Interactive Visualization with Bokeh - 2

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Transform and prepare data for creating visualizations	
Create simple plots using Bokeh	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your course materials folder and `data_dir` be the variable corresponding to your `data` folder

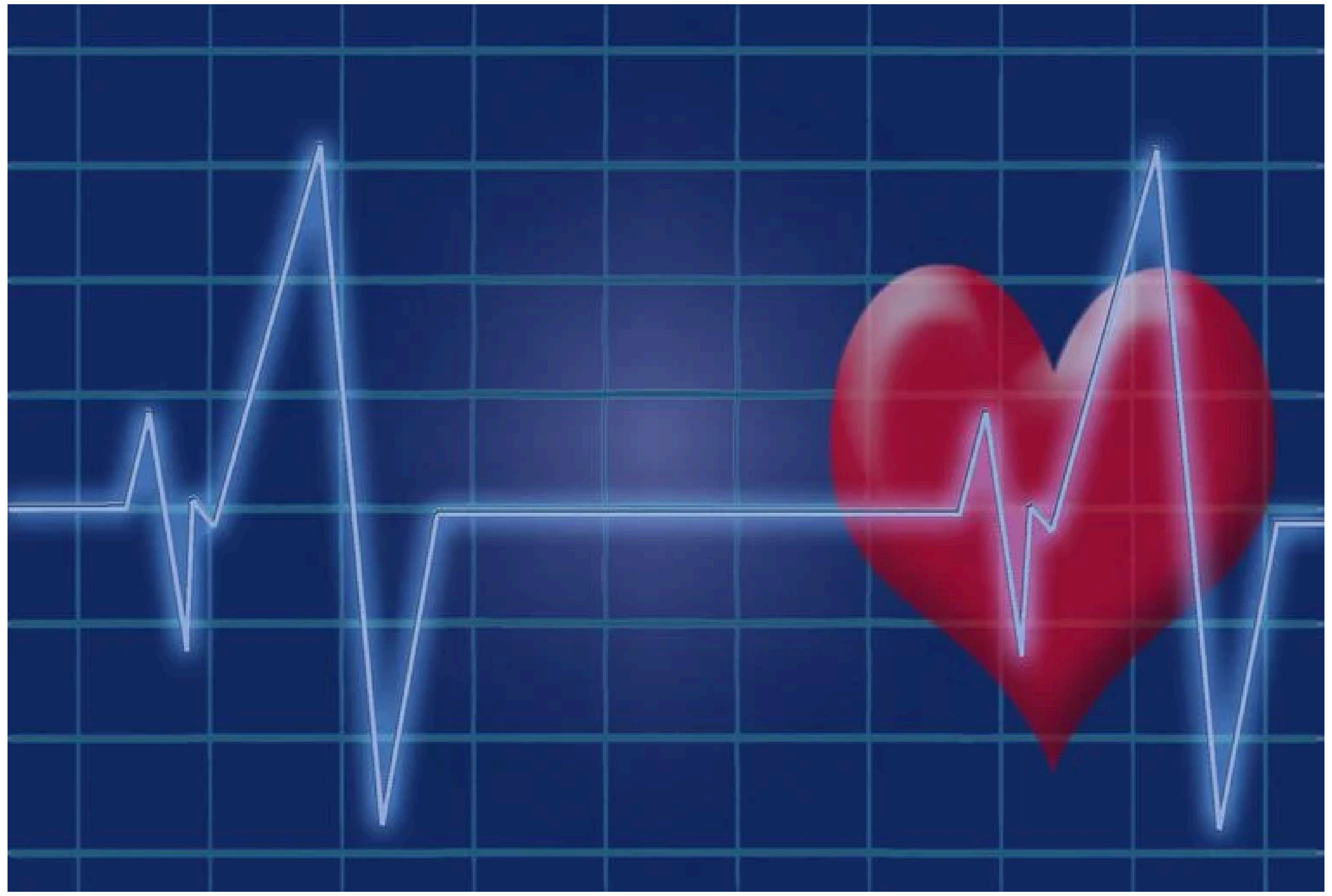
```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)

plot_dir = str(main_dir) + "/plots"
print(plot_dir)
```

Heart Disease survey: case study

- According to the World Health Organization (WHO), stroke is the 2nd leading cause of death globally
- *Click here* to see the dataset showing the results of a clinical trial of a heart-disease drug survey on a sample of US adults
- Each row in the data provides relevant information about the adult, including whether they had a stroke
- Using this data we want to predict whether a patient will likely have a stroke based on their demographic information and medical history



Dataset

- To implement everything we learn in this course, we will use the healthcare-dataset-stroke-data.csv dataset
- We will work with columns such as:
    - stroke
    - gender
    - age
    - hypertension
    - heart_disease
    - ever_married
- We will use different columns of the dataset to analyze stroke dataset

Load data into Python

- First, load the entire dataset
- Then, use the function `read_csv` to read in the `healthcare-dataset-stroke-data.csv` dataset

```
df = pd.read_csv(str(data_dir)+"/"+ 'healthcare-dataset-stroke-data.csv')
print(df.head())
```

```
      id  gender   age  ...   bmi   smoking_status stroke
0   9046    Male  67.0  ...  36.6  formerly smoked      1
1  51676  Female  61.0  ...   NaN     never smoked      1
2  31112    Male  80.0  ...  32.5     never smoked      1
3  60182  Female  49.0  ...  34.4           smokes      1
4   1665  Female  79.0  ...  24.0     never smoked      1

[5 rows x 12 columns]
```

Subset data

- Remove any columns from the dataframe that are not numeric or categorical, as we will not use them in our models

```
df = df[['age', 'avg_glucose_level', 'heart_disease', 'ever_married', 'hypertension',
'Residence_type', 'gender', 'smoking_status', 'work_type', 'stroke']]
print(df.head())
```

```
    age  avg_glucose_level  ...      work_type stroke
0  67.0             228.69  ...        Private      1
1  61.0             202.21  ...  Self-employed      1
2  80.0             105.92  ...        Private      1
3  49.0             171.23  ...        Private      1
4  79.0             174.12  ...  Self-employed      1

[5 rows x 10 columns]
```

Convert target to binary

- Let's check if the target (*stroke*) is binary; and if not, convert it to binary

```
# Target not binary - calculate the mean and assign the above mean to 1 and below to 0
print(df['stroke'].value_counts())
```

```
0    4861
1     249
Name: stroke, dtype: int64
```

- Since our target variable `stroke` is binary already, we need not convert it

- However, here's the code if we need to convert target variables to binary:

```
threshold = np.mean(df['target'])
df['target'] = np.where(df['target'] > threshold, 1,0)
# Target is binary
print(df['target'])
```

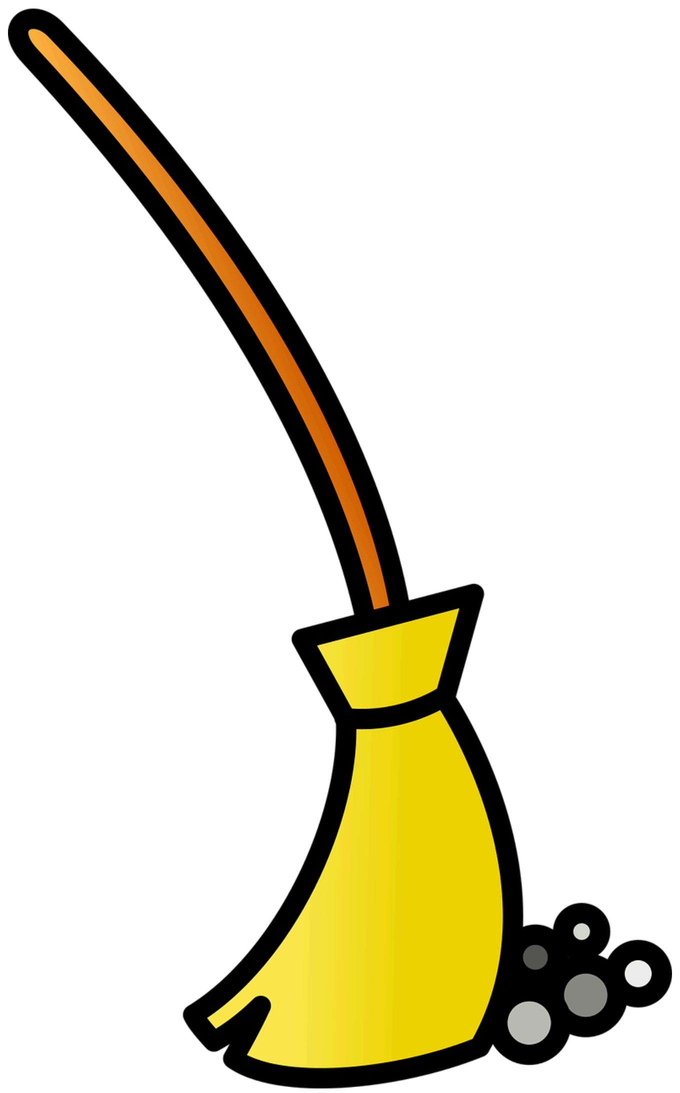
ID variables

- We will not use columns such as ID variables or variables which have more than 50% NAs: **`id`**

Data cleaning steps for visualization

- Before jumping into creating data visualizations, we must take a few important steps:

   i. Make sure the target is labeled
   ii. Check for NAs *(null values)*



The data at first glance

- We will start by looking at the first three rows of the data as well as the data types

```
# The first 3 rows.
print(df.head(3))
```

```
    age  avg_glucose_level  ...
work_type stroke
0  67.0             228.69  ...
Private        1
1  61.0             202.21  ...   Self-
employed        1
2  80.0             105.92  ...
Private        1

[3 rows x 10 columns]
```

```
# The data types.
print(df.dtypes)
```

```
age                  float64
avg_glucose_level    float64
heart_disease          int64
ever_married          object
hypertension           int64
Residence_type        object
gender                object
smoking_status        object
work_type             object
stroke                 int64
dtype: object
```

- We can also get the frequency table of the target variable

```
print(df['stroke'].value_counts())
```

```
0    4861
1     249
Name: stroke, dtype: int64
```

Data prep: label target data

- Now, let's create a new column to label our target variable `stroke`

```
df['Target_class'] = np.where(df['stroke']==1, 'affected','not_affected')
```

Data prep: check for NAs

- Next, we will check for NAs
- There are multiple methods to deal with them

```
 # Check for NAs.
print(df.isnull().sum())
```

```
age                    0
avg_glucose_level      0
heart_disease          0
ever_married           0
hypertension           0
Residence_type         0
gender                 0
smoking_status      1544
work_type              0
stroke                 0
Target_class           0
dtype: int64
```

- If we do have NAs, we can replace them with a **mean** or **0**

```
percent_missing = df.isnull().sum() *
100 / len(df)
print(percent_missing)
```

```
age                   0.000000
avg_glucose_level     0.000000
heart_disease         0.000000
ever_married          0.000000
hypertension          0.000000
Residence_type        0.000000
gender                0.000000
smoking_status       30.215264
work_type             0.000000
stroke                0.000000
Target_class          0.000000
dtype: float64
```

Data prep: check for NAs

- Here's a convenience function which will help impute missing data if it exists in the dataset

```
# Delete columns containing either 50% or more than 50% NaN Values
perc = 50.0
min_count =  int(((100-perc)/100)*df.shape[0] + 1)
df = df.dropna(axis=1,
               thresh=min_count)
print(df.shape)
```

```
(5110, 11)
```

```
# Function to impute NA in both numeric and categorical columns
def fillna(df):
# Fill numerical columns with mean
    numerical_columns = df.select_dtypes(include=['number'])
    numerical_columns = numerical_columns.fillna(numerical_columns.mean())

    # Fill categorical columns with median
    categorical_columns = df.select_dtypes(exclude=['number'])
    categorical_columns = categorical_columns.fillna(categorical_columns.mode().iloc[0])

    # Combine the numerical and categorical columns back into the original DataFrame
    filled_df = pd.concat([numerical_columns, categorical_columns], axis=1)
    return filled_df

df = fillna(df)
```

Module completion checklist

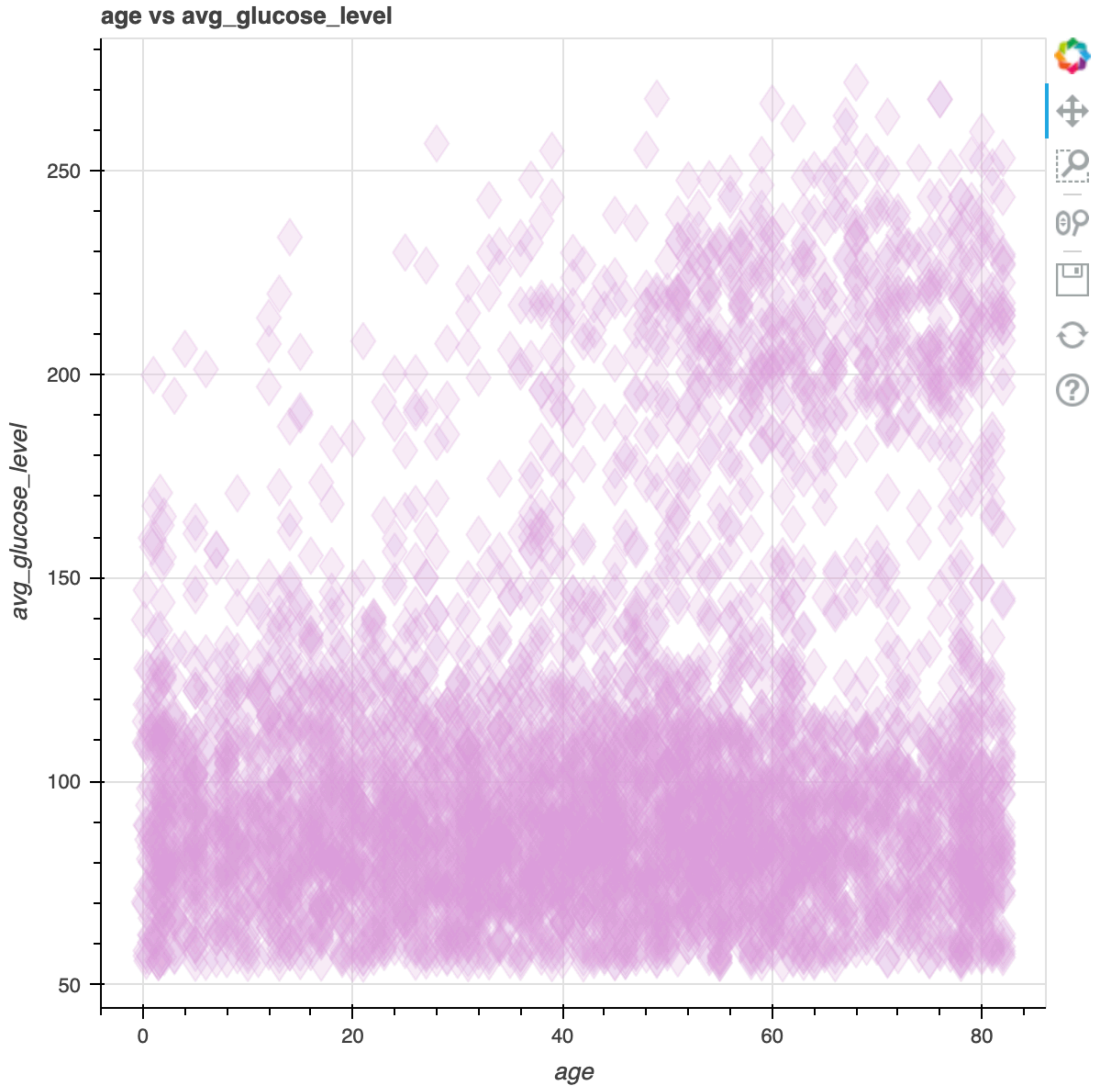
Objective	Complete
Transform and prepare data for creating visualizations	✔
Create simple plots using Bokeh	

Use stroke data for plots

- We are ready to create plots with `df`

```
p = figure(title = "age vs avg_glucose_level",
           x_axis_label = 'age',
           y_axis_label = 'avg_glucose_level',
           width = 600, height = 600)

p.diamond(df['age'],
          df['avg_glucose_level'],
          size = 20,
          color = "plum",
          alpha = 0.2)
show(p)
```



vbar() and hbar()

- To see the count of the categorical levels, we will use the `stroke` variable

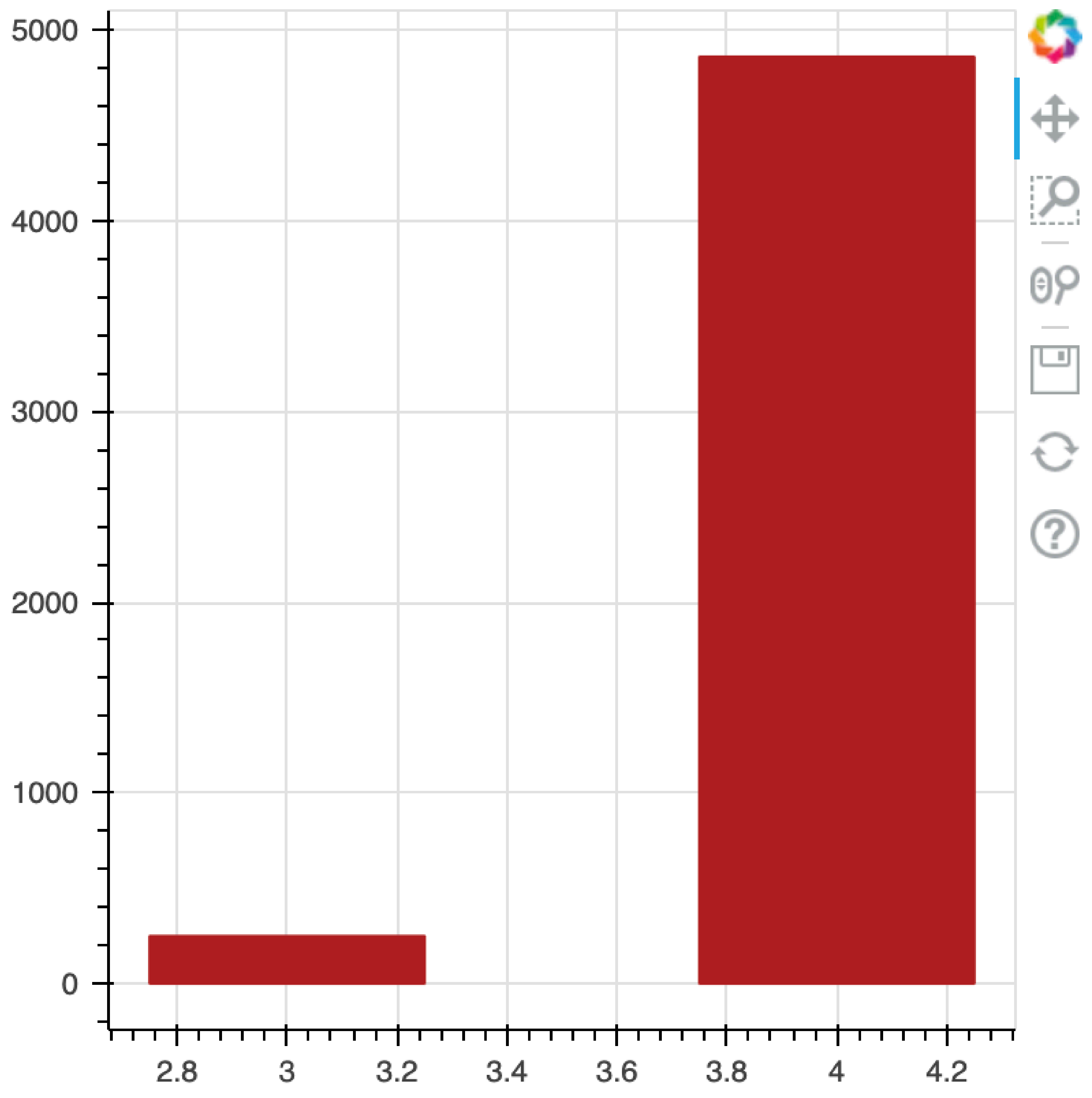
```
df.stroke.value_counts()
```

```
0     4861
1      249
Name: stroke, dtype: int64
```

```
p = figure(width=400, height=400)

p.vbar(x = [0, 1],
       width = 0.2,
       bottom = 0,
       top = df.stroke.value_counts(),
       color = "firebrick")

show(p)
```



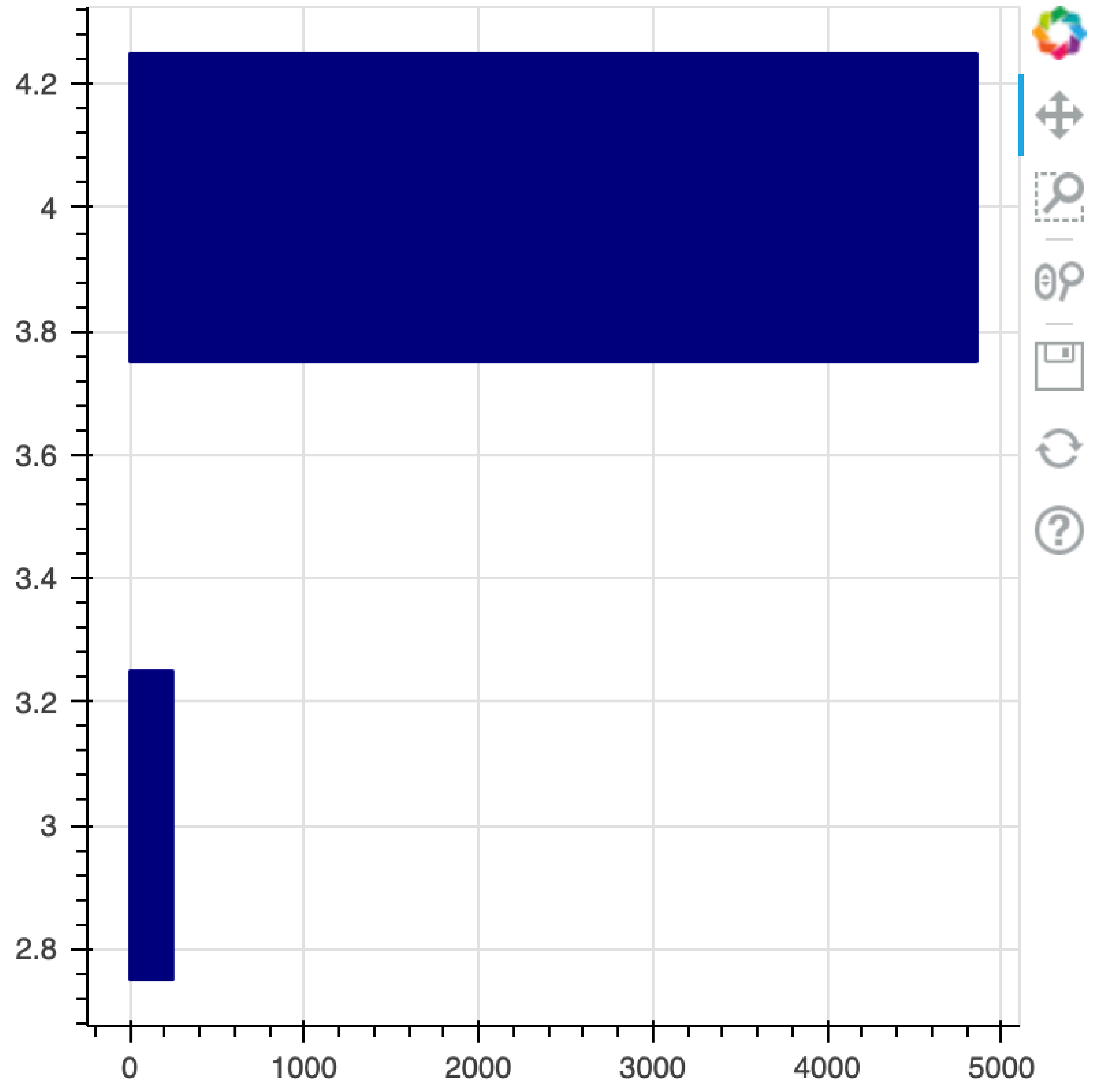
vbar() and hbar() (cont'd)

- We can also create horizontal bar charts using `.hbar()`

```
p = figure(width = 400, height = 400)

p.hbar(y = [0, 1],
       height = 0.2,
       left = 0,
       right = df.stroke.value_counts(),
       color = "navy")

show(p)
```



Markers for categorical data

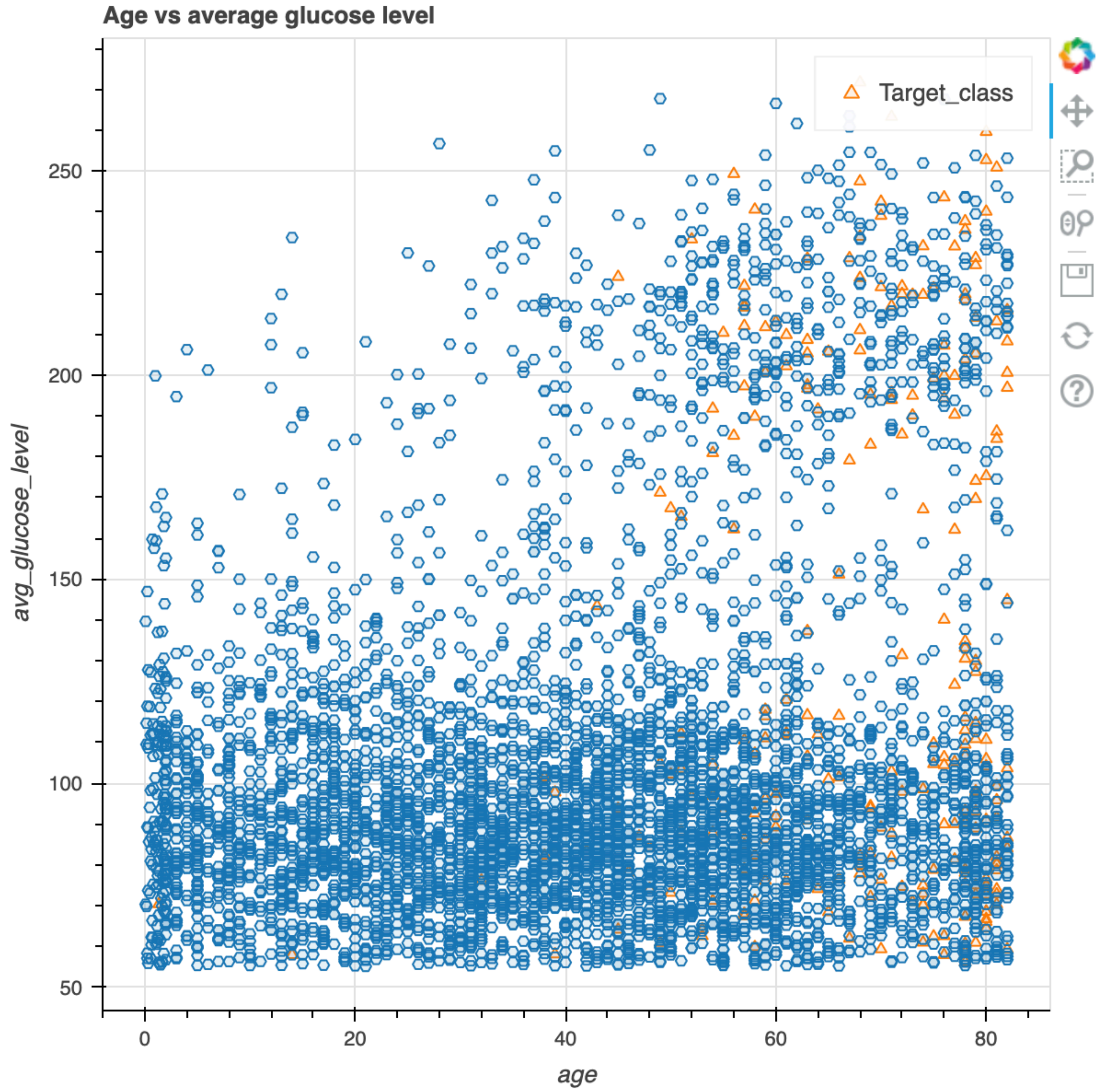
- It is also possible to map categorical data to marker types
- This example shows the use of `factor_mark()` to display different markers or different categories in the input data
- It also demonstrates the use of `factor_cmap()` to color map those same categories

```
LEVELS = ['not_affected', 'affected']
MARKERS = ['hex', 'triangle']

p = figure(title = "Age vs average glucose level",
           x_axis_label = 'age',
           y_axis_label = 'avg_glucose_level')
```

Markers for categorical data (cont'd)

```
p.scatter("age", "avg_glucose_level",
           source = df,
           legend_label = "Target_class",
           fill_alpha = 0.1,
           size = 6,
           marker = factor_mark('Target_class',
                                MARKERS,
                                LEVELS),
           color = factor_cmap('Target_class',
                               'Category10_7',
                               LEVELS))
show(p)
```



Knowledge check

Module completion checklist

Objective	Complete
Transform and prepare data for creating visualizations	✔
Create simple plots using Bokeh	✔

Congratulations on completing this module!

You are now ready to try tasks 3-8 in the Exercise for this topic