Use this code to upload requirements.txt.

```
from google.colab import files

uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving mc1-reports-data-cleaned.csv to mc1-reports-data-cleaned.csv
```

```
!pip install --upgrade pip
!pip install --upgrade -r requirements.txt
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.10/dist-
packages (24.0)
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from -r requirements.txt
(line 1)) (3.7.1)
Collecting matplotlib (from -r requirements.txt (line 1))
  Downloading matplotlib-3.8.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.8 kB)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.10/dist-packages (from -r requirements.txt
(line 2)) (0.13.1)
Collecting seaborn (from -r requirements.txt (line 2))
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (from -r requirements.txt
(line 3)) (2.0.3)
Collecting pandas (from -r requirements.txt (line 3))
  Downloading pandas-2.2.2-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (19 kB)
Requirement already satisfied: altair in
/usr/local/lib/python3.10/dist-packages (from -r requirements.txt
(line 4)) (4.2.2)
Collecting altair (from -r requirements.txt (line 4))
  Downloading altair-5.3.0-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from -r requirements.txt
(line 5)) (1.25.2)
Collecting numpy (from -r requirements.txt (line 5))
  Downloading numpy-1.26.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 2.4 MB/s eta
```

```
0:00:00
ent already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from -r requirements.txt (line 6)) (3.3)
Requirement already satisfied: plotly in
/usr/local/lib/python3.10/dist-packages (from -r requirements.txt
(line 7)) (5.15.0)
Collecting plotly (from -r requirements.txt (line 7))
  Downloading plotly-5.22.0-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: ipywidgets in
/usr/local/lib/python3.10/dist-packages (from -r requirements.txt
(line 8)) (7.7.1)
Collecting ipywidgets (from -r requirements.txt (line 8))
  Downloading ipywidgets-8.1.2-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (24.0)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
requirements.txt (line 1)) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->-r
requirements.txt (line 3)) (2023.4)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas->-r
requirements.txt (line 3)) (2024.1)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from altair->-r
requirements.txt (line 4)) (3.1.3)
Requirement already satisfied: jsonschema>=3.0 in
/usr/local/lib/python3.10/dist-packages (from altair->-r
requirements.txt (line 4)) (4.19.2)
```

```
Requirement already satisfied: toolz in
/usr/local/lib/python3.10/dist-packages (from altair->-r
requirements.txt (line 4)) (0.12.1)
Requirement already satisfied: typing-extensions>=4.0.1 in
/usr/local/lib/python3.10/dist-packages (from altair->-r
requirements.txt (line 4)) (4.11.0)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from plotly->-r
requirements.txt (line 7)) (8.2.3)
Collecting comm>=0.1.3 (from ipywidgets->-r requirements.txt (line 8))
  Downloading comm-0.2.2-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: ipython>=6.1.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets->-r
requirements.txt (line 8)) (7.34.0)
Requirement already satisfied: traitlets>=4.3.1 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets->-r
requirements.txt (line 8)) (5.7.1)
Collecting widgetsnbextension~=4.0.10 (from ipywidgets->-r
requirements.txt (line 8))
  Downloading widgetsnbextension-4.0.10-py3-none-any.whl.metadata (1.6
kB)
Requirement already satisfied: jupyterlab-widgets~=3.0.10 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets->-r
requirements.txt (line 8)) (3.0.10)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (67.7.2)
Requirement already satisfied: jedi>=0.16 in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (0.19.1)
Requirement already satisfied: decorator in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (4.4.2)
Requirement already satisfied: pickleshare in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from
ipython>=6.1.0->ipywidgets->-r requirements.txt (line 8)) (3.0.43)
Requirement already satisfied: pygments in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (2.16.1)
Requirement already satisfied: backcall in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (0.1.7)
Requirement already satisfied: pexpect>4.3 in
```

```
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets->-r requirements.txt (line 8)) (4.9.0)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair-
>-r requirements.txt (line 4)) (23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair-
>-r requirements.txt (line 4)) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair-
>-r requirements.txt (line 4)) (0.35.0)
Requirement already satisfied: rpds-py>=0.7.1 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair-
>-r requirements.txt (line 4)) (0.18.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib->-r requirements.txt (line 1)) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->altair->-r
requirements.txt (line 4)) (2.1.5)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from jedi>=0.16-
>ipython>=6.1.0->ipywidgets->-r requirements.txt (line 8)) (0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.10/dist-packages (from pexpect>4.3-
>ipython>=6.1.0->ipywidgets->-r requirements.txt (line 8)) (0.7.0)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0->ipython>=6.1.0->ipywidgets->-r requirements.txt
(line 8)) (0.2.13)
Downloading matplotlib-3.8.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 39.0 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 294.9/294.9 kB 16.4 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.0/13.0 MB 43.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 857.8/857.8 kB 33.0 MB/s eta
0:00:00
py-1.26.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(18.2 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.2/18.2 MB 36.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 37.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.4/139.4 kB 8.6 MB/s eta
0:00:00
```

```
m-0.2.2-py3-none-any.whl (7.2 kB)
Downloading widgetsnbextension-4.0.10-py3-none-any.whl (2.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 51.1 MB/s eta
0:00:00
py, comm, pandas, matplotlib, ipywidgets, seaborn, altair
  Attempting uninstall: widgetsnbextension
    Found existing installation: widgetsnbextension 3.6.6
    Uninstalling widgetsnbextension-3.6.6:
      Successfully uninstalled widgetsnbextension-3.6.6
  Attempting uninstall: plotly
    Found existing installation: plotly 5.15.0
    Uninstalling plotly-5.15.0:
      Successfully uninstalled plotly-5.15.0
  Attempting uninstall: numpy
    Found existing installation: numpy 1.25.2
    Uninstalling numpy-1.25.2:
      Successfully uninstalled numpy-1.25.2
  Attempting uninstall: pandas
    Found existing installation: pandas 2.0.3
    Uninstalling pandas-2.0.3:
      Successfully uninstalled pandas-2.0.3
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.7.1
    Uninstalling matplotlib-3.7.1:
      Successfully uninstalled matplotlib-3.7.1
  Attempting uninstall: ipywidgets
    Found existing installation: ipywidgets 7.7.1
    Uninstalling ipywidgets-7.7.1:
      Successfully uninstalled ipywidgets-7.7.1
  Attempting uninstall: seaborn
    Found existing installation: seaborn 0.13.1
    Uninstalling seaborn-0.13.1:
      Successfully uninstalled seaborn-0.13.1
  Attempting uninstall: altair
    Found existing installation: altair 4.2.2
    Uninstalling altair-4.2.2:
      Successfully uninstalled altair-4.2.2
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
google-colab 1.0.0 requires pandas==2.0.3, but you have pandas 2.2.2
which is incompatible.
Successfully installed altair-5.3.0 comm-0.2.2 ipywidgets-8.1.2
matplotlib-3.8.4 numpy-1.26.4 pandas-2.2.2 plotly-5.22.0 seaborn-
0.13.2 widgetsnbextension-4.0.10
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

{"id":"4f6b65c52277430fb44414c49e20cfdf","pip_warning":{"packages":
["_plotly_utils","ipywidgets","matplotlib","mpl_toolkits","plotly"]}}

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import altair as alt
import numpy as np
from altair import Scale
import networkx as nx
import plotly.express as px
import plotly.figure_factory as ff
import plotly.io as pio
pio.renderers.default = 'colab'
from ipywidgets import interact, Dropdown
alt.data_transformers.disable_max_rows()
```

```
DataTransformerRegistry.enable('default')
```

Use code below to import dataset

```
from google.colab import files

uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

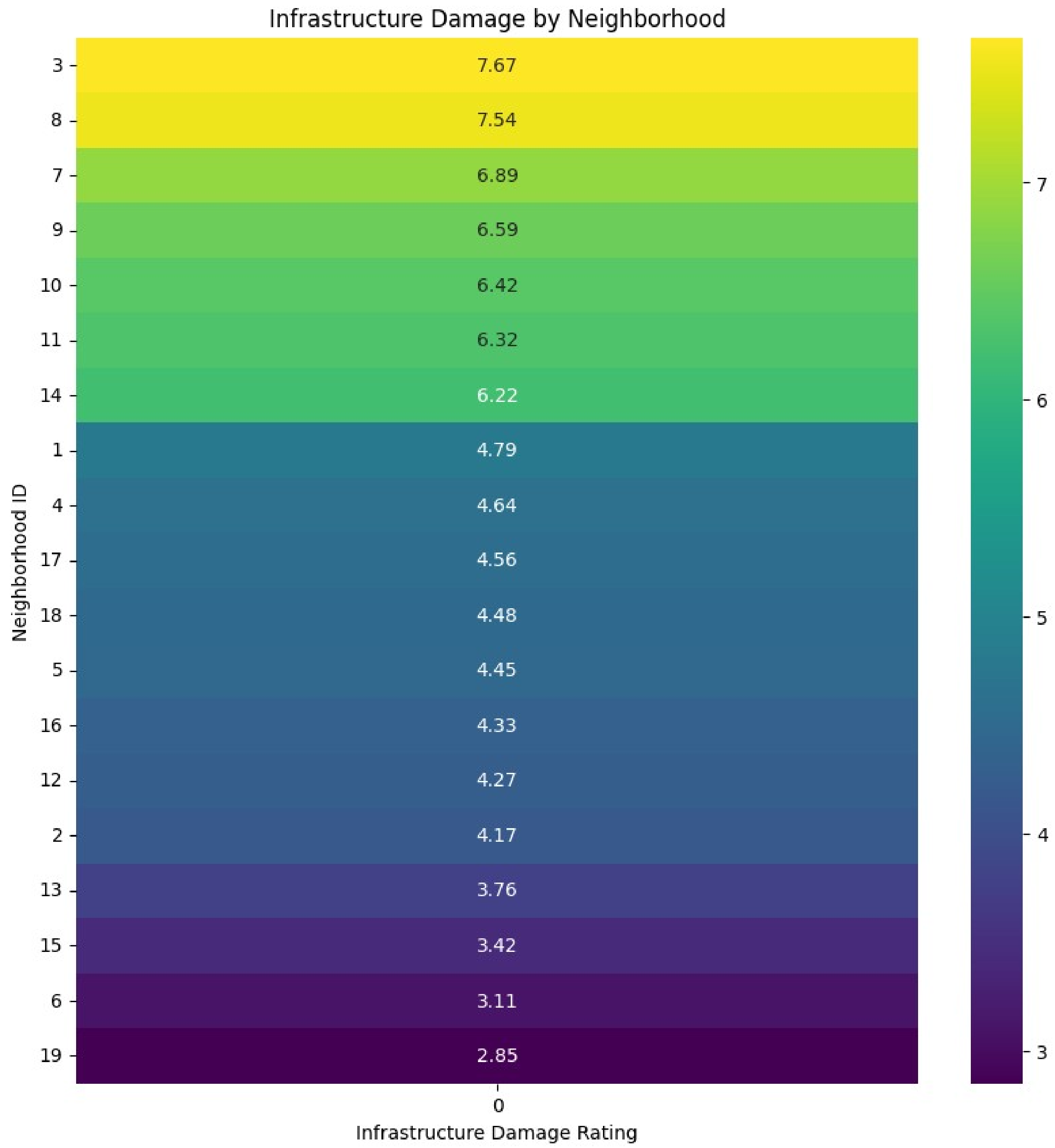
```
Saving requirements.txt to requirements (1).txt
```

```
data = pd.read_csv('mc1-reports-data-cleaned.csv')
data['infrastructure_damage'] = data[['sewer_and_water', 'power',
'roads_and_bridges']].mean(axis=1)
damage_by_neighborhood = data.groupby('location')
['infrastructure_damage'].mean().reset_index()
damage_by_neighborhood_sorted =
damage_by_neighborhood.sort_values(by='infrastructure_damage',
ascending=False)
```

Question 1: Emergency responders will base their initial response on the earthquake shake map. Use visual analytics to determine how their response should change based on damage reports from citizens on the ground. How would you prioritize neighborhoods for response? Which parts of the city are hardest hit? Limit your response to 1000 words and 10 images.

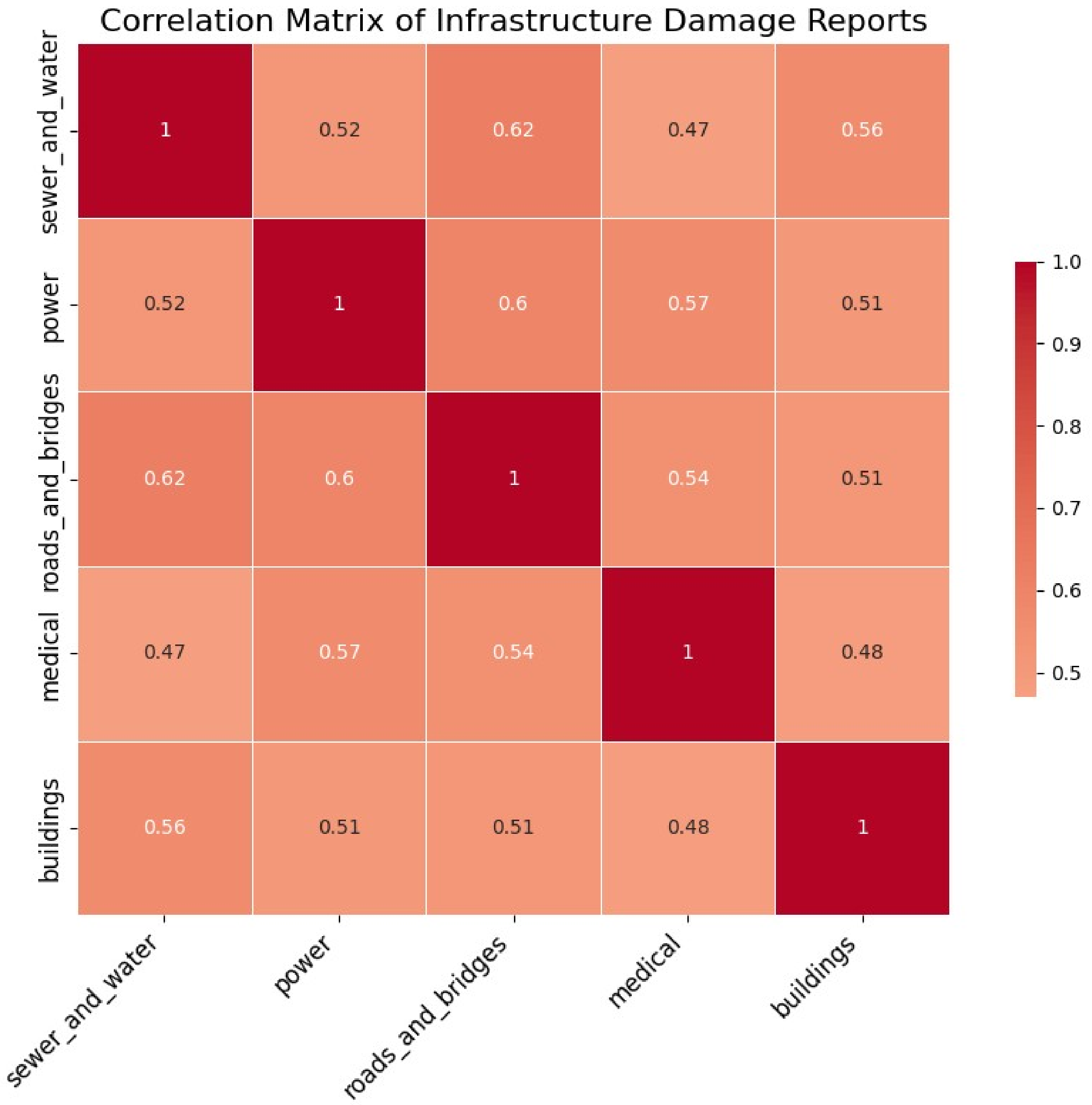
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# This matrix will have a single value per row, representing the
```

```
infrastructure damage
data_matrix =
damage_by_neighborhood_sorted[['infrastructure_damage']].values

# Creating a figure and axis for the plot
plt.figure(figsize=(10, 10))

# Creating a heatmaps
sns.heatmap(data_matrix, annot=True, fmt=".2f", cmap="viridis",
yticklabels=damage_by_neighborhood_sorted['location'].astype(str))

# Adding labels and title for clarity
plt.title("Infrastructure Damage by Neighborhood")
plt.ylabel("Neighborhood ID")
plt.xlabel("Infrastructure Damage Rating")

# Adjusting the y-axis for better visibility
plt.yticks(rotation=0)

plt.show()
```



```python
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming `data` includes the relevant columns
correlation_matrix = data[['sewer_and_water', 'power',
'roads_and_bridges', 'medical', 'buildings']].corr()

# Plotting using seaborn for a refined implementation
plt.figure(figsize=(10, 8))
```

```
# Create a heatmap with additional options for better visualization
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,
            linewidths=.5, cbar_kws={"shrink": .5}, square=True)

# Enhancements for readability
plt.title('Correlation Matrix of Infrastructure Damage Reports',
fontsize=16)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)

# Tight layout often provides a better visual arrangement
plt.tight_layout()

plt.show()
```

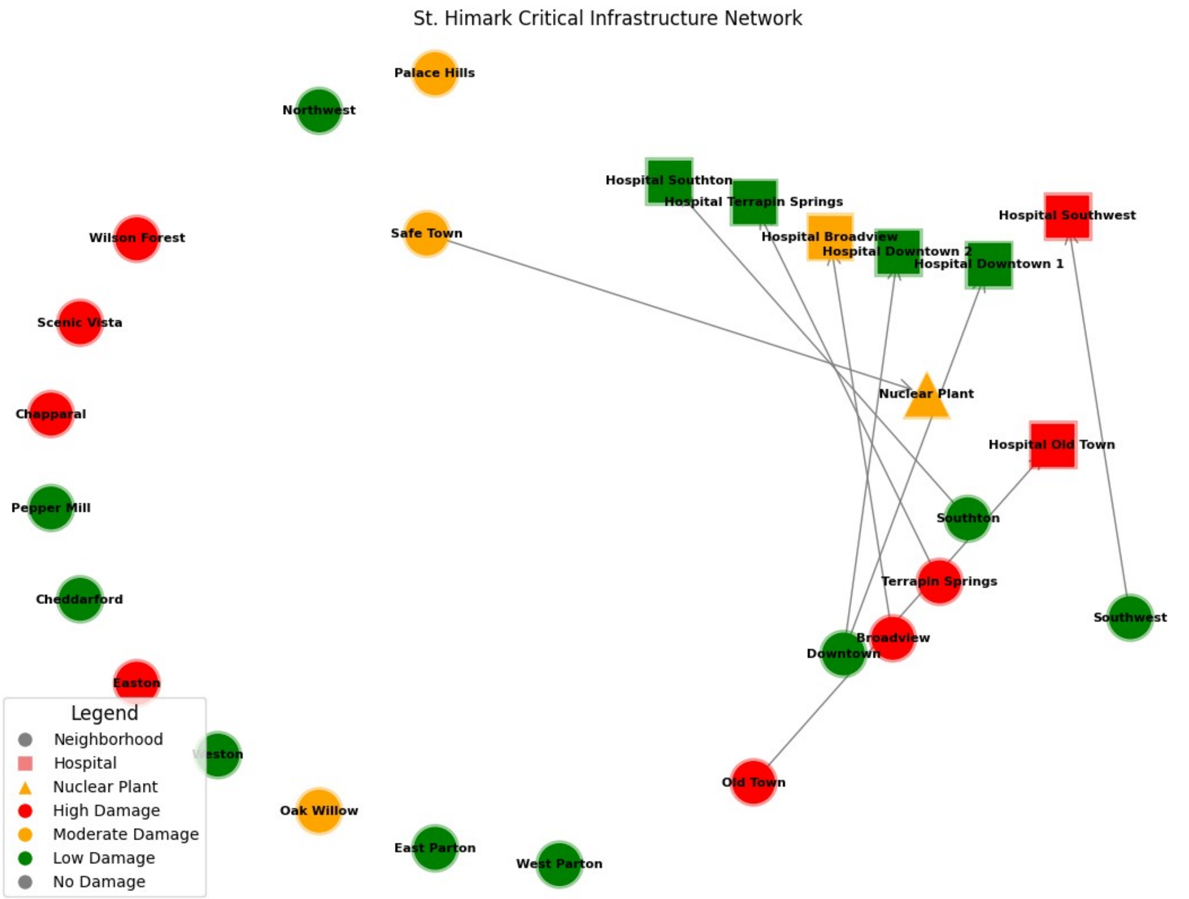


```
data['location'] = pd.to_numeric(data['location'],
errors='coerce').astype(str)

# Create a sorted list of unique location values as strings for the
dropdown
# Convert NaNs to a placeholder value and then to string
data['location'] = pd.to_numeric(data['location'],
errors='coerce').fillna(-1).astype(int).astype(str)
data['location'] = data['location'].replace('-1', 'Unknown')  #
Replace the placeholder with a descriptive string

# Now you can safely sort the locations as strings including the
```

```
'Unknown'
location_options = sorted(data['location'].unique())

location_options = sorted(data['location'].unique(), key=lambda x:
int(x))

# Define the dropdown selection for locations
input_dropdown = alt.binding_select(options=location_options,
name='Location ')
location_selection = alt.selection_single(fields=['location'],
bind=input_dropdown, clear=False, name='Selector')

# Construct the interactive bar chart
bar_chart = alt.Chart(data).transform_fold(
    fold=['sewer_and_water', 'power', 'roads_and_bridges', 'medical',
'buildings'],
    as_=['Infrastructure Type', 'Severity']
).mark_bar().encode(
    x=alt.X('average(Severity):Q', title='Average Severity'),
    y=alt.Y('Infrastructure Type:N', title='Infrastructure Type'),
    color='Infrastructure Type:N',
    tooltip=['Infrastructure Type:N',
alt.Tooltip('average(Severity):Q', title='Average Severity'),
'location:N']
).properties(
    title='Comparative Damage Severity Distribution by Location'
).add_selection(
    location_selection
).transform_filter(
    location_selection
)

bar_chart

alt.Chart(...)

import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

# Initialize the graph
G = nx.DiGraph()

# Adding critical infrastructure nodes with attributes
critical_infrastructures = [
    ('Nuclear Plant', {'type': 'nuclear', 'neighborhood': 'Safe Town',
'damage_level': 'Moderate'}),
    ('Hospital Old Town', {'type': 'hospital', 'neighborhood': 'Old
Town', 'damage_level': 'High'}),
    ('Hospital Southwest', {'type': 'hospital', 'neighborhood':
```

```
'Southwest', 'damage_level': 'High'}),
    ('Hospital Downtown 1', {'type': 'hospital', 'neighborhood':
'Downtown', 'damage_level': 'Low'}),
    ('Hospital Downtown 2', {'type': 'hospital', 'neighborhood':
'Downtown', 'damage_level': 'Low'}),
    ('Hospital Broadview', {'type': 'hospital', 'neighborhood':
'Broadview', 'damage_level': 'Moderate'}),
    ('Hospital Terrapin Springs', {'type': 'hospital', 'neighborhood':
'Terrapin Springs', 'damage_level': 'Low'}),
    ('Hospital Southton', {'type': 'hospital', 'neighborhood':
'Southton', 'damage_level': 'Low'}),
    # ... [Add the rest of your infrastructure nodes with damage_level
if known]
]
G.add_nodes_from(critical_infrastructures)

# Adding neighborhood nodes
neighborhoods = [
    'Palace Hills', 'Northwest', 'Safe Town', 'Wilson Forest', 'Scenic
Vista',
    'Chapparal', 'Pepper Mill', 'Cheddarford', 'Easton', 'Weston',
    'Oak Willow', 'East Parton', 'West Parton', 'Old Town',
'Downtown',
    'Broadview', 'Terrapin Springs', 'Southton', 'Southwest',
    # ... [Add the rest of your neighborhoods]
]
for neighborhood in neighborhoods:
    G.add_node(neighborhood, type='neighborhood', damage_level='No
Damage')

# Adding edges to represent "contains" relationships
edges = [
    ('Safe Town', 'Nuclear Plant', {'relationship': 'contains'}),
    ('Old Town', 'Hospital Old Town', {'relationship': 'contains'}),
    ('Southwest', 'Hospital Southwest', {'relationship': 'contains'}),
    ('Downtown', 'Hospital Downtown 1', {'relationship': 'contains'}),
    ('Downtown', 'Hospital Downtown 2', {'relationship': 'contains'}),
    ('Broadview', 'Hospital Broadview', {'relationship': 'contains'}),
    ('Terrapin Springs', 'Hospital Terrapin Springs', {'relationship':
'contains'}),
    ('Southton', 'Hospital Southton', {'relationship': 'contains'}),
    # ... [Add the rest of your edges]
]

G.nodes['Nuclear Plant']['damage_level'] = 'Moderate'
G.nodes['Hospital Downtown 1']['damage_level'] = 'Low'
G.nodes['Hospital Southton']['damage_level'] = 'Low'
G.nodes['Hospital Terrapin Springs']['damage_level'] = 'Low'
G.nodes['Hospital Broadview']['damage_level'] = 'Moderate'
G.nodes['Hospital Downtown 2']['damage_level'] = 'Low'
```

```
G.nodes['Hospital Southwest']['damage_level'] = 'High'
G.nodes['Hospital Old Town']['damage_level'] = 'High'
G.nodes['Old Town']['damage_level'] = 'High'
G.nodes['Scenic Vista']['damage_level'] = 'High'
G.nodes['Wilson Forest']['damage_level'] = 'High'
G.nodes['Broadview']['damage_level'] = 'High'
G.nodes['Chapparal']['damage_level'] = 'High'
G.nodes['Terrapin Springs']['damage_level'] = 'High'
G.nodes['Easton']['damage_level'] = 'High'
G.nodes['Palace Hills']['damage_level'] = 'Moderate'
G.nodes['Safe Town']['damage_level'] = 'Moderate'
G.nodes['Oak Willow']['damage_level'] = 'Moderate'
G.nodes['East Parton']['damage_level'] = 'Low'
G.nodes['Southwest']['damage_level'] = 'Low'
G.nodes['Southton']['damage_level'] = 'Low'
G.nodes['Pepper Mill']['damage_level'] = 'Low'
G.nodes['Northwest']['damage_level'] = 'Low'
G.nodes['Cheddarford']['damage_level'] = 'Low'
G.nodes['Weston']['damage_level'] = 'Low'
G.nodes['Downtown']['damage_level'] = 'Low'
G.nodes['West Parton']['damage_level'] = 'Low'
G.add_edges_from(edges)
node_shapes = {'neighborhood': 'o', 'hospital': 's', 'nuclear': '^'}
damage_colors = {'High': 'red', 'Moderate': 'orange', 'Low': 'green',
'No Damage': 'gray'}

pos = nx.kamada_kawai_layout(G)

# Begin plotting
plt.figure(figsize=(14, 10))

# Draw edges
nx.draw_networkx_edges(G, pos, arrows=True, arrowstyle='->',
arrowsize=20, edge_color='gray')

# Draw node highlights for visual effect (halo effect)
highlight_size_factor = 1.3  # Adjust size for the highlight effect
for node_type, shape in node_shapes.items():
    filtered_nodes = [node for node in G.nodes if G.nodes[node]
['type'] == node_type]
    node_color = [damage_colors[G.nodes[node]['damage_level']] for
node in filtered_nodes]
    nx.draw_networkx_nodes(G, pos, nodelist=filtered_nodes,
node_shape=shape,
                           node_color=node_color, node_size=700 *
highlight_size_factor, alpha=0.3)

# Draw nodes with shape and color based on type and damage level
for node_type, shape in node_shapes.items():
    filtered_nodes = [node for node in G.nodes if G.nodes[node]
['type'] == node_type]
    node_color = [damage_colors[G.nodes[node]['damage_level']] for
node in filtered_nodes]
    nx.draw_networkx_nodes(G, pos, nodelist=filtered_nodes,
node_shape=shape,
                           node_color=node_color, node_size=700)
# Draw labels
nx.draw_networkx_labels(G, pos, font_size=8, font_weight='bold')

# Custom legend for node types and damage levels
legend_elements = [
    Line2D([0], [0], marker='o', color='w', label='Neighborhood',
markerfacecolor='gray', markersize=10),
    Line2D([0], [0], marker='s', color='w', label='Hospital',
markerfacecolor='lightcoral', markersize=10),
    Line2D([0], [0], marker='^', color='w', label='Nuclear Plant',
markerfacecolor='orange', markersize=10),
    Line2D([0], [0], marker='o', color='w', label='High Damage',
markerfacecolor='red', markersize=10),
    Line2D([0], [0], marker='o', color='w', label='Moderate Damage',
markerfacecolor='orange', markersize=10),
    Line2D([0], [0], marker='o', color='w', label='Low Damage',
markerfacecolor='green', markersize=10),
    Line2D([0], [0], marker='o', color='w', label='No Damage',
markerfacecolor='gray', markersize=10),
]
plt.legend(handles=legend_elements, loc='best', title="Legend",
title_fontsize='large')

plt.title('St. Himark Critical Infrastructure Network')
plt.axis('off')
plt.show()
```



```
import pandas as pd
import plotly.express as px

# Load the data

# Ensure 'location' is the correct type, assuming it should match
integer keys in the dictionary
data['location'] = data['location'].astype(int)

# Define the numeric columns
numeric_cols = ['sewer_and_water', 'power', 'roads_and_bridges',
'medical', 'buildings', 'shake_intensity']

# Fill NaNs in numeric columns with the mean of each column
data[numeric_cols] =
data[numeric_cols].fillna(data[numeric_cols].mean())

# Aggregate data by 'location'
agg_reports = data.groupby('location')
[numeric_cols].mean().reset_index()

# Neighborhood mapping - Ensure these are integers if your locations
```

```
are integers
neighborhood_names = {
    1: 'Palace Hills', 2: 'Northwest', 3: 'Old Town', 4: 'Safe Town',
5: 'Southwest',
    6: 'Downtown', 7: 'Wilson Forest', 8: 'Scenic Vista', 9:
'Broadview', 10: 'Chapparal',
    11: 'Terrapin Springs', 12: 'Pepper Mill', 13: 'Cheddarford', 14:
'Easton',
    15: 'Weston', 16: 'Southton', 17: 'Oak Willow', 18: 'East Parton',
19: 'West Parton'
}

# Map neighborhood names
agg_reports['neighborhood'] =
agg_reports['location'].map(neighborhood_names)

# Population mapping
population_data = {
    'Palace Hills': 13000, 'Northwest': 15000, 'Old Town': 17000,
'Safe Town': 10000,
    'Southwest': 12000, 'Downtown': 30000, 'Wilson Forest': 8000,
'Scenic Vista': 9000,
    'Broadview': 11000, 'Chapparal': 7000, 'Terrapin Springs': 6000,
'Pepper Mill': 14000,
    'Cheddarford': 5000, 'Easton': 16000, 'Weston': 18000, 'Southton':
15000,
    'Oak Willow': 13000, 'East Parton': 19000, 'West Parton': 20000
}

# Map population, fill any missing data with the average population
average_population = pd.Series(population_data).mean()
agg_reports['population'] =
agg_reports['neighborhood'].map(population_data).fillna(average_popula
tion)

# Compute 'impact' and 'vulnerability'
agg_reports['impact'] = agg_reports['shake_intensity']
agg_reports['vulnerability'] = agg_reports[['sewer_and_water',
'power', 'roads_and_bridges', 'medical', 'buildings']].mean(axis=1)

# Create the scatter plot
fig = px.scatter(agg_reports, x="impact", y="vulnerability",
                 size="population", color="neighborhood",
                 hover_name="neighborhood", size_max=60,
                 title="Impact vs. Vulnerability Across
Neighborhoods")

fig.show()
```

```
neighborhood_names = {
    1: 'Palace Hills', 2: 'Northwest', 3: 'Old Town', 4: 'Safe Town',
    5: 'Southwest', 6: 'Downtown', 7: 'Wilson Forest', 8: 'Scenic
Vista',
    9: 'Broadview', 10: 'Chapparal', 11: 'Terrapin Springs', 12:
'Pepper Mill',
    13: 'Cheddarford', 14: 'Easton', 15: 'Weston', 16: 'Southton',
    17: 'Oak Willow', 18: 'East Parton', 19: 'West Parton'
}

# Mapping location IDs to neighborhood names
data['neighborhood'] = data['location'].map(neighborhood_names)

# Histogram of shake intensity by neighborhood
fig = px.histogram(
    data,
    x="shake_intensity",
    color="neighborhood",
    barmode='group',  # Change to 'group' to place bars side by side
    nbins=15,  # Adjust number of bins for better visibility
    title="Comparative Histogram of Shake Intensity by Neighborhood",
    labels={"shake_intensity": "Shake Intensity"},
    opacity=0.85,  # Increase opacity for less overlap confusion
    height=600,
    category_orders={"neighborhood":
list(neighborhood_names.values())}
)

# Update layout
fig.update_layout(
    xaxis_title="Shake Intensity",
    yaxis_title="Count of Reports",
    legend_title="Neighborhood",
    legend=dict(
        orientation="h",
        yanchor="bottom",
        y=1.02,
        xanchor="right",
        x=1
    ),
    colorway=px.colors.qualitative.Alphabet  # A colorway with
maximized contrast
)

# Display the figure
fig.show()


import pandas as pd
import altair as alt
```

```
# Remove duplicate import of pandas
# data = pd.read_csv('path_to_your_data.csv') # Uncomment and set the
path to your data file

# Ensure the 'location' column is in the correct format and contains
expected values
print("Unique 'location' values before mapping:",
data['location'].unique())

# Convert 'time' column to datetime
data['time'] = pd.to_datetime(data['time'], format='%Y-%m-%d %H:%M:
%S')
data['time'] = data['time'].dt.floor('h')  # Round down to the nearest
hour

# Mapping of location numbers to names
location_names = {  1: 'Palace Hills', 2: 'Northwest', 3: 'Old Town',
4: 'Safe Town',
    5: 'Southwest', 6: 'Downtown', 7: 'Wilson Forest', 8: 'Scenic
Vista',
    9: 'Broadview', 10: 'Chapparal', 11: 'Terrapin Springs', 12:
'Pepper Mill',
    13: 'Cheddarford', 14: 'Easton', 15: 'Weston', 16: 'Southton',
    17: 'Oak Willow', 18: 'East Parton', 19: 'West Parton'
    # Your mapping dictionary
}

# Apply the mapping to the 'location' column
data['location'] = data['location'].map(location_names)

# Check for NaNs after mapping
print("NaNs after mapping:", data['location'].isna().sum())
print("Unique 'location' values after mapping:",
data['location'].unique())

# If the 'location' column still contains NaNs, this indicates a
problem with the mapping
# You should not proceed with the melting and grouping until this is
resolved
# Once you are sure there are no NaNs, proceed with melting the
DataFrame
melted_data = data.melt(
    id_vars=['time', 'location'],
    value_vars=['sewer_and_water', 'power', 'roads_and_bridges',
'medical', 'buildings', 'shake_intensity'],
    var_name='category',
    value_name='value'
)
```

```
# Calculate the average damage value for each category and hour
avg_damage_per_category_hour = melted_data.groupby(['time',
'location', 'category']).mean().reset_index()

# Create a selection element for location
input_dropdown =
alt.binding_select(options=sorted(avg_damage_per_category_hour['locati
on'].unique()))
location_selection = alt.selection_point(fields=['location'],
bind=input_dropdown, name='Select')

# Define the chart
stacked_area_chart =
alt.Chart(avg_damage_per_category_hour).mark_area().encode(
    x='time:T',
    y=alt.Y('value:Q', stack='zero', axis=alt.Axis(title='Average
Damage Value')),
    color=alt.Color('category:N',
legend=alt.Legend(title="Category")),
    tooltip=['time:T', 'location:N', 'category:N', 'value:Q']
).add_selection(
    location_selection
).transform_filter(
    location_selection  # Filter by the location selection
).properties(
    width=800,
    height=500,
    title='Average Damage Value Area Chart'
)

stacked_area_chart.display()
```

```
Unique 'location' values before mapping: [ 1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19]
NaNs after mapping: 0
Unique 'location' values after mapping: ['Palace Hills' 'Northwest'
'Old Town' 'Safe Town' 'Southwest' 'Downtown'
 'Wilson Forest' 'Scenic Vista' 'Broadview' 'Chapparal' 'Terrapin
Springs'
 'Pepper Mill' 'Cheddarford' 'Easton' 'Weston' 'Southton' 'Oak Willow'
 'East Parton' 'West Parton']
```

```
/usr/local/lib/python3.10/dist-packages/altair/utils/
deprecation.py:65: AltairDeprecationWarning:

'add_selection' is deprecated. Use 'add_params' instead.
```

```
alt.Chart(...)
```

```
import altair as alt
import pandas as pd

data['time'] = pd.to_datetime(data['time'], format='%d/%m/%Y %H:%M')

# Round the time to the nearest hour to group by each hour
data['time'] = data['time'].dt.floor('h')

# Replace location numbers with names
location_names = {
    1: 'Palace Hills', 2: 'Northwest', 3: 'Old Town', 4: 'Safe Town',
    5: 'Southwest', 6: 'Downtown', 7: 'Wilson Forest', 8: 'Scenic
Vista',
    9: 'Broadview', 10: 'Chapparal', 11: 'Terrapin Springs', 12:
'Pepper Mill',
    13: 'Cheddarford', 14: 'Easton', 15: 'Weston', 16: 'Southton',
    17: 'Oak Willow', 18: 'East Parton', 19: 'West Parton'
}
data['location'] = data['location'].map(location_names)

# Melt the DataFrame to have category, time, and value
melted_data = data.melt(id_vars=['time', 'location'],
                                value_vars=['sewer_and_water',
'power', 'roads_and_bridges', 'medical', 'buildings',
'shake_intensity'],
                                var_name='category',
value_name='value')

# Calculate the average damage value for each category and hour
avg_damage_per_category_hour = melted_data.groupby(['time',
'location', 'category']).mean().reset_index()

# Create a selection element for location
input_dropdown =
alt.binding_select(options=sorted(avg_damage_per_category_hour['locati
on'].unique()))
location_selection = alt.selection_point(fields=['location'],
bind=input_dropdown, name='Select')

# Create a selection for the legend
legend_selection = alt.selection_point(fields=['category'],
bind='legend')

# Define the stacked bar chart with interactive legend
stacked_chart =
alt.Chart(avg_damage_per_category_hour).mark_bar().encode(
    x='time:T',
    y=alt.Y('value:Q', stack='zero', axis=alt.Axis(title='Average
Damage Value')),
    color=alt.Color('category:N',
legend=alt.Legend(title="Category")),
    opacity=alt.condition(legend_selection, alt.value(1),
alt.value(0.2)),  # Use the legend selection for opacity
    tooltip=['time:T', 'location:N', 'category:N', 'value:Q']
).add_params(
    location_selection,
    legend_selection
).transform_filter(
    location_selection
).properties(
    width=800,
    height=500,
    title='Average Damage Value of Categories'
)
stacked_area_chart.display()
stacked_chart.save('average_damage_bar.html')
```

```
alt.Chart(...)
```