[Lab 5]

Name: Nguyễn Thành Trung – MSSV: 19522431

Link github: [Data\_Mining/Week5 at main · Shu2301/Data\_Mining (github.com)](https://github.com/Shu2301/Data_Mining/tree/main/Week5)

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python # For example, here's several helpful packages to load

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the read-only "../input/" directory

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, \_, filenames in os.walk(' /kaggle/input'): for filename in filenames:

print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory ( /kaggle/working/) that gets preserved as output when you create a version using "Save & # You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

# In this workshop, you will continue using Matplotlib, Seaborn and Plotly to explore the data. Filter from the dataset all the “Accounting” job and visualize the total job postings of each sub-sectors.

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import plotly.express as px

df = pd.read\_csv(" /kaggle/input/shu/job-market.csv")

accounting\_jobs = df[df['Classification'] == 'Accounting']

subsector\_counts = accounting\_jobs['SubClassification'].value\_counts()

fig = px.bar(x=subsector\_counts.index, y=subsector\_counts.values, labels={'x':'Sub-sector', 'y':'Count'}, color=subsector\_counts.index) fig.update\_layout(title='Accounting')

fig.show()

# Count the job postings by month and visualize in line graph using Matplotlib, Seaborn and Plotly

import matplotlib.pyplot as plt

# Convert the 'Date' column to datetime format df['Date'] = pd.to\_datetime(df['Date'])

# Filter the data for the month of October 2018

df = df[(df['Date'] >= '2018-10-01') & (df['Date'] <= '2018-10-31')]

# Group the data by day and count the job postings

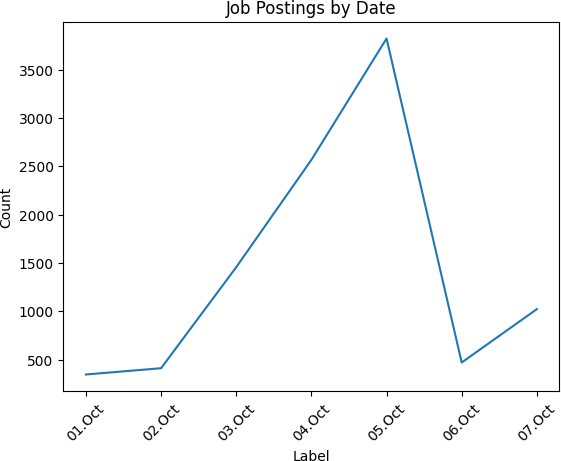
job\_postings\_by\_day = df.groupby(pd.Grouper(key='Date', freq='D'))['Title'].count()

# Create a line graph of the job postings by day using Matplotlib plt.plot(job\_postings\_by\_day.index, job\_postings\_by\_day.values)

plt.xticks(job\_postings\_by\_day.index, [d.strftime('%d.%b') for d in job\_postings\_by\_day.index], rotation=45) plt.xlabel('Label')

plt.ylabel('Count')

plt.title('Job Postings by Date') plt.show()



# Visualize salary distribution of sub-sector in Accounting using boxplot of Matplotlib, Seaborn and Plotly.

import seaborn as sns

accounting\_jobs = df[df['Classification'] == 'Accounting'] plt.figure(figsize=(10,8))

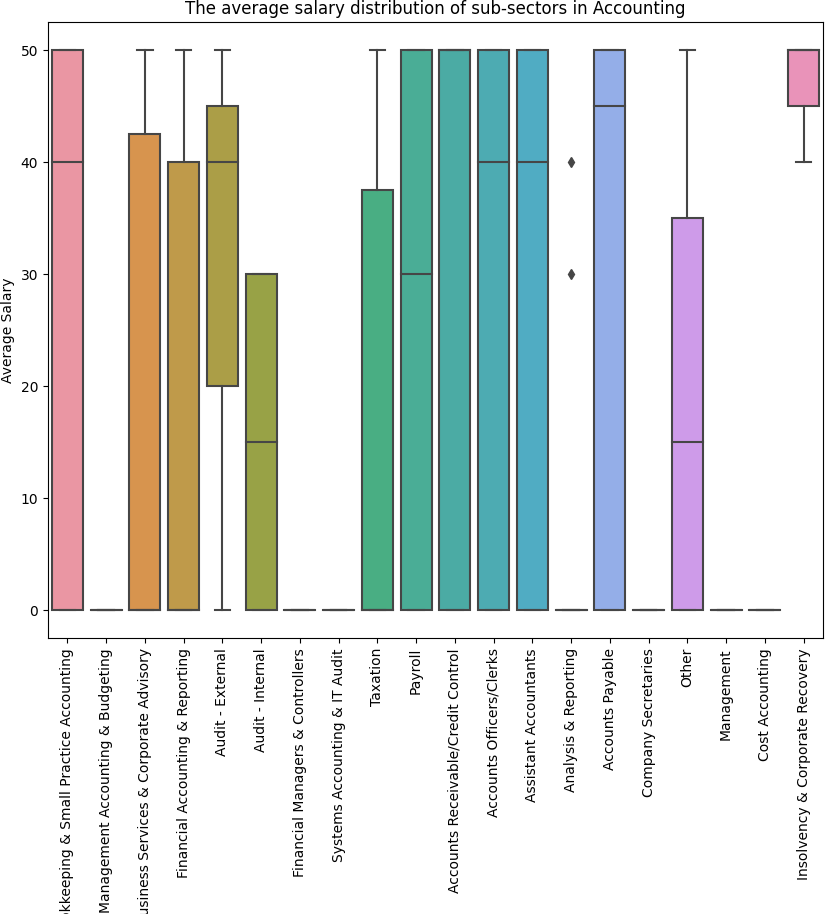
sns.boxplot(x=accounting\_jobs['SubClassification'], y=accounting\_jobs['LowestSalary']) plt.xlabel('Sub-sector')

plt.ylabel('Average Salary')

plt.title('The average salary distribution of sub-sectors in Accounting') plt.xticks(rotation=90)

plt.show()





/opt/conda/lib/python3.10/site-packages/scipy/ init .py:146: UserWarning:

A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

# Section 2: Moving Beyond Static Visualizations

Animating cumulative values over time

import pandas as pd

questions\_per\_library = pd.read\_csv(

' /kaggle/input/shu/stackoverflow/stackoverflow.csv', 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** | **vega** | **holoviews** | **hvplot** | **bokeh** |
| **2021-05-31** | 200734.0 | 57853.0 | 89812.0 | 6855.0 | 1456.0 | 57.0 | 716.0 | 46.0 | 532.0 | 513.0 | 84.0 | 4270.0 |
| **2021-06-30** | 205065.0 | 58602.0 | 91026.0 | 7021.0 | 1522.0 | 57.0 | 760.0 | 48.0 | 557.0 | 521.0 | 88.0 | 4308.0 |
| **2021-07-31** | 209235.0 | 59428.0 | 92254.0 | 7174.0 | 1579.0 | 62.0 | 781.0 | 50.0 | 572.0 | 528.0 | 89.0 | 4341.0 |
| **2021-08-31** | 213410.0 | 60250.0 | 93349.0 | 7344.0 | 1631.0 | 62.0 | 797.0 | 52.0 | 589.0 | 541.0 | 92.0 | 4372.0 |
| **2021-09-30** | 214919.0 | 60554.0 | 93797.0 | 7414.0 | 1652.0 | 63.0 | 804.0 | 54.0 | 598.0 | 542.0 | 92.0 | 4386.0 |

from matplotlib.animation import FuncAnimation

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\_000)

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>, <AxesSubplot: xlabel='total questions'>)



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

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'))

from pandas.core.strings.base import annotations 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

from numpy import repeat

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.mp4',

writer='ffmpeg', fps=10, bitrate=100, dpi=300

)

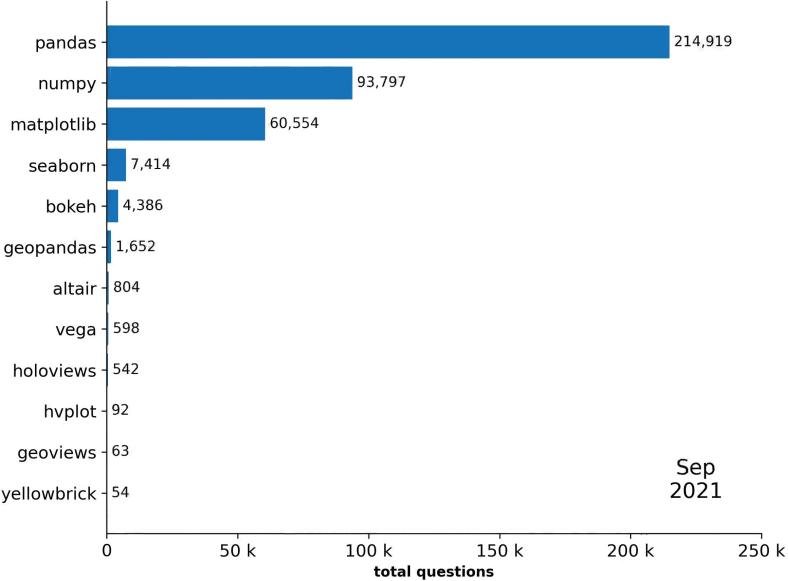
plt.close()

from IPython import display

display.Video(

'/kaggle/working/stackoverflow\_questions.mp4', width=600, height=400, embed=True, html\_attributes='controls muted autoplay'

)



0:15 / 0:15

# Animating distributions over time

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation from matplotlib import ticker

from functools import partial

subway = pd.read\_csv(

' /kaggle/input/shu/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']

count\_per\_bin, bin\_ranges = np.histogram(manhattan\_entries, bins=30)

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': 'Weekday', '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('Histogram of Daily Subway Entries in Manhattan') fig.tight\_layout()

return fig, axes, bin\_ranges, configs

\_ = subway\_histogram(manhattan\_entries, bins=30, date\_range='2017')





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

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

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

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.mp4',

writer='ffmpeg', fps=30, bitrate=500, dpi=300

)

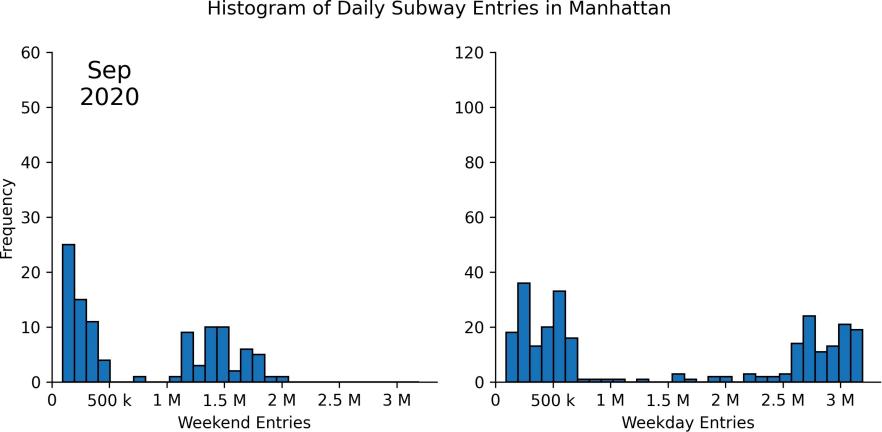
plt.close()

from IPython import display

display.Video(

'/kaggle/working/subway\_entries\_subplots.mp4', width=600, height=400, embed=True, html\_attributes='controls muted autoplay'

)



0:13 / 0:18

# Animating geospatial data with HoloViz

import pandas as pd

import geopandas as gpd

earthquakes = gpd.read\_file(' /kaggle/input/shu/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

**0** POINT Z (-67.12750 19.21750 12.00000) 2.75 2020-01-01 00:01:56.590 1



import geoviews as gv

import geoviews.feature as gf import holoviews as hv

gv.extension('matplotlib')

**1** POINT Z (-67.09010 19.07660 6.00000) 2.55 2020-01-01 00:03:38.210 1

**2** POINT Z (-66.85410 17.87050 6.00000) 1.81 2020-01-01 00:05:09.440 1

**3** POINT Z (-66.86360 17.89930 8.00000) 1.84 2020-01-01 00:05:36.930 1

**4** POINT Z (-66.86850 17.90660 8.00000) 1.64 2020-01-01 00:09:20.060 1

import calendar

def plot\_earthquakes(data, month\_num): points = gv.Points(

data.query(f'month == {month\_num}'), kdims = ['longitude', 'latitude'],

vdims = ['mag']

).redim.range(mag=(-2, 5), latitude=(-45,45))

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=f'{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)

)

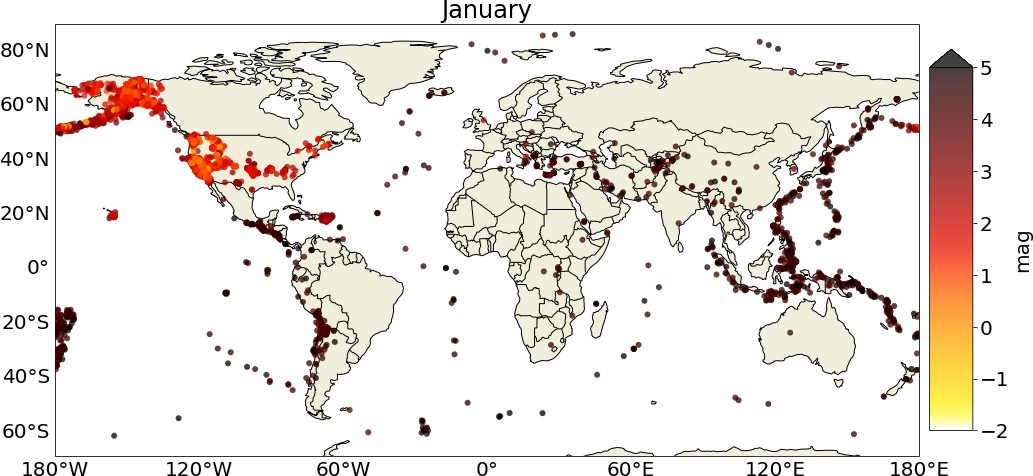
/opt/conda/lib/python3.10/site-packages/cartopy/io/ init .py:241: DownloadWarning: Downloading: [h ttps://naturalearth.s3.amazonaws.com/110m\_physical/ne\_110m\_land.zip](https://naturalearth.s3.amazonaws.com/110m_physical/ne_110m_land.zip)

/opt/conda/lib/python3.10/site-packages/cartopy/io/ init .py:241: DownloadWarning:

Downloading: [h ttps://naturalearth.s3.amazonaws.com/110m\_physical/ne\_110m\_coastline.zip](https://naturalearth.s3.amazonaws.com/110m_physical/ne_110m_coastline.zip)

/opt/conda/lib/python3.10/site-packages/cartopy/io/ init .py:241: DownloadWarning:

Downloading: [h ttps://naturalearth.s3.amazonaws.com/110m\_cultural/ne\_110m\_admin\_0\_boundary\_lines\_land.zip](https://naturalearth.s3.amazonaws.com/110m_cultural/ne_110m_admin_0_boundary_lines_land.zip)



frames = {

month\_num: plot\_earthquakes(earthquakes, month\_num)

for month\_num in range(1, 13)

}

holomap = hv.HoloMap(frames)

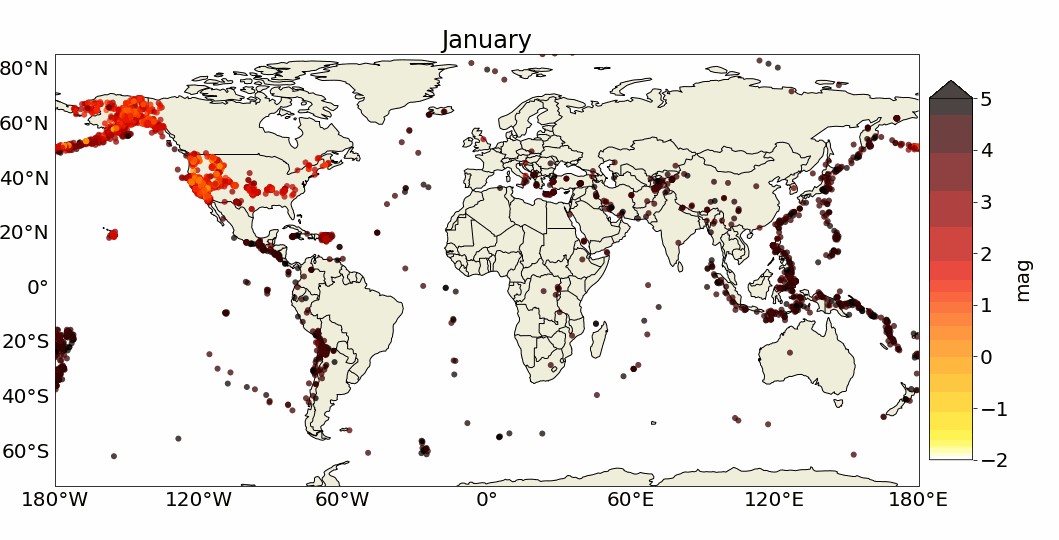
hv.output(

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

)



hv.save(

holomap.opts(

fig\_inches=(6,3), aspect=2, fig\_size=250, fig\_bounds=(0.07, 0.05, 0.87, 0.95)

), 'earthquakes.gif', fps=5

)

# Up Next: Building interactive Visualizations for Data ExplorationExcercises Excercises

1. Modify the animation of subway entries from this section to show both the weekday and weekend histograms on the same subplot
2. Modify the earthquake animation to show earthquakes per day in April 2020.

Section 3: Building interactive visualizations for data exploration

import pandas as pd

import geopandas as gpd

earthquakes = gpd.read\_file('/kaggle/input/shu/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 |
| **1** 2.55 | 64 km N of Isabela, Puerto Rico | 2020-01-01 00:03:38.210 | 0 | md | POINT Z (-67.09010 19.07660 6.00000) | 1 |
| **2** 1.81 | 12 km SSE of Maria Antonia, Puerto Rico | 2020-01-01 00:05:09.440 | 0 | md | POINT Z (-66.85410 17.87050 6.00000) | 1 |
| **3** 1.84 | 9 km SSE of Maria Antonia, Puerto Rico | 2020-01-01 00:05:36.930 | 0 | md | POINT Z (-66.86360 17.89930 8.00000) | 1 |
| **4** 1.64 | 8 km SSE of Maria Antonia, Puerto Rico | 2020-01-01 00:09:20.060 | 0 | md | POINT Z (-66.86850 17.90660 8.00000) | 1 |

from cartopy import crs import geoviews as gv

import geoviews.feature as gf gv.extension('bokeh')



points = gv.Points( earthquakes,

kdims = ['longitude', 'latitude'],

vdims = ['month', 'place', 'tsunami', 'mag', 'magType']

)

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'])

)

import panel as pn

earthquake\_viz = pn.panel(interactive\_map, widget\_location='bottom')

earthquake\_viz.embed()

# Linking plots

january\_earthquakes = earthquakes.query('month == 1').assign( longitude = lambda x: x.geometry.x,

latitude = lambda x: x.geometry.y

).drop(columns=['month', 'geometry'])

import hvplot.pandas

geo = january\_earthquakes.hvplot(

x='longitude', y='latitude', kind='points', color='mag', cmap='fire\_r', clim=(-2,10),

titles='CartoLight', geo=True, global\_extent=True,

xlabel='Longitude', ylabel='Latitude', title='January 2020 Earthquakes', frame\_height=450

)

WARNING:param.main: titles option not found for points plot with bokeh; similar options include: ['title', 'tiles']

table = january\_earthquakes.sort\_values(['longitude', 'latitude']).hvplot( kind='table', width=650, height=450, title='Raw Data'

)

layout = geo + table

selection = hv.link\_selections.instance()

map\_and\_table\_tabs = selection(layout).opts(tabs=True)

map\_and\_table\_tabs

selection.filter(january\_earthquakes).nlargest(3, 'mag')

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **mag** | **place** | **time** | **tsunami** | **magType** | **longitude** | **latitude** |
| **15154** | 7.7 | 123 km NNW of Lucea, Jamaica | 2020-01-28 19:10:24.918 | 1 | mww | -78.7560 | 19.4193 |
| **13296** | 6.7 | 13 km N of Do?anyol, Turkey | 2020-01-24 17:55:14.147 | 0 | mww | 39.0609 | 38.4312 |
| **4062** | 6.4 | 13 km SSE of Maria Antonia, Puerto Rico | 2020-01-07 08:24:25.262 | 1 | mww | -66.8266 | 17.8686 |

# Additional plot types

import pandas as pd import numpy as np

flight\_stats = pd.read\_csv(

' /kaggle/input/shu/T100\_MARKET\_ALL\_CARRIER/865214564\_T\_T100\_MARKET\_ALL\_CARRIER.csv', 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()

## ity\_name

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **passengers** | **freight** | **mail** | **unique\_carrier\_name** | **region** | **origin origin\_c** |
| **0** 0.0 | 53185.0 | 0.0 | Emirates | International | DXB Dubai, |
| **1** 0.0 | 9002.0 | 0.0 | Emirates | International | DXB Dubai, |
| **2** 0.0 | 2220750.0 | 0.0 | Emirates | International | DXB Dubai, |

United Arab Emirates

United Arab Emirates

United Arab Emirates

**te**

B- IAH

|  |  |  |
| --- | --- | --- |
| **dest** | **dest\_city\_name** | **class rou** |
| IAH | Houston, TX | G DX |
| JFK | New York, NY | G DX |
| ORD | Chicago, IL  Dubai, United Arab | G DX  OR  DX |

B- JFK

B- D

B-

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(

f'`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 |

import holoviews as hv 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)

# 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

Corp ORD

|  |  |  |
| --- | --- | --- |
| **0** Domestic Air Wisconsin Airlines ATL- Atlanta, GA | Chicago, IL | 915.0 |
| **1** Domestic Air Wisconsin Airlines ATL- Chicago, IL | Atlanta, GA | 556.0 |
| **2** Domestic Alaska Airlines Inc. JFK- Los Angeles, CA | New York, NY | 265307.0 |

Corp ORD

LAX

JFK-

domestic\_passenger\_travel.unique\_carrier\_name.replace( '^(?!' + '|'.join(top\_airlines) + ').\*$',

'Other Airlines',

regex=True, inplace=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') 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

)

import holoviews as hv hv.extension('bokeh')

sankey = hv.Sankey( all\_edges,

kdims=['source', 'target'],

vdims=hv.Dimension('passengers', unit='M')

).opts(

labels='index', label\_position='right', cmap='Set1', edge\_color='lightgray',

width=750, height=600,

title='Travel Between the Top 5 Cities in 2019'

)



sankey

# Exercises

1. For the 10 carriers that transported the most freight, create a bar plot showing total freight transported per carrier.

import matplotlib.pyplot as plt

# Group by carrier and sum the freight

top\_10\_carriers = flight\_stats.groupby('unique\_carrier\_name')['freight'].sum().nlargest(10)

# Create a bar plot

plt.bar(top\_10\_carriers.index, top\_10\_carriers.values)

# Set labels and title plt.xlabel('Carrier')

plt.ylabel('Total Freight (in millions)')

plt.title('Total Freight Transported by Top 10 Carriers')

# Rotate x-axis labels diagonally for better visibility plt.xticks(rotation=45, ha='right')

# Show the plot plt.show()

import pandas as pd

import geopandas as gpd

from bokeh.plotting import figure, show from bokeh.models import HoverTool

# Load earthquake data into a geopandas DataFrame

earthquakes = gpd.read\_file('/kaggle/input/shu/earthquakes.geojson')

# Convert the 'time' column to a datetime object

earthquakes['time'] = pd.to\_datetime(earthquakes['time'], unit='ms')

# Group by day and count the number of earthquakes

earthquakes\_per\_day = earthquakes.groupby(pd.Grouper(key='time', freq='D')).size()

# Create a line plot with tooltips p = figure(

title='Total Earthquakes per Day', x\_axis\_label='Date',

y\_axis\_label='Total Earthquakes', x\_axis\_type='datetime',

tools='hover',

tooltips=[('Date', '@time{%F}'), ('Total Earthquakes', '@value')], sizing\_mode="stretch\_both"

)

p.line(

x=earthquakes\_per\_day.index, y=earthquakes\_per\_day.values, line\_width=2,

)

# Format the hover tooltip to show the date in YYYY-MM-DD format p.hover.formatters = {'@time': 'datetime'}

# Show the plot show(p)

# 3. Make histograms of earthquake magnitude (mag) for each magnitude type (magType) with a dropdown to select the magnitude type.

import pandas as pd

import geopandas as gpd

from bokeh.layouts import column

from bokeh.plotting import figure, show

from bokeh.models import ColumnDataSource, Select

# Load earthquake data into a geopandas DataFrame

earthquakes = gpd.read\_file('/kaggle/input/shu/earthquakes.geojson')

# Filter out NaN values from the magType column

earthquakes = earthquakes.dropna(subset=['magType'])

# Create a ColumnDataSource for the histograms

source = ColumnDataSource(data=dict(mag=[], magType=[]))

# Create a histogram plot p = figure(

title='Histogram of Earthquake Magnitude', x\_axis\_label='Magnitude',

y\_axis\_label='Count',

tools='box\_select, lasso\_select', sizing\_mode="stretch\_both"

)

hist, edges = np.histogram([], bins=50) p.quad(

top=hist, bottom=0,

left=edges[:-1], right=edges[1:], fill\_color='navy',

line\_color='white', alpha=0.5

)

# Create a dropdown menu to select the magnitude type (sorted by magnitude type) magTypes = sorted(earthquakes.magType.unique().tolist())

magType\_menu = Select(

title='Magnitude Type', options=magTypes,

value=magTypes[0]

)

# Define a callback function to update the histogram when the magnitude type is changed def update\_hist(attr, old, new):

selected\_data = earthquakes[earthquakes.magType == magType\_menu.value] hist, edges = np.histogram(selected\_data.mag, bins=50)

source.data = dict(mag=selected\_data.mag, magType=[magType\_menu.value] \* len(selected\_data)) p.title.text = f'Histogram of Earthquake Magnitude ({magType\_menu.value})'

p.quad(

top=hist, bottom=0,

left=edges[:-1], right=edges[1:], fill\_color='navy',

line\_color='white', alpha=0.5

)

# Attach the update\_hist callback function to the magnitude type dropdown magType\_menu.on\_change('value', update\_hist)

# Create a layout with the histogram plot and the magnitude type dropdown layout = column(magType\_menu, p)

# Show the plot

update\_hist(None, None, magType\_menu.value) show(layout)



 0 giây hoàn thành lúc 21:29