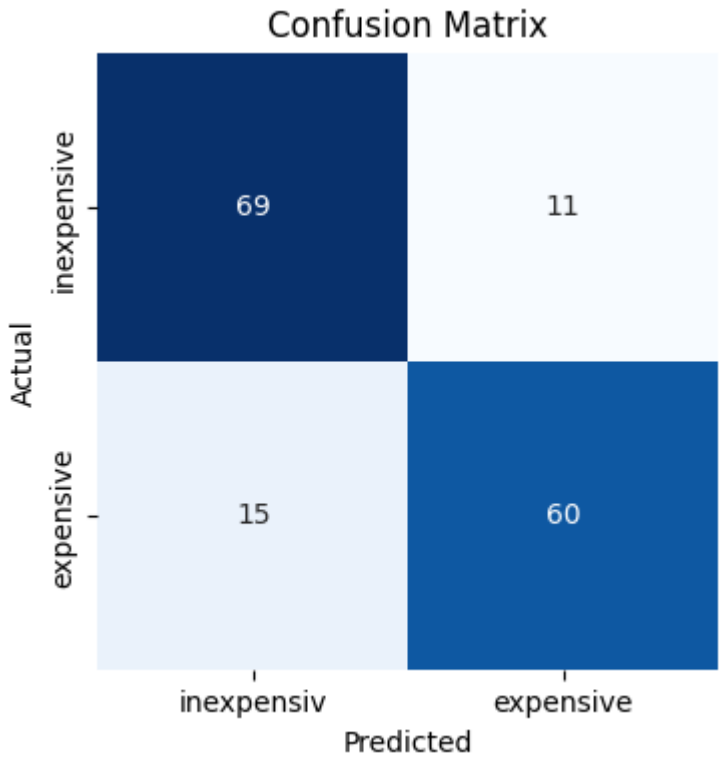
	Feature	Coefficient
0	rooms	-0.277112
1	area	-0.622674
2	luxurious	0.061063
3	pop_dens	1.316716
4	mean_taxable_income	0.558941
5	dist_supermarket	0.005065

Model Evaluation

Let's evaluate the model on the test set using various metrics.

Confusion Matrix and Classification Report

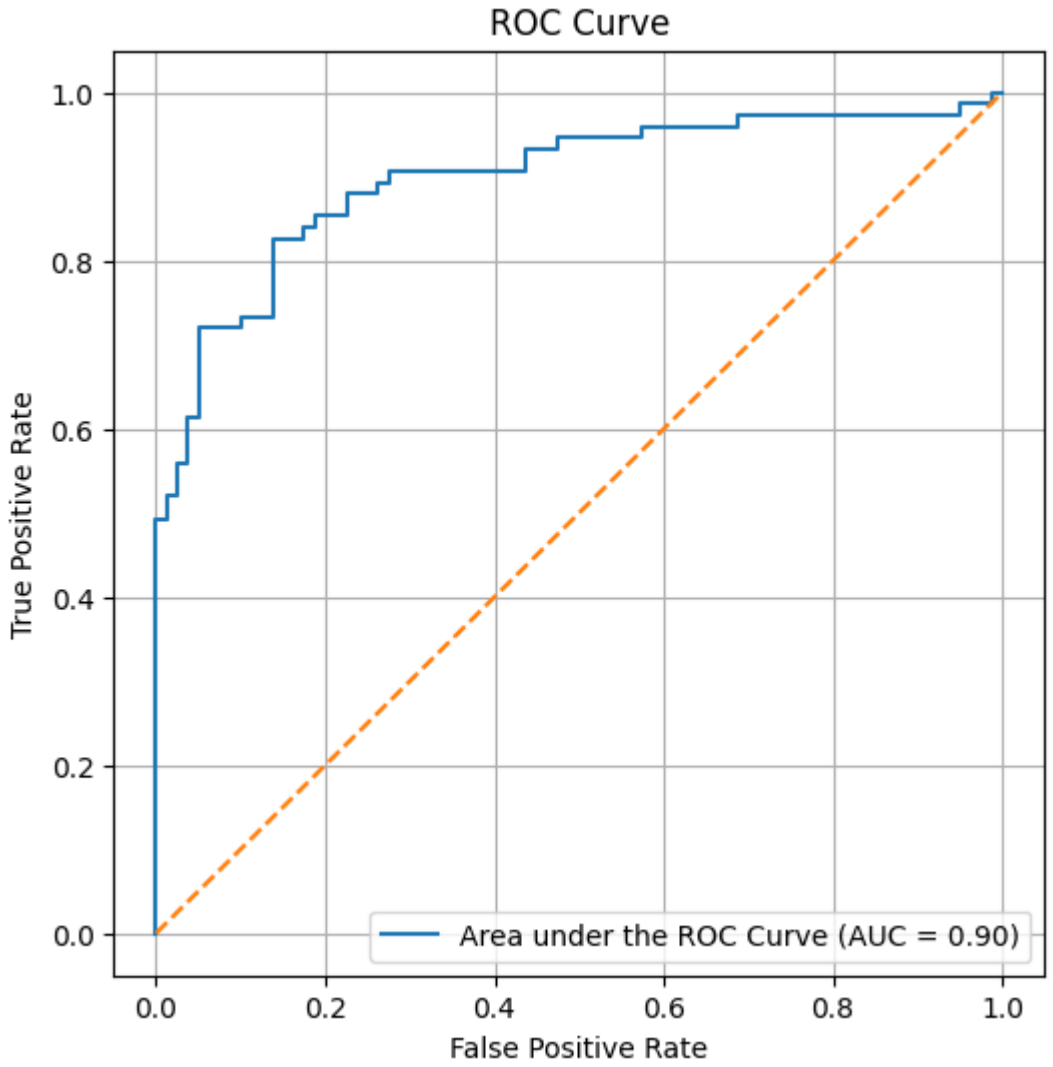
In [6]:
```
# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(4, 4))
sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='Blues',
            cbar=False,
            xticklabels=['inexpensiv', 'expensive'],
            yticklabels=['inexpensive', 'expensive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print('\nClassification report:\n', classification_report(y_test, y_pred))
```




```
Classification report:
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        80
           1       0.85      0.80      0.82        75

    accuracy                           0.83       155
   macro avg       0.83      0.83      0.83       155
weighted avg       0.83      0.83      0.83       155
```

ROC Curve and AUC

In [7]:
```
# ROC Curve
y_prob = model.predict_proba(X_test_scaled)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, label=f'Area under the ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Conclusion

We have successfully trained a logistic regression model to classify apartments as "expensive" or "inexpensive". The model was evaluated using metrics like accuracy, precision, recall, and the ROC curve.

Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [8]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:36:36
Python Version: 3.13.9
-----------------------------------
```

Logistic Regression (Iris Data)

This notebook demonstrates a simple logistic regression model using a dataset related to iris flowers. The goal is to predict whether a given iris species is of the species **Iris Virginica** based on its petal width. We'll walk through data preparation, model training, evaluation, and visualization.

Attribute	Value
Species	Iris Virginica
Petal Length	4.7 - 6.9 cm
Petal Width	1.4 - 2.5 cm
Sepal Length	6.0 - 7.9 cm
Sepal Width	2.2 - 3.8 cm
Color	Blue to Purple

Libraries and settings

In [1]:
```
# Libraries
import os
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Logistic_Regression
```

Data Preparation

In this section, we load the dataset and prepare it for the logistic regression model. We will use the famous Iris dataset, but only focus on one feature (petal width) to predict one specific class.

In [2]:
```
# Load the Iris dataset
iris = datasets.load_iris()

# Provide a description of the dataset
# print(iris.DESCR)
```

```
# We are only interested in the petal width
X = iris["data"][:, 3:]

 # 1 if Iris-Virginica, else 0
y = (iris["target"] == 2).astype(np.int32)

# Display a sample of the data
X[:5], y[:5]
```

Out[2]:
```
(array([[0.2],
        [0.2],
        [0.2],
        [0.2],
        [0.2]]),
 array([0, 0, 0, 0, 0], dtype=int32))
```

Model Training

We will use Scikit-Learn's `LogisticRegression` model to train on the data. The model will learn the relationship between petal width and the probability of the iris being of the species **Iris Virginica**.

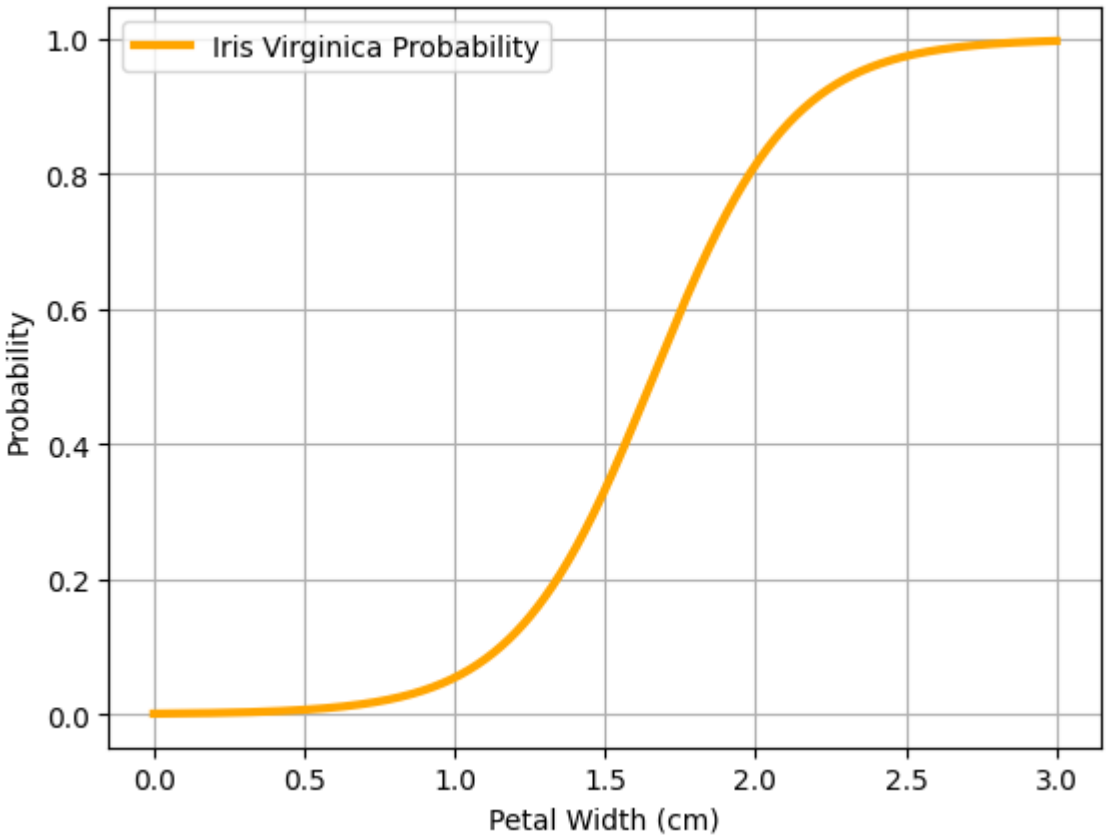
In [3]:
```
# Create and train the logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Display the model's learned coefficients
model.coef_, model.intercept_
```

Out[3]:
```
(array([[4.33251974]]), array([-7.19342106]))
```

Model Evaluation

Next, we evaluate the model by generating predictions for new data points. We also visualize the probability estimates to better understand the decision boundary.

In [4]:
```
# Generate new data points and predict probabilities
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = model.predict_proba(X_new)[:, 1]

# Plot the predictions
plt.plot(X_new,
         y_proba,
         color="orange",
         linewidth=3,
         label="Iris Virginica Probability")
plt.xlabel("Petal Width (cm)")
plt.ylabel("Probability")
plt.legend()
plt.grid()
plt.show()
```



Conclusion

In this notebook, we trained a logistic regression model to predict whether a flower is of the species **Iris Virginica** based on petal width. We visualized the probability predictions and demonstrated how logistic regression finds the decision boundary between classes. This method is useful for binary classification tasks like this one.

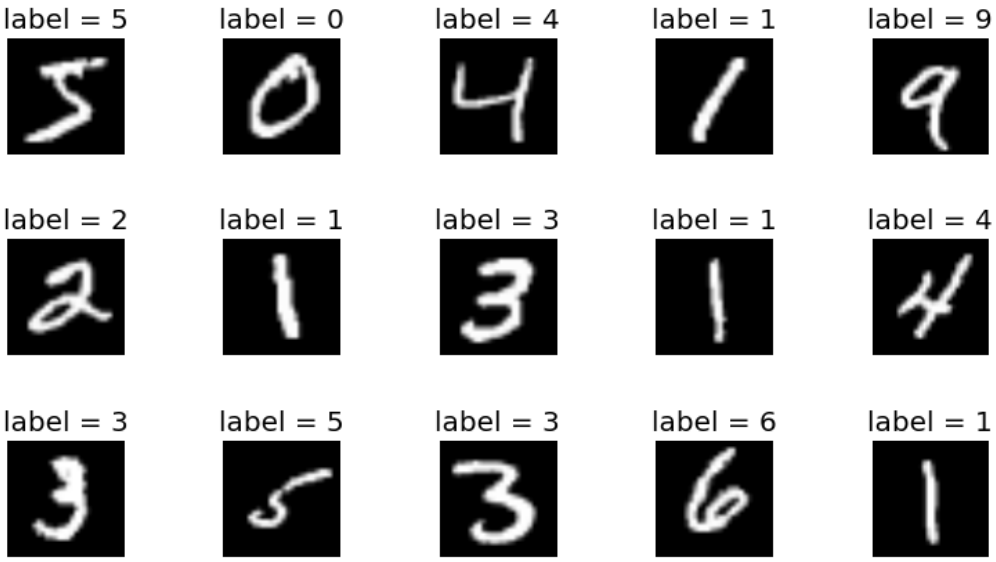
Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [5]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:37:32
Python Version: 3.13.9
-----------------------------------
```

MNIST handwritten digits prediction

The MNIST (Modified National Institute of Standards and Technology) digits dataset is a widely used dataset in the field of machine learning and computer vision. It is commonly used for training and testing image processing systems, particularly in the context of handwritten digit recognition.



Import Python libraries

In [1]:
```
# Libraries
import os
import logging
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set TensorFlow log level to suppress warnings
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# TensorFlow and Keras
from tensorflow import keras
from tensorflow.keras import layers

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Neural_Networks
```

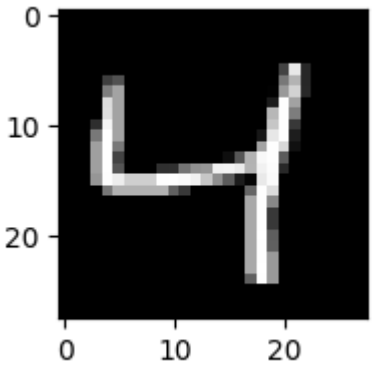
Prepare the data

In [2]:
```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# The data, split between train and test sets
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# Make sure images have shape (28, 28, 1)
X_train = np.expand_dims(X_train, -1)
X_test = np.expand_dims(X_test, -1)
print("X_train shape:", X_train.shape)
print(X_train.shape[0], "train samples")
print(X_test.shape[0], "test samples")

# Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
s/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
X_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Show single handwritten digits

In [3]:
```
# Show single digit image
image  = X_train[2] # Change index in [] to show other digits
fig    = plt.figure(figsize=(2,2))
plt.imshow(image, cmap='gray')
plt.show()
```



In [4]:
```
# Show shape of single digit image
X_train[2].shape
```

Out[4]: (28, 28, 1)

Initialize the model

In [5]:
```
# Linear stack of layers
```

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
# model.summary()
```

Train the model

In [6]:
```
# Define batch size and epochs
batch_size = 128
epochs = 5

# Compile the model
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])

# Train the model
model.fit(X_train,
          y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.1)
```

```
Epoch 1/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - accuracy: 0.8860 - loss: 0.3735 - val
_accuracy: 0.9782 - val_loss: 0.0819
Epoch 2/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy: 0.9660 - loss: 0.1117 - val
_accuracy: 0.9847 - val_loss: 0.0581
Epoch 3/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.9745 - loss: 0.0826 - val
_accuracy: 0.9883 - val_loss: 0.0460
Epoch 4/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.9789 - loss: 0.0700 - val
_accuracy: 0.9895 - val_loss: 0.0393
Epoch 5/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - accuracy: 0.9817 - loss: 0.0604 - val
_accuracy: 0.9917 - val_loss: 0.0371
```

Out[6]: <keras.src.callbacks.history.History at 0x2ad5351d7f0>

Evaluate the trained model

In [7]:
```
# Calculate the test loss and accuracy
score = model.evaluate(X_test, y_test, verbose=0)
print(f"Test loss: {score[0]:.4f}")
print(f"Test accuracy: {score[1]:.4f}")
```

```
Test loss: 0.0378
Test accuracy: 0.9870
```

Jupyter notebook --footer info-- (please always provide this at the end of each submitted notebook)

In [8]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:39:07
Python Version: 3.13.9
-----------------------------------
```

Multi-Layer Perceptron (MLP) Model (apartment data)

In this notebook, we will use a Multi-Layer Perceptron (MLP) Model to predict rental prices of apartments. A Multi-Layer Perceptron (MLP) model is a type of artificial neural network that consists of multiple layers of neurons. Each neuron in a layer is connected to every neuron in the previous and next layers, forming a fully connected network. The MLP model is capable of learning complex patterns in the data through backpropagation and gradient descent.

The key components of an MLP model include:

- **Input Layer**: The layer that receives the input features.
- **Hidden Layers**: One or more layers with weights and activation functions to learn intermediate representations.
- **Output Layer**: The layer that produces the final prediction.

In this notebook, we will:

1. Preprocess the apartment data, including scaling the features.
2. Define and compile the MLP model using TensorFlow/Keras.
3. Train the model on the training data.
4. Evaluate the model on the test data.
5. Visualize the training process and the model's performance.

Libraries and settings

In [1]:
```
# Libraries
import os
import logging
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Set TensorFlow log level to suppress warnings
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Neural_Networks
```

Import the apartment data

In [2]:

```
# Define columns for import
columns = [ 'web-scraper-order',
            'address_raw',
            'rooms',
            'area',
            'luxurious',
            'price',
            'price_per_m2',
            'lat',
            'lon',
            'bfs_number',
            'bfs_name',
            'pop',
            'pop_dens',
            'frg_pct',
            'emp',
            'mean_taxable_income',
            'dist_supermarket']

# Read and select variables
df_orig = pd.read_csv("./Data/apartments_data_enriched_cleaned.csv", sep=";", en

# Rename variable 'web-scraper-order' to 'apmt_id'
df_orig = df_orig.rename(columns={'web-scraper-order': 'id'})

# Remove missing values
df = df_orig.dropna()
df.head(5)

# Remove duplicates
df = df.drop_duplicates()

# Remove some 'extreme' values
df = df.loc[(df['price'] >= 1000) &
            (df['price'] <= 5000)]

# Reset index
df = df.reset_index(drop=True)

print(df.shape)
df.head(5)
```

```
(722, 17)
```

Out[2]:

	id	address_raw	rooms	area	luxurious	price	price_per_m2	
0	1693998201-1	Neuhusstrasse 6, 8630 Rüti ZH, ZH	3.0	49	0	1441	29.41	47.252
1	1693998233-172	Widacherstrasse 5, 8630 Rüti ZH, ZH	3.0	111	0	2600	23.42	47.252
2	1693998256-331	Widenweg 14, 8630 Rüti ZH, ZH	3.0	58	0	1490	25.69	47.253
3	1693998265-381	Rain 1, 8630 Rüti ZH, ZH	4.0	118	0	3240	27.46	47.259
4	1693998276-419	Bachtelstrasse 24b, 8630 Rüti ZH, ZH	3.0	66	0	1450	21.97	47.266

Rescale features using a Min-Max Scaler

In [3]:
```
# List of features to re-scale
features_to_scale = ['area',
                     'rooms',
                     'lat',
                     'lon',
                     'pop',
                     'pop_dens',
                     'frg_pct',
                     'emp',
                     'mean_taxable_income',
                     'dist_supermarket']

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the features
df[features_to_scale] = scaler.fit_transform(df[features_to_scale])
```

Create train and test samples (train = 80%, test = 20% of the data)

In [4]:
```
# Create train and test samples
X_train, X_test, y_train, y_test = train_test_split(df[features_to_scale],
                                                    df['price'],
                                                    test_size=0.20,
                                                    random_state=42)

# Show X_train
print('X_train:')
print(X_train.head(), '\n')

# Show y_train
print('y_train:')
```

```
print(y_train.head())
```

```
X_train:
         area     rooms       lat       lon       pop  pop_dens   frg_pct  \
456  0.411765  0.846154  0.361445  0.887293  0.009952  0.016427  0.206503
6    0.196078  0.384615  0.637867  0.195518  0.005904  0.093306  0.149322
362  0.294118  0.384615  0.749858  0.192933  0.010772  0.071666  0.439201
594  0.188235  0.230769  0.500230  0.406181  0.046320  0.203936  0.669363
439  0.298039  0.384615  0.145885  1.000000  0.022402  0.066417  0.491679

          emp  mean_taxable_income  dist_supermarket
456  0.003682             0.030066          0.591051
6    0.001043             0.439563          0.082871
362  0.001985             0.090126          0.787478
594  0.075905             0.079723          0.168520
439  0.006783             0.000000          0.193446

y_train:
456    1900
6      1850
362    2090
594    2102
439    1800
Name: price, dtype: int64
```

Train the Multi-Layer Perceptron (MLP) Model

In [5]:

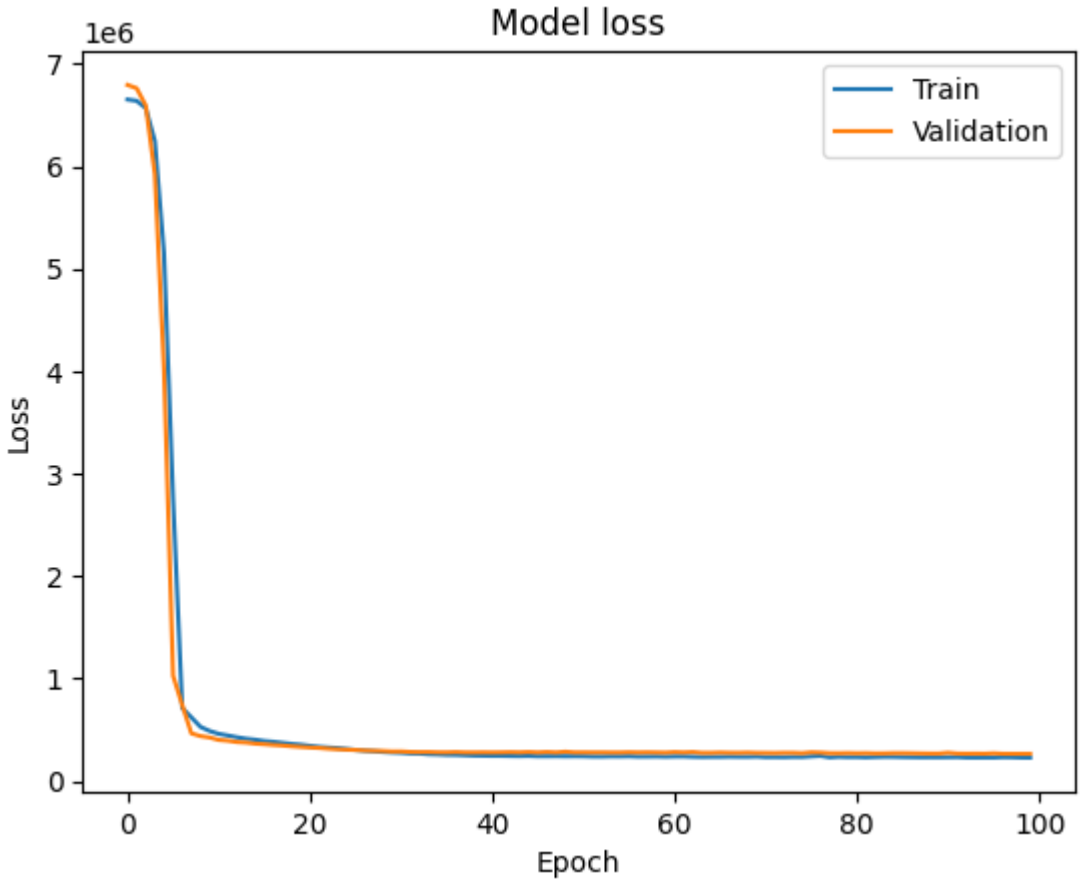
```
# Define the number of features
num_features = X_train.shape[1]

# Define the model
model = keras.Sequential([
    keras.Input(shape=(num_features,)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1),
])

# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mape'])

# Train the model
history = model.fit(X_train,
                    y_train,
                    epochs=100,
                    validation_split=0.20,
                    batch_size=32,
                    verbose=0)

# Predict the response for test dataset
y_pred = model.predict(X_test)

# Evaluate the model on the test set using the mean absolute error (MAPE)
test_loss, test_mape = model.evaluate(X_test, y_test)
print(f"\nMAPE: {test_mape:.2f}")

# Calculate R2 score
```

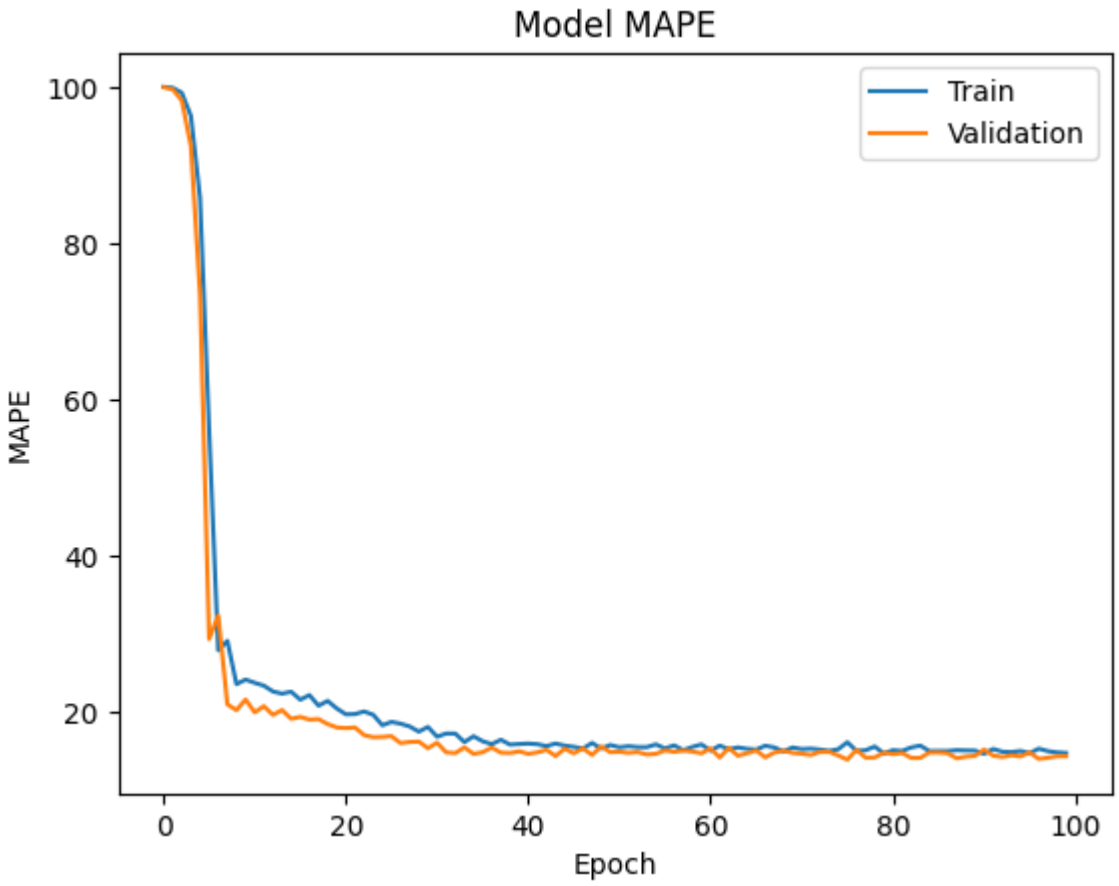
```
r2 = r2_score(y_test, y_pred)
print(f"R2 score: {r2:.4f}")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 302156.1250 - mape: 16.4172

MAPE: 16.42
R2 score: 0.5788
```

Plot training & validation loss values

In [6]:
```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

# Plot training & validation MAE values
plt.plot(history.history['mape'])
plt.plot(history.history['val_mape'])
plt.title('Model MAPE')
plt.ylabel('MAPE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```





Jupyter notebook --footer info-- (please always provide this at the end of each notebook)

In [7]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:39:44
Python Version: 3.13.9
-----------------------------------
```

Creating a Neural Network to classify diabetic patients

Libraries and settings

In [1]:
```
# Libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from prettytable import PrettyTable

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Show current working directory
print(os.getcwd())
```

```
c:\Users\Phil\Documents\GitHub\python_machine_learning_basics\Neural_Networks
```

The Problem

Suppose we have some information about obesity, smoking habits, and exercise habits of five people. We also know whether these people are diabetic or not. The dataset looks like this:

Person	Smoking	Obesity	Exercise	Diabetic
Person 1	0	1	0	1
Person 2	0	0	1	0
Person 3	1	0	0	0

Person	Smoking	Obesity	Exercise	Diabetic
Person 4	1	1	0	1
Person 5	1	1	1	1

In this table, there are five columns: Person, Smoking, Obesity, Exercise, and Diabetic. Here 1 refers to true and 0 refers to false. For instance, the first person has values of 0, 1, 0 which means that the person doesn't smoke, is obese, and doesn't exercise. The person is also diabetic. It is clearly evident from the dataset that a person's obesity is indicative of him being diabetic. Our task is to create a neural network that is able to predict whether an unknown person is diabetic or not given data about his exercise habits, obesity, and smoking habits.

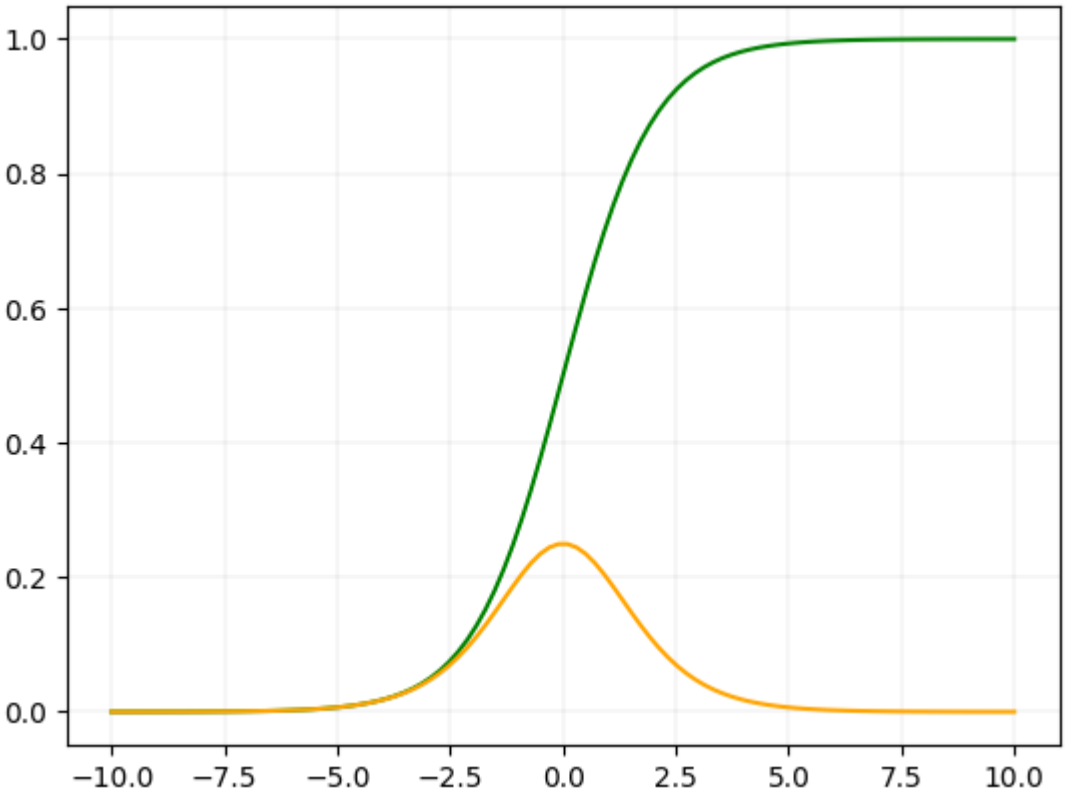
In [2]:
```
# Function to generate pretty tables
def generate_ascii_table(df):
    x = PrettyTable()
    x.field_names = df.columns.tolist()
    for row in df.values:
        x.add_row(row)
    print(x)
    return x

# Activation function (sigmoid)
def sigmoid(x):
    return 1/(1+np.exp(-x))

# Derivative of the activation function
# -> When updating the curve, to know in which
#    direction and how much to change or update
#    the curve depending upon the slope.
def sigmoid_der(x):
    return sigmoid(x)*(1-sigmoid(x))
```

In [3]:
```
# Create data frame
data = np.array([[0,1,0,1],[0,0,1,0],[1,0,0,0],[1,1,0,1],[1,1,1,1]])
df = pd.DataFrame(data, columns=['Smoking', 'Obesity', 'Exercise', 'Diabetic'])

# Show table
generate_ascii_table(df)

# Matrix with features (X) and labels (y)
X = data[0:5,0:3]
y = data[0:5,3]
y = y.reshape(5,1)
```

```
+---------+---------+----------+----------+
| Smoking | Obesity | Exercise | Diabetic |
+---------+---------+----------+----------+
|    0    |    1    |    0     |    1     |
|    0    |    0    |    1     |    0     |
|    1    |    0    |    0     |    0     |
|    1    |    1    |    0     |    1     |
|    1    |    1    |    1     |    1     |
+---------+---------+----------+----------+
```

In [4]:
```
# Plots the sigmoid function and its derivative
```

```
input = np.linspace(-10, 10, 100)
plt.plot(input, sigmoid(input), color="green")
plt.plot(input, sigmoid(input)*(1-sigmoid(input)), color="orange")
plt.grid(color='gray', linestyle='-', linewidth=0.1)
```



In [5]:
```
# Initialize weights and bias & define learning rate und the number of epochs
np.random.seed(42)
weights        = np.random.rand(3,1)
bias           = np.random.rand(1)
learning_rate  = 0.25
num_epochs     = 200
```

In [6]:
```
# Calculations step by step

# Inputs
inputs = X

# Print X, y, weights and bias
print('X:', '\n', inputs, '\n')
print('y:', '\n', y, '\n')
print('weights:', '\n', weights,'\n')
print('bias:', '\n', bias,'\n')

# Calculate X * weigths + bias
XW = np.dot(X, weights) + bias
print('XW:', '\n', XW, '\n')

# Apply activation function to XW
z = sigmoid(XW)
print('sigmoid(XW) = z', '\n', z, '\n')

# Calculate the error (difference between labels and z)
error = z - y
print('error:', '\n', error, '\n')
```

```
# Backpropagation (weights and bias adaptation)
dcost_dpred = error
dpred_dz    = sigmoid_der(z)
print('sigmoid_der(z):', '\n', dpred_dz, '\n')

# Calculate
z_delta = dcost_dpred * dpred_dz
print('z_delta:', '\n', z_delta, '\n')

# Adapt weights
inputs = X.T
weights -= learning_rate * np.dot(inputs, z_delta)
print('adapted weights', '\n', weights, '\n')

print('adapted bias:')
for num in z_delta:
    bias -= learning_rate * num
    print(bias)
```

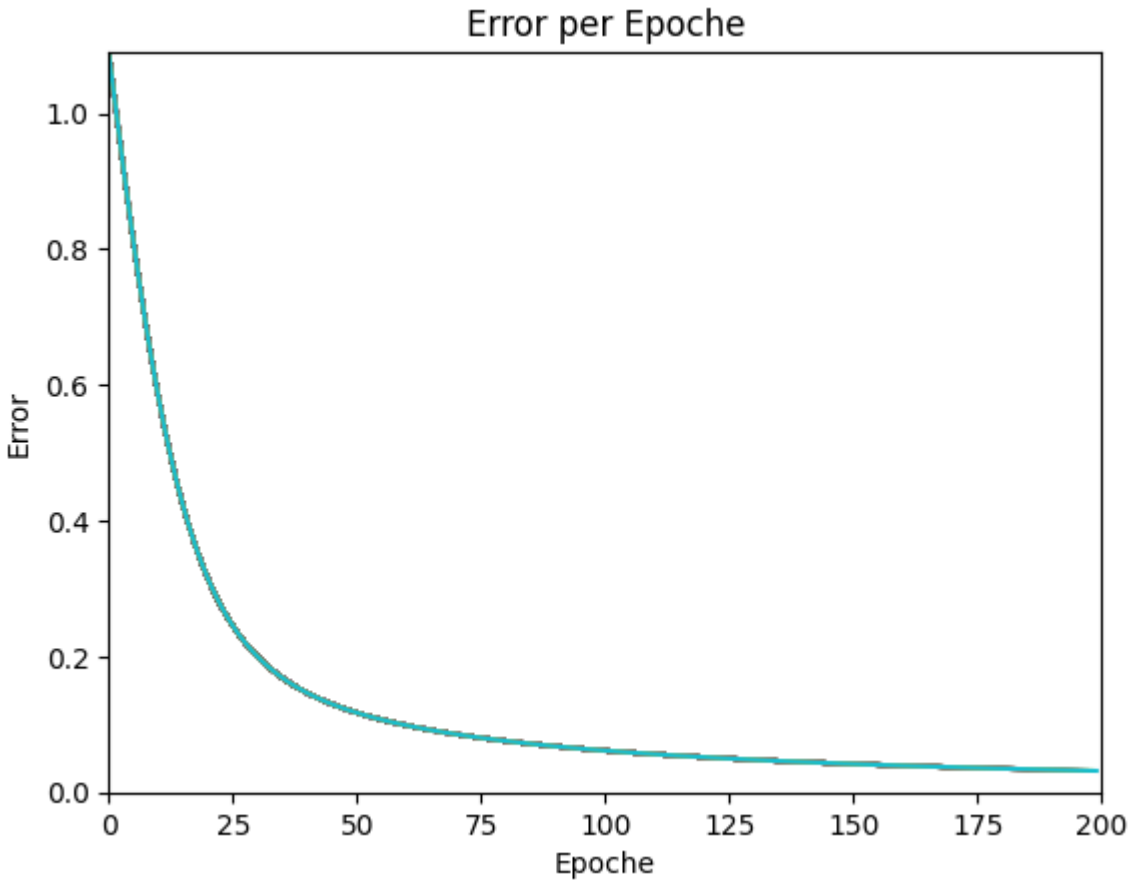
```
X:
 [[0 1 0]
 [0 0 1]
 [1 0 0]
 [1 1 0]
 [1 1 1]]

y:
 [[1]
 [0]
 [0]
 [1]
 [1]]

weights:
 [[0.37454012]
 [0.95071431]
 [0.73199394]]

bias:
 [0.59865848]

XW:
 [[1.54937279]
 [1.33065243]
 [0.9731986 ]
 [1.92391291]
 [2.65590685]]

sigmoid(XW) = z
 [[0.82482312]
 [0.79094853]
 [0.72575659]
 [0.87257414]
 [0.93437412]]

error:
 [[-0.17517688]
 [ 0.79094853]
 [ 0.72575659]
 [-0.12742586]
 [-0.06562588]]

sigmoid_der(z):
 [[0.21187385]
 [0.21464285]
 [0.21976804]
 [0.20785969]
 [0.20249263]]

z_delta:
 [[-0.0371154 ]
 [ 0.16977145]
 [ 0.1594981 ]
 [-0.0264867 ]
 [-0.01328876]]

adapted weights
 [[0.34460946]
 [0.96993702]
```

```
 [0.69287327]]

adapted bias:
[0.60793733]
[0.56549447]
[0.52561995]
[0.53224162]
[0.53556381]
```

In [7]:

```python
# Neural network

# Prepare figure
fig = plt.figure()
ax  = fig.add_subplot(111)
ax.set_xlabel('Epoche')
ax.set_ylabel('Error')
plt.title('Error per Epoche')
plt.ion()

fig.show()
fig.canvas.draw()

#--------------------------------------
# Loop with backpropagation
#--------------------------------------

# Initialize list
d = []

for epoch in range(num_epochs):

    inputs = X

    # Calculate (X * weigths) + bias
    XW = np.dot(X, weights) + bias

    # Apply activation function to XW
    z = sigmoid(XW)

    # Calculate the error (difference between label-value and z)
    error = z - y

    # Store error in data frame
    d.append(
        {
            'Epoch': epoch,
            'Error': error.sum()
        }
    )
    df = pd.DataFrame(d)

    # Backpropagation (change weights and bias)
    dcost_dpred = error
    dpred_dz    = sigmoid_der(z)
    z_delta     = dcost_dpred * dpred_dz

    inputs      = X.T
    weights    -= learning_rate * np.dot(inputs, z_delta)

    for num in z_delta:
```

```
        bias -= learning_rate * num

    #-----------------------
    # Plot error per epoche
    #-----------------------
    plt.xlim([0,num_epochs])
    plt.ylim([0, df.Error[0]])
    ax.plot(df.Epoch, df.Error)
    fig.canvas.draw()
```



In [8]:
```
# Model-prediction based on new data
data_new = np.array([[0,1,0]])
df_new   = pd.DataFrame(data_new, columns=['Smoking', 'Obesity', 'Exercise'])

print('New Data (basis for prediction):', '\n')
generate_ascii_table(df_new)

# Calculate probability of having Diabetes
result       = sigmoid(np.dot(data_new, weights) + bias)
print('\n', 'Predicted probability of having Diabetes:', result)
```

```
New Data (basis for prediction):

+---------+---------+----------+
| Smoking | Obesity | Exercise |
+---------+---------+----------+
|    0    |    1    |    0     |
+---------+---------+----------+

 Predicted probability of having Diabetes: [[0.95602528]]
```

Jupyter notebook --footer info-- (please always provide this

at the end of each submitted notebook)

In [9]:
```
import os
import platform
import socket
from platform import python_version
from datetime import datetime

print('-----------------------------------')
print(os.name.upper())
print(platform.system(), '|', platform.release())
print('Datetime:', datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
print('Python Version:', python_version())
print('-----------------------------------')
```

```
-----------------------------------
NT
Windows | 11
Datetime: 2025-11-09 16:40:08
Python Version: 3.13.9
-----------------------------------
```