Auto Visualization on Uploading Any Dataset

Auto visualization tools provide a seamless way to gain insights from uploaded datasets by automatically generating visual representations of the data's key characteristics. These tools eliminate the need for manual exploration and visualization setup, making it convenient for users to quickly comprehend data trends, patterns, and distributions. By dynamically generating charts, graphs, and plots based on the dataset's attributes, auto visualization tools cater to users regardless of their technical expertise, enabling efficient decision-making and data-driven analysis.

In [1]:
```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output, State
import pandas as pd
import io
import base64
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
from plotly.subplots import make_subplots
```

In [2]:
```
# Initialize the Dash app
app = dash.Dash(__name__)
```

Note

This Dash app layout includes components for uploading a CSV file, displaying uploaded data, input fields for column names, buttons for different types of visualizations, and a placeholder for displaying the generated visualization. The uploaded_df variable is initialized to None and will be used to store the DataFrame once the CSV file is uploaded and processed.

In [3]:
```
# Set up the layout of the Dash app
app.layout = html.Div([
    # Upload component for selecting and uploading a CSV file
    dcc.Upload(
        id='upload-data',
        children=html.Button('Upload CSV File'),
        multiple=False
```

```
    ),
    # Placeholder for displaying the uploaded data
    html.Div(id='output-data'),

    # Input field for column name(s) used in visualizations
    dcc.Input(id='visualization-column', type='text', placeholder='Enter column name(s)'),

    # Buttons for different types of visualizations
    html.Button('Box Plot', id='box-plot-button', n_clicks=0),
    html.Button('Scatter Plot', id='scatter-plot-button', n_clicks=0),
    html.Button('Bar Chart', id='bar-chart-button', n_clicks=0),
    html.Button('Histogram', id='histogram-button', n_clicks=0),
    html.Button('Line Plot', id='line-plot-button', n_clicks=0),

    # Placeholder for displaying the generated visualization
    dcc.Graph(id='visualization-output'),
])

# Initialize a variable to store the uploaded DataFrame
uploaded_df = None
```

Note

The provided code defines two Dash callbacks. The first callback handles the parsing of uploaded CSV contents and displays information about the uploaded file. The second callback generates different visualizations based on button clicks and the selected column name for visualization. The specific visualization functions like generate_box_plot need to be implemented separately, and they should return the corresponding visualization figure using Plotly's figure definitions.

In [4]:

```
# Define a callback to parse the uploaded CSV contents
def parse_contents(contents):
    content_type, content_string = contents.split(',')
    decoded = base64.b64decode(content_string)
    df = pd.read_csv(io.StringIO(decoded.decode('utf-8')))
    return df

# Callback to update the output display with uploaded file information
@app.callback(
    Output('output-data', 'children'),
    Input('upload-data', 'contents'),
    State('upload-data', 'filename')
)
```

```
def update_output(contents, filename):
    global uploaded_df

    if contents is not None:
        df = parse_contents(contents)
        uploaded_df = df
        return [
            html.H4(f'Uploaded File: {filename}'),
            html.H5('Column Names:'),
            html.P(', '.join(df.columns.tolist())),
        ]
    return []

# Callback to update the visualization output based on button clicks
@app.callback(
    Output('visualization-output', 'figure'),
    Input('box-plot-button', 'n_clicks'),
    Input('scatter-plot-button', 'n_clicks'),
    Input('bar-chart-button', 'n_clicks'),
    Input('histogram-button', 'n_clicks'),
    Input('line-plot-button', 'n_clicks'),
    State('visualization-column', 'value')
)
def update_visualization_output(box_n_clicks, scatter_n_clicks, bar_n_clicks, hist_n_clicks, line_n_clicks, visualizati
    ctx = dash.callback_context

    if not ctx.triggered:
        return {}

    button_id = ctx.triggered[0]['prop_id'].split('.')[0]

    # Generate the appropriate visualization based on the clicked button
    if button_id == 'box-plot-button':
        return generate_box_plot(visualization_column)
    elif button_id == 'scatter-plot-button':
        return generate_scatter_plot(visualization_column)
    elif button_id == 'bar-chart-button':
        return generate_bar_chart(visualization_column)
    elif button_id == 'histogram-button':
        return generate_histogram(visualization_column)
    elif button_id == 'line-plot-button':
        return generate_line_plot(visualization_column)
    return {}
```

The provided code defines a function generate_box_plot(column_names) that generates a box plot visualization using Plotly. The function takes a string of column names as input and creates a separate box plot for each specified column.

In [5]:
```
import plotly.graph_objs as go

def generate_box_plot(column_names):
    # Create an empty Plotly figure
    fig = go.Figure()

    # Split the column_names string into individual column names
    for col in column_names.split(','):
        # Check if the column exists in the uploaded DataFrame
        if col in uploaded_df.columns:
            # Add a box plot trace for the current column
            fig.add_trace(go.Box(y=uploaded_df[col], name=col))

    # Customize the layout of the figure
    fig.update_layout(title=f'Box Plot of {column_names}')

    # Return the generated Plotly figure
    return fig
```

The provided code defines a function generate_scatter_plot(column_names) that generates a scatter plot visualization using Plotly. The function takes a string of two column names as input and creates a scatter plot using the data from those columns.

In [6]:
```
def generate_scatter_plot(column_names):
    # Create an empty Plotly figure
    fig = go.Figure()

    # Split the column_names string into individual column names
    cols = column_names.split(',')

    # Check if there are exactly two valid columns
    if len(cols) == 2 and all(col in uploaded_df.columns for col in cols):
        # Add a scatter plot trace for the two columns
        fig.add_trace(go.Scatter(x=uploaded_df[cols[0]], y=uploaded_df[cols[1]], mode='markers'))

        # Customize the layout of the figure
        fig.update_layout(title=f'Scatter Plot: {cols[0]} vs {cols[1]}')

    # Return the generated Plotly figure
    return fig
```

The provided code defines a function generate_bar_chart(column_names) that generates a horizontal bar chart visualization using Plotly. The function takes a string of column names as input and creates separate bar chart traces for each specified column.

In [7]:
```
def generate_bar_chart(column_names):
    # Create an empty Plotly figure
    fig = go.Figure()

    # Split the column_names string into individual column names
    for col in column_names.split(','):
        # Check if the column exists in the uploaded DataFrame
        if col in uploaded_df.columns:
            # Add a bar chart trace for the current column
            fig.add_trace(go.Bar(x=uploaded_df[col], y=uploaded_df.index, orientation='h', name=col))

    # Customize the layout of the figure
    fig.update_layout(title=f'Bar Chart of {column_names}')

    # Return the generated Plotly figure
    return fig
```

The provided code defines a function generate_histogram(column_names) that generates a histogram visualization using the Plotly Express library. The function takes a string of column names as input and creates a histogram for each specified column.

In [10]:
```
def generate_histogram(column_names):
    # Use Plotly Express to create a histogram
    fig = px.histogram(uploaded_df, x=column_names.split(','), title=f'Histogram of {column_names}')

    # Return the generated Plotly figure
    return fig
```

In [11]:
```
def generate_line_plot(column_names):
     # Use Plotly Express to create a line plot
    fig = px.line(uploaded_df, x=uploaded_df.index, y=column_names.split(','), title=f'Line Plot of {column_names}')
    # Return the generated Plotly figure
    return fig
```

In [ ]:
```
if __name__ == '__main__':
    app.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8050
Press CTRL+C to quit
127.0.0.1 - - [15/Aug/2023 11:52:06] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:52:07] "GET /_dash-layout HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:52:07] "GET /_dash-dependencies HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:52:08] "GET /_dash-component-suites/dash/dcc/async-upload.js HTTP/1.1" 304 -
127.0.0.1 - - [15/Aug/2023 11:52:08] "GET /_dash-component-suites/dash/dcc/async-graph.js HTTP/1.1" 304 -
127.0.0.1 - - [15/Aug/2023 11:52:08] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:52:08] "GET /_dash-component-suites/dash/dcc/async-plotlyjs.js HTTP/1.1" 304 -
127.0.0.1 - - [15/Aug/2023 11:52:08] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:57:05] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:57:28] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:58:16] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 11:58:58] "POST /_dash-update-component HTTP/1.1" 200 -
Exception on /_dash-update-component [POST]
Traceback (most recent call last):
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 2525, in wsgi_app
    response = self.full_dispatch_request()
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 1822, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 1820, in full_dispatch_request
    rv = self.dispatch_request()
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 1796, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\dash\dash.py", line 1274, in dispatch
    ctx.run(
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\dash\_callback.py", line 440, in add_context
    output_value = func(*func_args, **func_kwargs)  # %% callback invoked %%
  File "C:\Users\Syed Muqtasid Ali\AppData\Local\Temp\ipykernel_2652\3613855840.py", line 53, in update_visualization_o
utput
    return generate_histogram(visualization_column)
  File "C:\Users\Syed Muqtasid Ali\AppData\Local\Temp\ipykernel_2652\561267043.py", line 3, in generate_histogram
    fig = px.histogram(uploaded_df, x=column_names.split(','), title=f'Histogram of {column_names}')
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\plotly\express\_chart_types.py", line 480, in histogram
    return make_figure(
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\plotly\express\_core.py", line 1990, in make_figure
    args = build_dataframe(args, constructor)
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\plotly\express\_core.py", line 1452, in build_dataframe
    raise ValueError(
ValueError: Plotly Express cannot process wide-form data with columns of different type.
```

```
127.0.0.1 - - [15/Aug/2023 11:59:25] "POST /_dash-update-component HTTP/1.1" 500 -
Exception on /_dash-update-component [POST]
Traceback (most recent call last):
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 2525, in wsgi_app
    response = self.full_dispatch_request()
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 1822, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 1820, in full_dispatch_request
    rv = self.dispatch_request()
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\flask\app.py", line 1796, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\dash\dash.py", line 1274, in dispatch
    ctx.run(
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\dash\_callback.py", line 440, in add_context
    output_value = func(*func_args, **func_kwargs)  # %% callback invoked %%
  File "C:\Users\Syed Muqtasid Ali\AppData\Local\Temp\ipykernel_2652\3613855840.py", line 55, in update_visualization_o
utput
    return generate_line_plot(visualization_column)
  File "C:\Users\Syed Muqtasid Ali\AppData\Local\Temp\ipykernel_2652\2985509218.py", line 3, in generate_line_plot
    fig = px.line(uploaded_df, x=uploaded_df.index, y=column_names.split(','), title=f'Line Plot of {column_names}')
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\plotly\express\_chart_types.py", line 264, in line
    return make_figure(args=locals(), constructor=go.Scatter)
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\plotly\express\_core.py", line 1990, in make_figure
    args = build_dataframe(args, constructor)
  File "C:\Users\Syed Muqtasid Ali\anaconda3\lib\site-packages\plotly\express\_core.py", line 1452, in build_dataframe
    raise ValueError(
ValueError: Plotly Express cannot process wide-form data with columns of different type.
127.0.0.1 - - [15/Aug/2023 11:59:32] "POST /_dash-update-component HTTP/1.1" 500 -
127.0.0.1 - - [15/Aug/2023 11:59:51] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2023 12:00:12] "POST /_dash-update-component HTTP/1.1" 200 -
```

In [ ]: