I. Project Description Section

1. Student-Prepared Task Description (Objectives)

1.1 A clear definition of the chosen problem:

(Population data taken for problem formulation:
https://www.worldometers.info/world-population/lithuania-population/)
(Accidents info: from processing official datasets, described below)
Problem: High level of traffic accidents (resolving in death/injury) (about 2080 accidents each year since 2013. Current population of Lithuania is 2,818,518, so AT LEAST 0.074% of people suffer from it (Each incident can have several dead/injured participants – driver+passengers from two or more vehicles)

1.2. Identification and description of datasets used (with links and schema).

Data sources:
“Eismo įvykių, įvykusių Lietuvos Respublikoje, kurių metu žuvo ir (ar) buvo sužeisti žmonės, duomenys” (“Data on traffic accidents in the Republic of Lithuania resulting in fatalities and/or injuries”) (Licensing: CC BY 4.0)
https://data.gov.lt/datasets/509/

- Database not accessible by provided API; published structure is 28 columns wide; still able to download json data files
- *Contains broad information on the accident, such as surface type, speed limit, vehicle information, category of accident and more. Used as a main source of information.*

“Gyventojų skaičius pagal savivaldybes” (Population by municipality) (pdf/xlsx, converted manually)
https://www.registrucentras.lt/lt/atviri-duomenys-ir-statistika/gyventoju-skaicius-pagal-apskritis

- Does not provide a structure separately, but all columns in this Excel document are named.
- *Contains population at certain districts, with details such as population by gender and by age. Used as a reference.*
- *Also, it states that current population of Lithuania is 3,054,549. I did not find that information at the stage of problem definition, but it does not impact defined problem and my work.*

3. Steps of data cleaning and processing.

- Downloaded json files, converted them to csv by using pandas.DataFrame import and export functions
- Created translation table and renamed columns names using widely-used CLI tool sed (GNU implementation) (update: modified translation table, so categories are also renamed):

```
sed -i -f formatTranslation.sed data2.csv
```

- I fixed entry with most likely incorrect timestamp “1899-12-31 20:44:00” (registered in system at 2022-03-30) so I deleted such rows entirely (for reproducibility):

```
max_date = '1950-01-01'
df.drop(df.query(f"dateTime < @max_date").index, inplace=True)
```

- Converted population data to csv manually (copied values), as I was not be able to find library that easily allowed to extract data from .xlsx format

2. Project Architecture and Structure

2.1 Programming language justification

- Provides an easy way of installing necessary libraries, such as for plotting or for editing large volumes of data (>280,000 lines long csv file)
- Ease of integration – works without the server, unlike PHP, and could be easily called from bash script, and I used bash for filtering through the dataset

2.2 Program structure

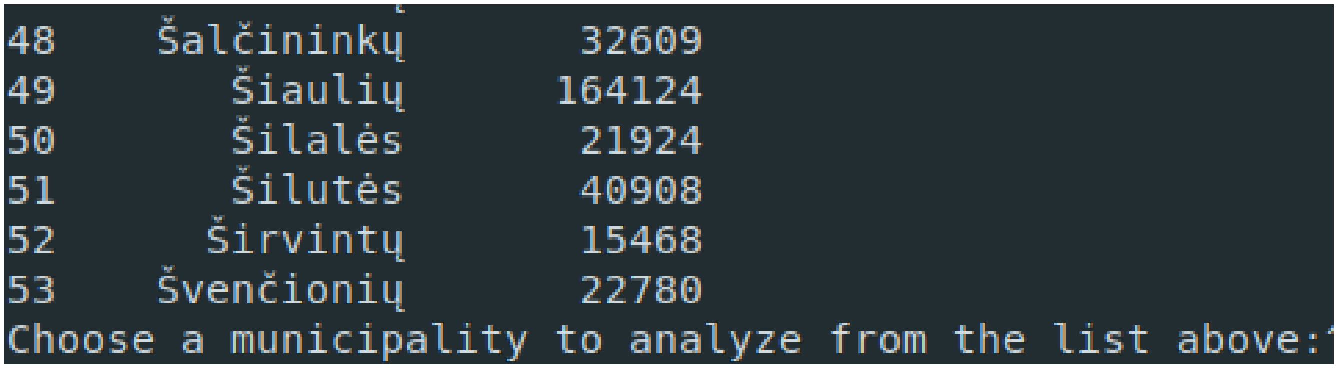
- Project consists of 5 files: project_setup.sh (downloads main ~800MB dataset, sets up the virtual environment, downloads libraries and runs "formatTranslation.sed" as a sed script and "convert_data.py"), formatTranslation.sed(file with direct translations), convert_data.py(convert .json to .csv for the ease of translation), dataviz2.py(all project actual logic and data visualization) and municipality_population.csv (973 bytes helping dataset, required manual processing)

2.3 Algorithm diagram

- Start → Download all json data with curl over https → Convert to CSV → Translate with SED script → Load CSV → Remove unnecessary characters(extra spaces and quotation marks) from category names → Fix dates → Capitalize all category names (as some of them are in caps) → Select municipalities that have data on both collisions and population at the same time → Remove rows that contain other municipalities → Calculate and print general metrics → Get user input → Calculate and output metrics based on user input → Exit

3. Result Examples

3.1 Examples of program's execution output:



```
48    Šalčininkų      32609
49       Šiaulių     164124
50       Šilalės      21924
51       Šilutės      40908
52      Širvintų      15468
53    Švenčionių      22780
Choose a municipality to analyze from the list above:
```

Figure 1: User interaction fragment

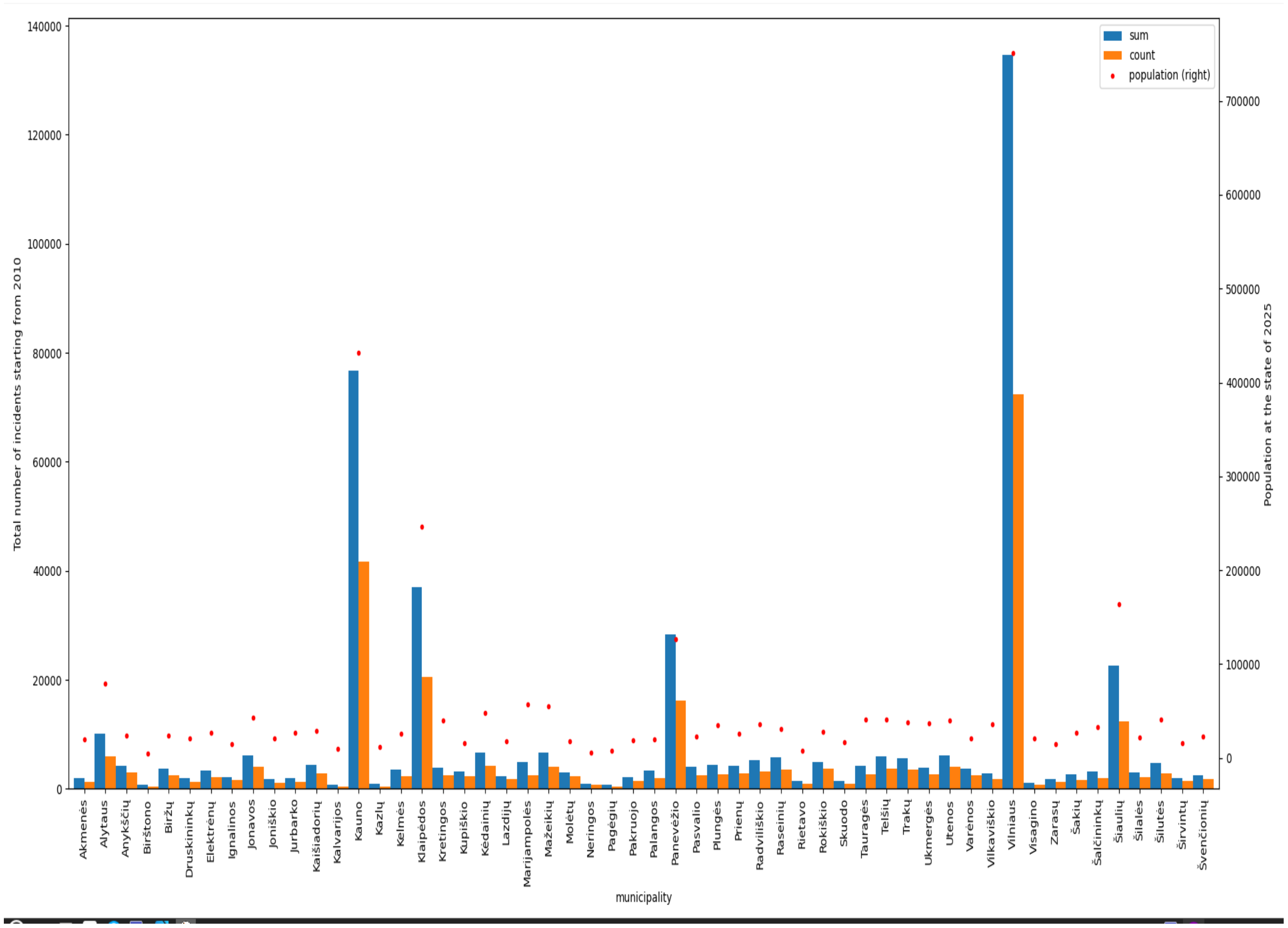


Figure 2: One of the general metrics: sum of people involved in accident; count of accidents, and population, all sorted by municipalities

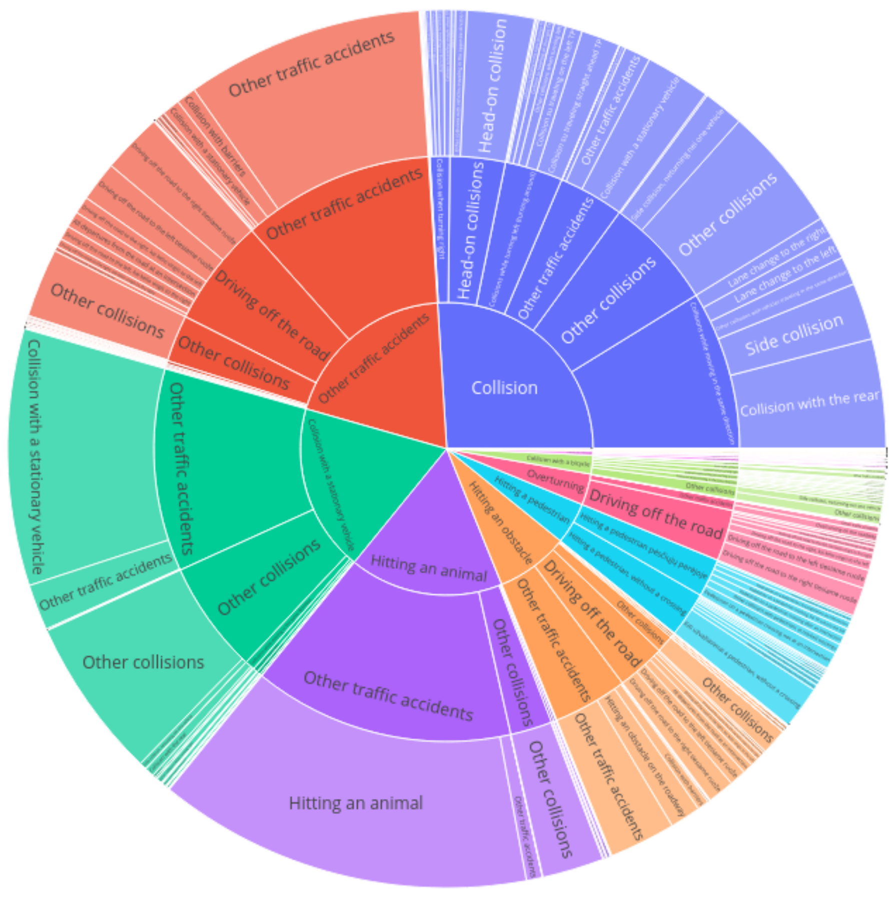


Figure 3: One of the sun charts

3.2 Explanation of results

- Calculated and displayed number of incidents and number of participants involved, distributed by savivaldybe (municipality) where incident occurred. It allows to see obvious data correlation, but at the same time, it turned out that although there are more accidents in big cities, there are less accident per person living there, meaning that they are actually safer.
- During the work on project, I had complications with matplotlib, as It stopped being able to draw graphs. I solved this by changing it to Qt5Agg rendering backend, which required installation of respectively named library. This behaviour was most likely caused by updating other packages and partially incorrect system shutdown.
- The data produced by determining the most widespread incident categories showed could be used to adjust road regulations per specific regions, as different regions prevail in different types of road traffic accidents(For example, increased collision with wild animals, that may require lowering the speed limit in certain areas). However, this topic might need deeper investigation, as it would definitely be useful to monitor efficiency of those policies by time. Stacked area chart would suit the most for this.

II. Program Code (Appendix)

```
import pandas as pd
import dask.dataframe as dd
import matplotlib.pyplot as plt
# import webbrowser

# matplotlib randomly decided to not build for GUI using pywebbrowser lib,
# so I had to change a rendering backend
import matplotlib
matplotlib.use("Qt5Agg")

import plotly.express as px



df = pd.read_csv("data2.csv")
df_mun = pd.read_csv("municipality_population.csv")

df["dateTime"] = pd.to_datetime(df["dateTime"])
df["registrationDate"] = pd.to_datetime(df["registrationDate"])
df["lastEditTime"] = pd.to_datetime(df["lastEditTime"])
df["category"] = df["category"].str.strip(' "')
df["category1"] = df["category1"].str.strip(' "')
df["category2"] = df["category2"].str.strip(' "')
```

```
MAX_DATE = "1950-01-01"
df.drop(df.query(f"dateTime < @MAX_DATE").index, inplace=True)
# print(df['municipality'].str.split(n=1).str[0])


# PREP
df["municipality"] = df["municipality"].str.split(n=1).str[0].str.capitalize()
df_mun["municipality"] = (
    df_mun["municipality"].str.split(n=1).str[0].str.capitalize()
)
df_mun = (
    df_mun.groupby("municipality")["population"].agg(["sum"]).reset_index()
)
df_mun = df_mun.rename(columns={"sum": "population"})
df_plot1 = (
    df.groupby("municipality")["numberOfParticipants"]
    .agg(["sum", "count"])
    .reset_index()
)

# excluding outer join on keys
# print(pd.concat([df_plot1['municipality'],
    # df_mun['municipality']],
    # ignore_index=True).drop_duplicates(keep=False))

# Checking if municipality has data on collisions and has data on population at the same time
CK_DF = df_plot1.merge(df_mun, how="inner", on="municipality")
common_keys = CK_DF["municipality"]
# print(common_keys)



# GLOBAL STATS
# Incidents per municipality
df = df.loc[df["municipality"].isin(common_keys)].reset_index(drop=True)
df_mun = df_mun.loc[df_mun["municipality"].isin(common_keys)].reset_index(
    drop=True
)
df_plot1 = df_plot1.loc[
    df_plot1["municipality"].isin(common_keys)
].reset_index(drop=True)

ax = df_plot1.plot(
    x="municipality",
    y=["sum", "count"],
    kind="bar",
    style=".-",
    rot=90,
    width=1,
)
ax.set_ylabel("Total number of incidents starting from 2010")
# population
ax = df_mun.plot(
    ax=ax,
    secondary_y=True,
    x="municipality",
    y=["population"],
    # kind="scatter",
    kind="line",
    # style=".-",
    style="r.",
    rot=90,
)
ax.set_ylabel("Population at the state of 2025")
plt.show()
plt.close()

# Incident type
sun1_df = df.loc[df["category"].str.len() > 0]
fig = px.sunburst(
    sun1_df,
    path=["category", "category1", "category2"],
    title="Reported incident category distribution:",
)
fig.show()
# fig.write_image("fig1.png")



# USER_INPUT
print("\n\n\n\n\n")
print(df_mun)
municipality_number = int(input("Choose a municipality to analyze from the list above:"))
user_municipality = common_keys.to_list()[municipality_number]
print(user_municipality)



# REQUESTED STATS
# Incidents per municipality
df_plot2 = df.loc[df["municipality"] == user_municipality]
df_2 = (
    df_plot2.set_index("dateTime")["numberOfParticipants"]
    # .resample("D")
    .resample("W")
    .agg(["sum", "count"])
)
```

```
# df_=df_1.merge(df_2,suffixes=('_D','_M'))

tmp_df = df_2.rolling(4, center=True).mean()
ax = tmp_df.plot(
    # x='dateTime'
    y=["sum", "count"],
    grid=True,
    kind="area",
    stacked=False,
    # ,rot=90
    title="Incidents per week at "
    + user_municipality
    + " municipality, data smoothed out across 4-week period",
)
ax.set_xlabel("Week #")
ax.set_ylabel("Incidents")
# df_= df_.reset_index()
# df_.rolling(30, center=True).mean(numeric_only=True).plot(
    # x="dateTime",
    # y=["sum", "count"],
    # kind="line",
    # style=".-", rot=90
# )

plt.show()
plt.close()
# print(df_plot2)


# Incident type
sun2_df = df_plot2.loc[df_plot2["category"].str.len() > 0]
fig = px.sunburst(
    sun2_df,
    path=["category", "category1", "category2"],
    title="Reported incident category distribution at "
    + user_municipality
    + " municipality:",
)
fig.show()

# Incidents per year
# # plot sum(numberOfParticipants) and count() over municipality
# # reindex and count occurences per "D"(1 day) period
# df_ = df.set_index('dateTime')['registrationCode'].resample('D').count()
# df_.plot(alpha=.5)
# # df_.rolling(30, center=True).mean().plot(ax=ax, grid=True, ylabel="Incidents")
# df_.rolling(30, center=True).mean().plot(grid=True, ylabel="Incidents")
# # plt.set_ylabel("Incidents")
# plt.show(); plt.close()
```