Section 2: Moving Beyond Static Visualizations

Create a dataset of cumulative questions per library over time

```
import pandas as pd

questions_per_library =pd.read_csv( 'stackoverflow.zip', parse_dates=True, index_col='creation_date'

).loc[:, 'pandas': 'bokeh'].resample('1M').sum().cumsum().reindex( pd.date_range('2008-08', '2021-10', freq='M')

).fillna(0)

questions_per_library.tail()
```

	pandas	matplotlib	numpy	seaborn	geopandas	geoviews	altair	yellowbrick
2021-05-31	200734.0	57853.0	89812.0	6855.0	1456.0	57.0	716.0	46.0
2021-06-30	205065.0	58602.0	91026.0	7021.0	1522.0	57.0	760.0	48.0
2021-07-31	209235.0	59428.0	92254.0	7174.0	1579.0	62.0	781.0	50.0

2. Import the FuncAnimation class

```
from matplotlib.animation import FuncAnimation
```

3. Write a function for generating the initial plot

```
import matplotlib.pyplot as plt
from matplotlib import ticker
def bar_plot(data):
    fig,ax=plt.subplots(figsize=(8,6))
    sort_order=data.last('1M').squeeze().sort_values().index
    bars=[
        bar.set_label(label) for label, bar in
        zip(sort_order,ax.barh(sort_order,[0]*data.shape[1]))
    ]

    ax.set_xlabel('total questions',fontweight='bold')
    ax.set_xlim(0,250_00)
    ax.xaxis.set_major_formatter(ticker.EngFormatter())
    ax.xaxis.set_tick_params(labelsize=12)
    ax.yaxis.set_tick_params(labelsize=12)
    for spine in ['top','right']:
        ax.spines[spine].set_visible(False)

    fig.tight_layout()
    return fig,ax


%config InlineBackend.figure_formats=['svg']
%matplotlib inline
bar_plot(questions_per_library)
```

```
(<Figure size 800x600 with 1 Axes>, <Axes: xlabel='total questions'>)
```

4. Write a function for generating annotations and plot text

```
def generate_plot_text(ax):
    annotations = [
        ax.annotate(
        '', xy=(0, bar.get_y()+bar.get_height()/2),
        ha='left', va = 'center'
```

```
        ) for bar in ax.patches
    ]
    time_text = ax.text(
    0.9, 0.1, '', transform=ax.transAxes,
    fontsize=15, ha='center', va='center'
    )
    return annotations, time_text
```

5. Define the plot update function

```
def update(frame,*,ax,df,annotations,time_text):
    data=df.loc[frame,:]

    #update bars
    for rect, text in zip(ax.patches, annotations):
        col = rect.get_label()
        if data[col]:
            rect.set_width(data[col])
            text.set_x(data[col])
            text.set_text(f'{data[col]:,.0f}')
     #update time
    time_text.set_text(frame.strftime('%b\n%Y'))
```

6. Bind arguments to the update function

```
from functools import partial
def bar_plot_init(questions_per_library):
    fig,ax = bar_plot(questions_per_library)
    annotations, time_text=generate_plot_text(ax)

    bar_plot_update=partial(
        update, ax=ax, df=questions_per_library,
        annotations=annotations, time_text=time_text
    )

    return fig, bar_plot_update
```

7. Animate the plot

```
fig, update_func=bar_plot_init(questions_per_library)

ani =FuncAnimation(
    fig,update_func, frames=questions_per_library.index,repeat=False
)
ani.save(
    'stackoverflow_questions.gif',
    writer='ffmpeg',fps=30,bitrate=100,dpi=300
)
plt.close()
```

```
 MovieWriter ffmpeg unavailable; using Pillow instead.
```

```
from IPython import display

display.Video(
    'stackoverflow_questions.json', width = 600, height = 400,
    embed = True, html_attributes = 'controls muted autoplay'
)
```

Animating distributions over time

1. Create a dataset of daily subway entries

```
import pandas as pd
subway = pd.read_csv('NYC_subway_daily.csv',parse_dates=['Datetime'],index_col=['Borough','Datetime']
                )
subway_daily = subway.unstack(0)
subway_daily.head()
```

	Entries				Exits			
Borough	**Bk**	**Bx**	**M**	**Q**	**Bk**	**Bx**	**M**	**Q**
Datetime								
2017-02-04	617650.0	247539.0	1390496.0	408736.0	417449.0	148237.0	1225689.0	279699.0
2017-02-05	542667.0	199078.0	1232537.0	339716.0	405607.0	139856.0	1033610.0	268626.0
2017-02-06	1184916.0	472846.0	2774016.0	787206.0	761166.0	267991.0	2240027.0	537780.0
2017-02-07	1192638.0	470573.0	2892462.0	790557.0	763653.0	270007.0	2325024.0	544828.0
2017-02-08	1243658.0	497412.0	2998897.0	825679.0	788356.0	275695.0	2389534.0	559639.0

```
manhattan_entries = subway_daily['Entries']['M']
```

2. Determine the bin ranges for the histograms

```
import numpy as np
count_per_bin, bin_ranges = np.histogram(manhattan_entries, bins=30)
```

3. Write a function for generating the initial histogram subplots

```
def subway_histogram(data ,bins, date_range):
    _, bin_ranges = np.histogram(data, bins=bins)

    weekday_mask = data.index.weekday < 5
    configs = [
        {'label':'Weekend','mask': ~weekday_mask, 'ymax':60},
         {'label':'Wekend','mask': weekday_mask, 'ymax':120}
    ]

    fig, axes = plt.subplots(1, 2, figsize=(8,4),sharex=True)
    for ax, config in zip(axes, configs):
        _, _, config['hist'] = ax.hist(
            data[config['mask']].loc[date_range], bin_ranges, ec='black'
        )
    ax.xaxis.set_major_formatter(ticker.EngFormatter())
    ax.set(
        xlim=(0, None), ylim=(0, config['ymax']),
        xlabel=f'{config["label"]} Entries'
    )

    for spine in ['top', 'right']:
        ax.spines[spine].set_visible(False)
```

```
    axes[0].set_ylabel('Frequency')
    fig.suptitle('Historam of daily subway in Manhattan')
    fig.tight_layout()
    return fig,axes,bin_ranges, configs


_ = subway_histogram(manhattan_entries, bins=30, date_range='2017')
```

4. Write a function for generating an annotation for the time period

```
def add_time_text(ax):
    time_text = ax.text(
        0.15, 0.9, '',transform=ax.transAxes,
        fontsize=15, ha='center', va='center'
    )
    return time_text
```

5. Define the plot update function

```
def update(frame, *,data,configs,time_text,bin_ranges):
    artists = []

    time = frame.strftime('%b\n%Y')
    if time != time_text.get_text():
        time_text.set_text(time)
        artists.append(time_text)

    for config in configs:
        time_frame_mask = \
            (data.index > frame - pd.Timedelta(days=365)) & (data.index <= frame)
        counts, _= np.histogram(
            data[time_frame_mask & config['mask']],
            bin_ranges
        )
        for count, rect in zip(counts, config['hist'].patches):
            if count != rect.get_height():
                rect.set_height(count)
                artists.append(rect)
    return artists
```

6. Bind arguments for the update function

```
def histogram_init(data, bins, initial_date_range):
    fig, axes, bin_ranges, configs = subway_histogram(data, bins, initial_date_range)

    update_func = partial(
        update, data = data,configs=configs,
        time_text=add_time_text(axes[0]),
        bin_ranges=bin_ranges
    )
    return fig, update_func
```

7. Animate the plot

```
fig, update_func = histogram_init(
    manhattan_entries, bins = 30, initial_date_range=slice('2017','2019-07')
)
ani = FuncAnimation(
    fig, update_func, frames=manhattan_entries['2019-08':'2021'].index,
    repeat=False, blit=True
)
ani.save(
    'subway_entries_subplots.gif',
    writer='ffmpeg',fps=30,bitrate=500,dpi=300
)
plt.close()
```

```
 MovieWriter ffmpeg unavailable; using Pillow instead.
```

```
from IPython import display

display.Video(
    'subway_entries_subplots.json', width = 600, height = 400,
    embed = True, html_attributes = 'controls muted autoplay'
)
```

Animating geospatial data with HoloViz

1. Use GeoPandas to read in our data.

```
import geopandas as gpd

earthquakes = gpd.read_file('earthquakes.geojson').assign(
    time=lambda x: pd.to_datetime(x.time, unit='ms'),
    month=lambda x: x.time.dt.month
)[['geometry', 'mag', 'time', 'month']]

earthquakes.shape
```

```
 (188527, 4)
```

```
earthquakes.head()
```

```
                                geometry   mag                 time  month
1    POINT Z (-67.09010 19.07660 6.00000)  2.55  2020-01-01 00:03:38.210      1
```

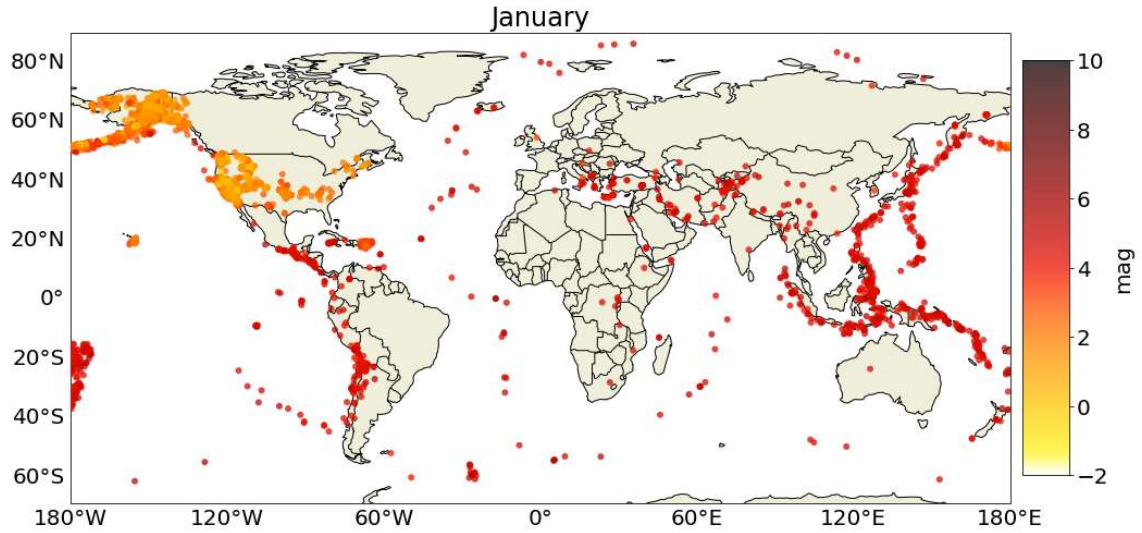
2. Handle HoloViz imports and set up the Matplotlib backend.

```
import geoviews as gv
import geoviews.feature as gf
import holoviews as hv

gv.extension('matplotlib')
```

3. Define a function for plotting earthquakes on a map using GeoViews.

```
import calendar

def plot_earthquakes(data, month_num):
    points = gv.Points(
        data.query(f'month == {month_num}'),
        kdims=['longitude', 'latitude'], # key dimensions (for coordinates in this case)
        vdims=['mag'] # value dimensions (for modifying the plot in this case)
    ).redim.range(mag=(-2, 10), latitude=(-90, 90))

    # create an overlay by combining Cartopy features and the points with *
    overlay = gf.land * gf.coastline * gf.borders * points

    return overlay.opts(
        gv.opts.Points(color='mag', cmap='fire_r', colorbar=True, alpha=0.75),
        gv.opts.Overlay(
            global_extent=False, title=calendar.month_name[month_num], fontscale=2
        )
    )


plot_earthquakes(earthquakes, 1).opts(
    fig_inches=(6, 3), aspect=2, fig_size=250, fig_bounds=(0.07, 0.05, 0.87, 0.95)
)
```

```
C:\Users\trung\anaconda3\lib\site-packages\cartopy\io\__init__.py:241: DownloadWarning: Downloading: https:
  warnings.warn(f'Downloading: {url}', DownloadWarning)
C:\Users\trung\anaconda3\lib\site-packages\cartopy\io\__init__.py:241: DownloadWarning: Downloading: https:
  warnings.warn(f'Downloading: {url}', DownloadWarning)
C:\Users\trung\anaconda3\lib\site-packages\cartopy\io\__init__.py:241: DownloadWarning: Downloading: https:
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

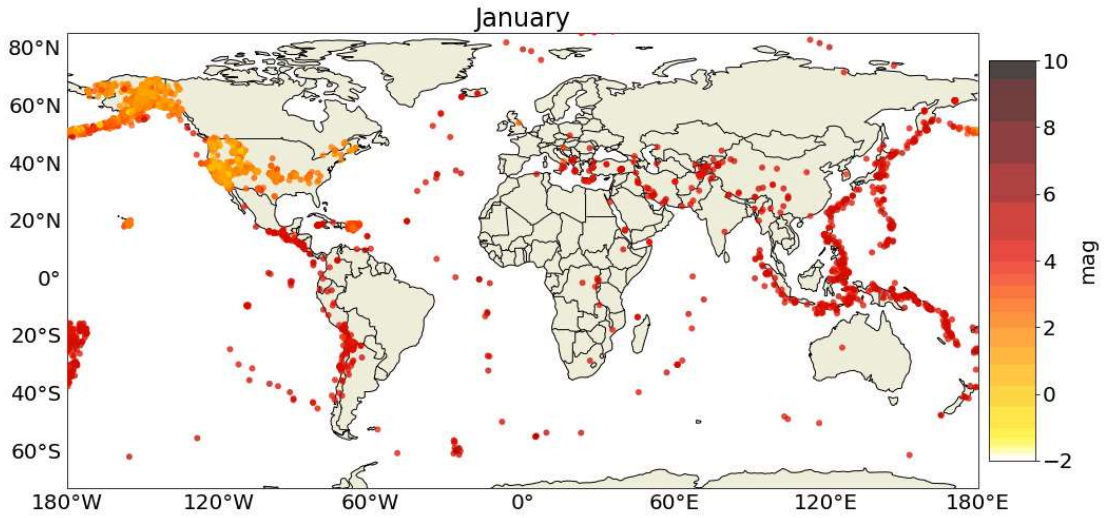


4. Create a mapping of frames to plots using HoloViews.

```
frames = {
    month_num: plot_earthquakes(earthquakes, month_num)
    for month_num in range(1, 13)
}
holomap = hv.HoloMap(frames)
```

5. Animate the plot.

```
hv.output(
    holomap.opts(
        fig_inches=(6, 3), aspect=2, fig_size=250,
        fig_bounds=(0.07, 0.05, 0.87, 0.95)
    ), holomap='gif', fps=5
)
```



Section 3: Building Interactive Visualizations for Data Exploration

Adding tooltips and sliders

1. Read in and prepare the data.

```
import geopandas as gpd
import pandas as pd

earthquakes = gpd.read_file('earthquakes.geojson').assign(
    time=lambda x: pd.to_datetime(x.time, unit='ms'),
    month=lambda x: x.time.dt.month
).dropna()

earthquakes.head()
```

	mag	place	time	tsunami	magType	geometry	month
0	2.75	80 km N of Isabela, Puerto Rico	2020-01-01 00:01:56.590	0	md	POINT Z (-67.12750 19.21750 12.00000)	1
2	1.81	12 km SSE of Maria Antonia,	2020-01-01	0	md	POINT Z (-66.65410 17.97050	1

2. Import the required libraries and set up the Bokeh backend.

```
from cartopy import crs
import geoviews as gv
import geoviews.feature as gf

gv.extension('bokeh')
```

3. Create an overlay with tooltips and a slider.

```
points = gv.Points(
    earthquakes,
    kdims=['longitude', 'latitude'],
    vdims=['month', 'place', 'tsunami', 'mag', 'magType']
)

# set colorbar limits for magnitude and axis limits
points = points.redim.range(
    mag=(-2, 10), longitude=(-180, 180), latitude=(-90, 90)
)


overlay = gf.land * gf.coastline * gf.borders * points.groupby('month')


interactive_map = overlay.opts(
    gv.opts.Feature(projection=crs.PlateCarree()),
    gv.opts.Overlay(width=700, height=450),
    gv.opts.Points(color='mag', cmap='fire_r', colorbar=True, tools=['hover'])
)
```

4. Render the visualization.

```
import panel as pn

earthquake_viz = pn.panel(interactive_map, widget_location='bottom')


earthquake_viz.embed()
```

Additional plot types

```
import numpy as np

flight_stats = pd.read_csv(
    'T100_MARKET_ALL_CARRIER.zip',
    usecols=[
        'CLASS', 'REGION', 'UNIQUE_CARRIER_NAME', 'ORIGIN_CITY_NAME', 'ORIGIN',
        'DEST_CITY_NAME', 'DEST', 'PASSENGERS', 'FREIGHT', 'MAIL'
    ]
).rename(lambda x: x.lower(), axis=1).assign(
    region=lambda x: x.region.replace({
        'D': 'Domestic', 'I': 'International', 'A': 'Atlantic',
        'L': 'Latin America', 'P': 'Pacific', 'S': 'System'
    }),
    route=lambda x: np.where(
        x.origin < x.dest,
        x.origin + '-' + x.dest,
        x.dest + '-' + x.origin
```

```
    )
)
```

```
flight_stats.head()
```

	passengers	freight	mail	unique_carrier_name	region	origin	origin_city_name	dest	dest_city_na
0	0.0	53185.0	0.0	Emirates	International	DXB	Dubai, United Arab Emirates	IAH	Houston,
1	0.0	9002.0	0.0	Emirates	International	DXB	Dubai, United Arab Emirates	JFK	New York,
2	0.0	2220750.0	0.0	Emirates	International	DXB	Dubai, United Arab Emirates	ORD	Chicago
3	0.0	1201488.0	0.0	Emirates	International	DXB	Dubai, United Arab Emirates	IAH	Houston, TX
4	0.0	1201488.0	0.0	Emirates	International	DXB	Dubai, United Arab Emirates	DXB	Dubai, Unit

```
cities = [
    'Atlanta, GA', 'Chicago, IL', 'New York, NY', 'Los Angeles, CA',
    'Dallas/Fort Worth, TX', 'Denver, CO', 'Houston, TX',
    'San Francisco, CA', 'Seattle, WA', 'Orlando, FL'
]

top_airlines = [
    'American Airlines Inc.', 'Delta Air Lines Inc.', 'JetBlue Airways',
    'Southwest Airlines Co.', 'United Air Lines Inc.'
]
```

Chord diagram

```
total_flight_stats = flight_stats.query(
    '`class` == "F" and origin_city_name != dest_city_name'
    f' and origin_city_name.isin({cities}) and dest_city_name.isin({cities})'
).groupby([
    'origin', 'origin_city_name', 'dest', 'dest_city_name'
])[['passengers', 'freight', 'mail']].sum().reset_index().query('passengers > 0')
```

```
total_flight_stats.sample(10, random_state=1)
```

	origin	origin_city_name	dest	dest_city_name	passengers	freight	mail
78	LGA	New York, NY	DEN	Denver, CO	589190.0	506023.0	293108.0
117	ORD	Chicago, IL	SEA	Seattle, WA	810594.0	1063463.0	2627325.0
31	DFW	Dallas/Fort Worth, TX	MCO	Orlando, FL	683700.0	187672.0	95570.0
5	ATL	Atlanta, GA	LAX	Los Angeles, CA	1121378.0	8707125.0	3267077.0
126	SEA	Seattle, WA	LGA	New York, NY	24.0	0.0	0.0
45	IAH	Houston, TX	ATL	Atlanta, GA	566369.0	367543.0	726670.0
14	DEN	Denver, CO	HOU	Houston, TX	305193.0	363119.0	0.0
44	HOU	Houston, TX	SFO	San Francisco, CA	1843.0	5523.0	0.0
73	LAX	Los Angeles, CA	MDW	Chicago, IL	277226.0	2022416.0	0.0
89	MCO	Orlando, FL	DEN	Denver, CO	594878.0	368516.0	138811.0

```
chord = hv.Chord(
    total_flight_stats,
    kdims=['origin', 'dest'],
    vdims=['passengers', 'origin_city_name', 'dest_city_name', 'mail', 'freight']
)
```

```
from bokeh.models import HoverTool

tooltips = {
    'Source': '@origin_city_name (@origin)',
    'Target': '@dest_city_name (@dest)',
    'Passengers': '@passengers{0,.}',
    'Mail': '@mail{0,.} lbs.',
```

```
    'Freight': '@freight{0,.} lbs.',
}
hover = HoverTool(tooltips=tooltips)


chord = chord.opts(
    labels='index', node_color='index', cmap='Category20', # node config
    edge_color='origin', edge_cmap='Category20', directed=True, # edge config
    inspection_policy='edges', tools=[hover, 'tap'], # tooltip config
    frame_width=500, aspect=1, # plot size config
    title='Total Passenger Service Travel Between Top 10 Cities in 2019'
)


chord
```

Sankey plot

```
top_cities = cities[:5]

domestic_passenger_travel = flight_stats.query(
    'region == "Domestic" and `class` == "F" and origin_city_name != dest_city_name '
    f'and origin_city_name.isin({top_cities}) and dest_city_name.isin({top_cities})'
).groupby([
    'region', 'unique_carrier_name', 'route', 'origin_city_name', 'dest_city_name'
]).passengers.sum().reset_index()

domestic_passenger_travel.head()
```

	region	unique_carrier_name	route	origin_city_name	dest_city_name	passengers
0	Domestic	Air Wisconsin Airlines Corp	ATL-ORD	Atlanta, GA	Chicago, IL	915.0
1	Domestic	Air Wisconsin Airlines Corp	ATL-ORD	Chicago, IL	Atlanta, GA	556.0
2	Domestic	Alaska Airlines Inc.	JFK-LAX	Los Angeles, CA	New York, NY	265307.0
3	Domestic	Alaska Airlines Inc.	JFK-LAX	New York, NY	Los Angeles, CA	257685.0
4	Domestic	Alaska Airlines Inc.	LAX-ORD	Chicago, IL	Los Angeles, CA	48269.0

```
domestic_passenger_travel.unique_carrier_name = (
    domestic_passenger_travel.unique_carrier_name.replace(
        '^(?!' + '|'.join(top_airlines) + ').*$',
        'Other Airlines',
        regex=True
    )
)


domestic_passenger_travel.groupby('unique_carrier_name').passengers.sum().div(
    domestic_passenger_travel.passengers.sum()
)
```

```
unique_carrier_name
American Airlines Inc.    0.337186
Delta Air Lines Inc.      0.312187
JetBlue Airways           0.049500
Other Airlines            0.120544
Southwest Airlines Co.    0.079074
United Air Lines Inc.     0.101509
Name: passengers, dtype: float64
```

```
def get_edges(data, *, source_col, target_col):
    aggregated = data.groupby([source_col, target_col]).passengers.sum()
    return aggregated.reset_index().rename(
        columns={source_col: 'source', target_col: 'target'}
    ).query('passengers > 0')


carrier_edges = get_edges(
    domestic_passenger_travel,
    source_col='region',
    target_col='unique_carrier_name'
```

```
).replace('^Domestic$', 'Top Routes', regex=True)

carrier_edges
```

	source	target	passengers
0	Top Routes	American Airlines Inc.	9426060.0
1	Top Routes	Delta Air Lines Inc.	8727210.0
2	Top Routes	JetBlue Airways	1383776.0
3	Top Routes	Other Airlines	3369815.0
4	Top Routes	Southwest Airlines Co.	2210533.0
5	Top Routes	United Air Lines Inc.	2837682.0

```
carrier_to_route_edges = get_edges(
    domestic_passenger_travel,
    source_col='unique_carrier_name',
    target_col='route'
)

carrier_to_route_edges.sample(10, random_state=1)
```

	source	target	passengers
39	Other Airlines	DFW-LGA	157366.0
41	Other Airlines	JFK-LAX	523222.0
2	American Airlines Inc.	ATL-LAX	294304.0
48	Southwest Airlines Co.	ATL-MDW	498481.0
50	Southwest Airlines Co.	LAX-MDW	558574.0
44	Other Airlines	LAX-ORD	378552.0
33	Other Airlines	ATL-LAX	146882.0
35	Other Airlines	ATL-MDW	1201.0
40	Other Airlines	DFW-ORD	241147.0
27	JetBlue Airways	DFW-JFK	140.0

```
all_edges = pd.concat([carrier_edges, carrier_to_route_edges]).assign(
    passengers=lambda x: x.passengers / 1e6
)


sankey = hv.Sankey(
    all_edges,
    kdims=['source', 'target'],
    vdims=hv.Dimension('passengers', unit='M')
).opts(
    labels='index', label_position='right', cmap='Set1', # node config
    edge_color='lightgray', # edge config
    width=750, height=600, # plot size config
    title='Travel Between the Top 5 Cities in 2019'
)


sankey
```