gy78fjxmh

May 2, 2024

[2]:
```
import pandas as pd
import plotly.graph_objects as go
from ipywidgets import interact, Dropdown, SelectionRangeSlider
import datetime

# Load the data from a CSV file
data = pd.read_csv('mc1-reports-data-cleaned.csv')

# Define a mapping from location IDs to neighborhood names for better␣
 ↪readability
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 the 'location' column in the data using the provided neighborhood names
data['location'] = data['location'].map(neighborhood_names)

# Convert the 'time' column to datetime format for time-based operations
data['time'] = pd.to_datetime(data['time'])
```

```
# Create a dropdown widget for selecting a neighborhood to view in the plot
neighborhood_selector = Dropdown(
    options=['All Neighborhoods'] + list(neighborhood_names.values()),
    description='Neighborhood'
)

# Create a dropdown widget for selecting a category of report to view in the␣
 ↪plot
category_selector = Dropdown(
    options=['shake_intensity', 'sewer_and_water', 'power',␣
 ↪'roads_and_bridges', 'medical', 'buildings'],
    description='Category'
)

# Create a range slider for selecting a date range to filter the data
date_range_selector = SelectionRangeSlider(
    options=[(date.strftime('%Y-%m-%d'), date) for date in pd.
 ↪date_range(start=data['time'].min(), end=data['time'].max(), freq='D')],
    index=(0, len(pd.date_range(start=data['time'].min(), end=data['time'].
 ↪max(), freq='D')) - 1),
    description='Date Range',
    orientation='horizontal',
    readout=True
)

# Define a function to plot the boxplot based on selected neighborhood,␣
 ↪category, and date range
def plot_boxplot(neighborhood, category, date_range):
    # Filter the data for the selected neighborhood and date range
    # If 'All Neighborhoods' is selected, do not filter by neighborhood
    if neighborhood != 'All Neighborhoods':
        filtered_data = data[(data['location'] == neighborhood) & (data['time']␣
 ↪>= date_range[0]) & (data['time'] <= date_range[1])].copy()
    else:
        filtered_data = data[(data['time'] >= date_range[0]) & (data['time'] <=␣
 ↪date_range[1])].copy()

    # Drop rows where the location is NaN before processing further
    filtered_data = filtered_data.dropna(subset=['location'])

    # Transform the data to a long format, suitable for boxplotting with Plotly
    melted_data = filtered_data.melt(id_vars=['location', 'time'],␣
 ↪value_vars=[category], var_name='Category', value_name='Value')

    # Prepare statistical information for the boxplot hover text
```

```
    stats = melted_data.groupby('location')['Value'].describe(percentiles=[.25,␣
↪.75])
    stats['IQR'] = stats['75%'] - stats['25%']
    stats.dropna(subset=['count'], inplace=True)  # Drop rows without valid␣
↪count data

    # Initialize an empty figure object
    fig = go.Figure()

    # Iterate over each unique location and add a box trace for each one
    for location in melted_data['location'].unique():
        if pd.isna(location) or location not in stats.index:
            continue  # Skip NaN locations and those without stats

        # Filter the data for the current location
        location_data = melted_data[melted_data['location'] == location]

        # Get the report count and IQR for hover data
        count = stats.at[location, 'count']
        iqr = stats.at[location, 'IQR']
        hover_data = f"report count: {int(count)}, IQR: {iqr:.2f}"

        # Add a box trace to the figure for the current location
        fig.add_trace(go.Box(
            y=location_data['Value'],
            name=location,
            boxpoints='all',  # Show all data points
            jitter=0.5,  # Spread data points to avoid overlap
            whiskerwidth=0.2,  # Set whisker width for the boxplot
            marker_size=2,  # Set marker size for the data points
            line_width=1,  # Set line width for the boxplot
            hoverinfo='y+text',  # Set hover info (y-value and additional text)
            text=[hover_data] * int(count)  # Repeat the hover text for each␣
↪data point
        ))

    # Customize the figure's layout for better clarity and readability
    fig.update_layout(
        title=f'Variability of {category.capitalize()} Reports in␣
↪{neighborhood} from {date_range[0].strftime("%Y-%m-%d")} to {date_range[1].
↪strftime("%Y-%m-%d")}',
        xaxis_title='Neighborhood',
        yaxis_title=f'Reported {category.capitalize()} Level',
        template='plotly_white',
        hovermode='closest'
    )
```

```
    # Display the figure
    fig.show()

# Set up an interactive widget to plot the boxplot based on user input
interact(plot_boxplot, neighborhood=neighborhood_selector,␣
 ↪category=category_selector, date_range=date_range_selector)
```

```
interactive(children=(Dropdown(description='Neighborhood', options=('All␣
 ↪Neighborhoods', 'PALACE HILLS', 'NORT…
```

[2]:
```
<function __main__.plot_boxplot(neighborhood, category, date_range)>
```

[2]:
```
import pandas as pd
import plotly.figure_factory as ff
from ipywidgets import interact, Dropdown

# Load the dataset from a CSV file into a pandas DataFrame.
file_path = 'mc1-reports-data-cleaned.csv'
data = pd.read_csv(file_path)

# Mapping of numeric location IDs to more descriptive neighborhood names.
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"
}
# Replace numeric location IDs in the 'location' column with the corresponding␣
 ↪neighborhood names.
data['location'] = data['location'].map(neighborhood_names)
```

```
# Convert the 'time' column from a string type to datetime type for easier time␣
 ↪series manipulation.
data['time'] = pd.to_datetime(data['time'])

# Create a dropdown widget for selecting a neighborhood to view in the heatmap.
neighborhood_selector = Dropdown(options=list(neighborhood_names.values()),␣
 ↪description='Neighborhood')

def plot_correlation_heatmap(neighborhood):
    # Filter the dataset to include only the data from the selected␣
 ↪neighborhood.
    filtered_data = data[data['location'] == neighborhood]

    # Calculate the correlation matrix for specific report categories within␣
 ↪the filtered dataset.
    correlation_matrix = filtered_data[['shake_intensity', 'sewer_and_water',␣
 ↪'power', 'roads_and_bridges', 'medical', 'buildings']].corr()

    # Labels for the heatmap axes, corresponding to the categories included in␣
 ↪the correlation matrix.
    axis_names = ['Shake Intensity', 'Sewer and Water', 'Power', 'Roads and␣
 ↪Bridges', 'Medical', 'Buildings']

    # Create the heatmap using Plotly's figure factory module.
    fig = ff.create_annotated_heatmap(
        z=correlation_matrix.to_numpy(),  # Correlation values
        x=axis_names,  # Category names for x-axis
        y=axis_names,  # Category names for y-axis
        annotation_text=correlation_matrix.round(2).to_numpy(),  # Text␣
 ↪annotations on the heatmap cells
        colorscale='RdBu',  # Colour scale for the heatmap
        reversescale=True,  # Reverse the colour scale to align dark colours␣
 ↪with high values
        showscale=True  # Show the colour scale bar
    )

    # Update the plot with axis labels and a title.
    fig.update_xaxes(title_text='Reported Category')
    fig.update_yaxes(title_text='Reported Category')
    fig.update_layout(
        title={
            'text': f'Correlation Matrix of Reported Categories in␣
 ↪{neighborhood}',
            'y':0.95,
            'x':0.5,
            'xanchor': 'center',
```

```
            'yanchor': 'top'
        },
        xaxis={'title': 'Reported Categories', 'side': 'bottom'},
        yaxis={'title': 'Reported Categories'},
        margin=dict(l=150, r=50, t=50, b=150)  # Adjust margins to fit labels without clipping
    )

    # Display the heatmap.
    fig.show()

# Attach the function to the dropdown widget to create an interactive visualization.
interact(plot_correlation_heatmap, neighborhood=neighborhood_selector)
```

```
interactive(children=(Dropdown(description='Neighborhood', options=('PALACE HILLS', 'NORTHWEST', 'OLD TOWN', '…
```

[2]:
```
<function __main__.plot_correlation_heatmap(neighborhood)>
```

[1]:
```
import pandas as pd
import plotly.express as px
from ipywidgets import interact, Dropdown

# Load the dataset from a specified file path
file_path = 'mc1-reports-data-cleaned.csv'
data = pd.read_csv(file_path)

# Mapping dictionary to convert numeric location IDs into more descriptive neighborhood names
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"
}

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

# Convert the 'time' column from string to datetime format for easier␣
 ↪time-based operations
data['time'] = pd.to_datetime(data['time'])

# Create a dropdown widget for selecting a neighborhood
neighborhood_selector = Dropdown(options=list(neighborhood_names.values()),␣
 ↪description='Neighborhood')

# Create a dropdown widget for selecting a category of data to analyze
category_selector = Dropdown(options=['shake_intensity', 'sewer_and_water',␣
 ↪'power', 'roads_and_bridges', 'medical', 'buildings'],␣
 ↪description='Category')

# Define a function to generate a histogram based on user selections for␣
 ↪neighborhood and category
def plot_histogram(neighborhood, category):
    # Filter the data to include only entries from the selected neighborhood
    filtered_data = data[data['location'] == neighborhood]

    # Sort the filtered data by the selected category to improve clarity in the␣
 ↪histogram
    filtered_data = filtered_data.sort_values(by=category)

    # Use Plotly Express to generate a histogram of the data, colouring by the␣
 ↪selected category
    fig = px.histogram(
        filtered_data,
        x='time',
        color=category,
        title=f'Report Timing and Frequency for {category.capitalize()} in␣
 ↪{neighborhood}',
        labels={'time': 'Time of Report'},
        category_orders={category: sorted(filtered_data[category].unique())},
        color_discrete_sequence=px.colors.qualitative.G10,
        template='plotly_white',
        barnorm='',
        nbins=24  # Set number of bins to 24 to represent hourly data over a day
    )
```

```
    # Update the layout of the plot to improve readability
    fig.update_layout(
        xaxis_title='Time of Report',
        yaxis_title='Number of Reports',
        xaxis_tickangle=-45,
        bargap=0.1,
        legend_title_text='Category Level'
    )

    # Display the plot
    fig.show()

# Setup the plot to update interactively based on user inputs
interact(plot_histogram, neighborhood=neighborhood_selector,␣
 ↪category=category_selector)

# Additional code to filter data specifically for 'PALACE HILLS' and␣
 ↪'shake_intensity' on a specific date, and print the result
filtered_data = data[
    (data['location'] == 'PALACE HILLS') &
    (data['time'].dt.date == pd.to_datetime('2020-04-06').date())
][['time', 'power']]

# Print the filtered data for inspection
print(filtered_data)
```

```
interactive(children=(Dropdown(description='Neighborhood', options=('PALACE␣
 ↪HILLS', 'NORTHWEST', 'OLD TOWN', '…
```

```
                    time  power
14    2020-04-06 08:30:00    5.0
15    2020-04-06 10:25:00    2.0
16    2020-04-06 02:25:00    8.0
24    2020-04-06 14:10:00   10.0
26    2020-04-06 17:10:00    5.0
…                     …    …
11525 2020-04-06 16:20:00    0.0
11526 2020-04-06 16:15:00    1.0
11527 2020-04-06 16:00:00    0.0
11528 2020-04-06 16:40:00    0.0
11529 2020-04-06 19:45:00    1.0

[105 rows x 2 columns]
```

[7]:
```
import pandas as pd
import plotly.graph_objects as go
```

```
from ipywidgets import interact, Dropdown


# Check and convert the 'location' column
data['location'] = pd.to_numeric(data['location'], errors='coerce').fillna(-1).
 ↪astype(int)
data['location'] = data['location'].map(neighborhood_names).fillna('Unknown')

# Convert the 'time' column to datetime and create 'time_interval'
data['time'] = pd.to_datetime(data['time'])
data['time_interval'] = data['time'].dt.floor('H')  # Grouping by hour

# Prepare a dropdown for selecting neighborhood names, including 'Unknown' if␣
 ↪needed
location_options = ['All Locations'] + list(set(data['location']))
location_selector = Dropdown(options=location_options,␣
 ↪description='Neighborhood')

def plot_data(selected_location):
    # Filter data based on selected neighborhood
    if selected_location != 'All Locations':
        filtered_data = data[data['location'] == selected_location].copy()
    else:
        filtered_data = data.copy()

    # Calculate and plot the missing data proportion for each category
    fig = go.Figure()
    categories = ['sewer_and_water', 'power', 'roads_and_bridges', 'medical',␣
 ↪'buildings', 'shake_intensity']
    for category in categories:
        proportion_missing = filtered_data.groupby('time_interval')[category].
 ↪apply(lambda x: 100 * x.isna().mean())
        fig.add_trace(go.Scatter(
            x=proportion_missing.index,
            y=proportion_missing,
            mode='lines+markers',
            name=f'{category} Missing (%)'
        ))

    fig.update_layout(
        title=f'Missing Data Proportion Over Time for {selected_location}',
        xaxis_title='Time Interval',
        yaxis_title='Percentage Missing (%)',
        legend_title='Data Category'
    )

    fig.show()
```

```
interact(plot_data, selected_location=location_selector)
```

```
C:\Users\kwasi\AppData\Local\Temp\ipykernel_10716\2881081481.py:12:
FutureWarning:

'H' is deprecated and will be removed in a future version, please use 'h'
instead.
```

```
interactive(children=(Dropdown(description='Neighborhood', options=('All␣
 ↪Locations', 'Unknown'), value='All Lo…
```

[7]:
```
<function __main__.plot_data(selected_location)>
```

[3]:
```
import pandas as pd
import plotly.express as px
import ipywidgets as widgets
from IPython.display import display

# Load the dataset from a CSV file
data_path = 'mc1-reports-data-cleaned.csv'
data = pd.read_csv(data_path)

# Convert the 'time' column from string to datetime format to facilitate␣
 ↪time-based analysis
data['time'] = pd.to_datetime(data['time'])

# Create a new column for time intervals, rounding down the time to the nearest␣
 ↪hour to group data by hour
data['time_interval'] = data['time'].dt.floor('H')  # This helps in aggregating␣
 ↪data on an hourly basis

# Mapping of location IDs to more descriptive neighborhood names for better␣
 ↪readability and interpretation
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"
}
# Apply the mapping to the 'location' column to replace numeric IDs with the␣
 ↪corresponding neighborhood names
data['location'] = data['location'].map(neighborhood_names)

# Define a function to update the heatmap based on the selected category of␣
 ↪damage
def update_heatmap(category):
    # Group data by location and time intervals, and calculate the mean of each␣
 ↪category for aggregation
    grouped_data = data.groupby(['location', 'time_interval']).mean().
 ↪reset_index()

    # Convert 'time_interval' back to datetime if necessary, to ensure␣
 ↪compatibility with Plotly's time formatting
    grouped_data['time_interval'] = pd.
 ↪to_datetime(grouped_data['time_interval'])

    # Pivot the data to create a matrix suitable for a heatmap, where each row␣
 ↪is a neighborhood and each column is a time interval
    heatmap_data = grouped_data.pivot(index='location',␣
 ↪columns='time_interval', values=category).fillna(0)

    # Create a heatmap using Plotly Express
    fig = px.imshow(
        heatmap_data,
        labels=dict(x="Time Interval", y="Neighborhood", color=f"{category.
 ↪capitalize()} Severity"),
        x=heatmap_data.columns.strftime('%Y-%m-%d %H:%M'),  # Format the x-axis␣
 ↪labels for better readability
        y=[name for _, name in sorted(neighborhood_names.items())],  # Ensure␣
 ↪y-axis labels are ordered correctly
        title=f'Heatmap of {category.capitalize()} Damage Reports Over Time',
        aspect='auto',  # Set the aspect ratio to 'auto' to adjust cell width␣
 ↪automatically
        height=800  # Set a fixed height to ensure all y-axis labels are visible
    )
```

```
    # Update the layout to specify y-axis ticks manually to ensure all␣
 ↪neighborhoods are labeled
    fig.update_yaxes(tickmode='array',␣
 ↪tickvals=list(range(len(neighborhood_names))), ticktext=[name for _, name in␣
 ↪sorted(neighborhood_names.items())])
    fig.update_xaxes(tickangle=45)  # Rotate x-axis labels for better visibility

    # Customize hover tooltips to show detailed data about each cell
    fig.update_traces(
        hoverinfo="all",
        hovertemplate="<b>Time:</b> %{x}<br><b>Neighborhood:</b>␣
 ↪%{y}<br><b>{category.capitalize()} Severity:</b> %{z}<extra></extra>"
    )

    # Display the heatmap
    fig.show()

# Create a dropdown widget to select different categories of damage reports for␣
 ↪the heatmap
category_selector = widgets.Dropdown(
    options=['sewer_and_water', 'power', 'roads_and_bridges', 'medical',␣
 ↪'buildings', 'shake_intensity'],
    value='buildings',
    description='Category:',
    disabled=False,
)

# Display the widget and bind the function to update the heatmap when the␣
 ↪category selection changes
widgets.interactive(update_heatmap, category=category_selector)
```

```
C:\Users\kwasi\AppData\Local\Temp\ipykernel_29636\587309372.py:14:
FutureWarning:

'H' is deprecated and will be removed in a future version, please use 'h'
instead.
```

[3]:
```
interactive(children=(Dropdown(description='Category:', index=4,
options=('sewer_and_water', 'power', 'roads_a…
```

[12]:
```
import altair as alt
import pandas as pd

import pandas as pd

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

```
# 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['location'].
 ↪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 area chart with interactive legend
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")),
    opacity=alt.condition(legend_selection, alt.value(1), alt.value(0.2)),
    tooltip=['time:T', 'location:N', 'category:N', 'value:Q']
).add_params(
    location_selection,
    legend_selection
).transform_filter(
```

```
    location_selection  # Filter by the location selection
).properties(
    width=800,
    height=500,
    title='Average Damage Value Area Chart'
)

stacked_area_chart

stacked_area_chart.save('average_damalge_area.html')
```