**Main.py**

import tkinter as tk

from tkinter import ttk

from zeeshansami\_fticonsulting.visualization import \*

# Load data once to avoid multiple reads

carshare\_data = load\_carshare\_data()

election\_data = load\_election\_data()

melted\_data = melt\_election\_data(election\_data)

def visualize\_chart():

chart\_type = chart\_combobox.get()

dataset = dataset\_combobox.get()

if dataset == 'Carshare Data':

if chart\_type == 'Bar Chart':

fig = barchart\_carsharedata()

elif chart\_type == 'Box and Whisker Plot':

fig = boxandwhiskerplot\_carsharedata(carshare\_data)

elif chart\_type == 'Dot Plot':

fig = dotplots\_carsharedata(carshare\_data)

elif chart\_type == 'Heatmap':

fig = heatmaps\_carsharedata(carshare\_data)

elif chart\_type == 'Radar/spider Chart':

fig = radarspiderchart\_carsharedata(carshare\_data)

elif chart\_type == 'Treemaps':

fig = treemaps\_carsharedata(carshare\_data)

elif chart\_type == 'Waterfall Charts':

fig = waterfallcharts\_carsharedata()

fig.show()

else: # Assuming 'Election Data'

if chart\_type == 'Bar Chart':

fig = barchart\_electiondata(melted\_data)

elif chart\_type == 'Box and Whisker Plot':

fig = boxandwhiskerplot\_electiondata('Coderre')

elif chart\_type == 'Dot Plot':

fig = dotplots\_electiondata('Coderre')

elif chart\_type == 'Heatmap':

fig = heatmaps\_electiondata('Coderre')

elif chart\_type == 'Radar/spider Chart':

fig = radarspiderchart\_electiondata('Coderre')

elif chart\_type == 'Treemaps':

fig = treemaps\_electiondata('Coderre')

elif chart\_type == 'Waterfall Charts':

fig = waterfallcharts\_electiondata('Coderre')

fig.show()

# Creating main window

root = tk.Tk()

root.title("Visualization UI")

# Creating a frame for the combobox and label

frame = ttk.Frame(root, padding="10")

frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

# Adding a label

label = ttk.Label(frame, text="Select the type of chart:")

label.grid(column=0, row=0, sticky=tk.W, padx=5, pady=5)

# Combobox for chart selection

chart\_combobox = ttk.Combobox(frame, values=['Bar Chart', 'Box and Whisker Plot', 'Dot Plot', 'Heatmap', 'Radar/spider Chart', 'Treemaps', 'Waterfall Charts'])

chart\_combobox.grid(column=1, row=0, padx=5, pady=5)

chart\_combobox.set('Bar Chart')

# Label and combobox for dataset selection

dataset\_label = ttk.Label(frame, text="Select the dataset:")

dataset\_label.grid(column=0, row=1, sticky=tk.W, padx=5, pady=5)

dataset\_combobox = ttk.Combobox(frame, values=['Carshare Data', 'Election Data'])

dataset\_combobox.grid(column=1, row=1, padx=5, pady=5)

dataset\_combobox.set('Carshare Data')

# Button to visualize the chart

btn\_visualize = ttk.Button(frame, text="Visualize", command=visualize\_chart)

btn\_visualize.grid(column=1, row=2, pady=20)

root.mainloop()

**Visualizations.py**

# Barchart Carshare Data

import plotly.data as data

import plotly.express as px

# Data Handling

def load\_carshare\_data():

return data.carshare()

# Visualization Functions

def barchart\_carsharedata():

dataframe = load\_carshare\_data()

fig = px.histogram(dataframe, x="peak\_hour", nbins=24, title="Car Count by Peak Hour")

fig.show()

# Barchart ElectionData

# Import necessary libraries

import plotly.data as data

import plotly.express as px

import pandas as pd

# Load the election data

def load\_election\_data():

return data.election()

# Reshape the data using melt

def melt\_election\_data(election\_data):

return pd.melt(election\_data, id\_vars=['district'], value\_vars=['Coderre', 'Bergeron', 'Joly'],

var\_name='candidate', value\_name='votes')

# Visualization Function

def barchart\_electiondata(dataframe):

fig = px.bar(dataframe, x='district', y='votes', color='candidate', title="Votes by District and Candidate",

height=600, width=1200, barmode='group')

fig.update\_layout(xaxis\_tickangle=-45)

return fig

if \_\_name\_\_ == "\_\_main\_\_":

election\_data = load\_election\_data()

melted\_data = melt\_election\_data(election\_data)

# Visualize

fig = barchart\_electiondata(melted\_data)

fig.show()

# Box and Whisker Plots Carshare Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

def load\_carshare\_data():

"""Load the carshare dataset from plotly."""

return data.carshare()

def boxandwhiskerplot\_carsharedata(dataframe):

"""Plot car count by peak hour."""

df = dataframe.groupby('peak\_hour').size().reset\_index(name='counts')

fig = px.box(df, y='counts', title='Car Count by Peak Hour')

return fig

# Box and Whisker Plots Electiondata Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

# Load the election data

election\_data = data.election()

def boxandwhiskerplot\_electiondata(candidate\_name):

"""Generate a boxplot for votes distribution by district for a specific candidate."""

df = election\_data[['district', candidate\_name]]

df.columns = ['district', 'votes']

fig = px.box(df, y='votes', title=f'Votes Distribution by District for {candidate\_name}')

return fig

# Dot Plots Carshare Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

def load\_carshare\_data():

"""Load carshare dataset from plotly."""

return data.carshare()

def dotplots\_carsharedata(dataframe):

"""Generate a scatter plot showing car count by peak hour."""

df = dataframe.groupby('peak\_hour').size().reset\_index(name='counts')

fig = px.scatter(df, x='peak\_hour', y='counts', title='Car Count by Peak Hour')

fig.update\_traces(marker=dict(size=15,

line=dict(width=2,

color='DarkSlateGrey')),

selector=dict(mode='markers'))

return fig

# Dot Plots Electiondata Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

# Load the election data

election\_data = data.election()

def dotplots\_electiondata(candidate\_name):

"""Generate a scatter plot for votes distribution by district for a specific candidate."""

df = election\_data[['district', candidate\_name]]

df.columns = ['district', 'votes']

fig = px.scatter(df, x='votes', y='district', title=f'Votes by District for {candidate\_name}')

fig.update\_traces(marker=dict(size=15,

line=dict(width=2,

color='DarkSlateGrey')),

selector=dict(mode='markers'))

return fig

# Heatmaps Carshare Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

def load\_carshare\_data():

"""Load carshare dataset from plotly."""

return data.carshare()

def heatmaps\_carsharedata(dataframe):

"""Generate a heatmap showing car count by peak hour."""

df = dataframe.groupby('peak\_hour').size().reset\_index(name='counts')

fig = px.imshow(df, labels=dict(x="Peak Hour", y="Counts"), title='Car Count by Peak Hour')

return fig

# Heatmaps Electiondata Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

# Load the election data

election\_data = data.election()

def heatmaps\_electiondata(candidate\_name):

"""Generate a heatmap for votes distribution by district for a specific candidate."""

df = election\_data[['district', candidate\_name]]

df.columns = ['district', 'votes']

fig = px.imshow(df.pivot\_table(values='votes', index='district'),

title=f'Votes by District for {candidate\_name}')

return fig

# Radar Spider Chart carsharedata Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

def load\_carshare\_data():

"""Load carshare dataset from plotly."""

return data.carshare()

def radarspiderchart\_carsharedata(dataframe):

"""Generate a radar/spider chart showing car count by peak hour."""

df = dataframe.groupby('peak\_hour').size().reset\_index(name='counts')

fig = px.line\_polar(df, r='counts', theta='peak\_hour', line\_close=True)

fig.update\_traces(fill='toself')

fig.update\_layout(

title="Car Count by Peak Hour",

polar=dict(

radialaxis=dict(

visible=True,

range=[0, df['counts'].max()]

))

)

return fig

# Radar Spider Chart electiondata Dataset

# visualization.py

import plotly.data as data

import plotly.graph\_objects as go

# Load the election data

election\_data = data.election()

def radarspiderchart\_electiondata(candidate\_name):

"""Generate a radar/spider chart for votes distribution by district for a specific candidate."""

districts = election\_data['district'].tolist()

votes = election\_data[candidate\_name].tolist()

fig = go.Figure()

fig.add\_trace(go.Scatterpolar(

r=votes,

theta=districts,

fill='toself',

name=candidate\_name,

))

fig.update\_layout(

polar=dict(

radialaxis=dict(

visible=True,

range=[0, max(votes)+10]

)),

showlegend=False,

title=f"Votes by District for {candidate\_name}"

)

return fig

# Treemaps carsharedata Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

# Load the carshare data

def load\_carshare\_data():

return data.carshare()

def treemaps\_carsharedata(dataframe):

"""Generate a treemap for car counts by peak hour."""

df = dataframe.groupby('peak\_hour').size().reset\_index(name='counts')

fig = px.treemap(df, path=['peak\_hour'], values='counts',

title='Car Count by Peak Hour')

return fig

# Treemaps electiondata Dataset

# visualization.py

import plotly.data as data

import plotly.express as px

# Load the election data

def load\_election\_data():

return data.election()

def treemaps\_electiondata(candidate\_name):

"""Generate a treemap for votes by district for a specific candidate."""

df = load\_election\_data()[['district', candidate\_name]]

df.columns = ['district', 'votes']

fig = px.treemap(df, path=['district'], values='votes',

title=f'Votes by District for {candidate\_name}')

return fig

# waterfall Charts Carsharedata Dataset

# visualization.py

import plotly.data as data

import plotly.graph\_objects as go

# Load the carshare data

def load\_carshare\_data():

return data.carshare()

def waterfallcharts\_carsharedata():

"""Generate a waterfall chart for car counts by peak hour."""

dataframe = load\_carshare\_data()

df = dataframe.groupby('peak\_hour').size().reset\_index(name='counts')

fig = go.Figure(go.Waterfall(

name = "20",

orientation = "v",

measure = ["relative"]\*len(df['peak\_hour']),

x = df['peak\_hour'],

textposition = "outside",

text = df['counts'],

y = df['counts'],

connector = {"line":{"color":"rgb(63, 63, 63)"}},

))

fig.update\_layout(

title="Car Count by Peak Hour",

)

return fig

# waterfall Charts Electiondata Dataset

# visualization.py

import plotly.data as data

import plotly.graph\_objects as go

# Load the election data

def load\_election\_data():

return data.election()

def waterfallcharts\_electiondata(candidate\_name):

"""Generate a waterfall chart for votes by district for a given candidate."""

election\_data = load\_election\_data()

districts = election\_data['district'].tolist()

votes = election\_data[candidate\_name].tolist()

fig = go.Figure(go.Waterfall(

name = "20", orientation = "v",

measure = ["relative"]\*len(districts),

x = districts,

textposition = "outside",

text = votes,

y = votes,

connector = {"line":{"color":"rgb(63, 63, 63)"}},

))

fig.update\_layout(title=f"Votes by District for {candidate\_name}")

return fig

**test\_visualizations.py**

import unittest

# Import necessary functions and classes from the provided module

from zeeshansami\_fticonsulting.visualization import (

barchart\_carsharedata,

load\_election\_data,

melt\_election\_data,

barchart\_electiondata,

load\_carshare\_data,

boxandwhiskerplot\_carsharedata,

boxandwhiskerplot\_electiondata,

dotplots\_carsharedata,

dotplots\_electiondata,

heatmaps\_carsharedata,

heatmaps\_electiondata,

radarspiderchart\_carsharedata,

radarspiderchart\_electiondata,

treemaps\_carsharedata,

treemaps\_electiondata,

waterfallcharts\_carsharedata,

waterfallcharts\_electiondata

)

class VisualizationTest(unittest.TestCase):

def setUp(self):

# Load necessary data for the tests

self.election\_data = load\_election\_data()

self.carshare\_data = load\_carshare\_data()

self.melted\_data = melt\_election\_data(self.election\_data)

def test\_barchart\_carsharedata(self):

result = barchart\_carsharedata()

# Assert certain properties or values about the result, for example:

self.assertIsNotNone(result)

def test\_barchart\_electiondata(self):

fig = barchart\_electiondata(self.melted\_data)

self.assertIsNotNone(fig)

# ... Continue writing tests for other functions ...

def test\_heatmaps\_electiondata(self):

fig = heatmaps\_electiondata('Coderre')

self.assertIsNotNone(fig)

def test\_radarspiderchart\_carsharedata(self):

fig = radarspiderchart\_carsharedata(self.carshare\_data)

self.assertIsNotNone(fig)

# ... And so on for the remaining functions ...

def test\_waterfallcharts\_electiondata(self):

candidate = "Coderre"

fig = waterfallcharts\_electiondata(candidate)

self.assertIsNotNone(fig)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()