Recommender Systems

**Prof. Dietmar Jannach Alpen‐Adria‐Universität Klagenfurt**

**Assignment 1**

# Task 1) Installing and running Python

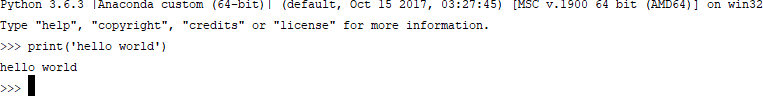
* Download and install the latest version of the community edition of the *PyCharm IDE* with

*Python version 3.8* (https://[www.jetbrains.com/pycharm/).1](http://www.jetbrains.com/pycharm/).1)

* Open a terminal window in the IDE, use the button on the lower end of the screen.



Type “python” to start the interactive interpreter. Type “print(‘Hello World’)” in the terminal.



A screen shot of a computer

Description automatically generated

* Create a file called “hello.py” and type “print(‘Hello World’)”. Run the file.

# Task 2) Basic language concepts2

**Task 2.1) Opening files3, list data structure, loops**

Download the “MovieLens ‐ recommended for education and development ‐ Small4” dataset from https://grouplens.org/datasets/movielens/.

Write a program that determines the mean rating in the dataset in the following way.

* Create a list of data type “float” to store all ratings in memory.
* Open the file “ratings.csv” and read the contents line by line.
* Store each rating in the list.
* Close the file.
* Iterate through the resulting list, sum up the values and calculate the average at the end.
* Print the result.  
  CODE:
* import zipfile  
  import csv  
    
  # Path to the zip file  
  zip\_file\_path = r'C:\Users\user\Documents\Uni\Semester 2\Information Search & Recommendation Systems\HW\ml-latest-small.zip' # Modify this line based on your specific path on your PC  
  csv\_file\_name = 'ml-latest-small/ratings.csv' # Include the subdirectory in the path  
    
  # Function to calculate the mean of ratings  
  def calculate\_mean(ratings):  
   total\_sum = sum(ratings)  
   count = len(ratings)  
   mean\_rating = total\_sum / count if count > 0 else 0  
   return mean\_rating  
    
  # Main function to handle the process  
  def process\_ratings():  
   ratings = [] # List to store ratings as floats  
    
   # Open the zip file  
   with zipfile.ZipFile(zip\_file\_path, 'r') as z:  
   # Open the CSV file within the zip file  
   with z.open(csv\_file\_name) as csvfile:  
   reader = csv.reader(map(lambda x: x.decode('utf-8'), csvfile))  
   next(reader) # Skip the header row  
   for row in reader:  
   rating = float(row[2]) # third column is the rating column  
   ratings.append(rating)  
    
   # Calculate the mean rating  
   mean\_rating = calculate\_mean(ratings)  
    
   # Print the result  
   print(f"The mean rating is: {mean\_rating}")  
    
  # Run the function  
  if \_\_name\_\_ == '\_\_main\_\_':  
   process\_ratings()

# Task 2.2) Functions and error handling

1 You can use https://[www.instructables.com/id/Python‐Hello‐World/](http://www.instructables.com/id/Python) for Python Hello World in PyCharm.

2 You can use https://[www.w3schools.com/python/](http://www.w3schools.com/python/) as a starting point.

3 You can use https://python4mpia.github.io/pure\_python/files.html for file handling in plain Python.

All calculations from Task 2.1 should now be done within a function called “computeMeanRating”, which takes a file name as an input and returns a float as a result.

Define this function, implement appropriate error handling procedures (including exception handling in case the file cannot be read or found), and write a main function that invokes the method.

CODE:

import zipfile  
import csv  
  
# Path to the zip file  
zip\_file\_path = r'C:\Users\user\Documents\Uni\Semester 2\Information Search & Recommendation Systems\HW\ml-latest-small.zip' # Modify this line based on your specific path on your PC  
  
# Function to calculate the mean of ratings  
def calculate\_mean(ratings):  
 total\_sum = sum(ratings)  
 count = len(ratings)  
 mean\_rating = total\_sum / count if count > 0 else 0  
 return mean\_rating  
  
# Function to compute the mean rating from a given file inside a zip archive  
def computeMeanRating(zip\_file\_path, internal\_csv\_path):  
 ratings = [] # List to store ratings as floats  
 try:  
 # Open the zip file  
 with zipfile.ZipFile(zip\_file\_path, 'r') as z:  
 # Open the CSV file within the zip file  
 with z.open(internal\_csv\_path) as csvfile:  
 reader = csv.reader(map(lambda x: x.decode('utf-8'), csvfile))  
 next(reader) # Skip the header row  
 for row in reader:  
 rating = float(row[2])  
 ratings.append(rating)  
  
 # Calculate and return the mean rating  
 return calculate\_mean(ratings)  
 except FileNotFoundError:  
 print(f"Error: The file '{zip\_file\_path}' does not exist.")  
 return None  
 except KeyError:  
 print(f"Error: The file '{internal\_csv\_path}' does not exist within the zip archive.")  
 return None  
 except Exception as e:  
 print(f"An error occurred: {e}")  
 return None  
  
# Main function to handle the process  
def main():  
 csv\_file\_name = 'ml-latest-small/ratings.csv' # Include the subdirectory in the path  
 mean\_rating = computeMeanRating(zip\_file\_path, csv\_file\_name)  
 if mean\_rating is not None:  
 print(f"The mean rating is: {mean\_rating:.2f}")  
 else:  
 print("Failed to compute the mean rating due to an earlier error.")  
  
# Run the function  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Task 2.3) Functions and return values

Extend the function from Task 2.2 so that it returns not only the mean value, but also the mode and the median5. Write a corresponding test method.

CODE:

import zipfile  
import csv  
  
# Function to calculate mean, mode, and median manually  
def calculate\_mean(ratings):  
 return sum(ratings) / len(ratings) if ratings else 0  
  
def calculate\_mode(ratings):  
 frequency = {}  
 for rating in ratings:  
 if rating in frequency:  
 frequency[rating] += 1  
 else:  
 frequency[rating] = 1  
 max\_count = max(frequency.values(), default=0)  
 modes = [rate for rate, count in frequency.items() if count == max\_count]  
 return modes[0] if modes else None # Return the first mode found  
  
def calculate\_median(ratings):  
 n = len(ratings)  
 if n == 0:  
 return None  
 sorted\_ratings = sorted(ratings)  
 mid = n // 2  
 if n % 2 == 1:  
 return sorted\_ratings[mid]  
 else:  
 return (sorted\_ratings[mid - 1] + sorted\_ratings[mid]) / 2.0  
  
# Function to compute the mean, mode, and median from a given file inside a zip archive  
def computeStats(zip\_file\_path, internal\_csv\_path):  
 ratings = [] # List to store ratings as floats  
 try:  
 # Open the zip file  
 with zipfile.ZipFile(zip\_file\_path, 'r') as z:  
 # Open the CSV file within the zip file  
 with z.open(internal\_csv\_path) as csvfile:  
 reader = csv.reader(map(lambda x: x.decode('utf-8'), csvfile))  
 next(reader) # Skip the header row  
 for row in reader:  
 rating = float(row[2]) # Assuming the third column is the rating  
 ratings.append(rating)  
  
 # Calculate and return the mean, mode, and median  
 mean\_rating = calculate\_mean(ratings)  
 mode\_rating = calculate\_mode(ratings)  
 median\_rating = calculate\_median(ratings)  
 return mean\_rating, mode\_rating, median\_rating  
 except FileNotFoundError:  
 print(f"Error: The file '{zip\_file\_path}' does not exist.")  
 return None  
 except KeyError:  
 print(f"Error: The file '{internal\_csv\_path}' does not exist within the zip archive.")  
 return None  
 except Exception as e:  
 print(f"An error occurred: {e}")  
 return None  
  
# Test method to verify the computeStats function  
def test\_computeStats():  
 zip\_file\_path = r'C:\Users\user\Documents\Uni\Semester 2\Information Search & Recommendation Systems\HW\ml-latest-small.zip' # Modify this line based on your specific path on your PC  
 csv\_file\_name = 'ml-latest-small/ratings.csv'  
 results = computeStats(zip\_file\_path, csv\_file\_name)  
 if results:  
 print("Test Passed: Mean={}, Mode={}, Median={}".format(\*results))  
 else:  
 print("Test Failed")  
  
# Main function to handle the process  
def main():  
 test\_computeStats()  
  
# Run the function  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Task 2.4) More data structures and file handling

Our next goal is to analyze the genres that are appearing in the file “movies.csv”.

Write a procedure that takes the file name as a parameter and prints the following on the screen:

* All distinct genre names that appear in the file. You can use the Python *csv* module.
* For each genre, determine to how many movies it was assigned. Use a dictionary (genre ‐> counter) to save the number of genre assignments.
  + Print the number of movies per genre
  + Determine and print out the most popular genre.
* Optional: Sort the genres by the number of movies they are assigned to in descending order. Use a suitable library function.

CODE:

import zipfile  
import csv  
from collections import defaultdict  
import operator  
  
def analyze\_genres(zip\_file\_path, internal\_csv\_path):  
 genre\_count = defaultdict(int) # Dictionary to store genre counts  
  
 try:  
 # Open the zip file  
 with zipfile.ZipFile(zip\_file\_path, 'r') as z:  
 # Open the movies CSV file within the zip file  
 with z.open(internal\_csv\_path) as csvfile:  
 reader = csv.reader(map(lambda x: x.decode('utf-8'), csvfile))  
 next(reader) # Skip the header row  
 for row in reader:  
 genres = row[2].split('|') # Assuming the third column is the genres  
 for genre in genres:  
 if genre != '(no genres listed)': # Filter out unlisted genres  
 genre\_count[genre] += 1  
  
 # Print all distinct genres and their counts  
 print("Genre counts:")  
 for genre, count in genre\_count.items():  
 print(f"{genre}: {count}")  
  
 # Determine the most popular genre  
 most\_popular\_genre = max(genre\_count.items(), key=operator.itemgetter(1))  
 print(f"The most popular genre is: {most\_popular\_genre[0]} with {most\_popular\_genre[1]} movies.")  
  
 # Optional: Sort the genres by number of movies in descending order  
 sorted\_genres = sorted(genre\_count.items(), key=lambda item: item[1], reverse=True)  
 print("\nGenres sorted by popularity:")  
 for genre, count in sorted\_genres:  
 print(f"{genre}: {count}")  
  
 except FileNotFoundError:  
 print(f"Error: The file '{zip\_file\_path}' does not exist.")  
 except KeyError:  
 print(f"Error: The file '{internal\_csv\_path}' does not exist within the zip archive.")  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
# Main function to invoke the analysis  
def main():  
 zip\_file\_path = r'C:\Users\user\Documents\Uni\Semester 2\Information Search & Recommendation Systems\HW\ml-latest-small.zip'  
 csv\_file\_name = 'ml-latest-small/movies.csv'  
 analyze\_genres(zip\_file\_path, csv\_file\_name)  
  
# Run the function  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Task 2.5) Modules and classes

Define a Python module “utilityModule” including a class “Statistics” and add the function defined in Task 2.2 as a method to this class.

Write a test program that invokes the method (and thus prints the mean rating in the dataset).

CODE:

utilityModule.py:

import zipfile  
import csv  
  
class Statistics:  
 def \_\_init\_\_(self, zip\_file\_path, internal\_csv\_path):  
 self.zip\_file\_path = zip\_file\_path  
 self.internal\_csv\_path = internal\_csv\_path  
  
 def calculate\_mean(self, ratings):  
 return sum(ratings) / len(ratings) if ratings else 0  
  
 def calculate\_mode(self, ratings):  
 frequency = {}  
 for rating in ratings:  
 if rating in frequency:  
 frequency[rating] += 1  
 else:  
 frequency[rating] = 1  
 max\_count = max(frequency.values(), default=0)  
 modes = [rate for rate, count in frequency.items() if count == max\_count]  
 return modes[0] if modes else None # Return the first mode found  
  
 def calculate\_median(self, ratings):  
 n = len(ratings)  
 if n == 0:  
 return None  
 sorted\_ratings = sorted(ratings)  
 mid = n // 2  
 if n % 2 == 1:  
 return sorted\_ratings[mid]  
 else:  
 return (sorted\_ratings[mid - 1] + sorted\_ratings[mid]) / 2.0  
  
 def compute\_stats(self):  
 ratings = [] # List to store ratings as floats  
 try:  
 with zipfile.ZipFile(self.zip\_file\_path, 'r') as z:  
 with z.open(self.internal\_csv\_path) as csvfile:  
 reader = csv.reader(map(lambda x: x.decode('utf-8'), csvfile))  
 next(reader) # Skip the header row  
 for row in reader:  
 rating = float(row[2]) # Assuming the third column is the rating  
 ratings.append(rating)  
  
 mean = self.calculate\_mean(ratings)  
 mode = self.calculate\_mode(ratings)  
 median = self.calculate\_median(ratings)  
 return mean, mode, median  
 except FileNotFoundError:  
 print(f"Error: The file '{self.zip\_file\_path}' does not exist.")  
 except KeyError:  
 print(f"Error: The file '{self.internal\_csv\_path}' does not exist within the zip archive.")  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
 return None

Test code:  
  
from utilityModule import Statistics  
  
def test\_compute\_stats():  
 zip\_path = r'C:\Users\user\Documents\Uni\Semester 2\Information Search & Recommendation Systems\HW\ml-latest-small.zip'  
 csv\_path = 'ml-latest-small/ratings.csv'  
 stats = Statistics(zip\_path, csv\_path)  
 result = stats.compute\_stats()  
 if result:  
 mean, mode, median = result  
 print(f"Mean: {mean}, Mode: {mode}, Median: {median}")  
 else:  
 print("Failed to compute statistics.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 test\_compute\_stats()