In [1]:

Visualization in Bokeh

Neba Nfonsang

In [2]:

```
import dataiku
from dataiku import pandasutils as pdu
import pandas as pd


# import functions from bokeh modules
from bokeh.plotting import figure
from bokeh.io import output_notebook, show, curdoc
from bokeh.models import ColumnDataSource, Range1d, LabelSet
from bokeh.models.widgets import Slider, TextInput, Select
from bokeh.layouts import row, widgetbox, gridplot
output_notebook()
```

Loading BokehJS ...

In [3]:

```
# Load a DSS dataset as a Pandas dataframe
salary_df = dataiku.Dataset("salary_data_prepared")
salary_df = salary_df.get_dataframe()
salary_df.head()
```

Out[3]:

	rank	discipline	yrs_since_phd	yrs_service	gender	salary
0	Prof	B	19	18	Male	139750
1	Prof	B	20	16	Male	173200
2	AsstProf	B	4	3	Male	79750
3	Prof	B	45	39	Male	115000
4	Prof	B	40	41	Male	141500

Setting up an Empty Figure

In [4]:

```
# Initialize an empty figure or plot
fig = figure(title="Plot example",
            width=600,
            height=300,
            x_axis_label="x label",
            y_axis_label="y label")
show(fig)
```

A Scatter Plot

In [5]:

```
# initialize the figure or plot
fig = figure(title="Plot example",
            width=600,
            height=400,
            x_axis_label="x label",
            y_axis_label="y label")
```

```
fig.scatter(x=salary_df["yrs_since_phd"], y=salary_df["yrs_service"])

show(fig)
```

In [6]:

```
# initialize the figure or plot
fig = figure(title="A Scatter Plot of Salary vs Years of Service",
             width=700,
             height=400,
             x_axis_label="Years of Service",
             y_axis_label="Salary")

fig.circle(x=salary_df["yrs_service"], y=salary_df["salary"], size=10)

show(fig)
```

Line Plot

In [7]:

```
# initialize the figure or plot
years = [2010, 2011, 2013, 2014, 2015, 2016]
sales = [500, 400, 700, 1000, 800, 750]
```

```
fig = figure(title="A line plot of Years Since Phd vs Years of Service",
            width=700,
            height=400,
            x_axis_label="Year",
            y_axis_label="Sales")

fig.line(x=years, y=sales, line_width=4, color="red")
show(fig)
```

A Plot of Categorical Data

In [8]:

```
sal_by_rank = salary_df.groupby(by="rank", as_index=False)["salary"].mean()
sal_by_rank
```

Out[8]:

In [9]:

```
fig = figure(title="A plot of categorical data",
            width=700,
            height=400,
            x_axis_label="Rank",
            y_axis_label="Mean Salary",
            x_range=sal_by_rank["rank"].values)

```

```
fig.vbar(x=sal_by_rank["rank"], width=0.8, top=sal_by_r
ank["salary"])

show(fig)
```

Layout

In [10]:

```
np.random.randint(0, 100, size=100)
```

Out[10]:

```
array([82, 45, 27, 90, 97, 94, 82, 38, 68, 98, 45, 81,
88, 29, 79, 60, 13,
        9, 54, 79, 21, 51, 88, 31, 25, 56,  9, 56, 33,
62, 73, 44, 96, 71,
       63, 14, 11, 86,  9, 91, 26, 19, 39, 92, 35, 14,
34, 81, 39, 52, 93,
       74, 99, 67, 64, 10, 40, 52, 50, 80, 70, 22, 47,
2, 60, 89,  7, 79,
       54, 83, 67, 79, 83, 70, 71, 82, 82, 93, 26, 33,
91, 47, 61, 40, 33,
       33,  0, 36, 45, 50, 67, 32, 82, 58, 68, 14, 96,
77, 87, 94])
```

In [11]:

```
plot1 = figure(title="A plot of x vs y",

            width=400,

            height=400,

            x_axis_label="x",

            y_axis_label="y")


# plot randomly generated values
```

```
plot1.circle(np.random.randint(0, 100, size=100),
             np.random.randint(0, 100, size=100), size=
10, color="green")


plot2 = figure(title="A plot of x vs y",
            width=700,
            height=400,
            x_axis_label="x",
            y_axis_label="y")


# plot randomly generated values
plot2.circle(np.random.randint(0, 100, size=100),
             np.random.randint(0, 100, size=100), size=
10)



layout = row(plot1, plot2)
show(layout)
```

Gridplot

In [12]:

```
plot1 = figure(title="A plot of x vs y",
            width=400,
            height=400,
            x_axis_label="x",
             y_axis_label="y")

# plot randomly generated values
plot1.circle(np.random.randint(0, 100, size=100),
              np.random.randint(0, 100, size=100), size=
10, color="green")

plot2 = figure(title="A plot of x vs y",
             width=700,
             height=400,
             x_axis_label="x",
             y_axis_label="y")

# plot randomly generated values
plot2.circle(np.random.randint(0, 100, size=100),
              np.random.randint(0, 100, size=100), size=
10)


plot3 = figure(title="A plot of x vs y",
             width=700,
             height=400,
             x_axis_label="x",
             y_axis_label="y")
```

```
# plot randomly generated values

plot3.circle(np.random.randint(0, 100, size=100),

             np.random.randint(0, 100, size=100), size=10)



layout = gridplot([[plot1, plot2], [plot3, None]], toolbar_location=None)

show(layout)
```

Configuration Tools

Here are five main tools as seen on the right side of the plot with the following names

- PanTool: pan ==> used to drag plot around
- BoxZoomTool: box_zoom ==> allows you to select a portion of the plot, then zoom into that
- WheelZoomTool: wheel_zoom ==> used to zoom the plot through scrolling
- Save: save ==> allows you to save the plot
- Reset: reset ==> reset to clear any action you have taken to go back to the original plot

Tools could be grouped as follows: Pan/drag tools

- pan
- boox_select
- box_zoom
- lasso_select

Click/tap tools

- poly_select
- tap

Scroll/pinch tools

- wheel_zoom
- xwheel_pan
- ywheel_pan

Inspectors

- croshair
- hover

Customize the Tools

In [13]:

```
tools = ["lasso_select", "box_zoom", "tap"]
fig = figure(title="A plot of categorical data",
             width=700,
             height=400,
             x_axis_label="Rank",
             y_axis_label="Mean Salary",
             x_range=sal_by_rank["rank"].values,
             tools=tools)


fig.vbar(x=sal_by_rank["rank"], width=0.8, top=sal_by_rank["salary"])
show(fig)
```

Plot using Data Source

In [14]:

```
fig = figure(title="A plot of categorical data",
            width=700,
            height=400,
            x_axis_label="Rank",
            y_axis_label="Mean Salary",
            x_range=sal_by_rank["rank"].values,
            tools=tools)

source = ColumnDataSource(sal_by_rank)

fig.vbar(x="rank", width=0.8, top="salary", source=source, color="skyblue")
show(fig)
```

In [15]:

```
fig = figure(title="A plot of categorical data",
            width=700,
            height=400,
            x_axis_label="Rank",
            y_axis_label="Mean Salary",
            x_range=sal_by_rank["rank"].values,
            tools=tools)

source = ColumnDataSource(sal_by_rank)

# set the range of the y axis
```

```
fig.y_range = Range1d(0, 200000)
fig.vbar(x="rank", width=0.8, top="salary", source=source)
show(fig)
```

Interactive Visualization

Note: bokeh server must be used when working with interactive widgets. The primary purpose of the bokeh server is to synchronize data between the underlying Python environment and the BokehJS library running in the browser.

In [16]:

```
import dataiku
from dataiku import pandasutils as pdu
import pandas as pd
import numpy as np

# import functions from bokeh modules
from bokeh.io import curdoc
from bokeh.layouts import row, widgetbox
from bokeh.models import ColumnDataSource, Range1d, LabelSet
from bokeh.models.widgets import Slider, TextInput, Select
from bokeh.plotting import figure

# parameters for webapp inputs
```

```
input_dataset = "salary_data_prepared"
x_column = "rank"
y_column = "salary"
discipline_column = "discipline"
gender_column = "gender"
service_column = "yrs_service"


# Set up the data
data = dataiku.Dataset(input_dataset)
data_df = data.get_dataframe()
# groupby rank and aggregrate with mean
sal_mean_by_rank_df = data_df.groupby(by=x_column, as_index=False)[y_column].mean()
sal_mean_by_rank_df = sal_mean_by_rank_df.round(2)
source = ColumnDataSource(sal_mean_by_rank_df)

# create the plot
tools = ["lasso_select", "box_zoom", "tap", "crosshair",
         "pan", "reset", "save", "wheel_zoom"]

plot = figure(title="Mean " + y_column + " by " + x_column,
            width=500,
            height=400,
            x_axis_label=x_column.title(),
            y_axis_label="Mean " + y_column.title(),
            x_range=sal_mean_by_rank_df[x_column].value
s,
            tools=tools)

plot.y_range = Range1d(0, 200000) ## set y range
## label bar plots with y data
labels = LabelSet(x=x_column, y=y_column, text=y_column
, level='glyph',
        x_offset=-13.5, y_offset=0, source=source, rend
er_mode='canvas')

plot.vbar(x=x_column, width=0.8, top=y_column, source=s
ource)

plot.add_layout(labels)

# set up the widget
title_text = TextInput(title="Title", value="Mean " + y
_column + " by " + x_column)

service_data = data_df[service_column].values
min_yrs_service = Slider(title="Min Years in Service",
                         value=min(service_data), start=min(service_data),
                         end=max(service_data), step=1)

max_yrs_service = Slider(title="Max Years in Service",
                         value=max(service_data), start=min(service_data),
                         end=max(service_data), step=1)

gender_categories = data_df[gender_column].unique().tolist()
gender_categories.insert(0, "All")
gender_cat = Select(title="Gender Category", value=gender_categories[0],
                    options=gender_categories)


discipline_categories = data_df[discipline_column].unique().tolist()
discipline_categories.insert(0, "All")
discipline_cat = Select(title="Discipline Category", value=discipline_categories[0],
                       options=discipline_categories)

#show(title_text)
```

```
# set up update functions and callbacks

def update_title(attrname, old, new):
    plot.title.text = title_text.value # get current title text value
title_text.on_change("value", update_title) # trigger the function upon changes in title_text


def update_data(attrname, old, new):
    selected = data_df[(service_data>=min_yrs_service.value) &
                       (service_data<=max_yrs_service.value)]
    if (gender_cat.value!=gender_categories[0]):
        selected = selected[selected[gender_column]==gender_cat.value]

    if (discipline_cat.value!=discipline_categories[0]):
        selected = selected[selected[discipline_column]==discipline_cat.value]

    df = selected.groupby(by=x_column, as_index=False)[y_column].mean()
    df = df.round(2)
    source.data = ColumnDataSource.from_df(df)


for w in [min_yrs_service, max_yrs_service, gender_cat, discipline_cat]:
    w.on_change("value", update_data)


# set up layout and add to document
menu = widgetbox(title_text, min_yrs_service, max_yrs_service, gender_cat, discipline_cat)
layout = row(plot, menu)
curdoc().add_root(layout)
show(layout)
```