Group160_NoteBook

January 30, 2026

1 TKO_7093 - Statistical Data Analysis Project

1.1 Group 160

1.2 Team Members

Name	Student ID
Ayana Kotuwegoda Guruge	2406865
Sheheryar Wahidi	2413773
Yagya Yadav	2409273

1.3 Data Preparation

[1]:

```
#Needed libraries
#from google.colab import drive
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy import stats
import scikit_posthocs as sp
from scipy.stats import chi2_contingency
```

[2]:

```
#drive.mount('/content/drive')
#!ls /content/drive/
```

1.3.1 Following steps are carried out for the data preparation:

This step is for checking the dataset, cleaning and structuring for further analysis (qualititative).

- Data loaded from habits.data into a DataFrame matching the variables.
- Missing values are checked and not removed.
- Codes are converted into readable lables.

- Checking all demographic values for sensible values.
- Checking the validity of the ranges.
- Replacing the invalid values of the variables accordingly.

[3]:
```
# Load the dataset from Path
data_path = "habits.data"
columns = ['kohde', 'jasen', 'pvknro', 'sp', 'ASALUE', 'IKAL1', 'A1', 'A2',␣
  ↪'A3', 'A4', 'A5']
df = pd.read_csv(data_path, sep=";", header=0, usecols=columns, na_values=['?'])

# Initial exploration of the dataset
df.head()
df.shape
df.info()
df.describe()

# Display the first few rows of the dataframe
display(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 745 entries, 0 to 744
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   kohde   745 non-null    int64
 1   jasen   745 non-null    int64
 2   pvknro  745 non-null    int64
 3   sp      745 non-null    int64
 4   ASALUE  745 non-null    float64
 5   IKAL1   745 non-null    int64
 6   A1      741 non-null    object
 7   A2      737 non-null    object
 8   A3      733 non-null    object
 9   A4      736 non-null    object
 10  A5      703 non-null    float64
dtypes: float64(2), int64(5), object(4)
memory usage: 64.2+ KB
```

```
   kohde  jasen  pvknro  sp  ASALUE  IKAL1   A1   A2  A3   A4   A5
0  50002      1       1   1     1.0     49    0  560   0   80  1.0
1  50002      1       2   1     1.0     49  380  450  10    0  1.0
2  50003      1       1   2     2.0     41    0  470  30  100  1.0
3  50003      1       2   2     2.0     41    0  550   0    0  1.0
4  50004      2       1   1     1.0     62  640  410   0    0  1.0
```

- Structure was checked because according to the instructions given as there are multiple rows of the same person.
- This is more to show that one person can be recorded for more than one day. Changes will be done in task 1

  - 6 columns (Household ID, Person ID, day type, sex, living area, age group) are demographic variables.
  - 5 columns (A1-A5) which are activity variables.
- A1 to A4 variables are shown as objects where it should be in numerics since it is the time spent in activities in minutes according to the text file. This is due to mixed formatting or missing values or both.
- The appearance of A5 variable (Visiting library) in float which aslo should be in numerics.

[4]:

```python
# Checking for missing values
print("Missing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
kohde      0
jasen      0
pvknro     0
sp         0
ASALUE     0
IKAL1      0
A1         4
A2         8
A3        12
A4         9
A5        42
dtype: int64
```

- It shows that there some missing values in the activity variables but no missing values in the demographic variables.

- Number of missing values of activity columns:

  - A1 - 4
  - A2 - 8
  - A3 - 12
  - A4 - 9
  - A5 - 42

- Missing values are kept as NaN.

- Missing values are not always "0 minutes spent"/" no activity". It can also mean that the data entry was missed to be recorded or an error. Therefore, it is better to keep them as NaN value guessing or maintaining as 0 assuming that there was no activity.

1.3.2 Data Type Conversion and Cleaning

[5]:

```python
# Replace '?' with NaN for consistent value handling
df_clean = df.replace('?', np.nan)

# Convert demographic variables to appropriate types respectrively (all numeric␣
 ↪in this case)
```

```python
df_clean['kohde'] = pd.to_numeric(df_clean['kohde'], errors='coerce')
df_clean['jasen'] = pd.to_numeric(df_clean['jasen'], errors='coerce')
df_clean['pvknro'] = pd.to_numeric(df_clean['pvknro'], errors='coerce')
df_clean['sp'] = pd.to_numeric(df_clean['sp'], errors='coerce')
df_clean['IKAL1'] = pd.to_numeric(df_clean['IKAL1'], errors='coerce')
df_clean['ASALUE'] = pd.to_numeric(df_clean['ASALUE'], errors='coerce')

# Activity variables converting to numeric, handling time format data
for col in ['A1', 'A2', 'A3', 'A4', 'A5']:
    # A1-A4 are minutes and A5 in yes/no format, convert accordingly
    df_clean[col] = pd.to_numeric(df_clean[col], errors='coerce')

print("Data types after conversion:")
print(df_clean.dtypes)
print("\nMissing values per column")
print(df_clean.isnull().sum())
print("\nDataFrame shape:", df_clean.shape)
```

```
Data types after conversion:
kohde       int64
jasen       int64
pvknro      int64
sp          int64
ASALUE    float64
IKAL1       int64
A1        float64
A2        float64
A3        float64
A4        float64
A5        float64
dtype: object

Missing values per column
kohde      0
jasen      0
pvknro     0
sp         0
ASALUE     0
IKAL1      0
A1        85
A2        88
A3        94
A4        91
A5        42
dtype: int64

DataFrame shape: (745, 11)
```

The above conversions will ensure that later calculations will be correct since all the incorrect and missing values are now treated as missing values (NaN). This was done using ‘errors=“coerce”’ where cleaning up will not create new missing data but changing the already existing ones.

[6]:

```python
# Checking for ranges from the text file for variables pvknro, sp, ASALUE, IKAL1
allowed_ranges = {
    'pvknro' : {1,2},
    'sp' : {1,2},
    'ASALUE' : {1,2,3},
    'IKAL1' : set(range(1,10)), # 1 to 9
    'A5' : {1,2} # yes/no
}

for col, valid_values in allowed_ranges.items():
    unique_values = set(df[col].dropna().unique())
    invalid_values = unique_values - valid_values
    print(f"{col}:")
    print(f"Unique values found: {sorted(unique_values)}")
    if invalid_values:
        print(f"Invalid values found: {sorted(invalid_values)}")
    else:
        print("All values are valid.")
    print()
```

```
pvknro:
Unique values found: [np.int64(1), np.int64(2)]
All values are valid.

sp:
Unique values found: [np.int64(1), np.int64(2)]
All values are valid.

ASALUE:
Unique values found: [np.float64(1.0), np.float64(2.0), np.float64(3.0)]
All values are valid.

IKAL1:
Unique values found: [np.int64(20), np.int64(21), np.int64(22), np.int64(23),
np.int64(24), np.int64(25), np.int64(26), np.int64(27), np.int64(28),
np.int64(29), np.int64(30), np.int64(31), np.int64(32), np.int64(33),
np.int64(34), np.int64(35), np.int64(36), np.int64(37), np.int64(38),
np.int64(39), np.int64(40), np.int64(41), np.int64(42), np.int64(43),
np.int64(44), np.int64(45), np.int64(46), np.int64(47), np.int64(48),
np.int64(49), np.int64(50), np.int64(51), np.int64(52), np.int64(53),
np.int64(54), np.int64(55), np.int64(56), np.int64(57), np.int64(58),
np.int64(59), np.int64(60), np.int64(61), np.int64(62), np.int64(63),
np.int64(64), np.int64(65), np.int64(66), np.int64(67), np.int64(68),
np.int64(69), np.int64(70), np.int64(71), np.int64(72), np.int64(73),
np.int64(74), np.int64(75), np.int64(76), np.int64(77), np.int64(78),
np.int64(79), np.int64(80), np.int64(81), np.int64(82), np.int64(83),
np.int64(84)]
Invalid values found: [np.int64(20), np.int64(21), np.int64(22), np.int64(23),
np.int64(24), np.int64(25), np.int64(26), np.int64(27), np.int64(28),
np.int64(29), np.int64(30), np.int64(31), np.int64(32), np.int64(33),
np.int64(34), np.int64(35), np.int64(36), np.int64(37), np.int64(38),
np.int64(39), np.int64(40), np.int64(41), np.int64(42), np.int64(43),
np.int64(44), np.int64(45), np.int64(46), np.int64(47), np.int64(48),
np.int64(49), np.int64(50), np.int64(51), np.int64(52), np.int64(53),
np.int64(54), np.int64(55), np.int64(56), np.int64(57), np.int64(58),
np.int64(59), np.int64(60), np.int64(61), np.int64(62), np.int64(63),
np.int64(64), np.int64(65), np.int64(66), np.int64(67), np.int64(68),
np.int64(69), np.int64(70), np.int64(71), np.int64(72), np.int64(73),
np.int64(74), np.int64(75), np.int64(76), np.int64(77), np.int64(78),
np.int64(79), np.int64(80), np.int64(81), np.int64(82), np.int64(83),
np.int64(84)]

A5:
Unique values found: [np.float64(0.0), np.float64(1.0), np.float64(2.0),
np.float64(60.0), np.float64(120.0), np.float64(420.0)]
Invalid values found: [np.float64(0.0), np.float64(60.0), np.float64(120.0),
np.float64(420.0)]
```

- Above verification is done because the missing values per column increased after converting the data types for the correct ones.
  - pvknro, sp, ASALUE are all valid.
  - IKAL1 which are age ranges which should only be from 1-9 have values more than 20.
    - Therefore, it is assumed that the ages are entered insted of the relevant age group number.
  - A5 should be having yes or no entries with either 1 or 2 to depict that the library was visited or not. But there are some float values entered.
    - Therefore, it is assumed that those float values are the number of hours spent in the library which will then be entered as 1 (yes) to which it will be changed.

Data correction Following steps are done in this phase. * Raw data frame will be created for reference. * All rows in the A5 will be assumed and conmverted to 1 and 2 (yes or no). * To avoid losing information, the original A5 variable was kept unchanged and a new binary variable (A5_binary) was created for analysis. * After conversion, A5_binary contains only valid categories (yes, no, and missing). * All IKAL1 ranges will be preserved in the raw DataFrame as reference. But the age groups will be sorted into 1-9 ranges as shown in the data.text file for clearer depictions in the coming tasks. This is done in the cleaned DataFrame.

[7]:
```
df_raw = df.copy()

# Age groups based on IKAL1
bins = [10, 14, 19, 24, 34, 44, 54, 64, 74, float('inf')]
```

```python
labels = ['10-14', '15-19', '20-24', '25-34',
          '35-44', '45-54', '55-64', '65-74', '75+']

df_clean['age_group'] = pd.cut(
    df_clean['IKAL1'],
    bins=bins,
    labels=labels,
    right=True
).astype('category')

df_clean['IKAL1'].describe()
df_clean['age_group'].value_counts(dropna=False).sort_index()

print("\nAge group check:")
print(df_clean[['IKAL1', 'age_group']].head(10))

def a5_to_binary(x):
    if pd.isna(x):
        return np.nan
    if x == 1 or x > 2:
        return 1  # yes
    if x == 2 or x == 0:
        return 2  # no
    return np.nan

# Create a NEW cleaned column (do NOT overwrite A5)
df_clean["A5_binary"] = df_clean["A5"].apply(a5_to_binary).astype("category")

# Verify: compare original vs cleaned
print("\n\nOriginal A5:")
print(df_clean["A5"].value_counts(dropna=False).head(10))

print("\nA5_binary distribution:")
print(df_clean["A5_binary"].value_counts(dropna=False))

df_clean.head()
```

```
Age group check:
   IKAL1 age_group
0     49     45-54
1     49     45-54
2     41     35-44
3     41     35-44
4     62     55-64
5     62     55-64
6     46     45-54
7     46     45-54
```

```
8     33     25-34
9     33     25-34


Original A5:
A5
1.0      474
2.0      209
NaN       42
0.0       17
120.0      1
60.0       1
420.0      1
Name: count, dtype: int64

A5_binary distribution:
A5_binary
 1.0    477
 2.0    226
NaN      42
Name: count, dtype: int64
```

[7]:

```
    kohde  jasen  pvknro  sp  ASALUE  IKAL1     A1     A2    A3     A4   A5  \
0  50002      1       1   1     1.0     49    0.0  560.0   0.0   80.0  1.0
1  50002      1       2   1     1.0     49  380.0  450.0  10.0    0.0  1.0
2  50003      1       1   2     2.0     41    0.0  470.0  30.0  100.0  1.0
3  50003      1       2   2     2.0     41    0.0  550.0   0.0    0.0  1.0
4  50004      2       1   1     1.0     62  640.0  410.0   0.0    0.0  1.0

  age_group A5_binary
0     45-54       1.0
1     45-54       1.0
2     35-44       1.0
3     35-44       1.0
4     55-64       1.0
```

Mapping and making sp, pvknro, ASALUE, IKAL1, A5 will be converted to categorical data types since it will help in the analysis and the data to be treated correctly in the descriptive statistics and statistical test.

[8]:

```python
# Convert coded variables to categorical with labels

df_clean["sp"] = df_clean["sp"].map({1: "Male", 2: "Female"}).astype("category")

df_clean["pvknro"] = df_clean["pvknro"].map(
    {1: "Weekday", 2: "Weekend"}
).astype("category")

```

```
df_clean["ASALUE"] = df_clean["ASALUE"].map(
    {1: "City", 2: "Municipality", 3: "Rural"}
).astype("category")


df_clean[["sp", "pvknro", "ASALUE", "age_group", "A5_binary"]].head()
df_clean.dtypes
```

[8]:

```
kohde           int64
jasen           int64
pvknro       category
sp           category
ASALUE       category
IKAL1           int64
A1            float64
A2            float64
A3            float64
A4            float64
A5            float64
age_group    category
A5_binary    category
dtype: object
```

1.4 Task 1 - Characterizing Individuals in the data

Each entry in the level is given in the data file is person-day level entries. But to keep the observations per individual has to be identified from the household ID and the person ID (kohde and jasen)

[9]:

```
persons = df_clean.drop_duplicates(subset=['kohde', 'jasen'])

print("df_clean shape:", df_clean.shape)
print("persons shape:", persons.shape)
```

```
df_clean shape: (745, 13)
persons shape: (378, 13)
```

[10]:

```
print("Number of individuals:", persons.shape[0])
print("Number of households:", persons["kohde"].nunique())
```

```
Number of individuals: 378
Number of households: 378
```

The above result shows that there are only 378 individuals in the data set. But some are duplicated because of the due to both weekend and weekday entries. 745 is a count of person-day observations.

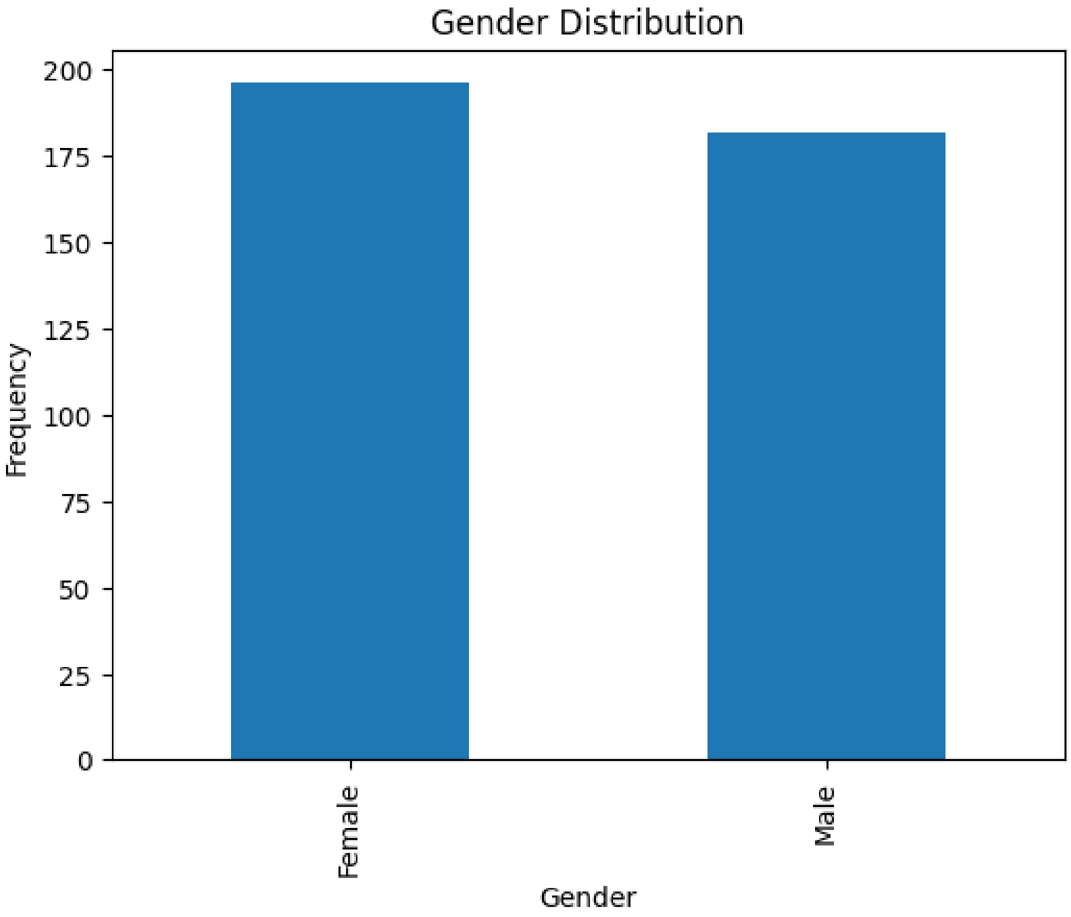
Gender, Age group, Living environment distribution

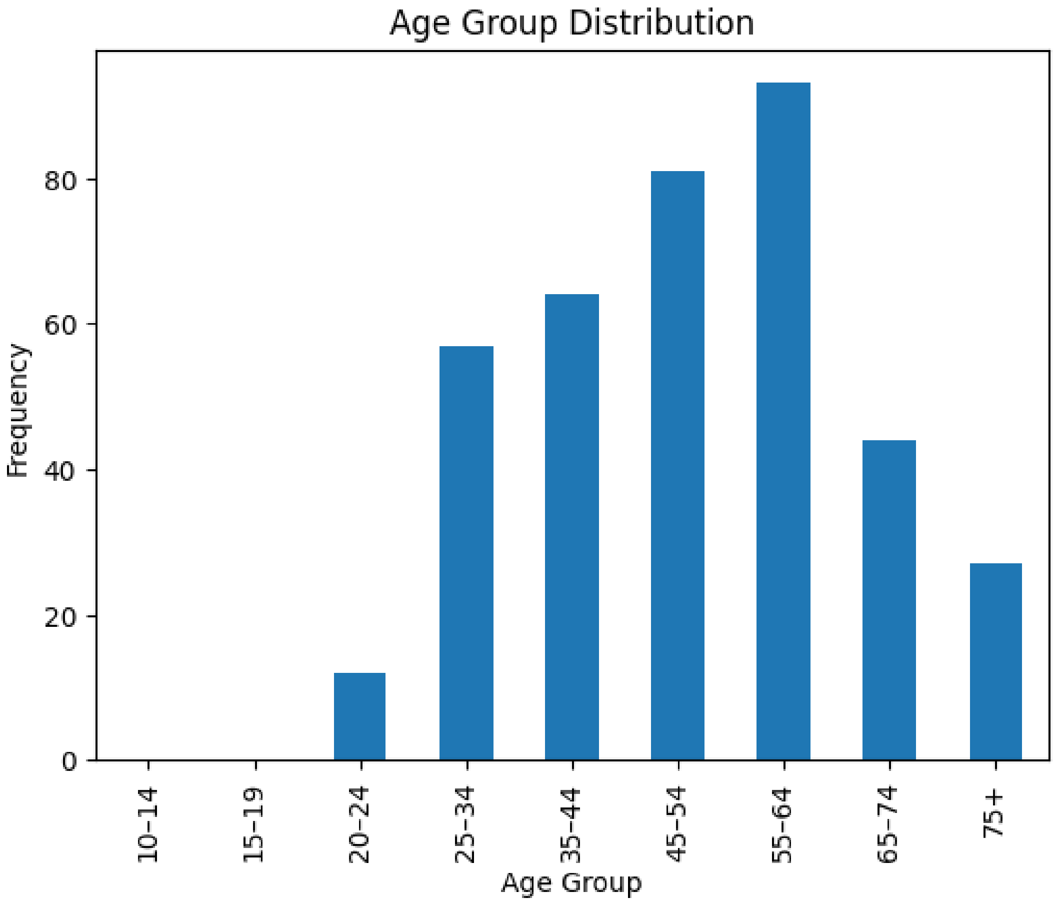
[11]:

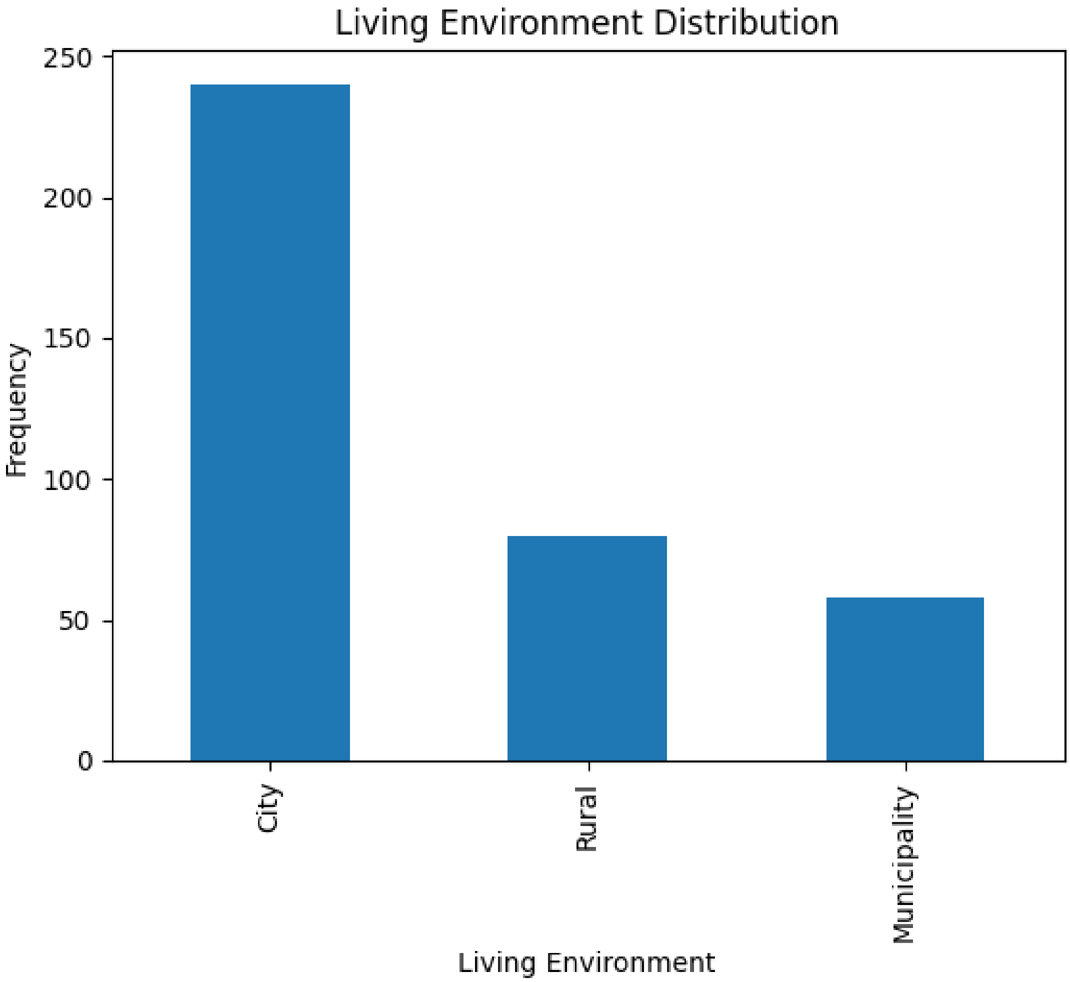
```
# Gender distribution
gender_counts = persons["sp"].value_counts()
```

```python
gender_pct = (gender_counts / len(persons) * 100).round(1)

pd.DataFrame({
    "Count": gender_counts,
    "Percentage (%)": gender_pct
})
print(pd.DataFrame({
    "Count": gender_counts,
    "Percentage (%)": gender_pct
}))

age_counts = persons["age_group"].value_counts().sort_index()
age_pct = (age_counts / len(persons) * 100).round(1)

pd.DataFrame({
    "Count": age_counts,
    "Percentage (%)": age_pct
})

print("\nAge Group Distribution:")
print(pd.DataFrame({
    "Count": age_counts,
    "Percentage (%)": age_pct
}))

env_counts = persons["ASALUE"].value_counts()
env_pct = (env_counts / len(persons) * 100).round(1)

pd.DataFrame({
    "Count": env_counts,
    "Percentage (%)": env_pct
})

print("\nLiving Environment Distribution:")
print(pd.DataFrame({
    "Count": env_counts,
    "Percentage (%)": env_pct
}))

gender_counts.plot(kind="bar", title="Gender Distribution")
plt.ylabel("Frequency")
plt.xlabel("Gender")
plt.show()

age_counts.plot(kind="bar", title="Age Group Distribution")
plt.ylabel("Frequency")
plt.xlabel("Age Group")
plt.show()

env_counts.plot(kind="bar", title="Living Environment Distribution")
plt.ylabel("Frequency")
plt.xlabel("Living Environment")
plt.show()
```

```
        Count  Percentage (%)
sp
Female    196            51.9
Male      182            48.1

Age Group Distribution:
           Count  Percentage (%)
age_group
10-14          0             0.0
15-19          0             0.0
20-24         12             3.2
25-34         57            15.1
35-44         64            16.9
45-54         81            21.4
55-64         93            24.6
65-74         44            11.6
75+           27             7.1

Living Environment Distribution:
              Count  Percentage (%)
ASALUE
City            240            63.5
Rural            80            21.2
Municipality     58            15.3
```


Gender Distribution
200
175
150
125
100
75
50
25
0
Frequency
Female
Male
Gender


Age Group Distribution
Frequency
0
20
40
60
80
10–14
15–19
20–24
25–34
35–44
45–54
55–64
65–74
75+
Age Group



- The analysis is based on individual-level data, where each person appears only once.
- The gender distribution is fairly balanced, with females making up about 52% of the sample and males about 48%.
- Most individuals belong to middle-aged groups, especially those aged 45–64 years, while younger and oldest age groups are less represented.
- The majority of individuals live in cities, with smaller proportions living in rural areas or municipalities.

[12]:
```
household_sizes = persons.groupby("kohde").size()

household_sizes.describe()
household_sizes.value_counts().sort_index()
print("Household Sizes Description:")
print(household_sizes.describe())
```

```
Household Sizes Description:
count    378.0
```

```
mean       1.0
std        0.0
min        1.0
25%        1.0
50%        1.0
75%        1.0
max        1.0
dtype: float64
```

- Household size was examined by counting the number of individuals per household.
- The results show that all households in the dataset consist of a single individual.
- As a result, household-level averages correspond directly to individual-level observations in the subsequent analysis.

1.5 Task 2 – Average daily time spent on activities

1.5.1 Estimating how much time on average the Finnish households spend on each activity.

- This is done in the household-level.
- Each household has one individual.
- Therefore, the averages of households are equal to the averages of individuals.
- Descriptve statistics are used (mean, standard deviation and intepretation)
- Only activity variables are used (A1-A4)

[13]:
```
activity_cols = ['A1', 'A2', 'A3', 'A4']

df_clean[activity_cols].describe()
```

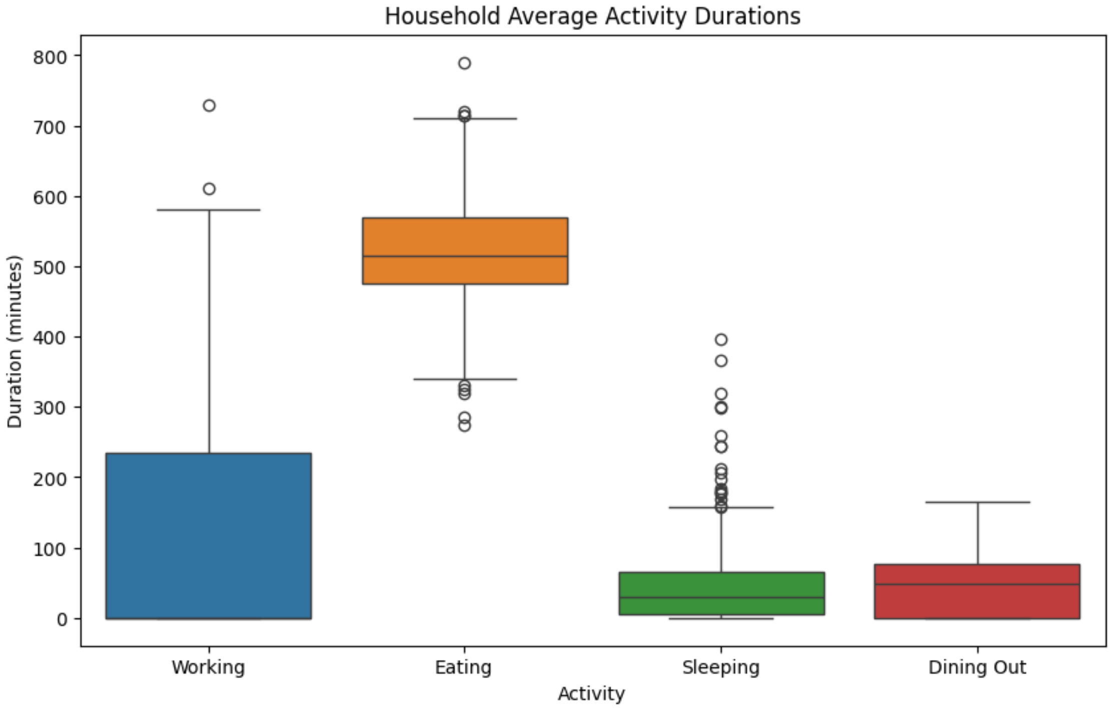
[13]:
```
                A1          A2          A3          A4
count   660.000000  657.000000  651.000000  654.000000
mean    120.606061  520.000000   47.983103   48.495413
std     208.142145  100.708617   64.043457   58.558586
min       0.000000   70.000000    0.000000    0.000000
25%       0.000000  465.000000    0.000000    0.000000
50%       0.000000  515.000000   30.000000    0.000000
75%     200.000000  580.000000   70.000000  100.000000
max    1050.000000  900.000000  474.000000  240.000000
```

- Above table shows a descriptive statistics for the daily time spent in activities in minutes.
- Most time is taken for sleep (A2) as an average. (520 mins)
- Working (A1) is varied and shows that not all invividuals work everyday.
- Reading (A3) and Dining (A4) have low averages and medians of 0. This means that they are occasionally done rather than daily activities.

[ ]:
```
household_activity_mean = (
    df_clean.groupby('kohde')[activity_cols]
    .mean()
)
```

```
print(household_activity_mean.head())

plt.figure(figsize=(10, 6))
sns.boxplot(data=household_activity_mean)
plt.title("Household Average Activity Durations")
plt.ylabel("Duration (minutes)")
plt.xlabel("Activity")

#rename x-ticks for clarity
plt.xticks(ticks=range(len(activity_cols)), labels=['Working', 'Sleeping',␣
  ↪'Reading', 'Dining Out'])
plt.show()
```

```
          A1     A2    A3     A4
kohde
50002  190.0  505.0   5.0   40.0
50003    0.0  510.0  15.0   50.0
50004  320.0  480.0  36.0   54.0
50005    0.0  545.0  46.0  108.0
50006    0.0  535.0  31.0   45.0
```



Following observations are done in the boxplot where average daily times for different activities are varied accross different households. - Sleeping is the most consistent activity, with high average durations and relatively little variation between households. - Working shows much larger variation, including many low or zero values, indicating that not everyone works every day or for the same amount of time. - Reading and dining out have much lower typical durations, suggesting these activities are done less frequently and for shorter periods. - A few households appear as outliers, especially for working and sleeping, reflecting unusually high time use on those activities.

1.5.2 Statistical estimations of average time spent on activities

- A 95% confidence interval is used. This is assuming the following:
  - Central limit theorom where large enough sample size is considered.
  - And taking household independantly.

[15]:

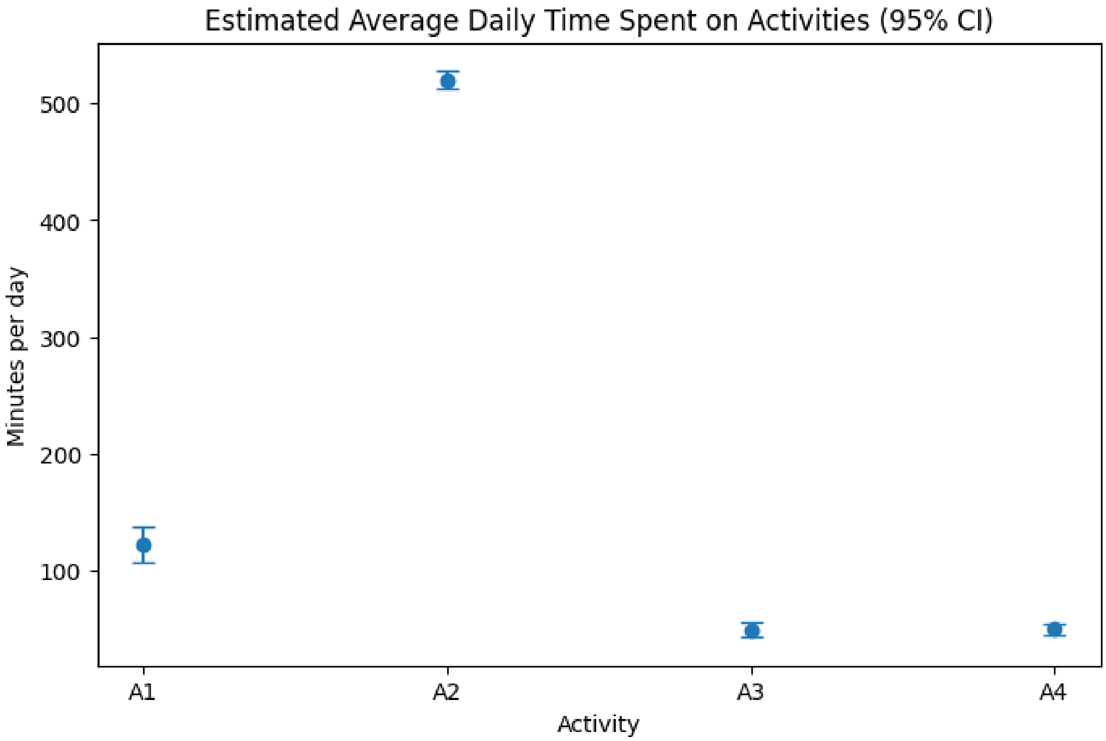
```
conint_results = []

n = len(household_activity_mean)

for col in activity_cols:
    mean = household_activity_mean[col].mean()
    std = household_activity_mean[col].std(ddof=1)
    se = std / np.sqrt(n)
    conint_low, conint_high = stats.t.interval(
        0.95, df=n-1, loc=mean, scale=se
    )
    conint_results.append((mean, conint_low, conint_high))

conint_df = pd.DataFrame(
    conint_results,
    index=activity_cols,
    columns=['Mean', '95% CI Lower', '95% CI Upper']
)
conint_df
```

[15]:

```
          Mean  95% CI Lower  95% CI Upper
A1  121.646884    106.590725    136.703043
A2  520.237389    512.625010    527.849768
A3   48.596439     42.511462     54.681416
A4   49.130952     44.721818     53.540087
```

[16]:

```
# Error bar plot for mean and 95% CI
means = conint_df["Mean"]
lower = conint_df["95% CI Lower"]
upper = conint_df["95% CI Upper"]

yerr = [means - lower, upper - means]

plt.figure(figsize=(8, 5))
plt.errorbar(conint_df.index, means, yerr=yerr, fmt="o", capsize=5)
plt.ylabel("Minutes per day")
plt.xlabel("Activity")
plt.title("Estimated Average Daily Time Spent on Activities (95% CI)")
plt.show()
```



- Sleeping (A2) has the highest estimated average time and a very narrow confidence interval, indicating a stable and consistent daily pattern across households.
- Working (A1) shows a lower average time and a wider confidence interval, reflecting substantial variation in working time between households.
- Reading (A3) and dining out (A4) have much lower estimated averages, with confidence intervals close to zero, confirming that these activities are not performed daily by most households.

Overall, the confidence intervals highlight clear differences in time allocation across activities while accounting for sampling uncertainty.

1.6 Task 3 – Differences between groups

This is to examine the time spent on activities between groups in Finland. The differences that are analyzed will be as follows: * Between weekdays and weekends. * Between living environments. * The analysis is done at the household level to get the independent observations. * Parametric statistical methods are applied to compare group means. Although individual activity durations are skewed and include many zero values, inference is based on household-level means. Given the moderate sample size, the Central Limit Theorem supports the use of mean-based inference.

1.6.1 Methods used

- Independent samples t-tests to compare weekends and weekdays.

- One-way ANOWA to compare living environments.
- 95% confidence intervals to quantify the estimation uncertainty.

[17]:

```
# Household-level averages by weekend and weekday
household_by_day = (
    df_clean
    .groupby(['kohde', 'pvknro'])[activity_cols]
    .mean()
    .reset_index()
)

household_by_day.head()
```

```
C:\Users\ayana\AppData\Local\Temp\ipykernel_397464\2454555527.py:4:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  .groupby(['kohde', 'pvknro'])[activity_cols]
```

[17]:

```
   kohde   pvknro     A1     A2    A3     A4
0  50002  Weekday    0.0  560.0   0.0   80.0
1  50002  Weekend  380.0  450.0  10.0    0.0
2  50003  Weekday    0.0  470.0  30.0  100.0
3  50003  Weekend    0.0  550.0   0.0    0.0
4  50004  Weekday  640.0  410.0   0.0    0.0
```

Weekday and weekend differences This is to examine the average time spent on activities that will differ between weekends and weekdays. Since they are independent in the household level, independent samples t-tests are used.

[18]:

```
# Splitting into weekday and weekend dataframes
weekday_data = household_by_day[household_by_day['pvknro'] == 'Weekday']
weekend_data = household_by_day[household_by_day['pvknro'] == 'Weekend']

print("Weekday Data Sample:")
print(weekday_data.head())
print("\nWeekend Data Sample:")
print(weekend_data.head())

# t-tests for each activity between weekday and weekend
t_test_results = []
for col in activity_cols:
    t_stat, p_val = stats.ttest_ind(
        weekday_data[col].dropna(),
        weekend_data[col].dropna(),
        equal_var=False # Welch's t-test which is safer for unequal variances
    )
    t_test_results.append((col, t_stat, p_val))
    ttest_df = pd.DataFrame(
        t_test_results,
        columns=['Activity', 't-statistic', 'p-value']
    )
print("\nT-test Results between Weekday and Weekend Activities:")
print(ttest_df)
```

```
Weekday Data Sample:
   kohde   pvknro     A1     A2    A3     A4
0  50002  Weekday    0.0  560.0   0.0   80.0
2  50003  Weekday    0.0  470.0  30.0  100.0
4  50004  Weekday  640.0  410.0   0.0    0.0
6  50005  Weekday    0.0  540.0  40.0    NaN
8  50006  Weekday    0.0  540.0   0.0   90.0

Weekend Data Sample:
   kohde   pvknro     A1     A2    A3     A4
1  50002  Weekend  380.0  450.0  10.0    0.0
3  50003  Weekend    0.0  550.0   0.0    0.0
5  50004  Weekend    0.0  550.0  72.0  108.0
7  50005  Weekend    0.0  550.0  52.0  108.0
9  50006  Weekend    0.0  530.0  62.0    0.0

T-test Results between Weekday and Weekend Activities:
  Activity  t-statistic   p-value
0       A1     0.324070  0.745988
1       A2    -0.834543  0.404282
2       A3    -1.657364  0.097940
3       A4    -2.775214  0.005682
```

Weekday vs Weekend Differences in Activity Time

- Differences between weekdays and weekends were tested using Welch's independent-samples t-tests, which are robust to unequal variances.
- No statistically significant differences were found for working (A1), sleeping (A2), or reading (A3) ($p > 0.05$).
- Dining out (A4) shows a statistically significant difference ($t = -2.78$, p 0.006), indicating higher average dining time on weekends.
- Overall, routine daily activities remain stable across the week, while leisure-related activities vary by day type.

[19]:
```
# Household level averages by living environment
household_by_env = (
    df_clean
    .groupby(['kohde'])[['ASALUE']+ activity_cols]
    .first()
    .reset_index()
)
print("\nHousehold Averages by Living Environment Sample:")
household_by_env.head()
```

```
Household Averages by Living Environment Sample:
```

[19]:

```
   kohde        ASALUE     A1     A2    A3     A4
0  50002          City    0.0  560.0   0.0   80.0
1  50003  Municipality    0.0  470.0  30.0  100.0
2  50004          City  640.0  410.0   0.0    0.0
3  50005          City    0.0  540.0  40.0  108.0
4  50006  Municipality    0.0  540.0   0.0   90.0
```

- Each household belongs to a single living environment, so the value of `ASALUE` does not change across observations.
- The `first()` function is used to retain one representative row per household after grouping.
- This avoids duplicate household entries and ensures independence of observations.
- The resulting dataset is suitable for comparing activity time across living environments using statistical tests.

Differences in Activity Time by Living Environment

- Using the Kruskal-Wallis test to find the strong skewness, zeros and the non-normal distributions.
- Examining whether the time spent on daily activities differs between living environments (city, municipality, and rural areas).
- The analysis is conducted at the household level to ensure independent observations.
- Because activity durations are skewed and contain many zero values, non-parametric statistical methods are used.

[20]:

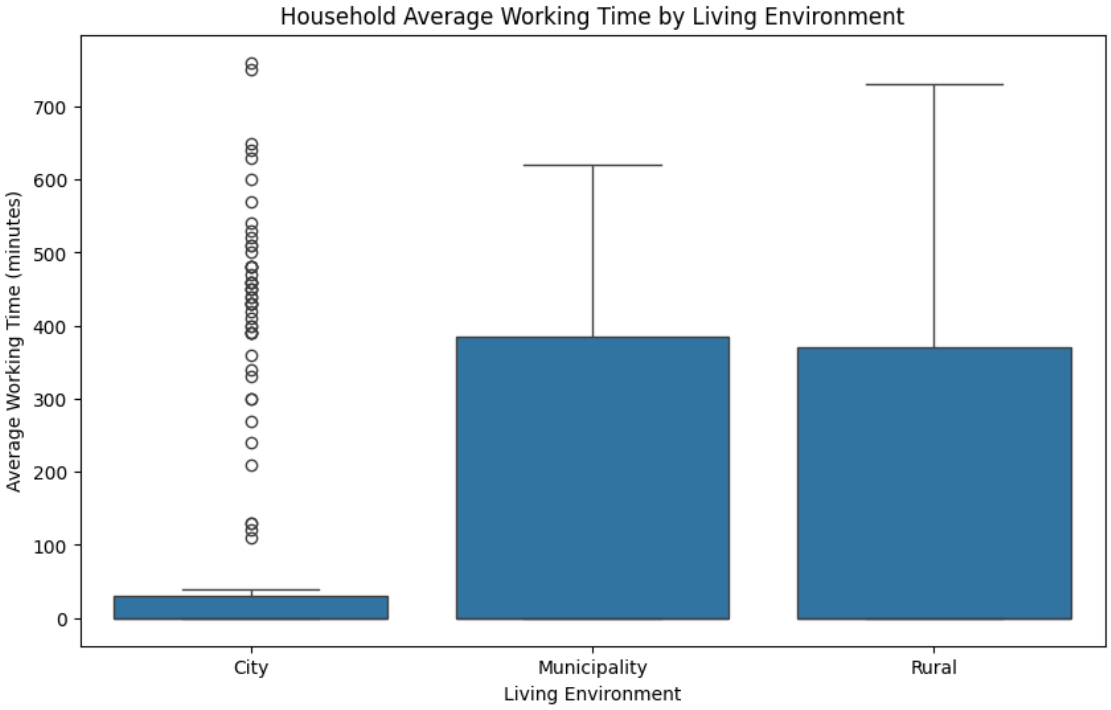
```
# Data preparation for ANOVA
city = household_by_env[household_by_env['ASALUE'] == 'City']
municipality = household_by_env[household_by_env['ASALUE'] == 'Municipality']
rural = household_by_env[household_by_env['ASALUE'] == 'Rural']
```

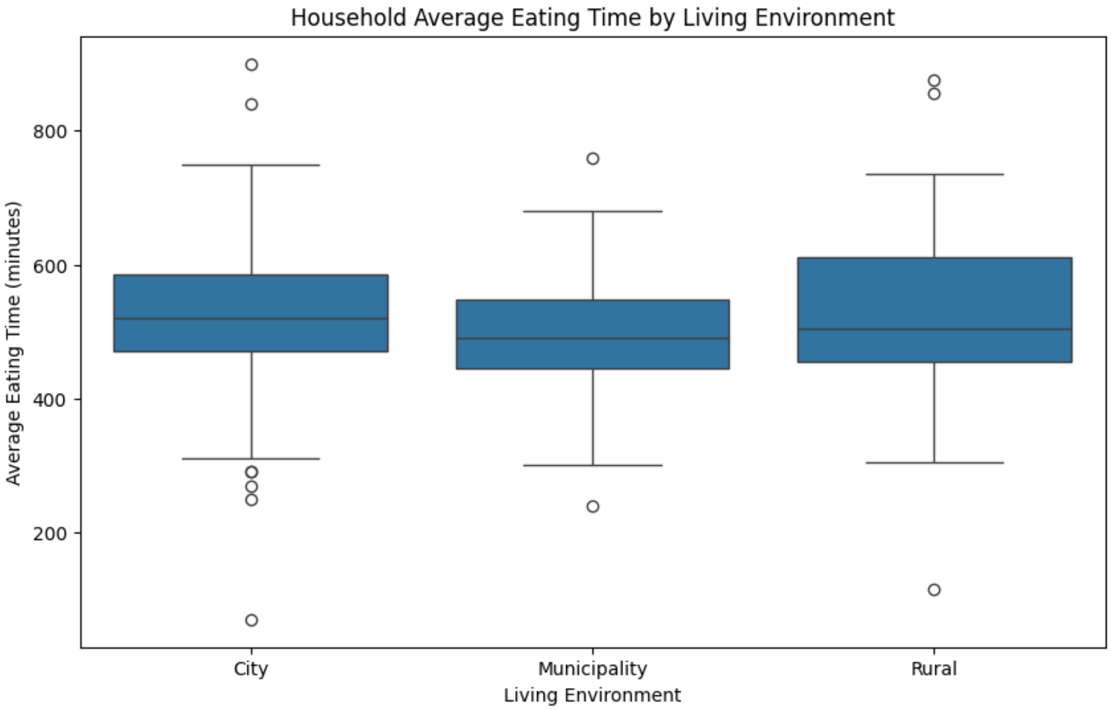
[ ]:

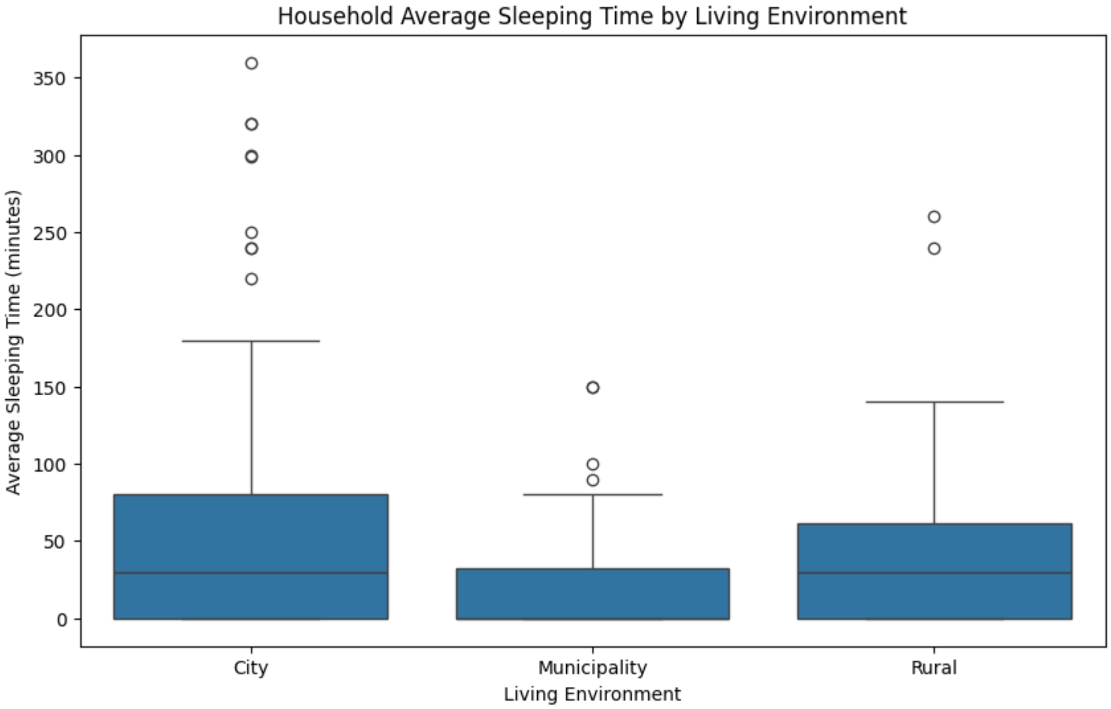
```
# Kruskal-Wallis ANOVA for each activity across living environments and␣
 ↪visualization
kw_results = []

for col in activity_cols:
    h_stat, p_val = stats.kruskal(
        city[col].dropna(),
        municipality[col].dropna(),
        rural[col].dropna()
    )
    kw_results.append((col, h_stat, p_val))

kw_df = pd.DataFrame(
```

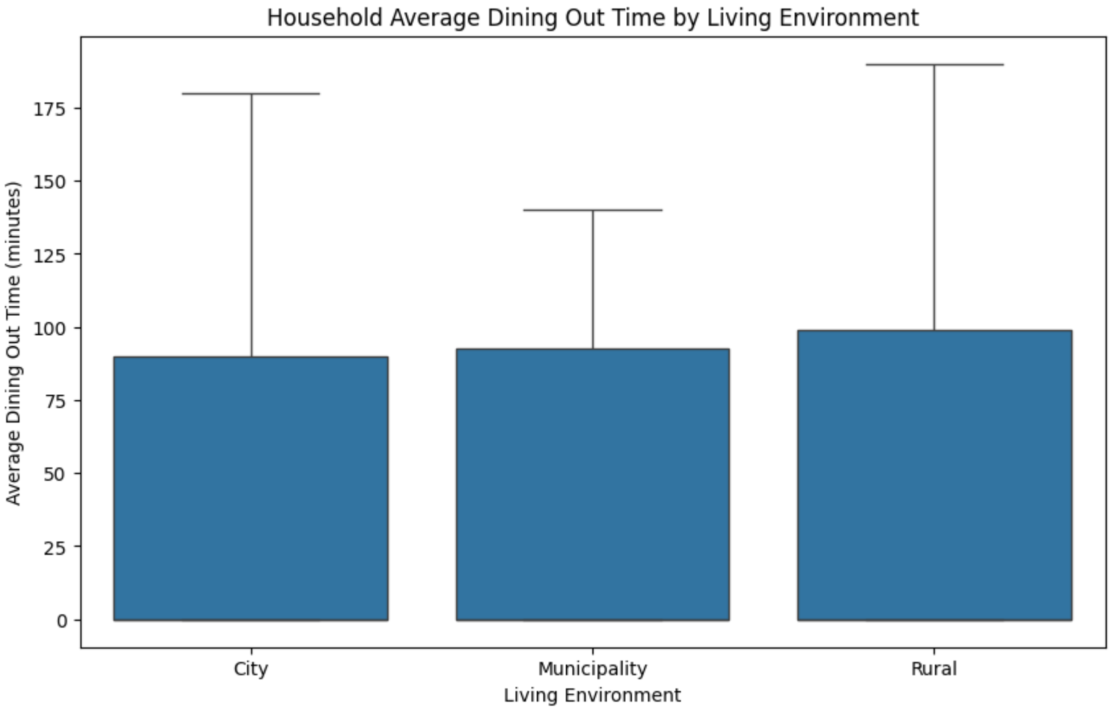
```python
    kw_results,
    columns=['Activity', 'H-statistic', 'p-value']
)

print("\nKruskal-Wallis ANOVA Results across Living Environments:")
print(kw_df)

plt.figure(figsize=(10, 6))
sns.boxplot(
    x='ASALUE',
    y='A1',
    data=household_by_env
)
plt.title("Household Average Working Time by Living Environment")
plt.ylabel("Average Working Time (minutes)")
plt.xlabel("Living Environment")
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(
    x='ASALUE',
    y='A2',
    data=household_by_env
)
plt.title("Household Average Sleeping Time by Living Environment")
plt.ylabel("Average Sleeping Time (minutes)")
plt.xlabel("Living Environment")
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(
    x='ASALUE',
    y='A3',
    data=household_by_env
)
plt.title("Household Average Reading Time by Living Environment")
plt.ylabel("Average Reading Time (minutes)")
plt.xlabel("Living Environment")
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(
    x='ASALUE',
    y='A4',
    data=household_by_env
)
plt.title("Household Average Dining Out Time by Living Environment")
plt.ylabel("Average Dining Out Time (minutes)")
plt.xlabel("Living Environment")
plt.show()
```

```
Kruskal-Wallis ANOVA Results across Living Environments:
  Activity  H-statistic   p-value
0       A1     5.931665  0.051518
1       A2     2.686724  0.260967
2       A3     8.899015  0.011684
3       A4     0.611458  0.736586
```




Household Average Eating Time by Living Environment
Average Eating Time (minutes)
800
600
400
200
City
Municipality
Rural
Living Environment


Household Average Sleeping Time by Living Environment
Average Sleeping Time (minutes)
350
300
250
200
150
100
50
0
City
Municipality
Rural
Living Environment



- Kruskal-Wallis H test shows the following.
  - Working (A1): Statistically no significant difference across living environments. This concludes that workiing times do not differ across environments.
  - Reading (A3): Differs significantly.
  - Sleeping (A2) and Dining out (A4) do not show significant variability to show there there are similar patterns across environments.

Therefore, post-hoc pairwise will not be tested since only one activity (sleeping) shows a significant difference. But a Dunn's test will be performed for A3 because of it.

[ ]:

```python
# Dunn's post-hoc test for A3
dunn_a3 = sp.posthoc_dunn(
    household_by_env,
    val_col='A3',
    group_col='ASALUE',
    p_adjust='bonferroni'
)

print("\nDunn's Post-hoc Test Results for A3 (Reading Time):")
print(dunn_a3)
```

```
Dunn's Post-hoc Test Results for A3 (Sleeping Time):
                   City  Municipality    Rural
City           1.000000      0.026893  1.00000
```

```
Municipality  0.026893      1.000000  0.12023
Rural         1.000000      0.120230  1.00000
```

- A statistically significant difference was found between city and municipality households (adjusted $p = 0.027$).
- No significant differences were observed between city and rural or municipality and rural households.
- This suggests that differences in Reading time across living environments are mainly driven by the contrast between city and municipality residents.

Visitng the library (A5) Since A5 was converted to a binary data type in the data cleaning and preparation stage due to the inconsistencies, it cannot be compared with means or ANOVA-type methods. Therefore, for further examination on how it has been differed across groups, categorical methods like cross-tabulations and Pearson's chi-square test are used for comparison rates. * Between living environments. * Between weekdays and weekends.

[23]:

```python
# Crosstabulation of A5_binary by ASALUE by living environment
a5_env_ct = pd.crosstab(
    df_clean['ASALUE'],
    df_clean['A5_binary'],
)

print("\nCrosstabulation of Visiting the library by living environment:")
print(a5_env_ct)

# Chi-square test of independence
chi2, p, dof, ex = chi2_contingency(a5_env_ct)

print(f"\nChi-square Test Results by living environments:")
print(f"Chi-square statistic: {chi2: .3f}")
print(f"Degrees of freedom: {dof}")
print(f"p-value: {p: .4f}")
```

```
Crosstabulation of Visiting the library by living environment:
A5_binary     1.0  2.0
ASALUE
City          311  128
Municipality   79   35
Rural          87   63

Chi-square Test Results by living environments:
Chi-square statistic:  8.584
Degrees of freedom: 2
p-value:  0.0137
```

- The test result is statistically significant ($^{2} = 8.58$, $p = 0.0137$) at the 5% significance level.
- This indicates that participation in A5 is not independent of living environment.
- City residents are more likely to participate in A5 compared to those living in municipalities

or rural areas.
- Therefore, living environment appears to be associated with engagement in activity A5.

[24]:
```
# Cross-tabulation of A5_binary by ASALUE by weekends and weekdays
a5_env_day_ct = pd.crosstab(
    df_clean['pvknro'],
    df_clean['A5_binary']
)

print("\nCrosstabulation of Visiting the library by Weekday/Weekend:")
print(a5_env_day_ct)

# Chi-square test of independence
chi2, p_day, dof_day, ex1 = chi2_contingency(a5_env_day_ct)
print(f"\nChi-square Test Results by Weekday/Weekend:")
print(f"Chi-square statistic: {chi2: .3f}")
print(f"Degrees of freedom: {dof}")
print(f"p-value: {p_day: .4f}")
```

```
Crosstabulation of Visiting the library by Weekday/Weekend:
A5_binary  1.0  2.0
pvknro
Weekday    239  116
Weekend    238  110

Chi-square Test Results by Weekday/Weekend:
Chi-square statistic:  0.049
Degrees of freedom: 2
p-value:  0.8243
```

- The test result is not statistically significant ($^2 = 0.05$, p = 0.824).
- This indicates that participation in A5 is independent of day type.
- Individuals are equally likely to engage in A5 on weekdays and weekends.
- Unlike living environment, day of the week does not appear to influence A5 participation.

Summarizing the comparisons Summary of Group Differences * **By day of week:** * Only dining out (A4) differs significantly, with higher values on weekends. * No significant differences were found for working (A1), eating (A2), sleeping (A3), or A5 participation.

- **By living environment:**
  - Kruskal–Wallis tests show a significant difference in sleeping time (A3) across living environments ($p < 0.05$).
  - No statistically significant differences were found for working (A1), eating (A2), or dining out (A4).
  - Post-hoc Dunn's test indicates that the significant effect in A3 is driven by a difference between city and municipality households.
  - Overall, living environment has a limited but detectable effect on sleeping behavior, while other daily activities remain broadly similar.

  - A5 participation differs significantly by living environment, with higher participation in cities.

1.7 Task 4 - Associations between activities in the Finnish population

Examining which daily activities are associated with others. The goal is to identify patterns such as whether working time is related to dining out or sleep etc.

1.7.1 Data used

- Durations of household-level activities.
- Activities

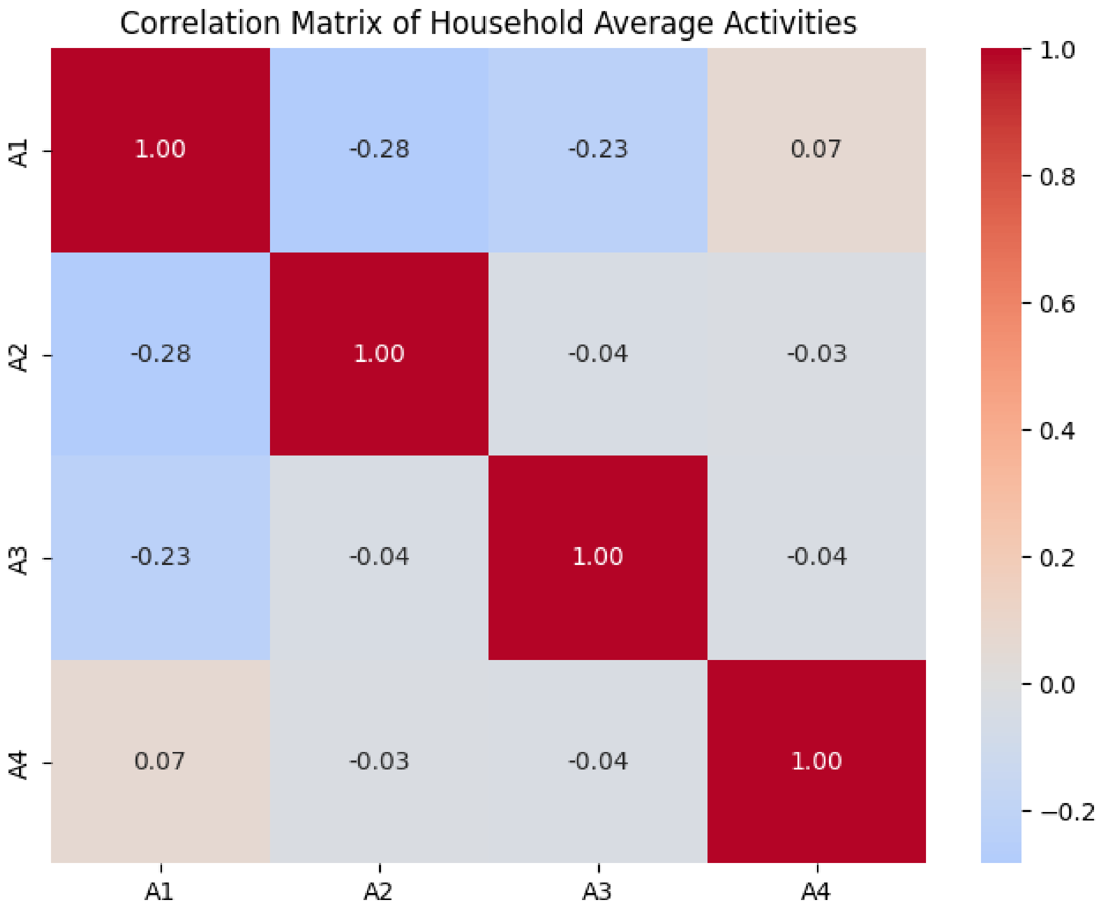
A corelation matrix will be implemented with a heatmap for the initial data mapping.

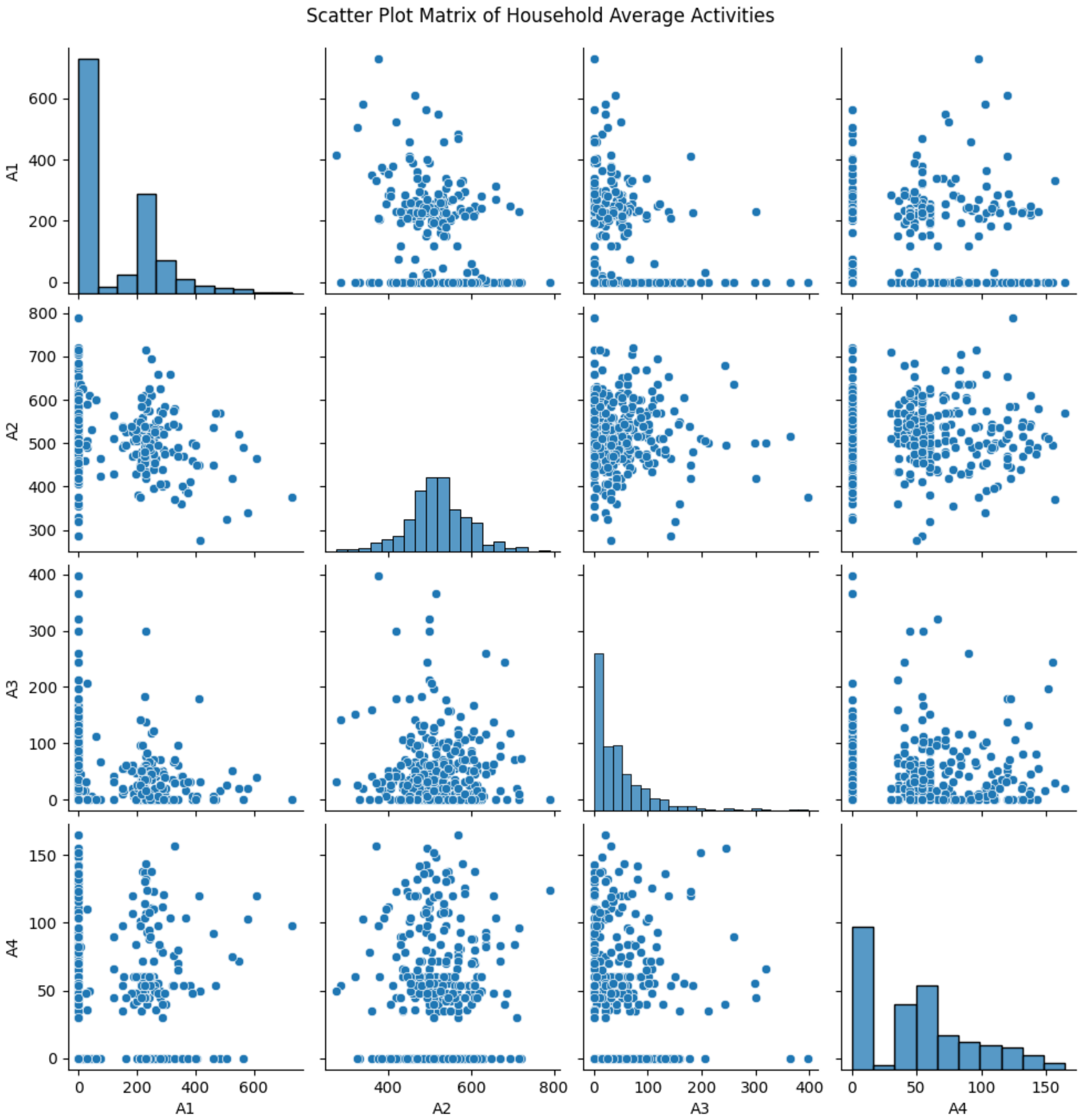
[25]:

```python
# Correlation matrix for activities

activity_cols = ['A1', 'A2', 'A3', 'A4']

# Household-level average activities
household_activity_mean = (
    df_clean.groupby('kohde')[activity_cols]
    .mean()
)

corr_matrix = household_activity_mean.corr()
print("\nCorrelation Matrix for Household Average Activities:")
print(corr_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    center=0
)
plt.title("Correlation Matrix of Household Average Activities")
plt.show()

# Scatter plot matrix
sns.pairplot(household_activity_mean)
plt.suptitle("Scatter Plot Matrix of Household Average Activities", y=1.02)
plt.show()
```

```
Correlation Matrix for Household Average Activities:
          A1        A2        A3        A4
A1  1.000000 -0.284277 -0.232733  0.067588
```

```
A2 -0.284277  1.000000 -0.040079 -0.026699
A3 -0.232733 -0.040079  1.000000 -0.035652
A4  0.067588 -0.026699 -0.035652  1.000000
```





Above correlation can be observed as follows: * The scatter plot matrix visualizes pairwise relationships between household-level average activity durations. * Most activity pairs show diffuse, cloud-like patterns, confirming the weak correlations observed numerically. * Strong zero-inflation is visible for working (A1), reading (A3), and dining out (A4), indicating that many households do not engage in these activities daily. * Sleeping time (A2) shows a more concentrated distribution, reflecting its regular and essential nature. * No clear linear or non-linear structures dominate the plots, suggesting that activity behaviors vary independently across households.

Overall, the low correlations suggest that each activity captures a different aspect of daily time use, and hence the use of multivariate techniques such as PCA to study underlying activity patterns further.

1.7.2 PCA Test to find common activity patterns

This is to combine the activities into few main patterns since the correlations are small. * Will be testing on household-level average minutes for A1-A4. * This is due to differenes in ranges and hence the standardization. Therefore the PCA will not be dominated by the largest variable like sleep.

[26]:

```python
activity_cols = ['A1', 'A2', 'A3', 'A4']

# Household-level averages
X = df_clean.groupby('kohde')[activity_cols].mean()

# Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

print("Households used in PCA:", X.shape[0])
print("\nMissing values in PCA data:", np.isnan(X_std).sum())
```

```
Households used in PCA: 378

Missing values in PCA data: 165
```

Note on missing values before PCA

Although missing values were handled during data preparation, additional missing values appear after aggregating the data to the household level. This occurs because some households did not engage in certain activities at all, resulting in undefined household-level averages.

Since PCA requires complete observations, households with missing activity averages were excluded from the PCA analysis.

[27]:

```python
# Remove households with missing values
X_complete = X.dropna()

print("Households used in PCA after removing missing values:", X_complete.
  ↪shape[0])
```

```
Households used in PCA after removing missing values: 336
```

- Households with missing averages (Ones that did not participate in certain activities) will be excluded for the PCA analysis.
- It is because PCA needs complete observations.

[28]:

```python
scaler = StandardScaler()
X_std = scaler.fit_transform(X_complete)

print("Standardized data shape:", X_std.shape)
print("\nMissing values in standardized data:", np.isnan(X_std).sum())
```

```
Standardized data shape: (336, 4)
```

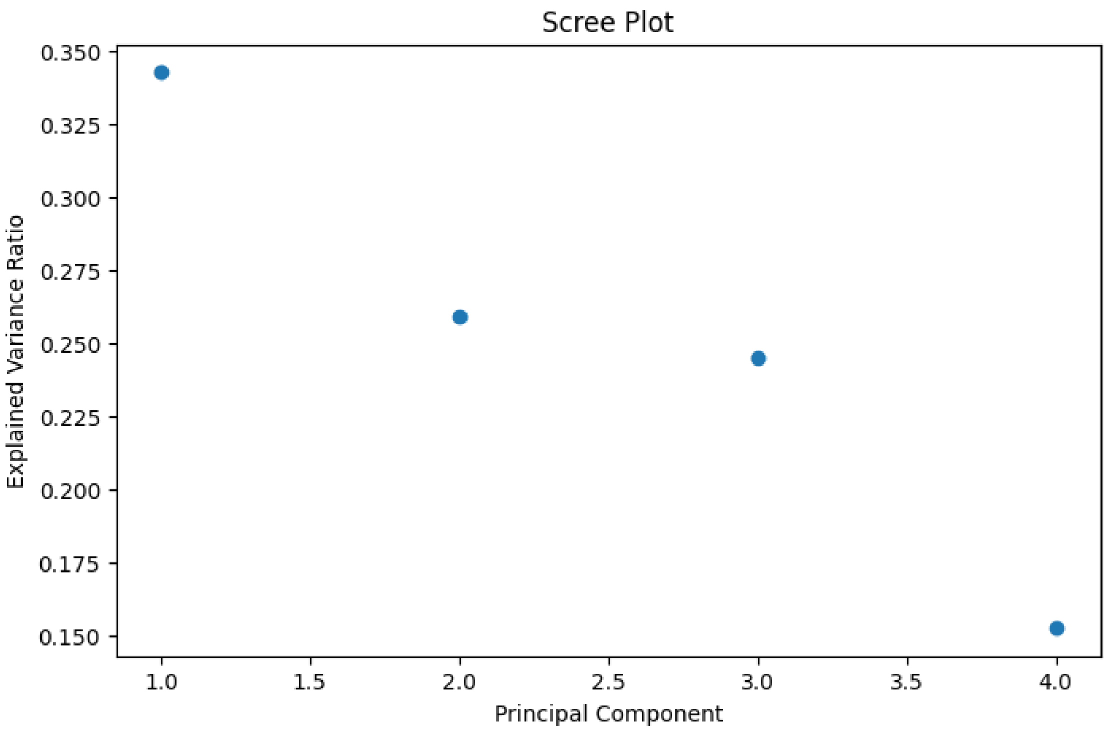
```
Missing values in standardized data: 0
```

- Activity variables are standardized to zero mean and unit variance (Large scales like sleep will not dominate)

[29]:

```
# PCA
pca = PCA()
X_pca = pca.fit_transform(X_std)

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
cum_var = explained_variance.cumsum()

for i, (ev, cv) in enumerate(zip(explained_variance, cum_var), start=1):
    print(f"PC{i}: Explained variance = {ev:.3f}, Cumulative = {cv:.3f}")

# Scree plot
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, 'o')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Scree Plot')
plt.show()
```

```
PC1: Explained variance = 0.343, Cumulative = 0.343
PC2: Explained variance = 0.259, Cumulative = 0.602
PC3: Explained variance = 0.245, Cumulative = 0.848
PC4: Explained variance = 0.152, Cumulative = 1.000
```



The varience ratios show how much of the total variation is daily acrtivity patterns are captured by each principal component. * PC1 shows about 34.4% of the total variance. * PC2 shows about 25.9% of the total vairance. * They bring the cumulative variance to 60% and then PC3 and PC4 have less information. Therefore, the main focus will be on PC1 and PC2 since they show an elbow at PC2.

[30]:
```
# PCA loadings
loadings = pd.DataFrame(
    pca.components_.T,
    index=activity_cols,
    columns=[f'PC{i}' for i in range(1, len(activity_cols) + 1)]
)

print("\nPCA Loadings:")
print(loadings)
```

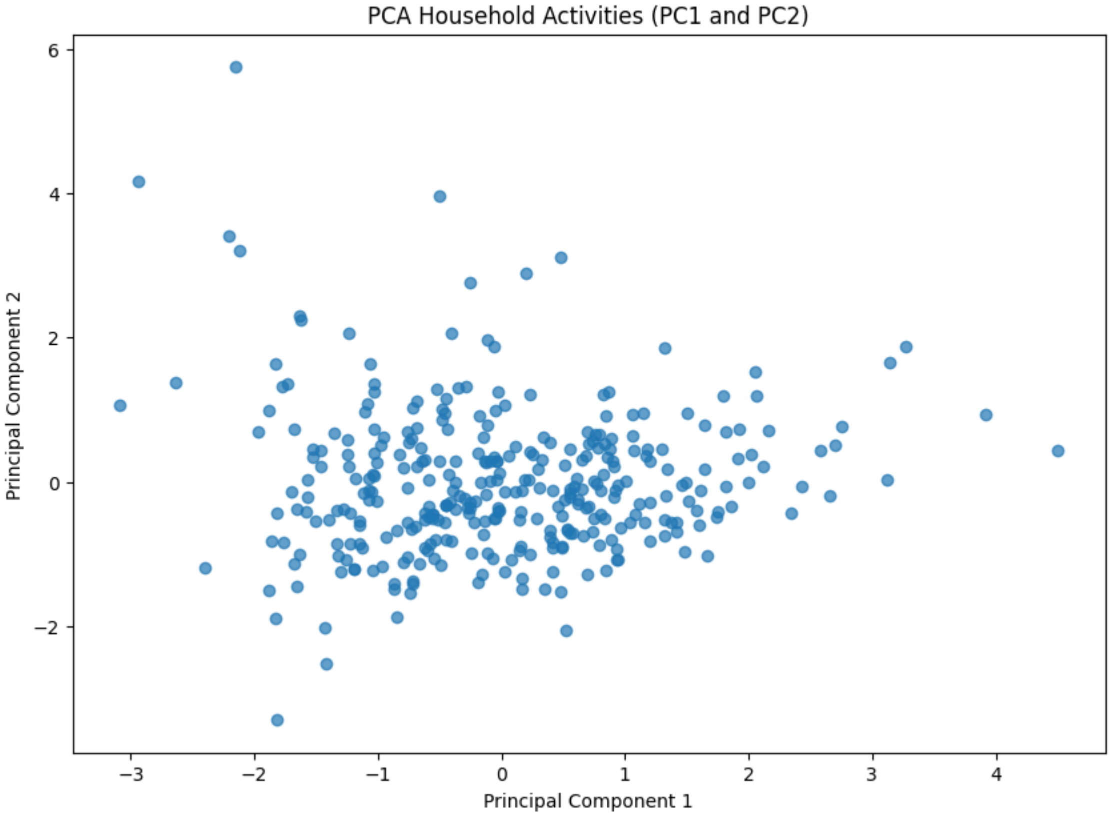
```
PCA Loadings:
         PC1       PC2       PC3       PC4
A1  0.710639  0.012470 -0.122302  0.692733
A2 -0.533103 -0.637129 -0.010121  0.556565
A3 -0.409898  0.737619  0.281346  0.456888
A4  0.206824 -0.223225  0.951727 -0.040125
```

- PC1 (Work–Rest Dimension):

    - Positive loading for working (A1) and negative loadings for sleeping (A2) and reading (A3).
    - Indicates a trade-off between working time and rest/leisure activities.
- PC2 (Leisure vs Sleep Dimension):
    - Strong positive loading for reading (A3) and negative loading for sleeping (A2).
    - Separates leisure-oriented households from those with longer sleep durations.
- PC3 (Occasional Activities):
    - Dominated by dining out (A4), reflecting discretionary and non-daily behavior.
- PC4:
    - Explains little additional variance and lacks a clear behavioral interpretation.

[31]:

```
# PCA with first two principal components
pca_2 = PCA(n_components=2)
X_pca_2 = pca_2.fit_transform(X_std)

# Scatter plot of the first two principal components
plt.figure(figsize=(10, 7))
plt.scatter(X_pca_2[:, 0], X_pca_2[:, 1], alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Household Activities (PC1 and PC2)')
plt.show()
```



The PCA plot shows a continuous spread of households with no clear group separation, indicating gradual variation in activity patterns. Thus, PCA mainly serves as a dimensionality-reduction and visualization tool rather than a clustering method on its own.

1.8 Conclusion

This analysis examined daily time-use patterns of individuals and households in Finland using survey data and a combination of descriptive, inferential, and multivariate methods.

Because no clear separation is visible in the PCA space and no clustering objective is defined in the research questions, no clustering algorithm was applied.

Sleeping accounts for the largest share of daily time, followed by working, while reading and dining out occur less frequently and are often zero. Average time estimates are stable and precisely estimated for core activities.

Differences between groups are limited. Dining out is the only activity that differs significantly between weekdays and weekends, with higher values on weekends. By living environment, only sleeping time shows a significant difference, mainly between city and municipality households. Other activities remain largely consistent across groups.

Library visits differ by living environment but not by day of the week, indicating location-related effects rather than weekly routines.

Correlations between activities are generally weak, suggesting that activities represent distinct aspects of daily time use. PCA supports this by showing gradual variation without clear clustering. Because no clear separation is visible in the PCA space and no clustering objective is defined in the research questions, no clustering algorithm was applied.

Overall, Finnish daily routines are highly stable, with only modest variation in discretionary activities depending on context.