STA130_HW03

September 26, 2024

Note: This Jupyter notebook is too large to display on GitHub.

On advice of the prof, it's exported as a PDF instead.

It still available on GitHub to downloaded (but unable to be displayed): https://github.com/jasonli0616/sta130/blob/main/HW/STA130_HW03.ipynb

See post on Piazza: https://piazza.com/class/m0584bs9t4thi/post/141

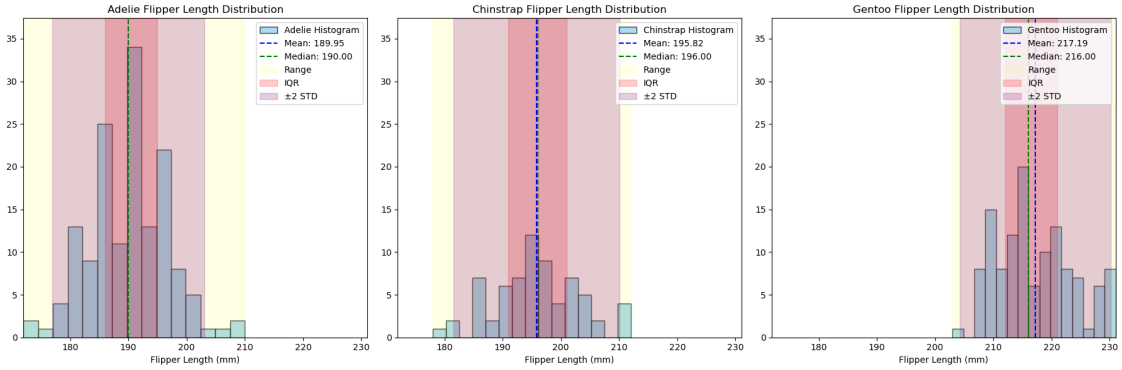
1 STA130 Homework 03 - Jason Li

1.0.1 1.

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset
pingees = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/
  ↪master/penguins.csv")

# Filter out rows with missing flipper_length_mm values
pingees = pingees.dropna(subset=['flipper_length_mm', 'species'])

# Create a list of species for looping
species_list = pingees['species'].unique()

# Determine the overall min and max values for flipper_length_mm
overall_min = pingees['flipper_length_mm'].min()
overall_max = pingees['flipper_length_mm'].max()

# Create subplots: 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharex=True, sharey=True)

# Define colors for mean, median, range, IQR, and ±2 STD
color_map = {
    'mean': 'blue',
    'median': 'green',
```

```
    'range': 'yellow',
    'iqr': 'red',
    '2std': 'purple'
}

# Track maximum frequency for y-axis scaling
max_frequency = 0

# Loop through species and create histograms
for i, species in enumerate(species_list):
    # Filter data for each species
    data = pingees[pingees['species'] == species]['flipper_length_mm']

    # Calculate statistics
    mean = data.mean()
    median = data.median()
    min_value = data.min()
    max_value = data.max()
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    std = data.std()

    # Plot the histogram
    hist = axes[i].hist(data, bins=15, color="skyblue", edgecolor="black",␣
↪alpha=0.7, label=f"{species} Histogram")

    # Update max_frequency with the maximum frequency value
    current_max_frequency = max(hist[0])
    if current_max_frequency > max_frequency:
        max_frequency = current_max_frequency

    # Add vertical lines for mean and median
    axes[i].axvline(mean, color=color_map['mean'], linestyle='--', label=f"Mean:
↪ {mean:.2f}")
    axes[i].axvline(median, color=color_map['median'], linestyle='--',␣
↪label=f"Median: {median:.2f}")

    # Shade the range
    axes[i].axvspan(min_value, max_value, color=color_map['range'], alpha=0.1,␣
↪label="Range")

    # Shade the IQR
    axes[i].axvspan(q1, q3, color=color_map['iqr'], alpha=0.2, label="IQR")

    # Shade ±2 standard deviations from the mean
    axes[i].axvspan(mean - 2*std, mean + 2*std, color=color_map['2std'],␣
↪alpha=0.2, label="±2 STD")
```

```
    # Add titles and labels
    axes[i].set_title(f"{species} Flipper Length Distribution")
    axes[i].set_xlabel("Flipper Length (mm)")
    axes[i].legend(loc='upper right')

# Set consistent x and y axis limits across all subplots
for ax in axes:
    ax.set_xlim(overall_min, overall_max)  # Consistent x-axis range
    ax.set_ylim(0, max_frequency * 1.1)  # Consistent y-axis range with a␣
 ↪little padding
    ax.tick_params(axis='y', which='both', labelleft=True)  # Force y-axis␣
 ↪ticks and labels to show

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```

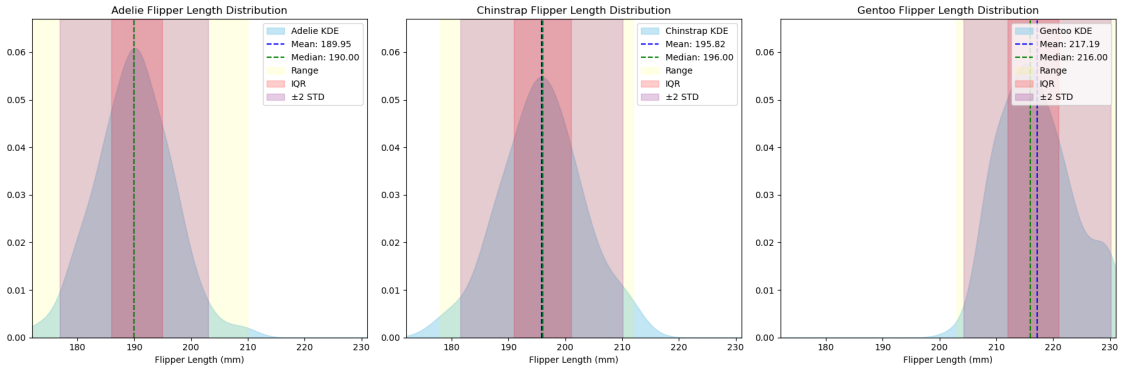


1.0.2 2

[2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import gaussian_kde

# Load the dataset
pingees = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/
 ↪master/penguins.csv")

# Filter out rows with missing flipper_length_mm values
pingees = pingees.dropna(subset=['flipper_length_mm', 'species'])
```

```
# Create a list of species for looping
species_list = pingees['species'].unique()

# Determine the overall min and max values for flipper_length_mm
overall_min = pingees['flipper_length_mm'].min()
overall_max = pingees['flipper_length_mm'].max()

# Create subplots: 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharex=True, sharey=True)

# Define colors for mean, median, range, IQR, and ±2 STD
color_map = {
    'mean': 'blue',
    'median': 'green',
    'range': 'yellow',
    'iqr': 'red',
    '2std': 'purple'
}

# Track maximum density for y-axis scaling
max_density = 0

# Loop through species and create KDE plots
for i, species in enumerate(species_list):
    # Filter data for each species
    data = pingees[pingees['species'] == species]['flipper_length_mm']

    # Calculate statistics
    mean = data.mean()
    median = data.median()
    min_value = data.min()
    max_value = data.max()
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    std = data.std()

    # Ensure there is variability in the data to plot a KDE
    if len(data.unique()) > 1:  # Proceed if there is more than one unique value
        # Use scipy gaussian_kde to compute KDE values
        kde_scipy = gaussian_kde(data)
        x_values = np.linspace(overall_min, overall_max, 1000)
        kde_values = kde_scipy(x_values)

        # Plot the KDE curve
        axes[i].fill_between(x_values, kde_values, color="skyblue", alpha=0.5,␣
 ↪label=f"{species} KDE")
```

```
        # Update max_density with the maximum density value
        current_max_density = kde_values.max()
        if current_max_density > max_density:
            max_density = current_max_density

        # Add vertical lines for mean and median
        axes[i].axvline(mean, color=color_map['mean'], linestyle='--',␣
↪label=f"Mean: {mean:.2f}")
        axes[i].axvline(median, color=color_map['median'], linestyle='--',␣
↪label=f"Median: {median:.2f}")

        # Shade the range
        axes[i].axvspan(min_value, max_value, color=color_map['range'], alpha=0.
↪1, label="Range")

        # Shade the IQR
        axes[i].axvspan(q1, q3, color=color_map['iqr'], alpha=0.2, label="IQR")

        # Shade ±2 standard deviations from the mean
        axes[i].axvspan(mean - 2*std, mean + 2*std, color=color_map['2std'],␣
↪alpha=0.2, label="±2 STD")

    else:
        # If no variability, just display a message
        axes[i].text(0.5, 0.5, "No variability in data",␣
↪horizontalalignment='center',
                     verticalalignment='center', transform=axes[i].transAxes,␣
↪fontsize=12)

    # Add titles and labels
    axes[i].set_title(f"{species} Flipper Length Distribution")
    axes[i].set_xlabel("Flipper Length (mm)")
    axes[i].legend(loc='upper right')

# Set consistent x and y axis limits across all subplots
for ax in axes:
    ax.set_xlim(overall_min, overall_max)  # Consistent x-axis range
    ax.set_ylim(0, max_density * 1.1)  # Consistent y-axis range with a little␣
↪padding
    ax.tick_params(axis='y', which='both', labelleft=True)  # Force y-axis␣
↪ticks and labels to show

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```



1.0.3 3.

See transcript with ChatGPT for more (linked below).

My preference for a data visualization method is the histogram. It is a straightforward way of visualizing data. Unlike the box plot, it doesn't hide the distribution, and it is easy to see multimodal distributions. However, it doesn't make the median, quartiles, min/max, and outliers as clear as box plots. It is intuitive and easy to create, by putting data in bins and visualizing the count. It is not susceptible to over-smoothing like KDEs, though it has its own issues with choosing a bin size.

1.0.4 4.

[3]:
```
from scipy import stats
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np

n = 1500
data1 = stats.uniform.rvs(0, 10, size=n)
data2 = stats.norm.rvs(5, 1.5, size=n)
data3 = np.r_[stats.norm.rvs(2, 0.25, size=int(n/2)), stats.norm.rvs(8, 0.5,␣
 ↪size=int(n/2))]
data4 = stats.norm.rvs(6, 0.5, size=n)

fig = make_subplots(rows=1, cols=4)

fig.add_trace(go.Histogram(x=data1, name='A', nbinsx=30,␣
 ↪marker=dict(line=dict(color='black', width=1))), row=1, col=1)
fig.add_trace(go.Histogram(x=data2, name='B', nbinsx=15,␣
 ↪marker=dict(line=dict(color='black', width=1))), row=1, col=2)
fig.add_trace(go.Histogram(x=data3, name='C', nbinsx=45,␣
 ↪marker=dict(line=dict(color='black', width=1))), row=1, col=3)
fig.add_trace(go.Histogram(x=data4, name='D', nbinsx=15,␣
 ↪marker=dict(line=dict(color='black', width=1))), row=1, col=4)
```

```
fig.update_layout(height=300, width=750, title_text="Row of Histograms")
fig.update_xaxes(title_text="A", row=1, col=1)
fig.update_xaxes(title_text="B", row=1, col=2)
fig.update_xaxes(title_text="C", row=1, col=3)
fig.update_xaxes(title_text="D", row=1, col=4)
fig.update_xaxes(range=[-0.5, 10.5])

for trace in fig.data:
    trace.xbins = dict(start=0, end=10)

# This code was produced by just making requests to Microsoft Copilot
# https://github.com/pointOfive/stat130chat130/blob/main/CHATLOG/wk3/COP/SLS/
 ↪0001_concise_makeAplotV1.md

fig.show() # USE `fig.show(renderer="png")` FOR ALL GitHub and MarkUs␣
 ↪SUBMISSIONS
```

Initial thoughts Here are my initial thoughts on this, just from looking at the graphs visually without doing any calculations or asking ChatGPT.

A and B appear to have a mean of 5. A has a (somewhat) uniform distribution between 0-10, so the mean would seem to be in the middle, 5. B follows a normal distribution centered around 5.

The mean of C could also appear to be 5? It is bimodal, but with data points more broadly distributed between 5-10 and more "focused" around ~3 (probably improper terms, but it gets the idea across).

The variance of A and C are both greater than B and D, however I'm not sure if the variance of A and C are similar. Regardless of whether the value is similar, the histogram shows us that the distribution is not similar.

The variance of D is probably the lowest, since it all the data are near 5 to 8. The mean of D appears to be around 6 to 7. It also follows a normal distribution, but is not similar to B since it is only distributed near 5 to 8 whereas B is distributed through 0 to 10. It also appears to be a little bit skewed, but not sure about that.

In short: A and B have similar means, and maybe C does as well. The mean of D appears to be different. None of the variances look super similar, but I would rank them from least to greatest as D, B, C, A.

Below, I asked ChatGPT to produce code to print the sample mean, sample variance, and sample standard deviation. The ChatGPT transcript is linked below.

[4]:
```
# Function to calculate and print sample statistics
def print_statistics(data, name):
    mean = np.mean(data)
    variance = np.var(data, ddof=1)  # Use ddof=1 for sample variance
    std_dev = np.std(data, ddof=1)   # Use ddof=1 for sample standard deviation
```

```
    print(f"Statistics for {name}:")
    print(f"  Sample Mean: {mean:.4f}")
    print(f"  Sample Variance: {variance:.4f}")
    print(f"  Sample Standard Deviation: {std_dev:.4f}\n")

# Print statistics for each dataset
print_statistics(data1, "Dataset A")
print_statistics(data2, "Dataset B")
print_statistics(data3, "Dataset C")
print_statistics(data4, "Dataset D")
```

```
Statistics for Dataset A:
  Sample Mean: 4.8907
  Sample Variance: 7.9829
  Sample Standard Deviation: 2.8254

Statistics for Dataset B:
  Sample Mean: 5.0735
  Sample Variance: 2.1133
  Sample Standard Deviation: 1.4537

Statistics for Dataset C:
  Sample Mean: 5.0026
  Sample Variance: 9.2188
  Sample Standard Deviation: 3.0362

Statistics for Dataset D:
  Sample Mean: 5.9957
  Sample Variance: 0.2354
  Sample Standard Deviation: 0.4852
```

I was right about the mean of A and B being similar and D being different, and how C was close to A and B but not as close as A and B to each other.

Surprisingly to me, the variance of C is greater than A, however the variance of all four datasets are really not that similar.

1.0.5 5.

Below is ChatGPT's code for creating some histograms. ChatGPT transcript is linked below. This code will be used to show some histograms in my explanation.

[5]:
```
from scipy import stats
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
```

```
# 1. Generate a Normally Distributed Dataset
normal_data = np.random.normal(loc=0, scale=1, size=1000)  # mean=0, std=1

# 2. Generate a Left-Skewed Dataset (Negative Gamma Distribution)
left_skewed_data = -stats.gamma(a=2, scale=2).rvs(size=1000)

# 3. Generate a Right-Skewed Dataset (Gamma Distribution)
right_skewed_data = stats.gamma(a=2, scale=2).rvs(size=1000)

# Function to create a histogram with mean and median labeled
def create_histogram(data, title):
    df = pd.DataFrame({'data': data})
    fig = px.histogram(df, x="data", nbins=30, title=title)

    # Calculate mean and median
    mean = data.mean()
    median = np.median(data)

    # Get the maximum y-value for the histogram
    y_max = max(np.histogram(data, bins=30)[0])

    # Add mean and median lines
    fig.add_trace(go.Scatter(x=[mean, mean], y=[0, y_max],
                             mode='lines', name='Mean', line=dict(color='red',␣
 ↪dash='dash')))
    fig.add_trace(go.Scatter(x=[median, median], y=[0, y_max],
                             mode='lines', name='Median',␣
 ↪line=dict(color='blue', dash='dash')))

    # Show the figure
    fig.show(renderer="png")
```

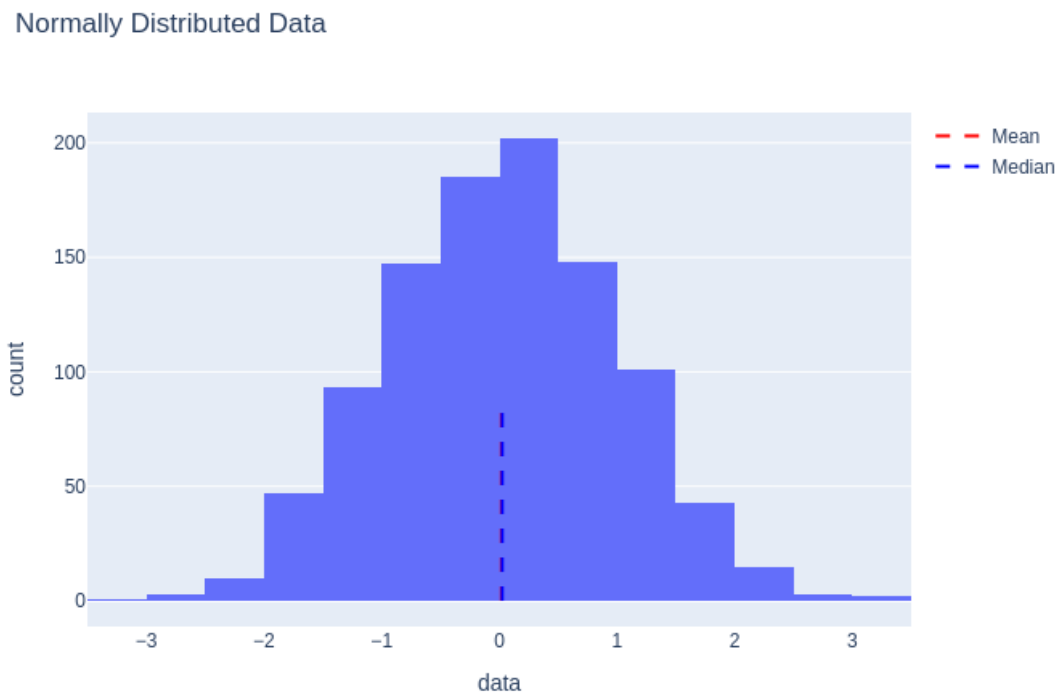
For a normal distribution, the mean and median are the same value, and will be at the center of the histogram.

Medians are simply the middle value in a dataset, whereas the mean is the total sum of values divided by the count. This means that the mean can be affected by outliers or skew, but the median should be more resistant. Regardless of outliers and skew, the middle value is the middle value and the median is less affected than the mean.

In a skew, there are more values that stretch out on one side. This is known as the tail.

Here is a normal distribution. The mean and mean are at the center of the histogram, and are (almost) equal to each other.

[6]:
```
create_histogram(normal_data, "Normally Distributed Data")
```

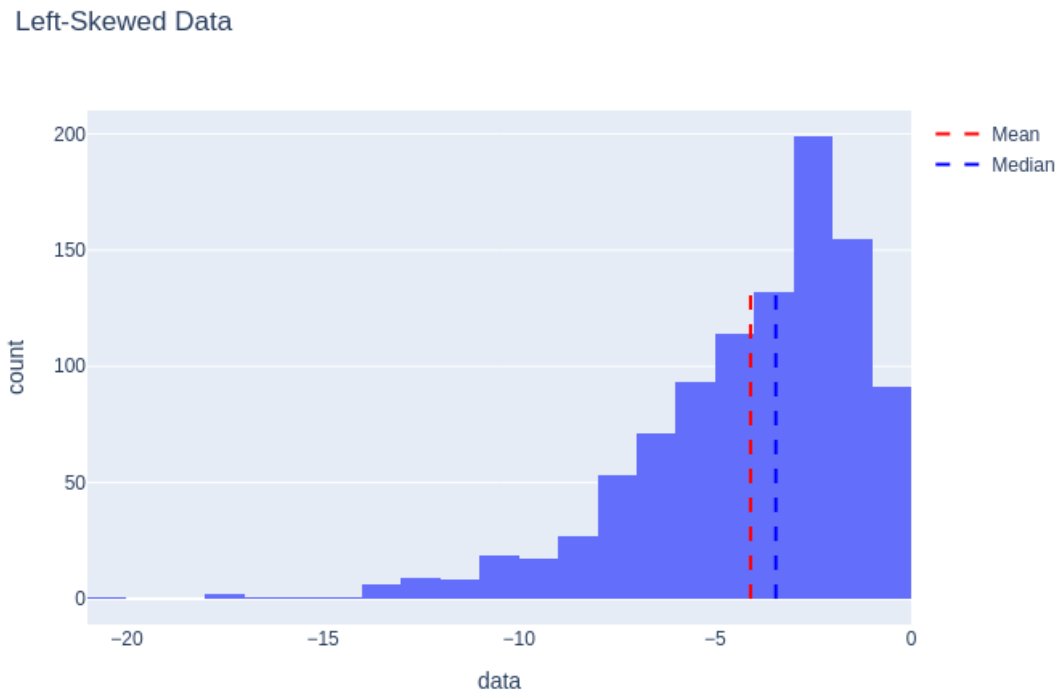


In a left skew (negative skew), the mean is to the left (negative side) of the median.

Most of the data occurs to the right of the median, but there are some extreme values to the left that skew the data and create a tail.

The mean is lower than the median.

[7]:
```
create_histogram(left_skewed_data, "Left-Skewed Data")
```

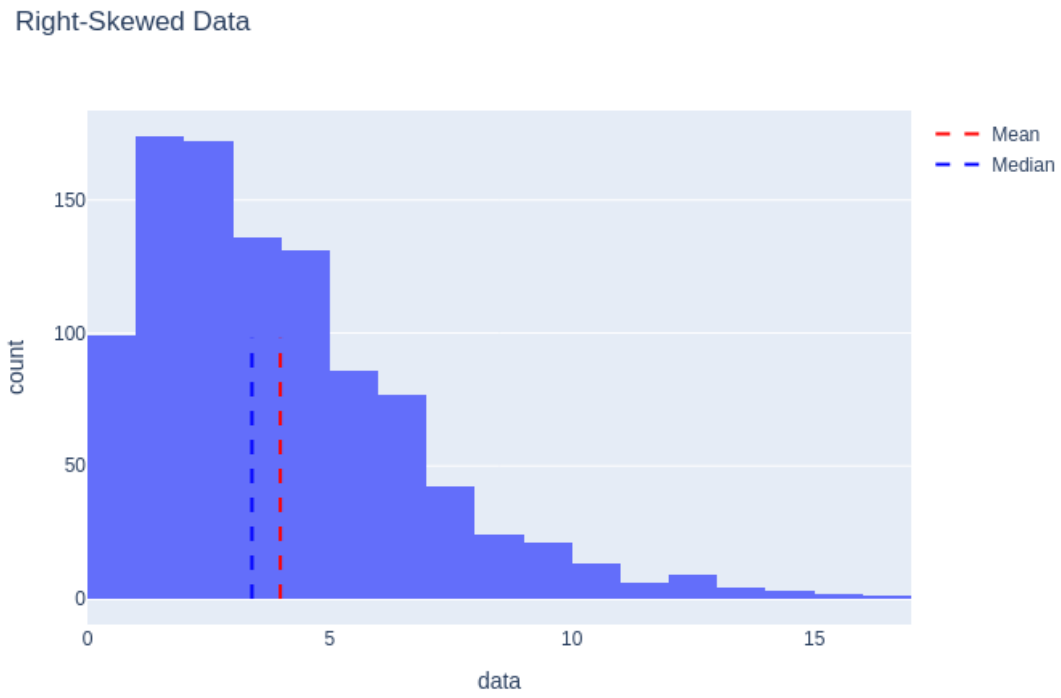


In a right skew (positive skew), the mean is to the right (positive side) of the median.

Most of the data occurs to the left of the median, but there are some extreme values to the right that skew the data and create a tail.

The mean is greater than the median.

[8]:
```
create_histogram(right_skewed_data, "Right-Skewed Data")
```



1.0.6 6.

[9]:
```
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/
↪master/data/2024/2024-08-06/olympics.csv")

df
```

[9]:

	id	name	sex	age	height	weight	\
0	1	A Dijiang	M	24.0	180.0	80.0	
1	2	A Lamusi	M	23.0	170.0	60.0	
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	
…	…	…	..	…	…	…	
271111	135569	Andrzej ya	M	29.0	179.0	89.0	
271112	135570	Piotr ya	M	27.0	176.0	59.0	
271113	135570	Piotr ya	M	27.0	176.0	59.0	
271114	135571	Tomasz Ireneusz ya	M	30.0	185.0	96.0	
271115	135571	Tomasz Ireneusz ya	M	34.0	185.0	96.0	

```
                  team  noc        games  year  season            city  \
0                China  CHN  1992 Summer  1992  Summer       Barcelona
1                China  CHN  2012 Summer  2012  Summer          London
2              Denmark  DEN  1920 Summer  1920  Summer       Antwerpen
3       Denmark/Sweden  DEN  1900 Summer  1900  Summer           Paris
4          Netherlands  NED  1988 Winter  1988  Winter         Calgary
…                  …  …          …   …     …               …
271111        Poland-1  POL  1976 Winter  1976  Winter       Innsbruck
271112          Poland  POL  2014 Winter  2014  Winter           Sochi
271113          Poland  POL  2014 Winter  2014  Winter           Sochi
271114          Poland  POL  1998 Winter  1998  Winter          Nagano
271115          Poland  POL  2002 Winter  2002  Winter  Salt Lake City

                sport                                      event medal
0          Basketball              Basketball Men's Basketball   NaN
1                Judo             Judo Men's Extra-Lightweight   NaN
2            Football                  Football Men's Football   NaN
3          Tug-Of-War              Tug-Of-War Men's Tug-Of-War  Gold
4       Speed Skating         Speed Skating Women's 500 metres   NaN
…                  …                                       …   …
271111           Luge               Luge Mixed (Men)'s Doubles   NaN
271112   Ski Jumping  Ski Jumping Men's Large Hill, Individual   NaN
271113   Ski Jumping       Ski Jumping Men's Large Hill, Team   NaN
271114     Bobsleigh                      Bobsleigh Men's Four   NaN
271115     Bobsleigh                      Bobsleigh Men's Four   NaN

[271116 rows x 15 columns]
```

[10]:
```
df.dtypes
```

[10]:
```
id          int64
name       object
sex        object
age       float64
height    float64
weight    float64
team       object
noc        object
games      object
year        int64
season     object
city       object
sport      object
event      object
medal      object
dtype: object
```

[11]:
```
import plotly.express as px

# Create the histogram for the 'age' column
fig_age = px.histogram(df, x='age', nbins=70, title='Histogram of Age')

# Customize the layout for a cleaner and more aesthetic look
fig_age.update_layout(
    xaxis_title='Age',
    yaxis_title='Count',
    title_x=0.5,  # Center the title
    bargap=0.1,   # Add gap between bars
    plot_bgcolor='rgba(0,0,0,0)',  # Set background to transparent
    paper_bgcolor='white',
    font=dict(family="Arial", size=14)
)

# Set bar color to blue
fig_age.update_traces(marker_color='#1f77b4')  # Blue color

# Display the plot
fig_age.show()
```

The age histogram for the Olympians dataset is as I expected. It follows a normal distribution with a right skew. This is because the majority of Olympians are young, with a smaller amount of much older people.

The distribution looks multimodal, which makes sense since the age of Olympians aren't really affected by other variables such as sex, sport, country, etc.

[12]:
```
# Create the histogram, separating by 'sex'
fig_age_sex = px.histogram(df, x='age', color='sex', nbins=70,
                           title='Histogram of Age by Sex',
                           labels={'sex': 'Sex', 'age': 'Age'},
                           barmode='overlay',
                           color_discrete_sequence=['#1f77b4', '#ff7f0e'])  #␣
 ↪Blue for Male, Red for Female

# Customize layout
fig_age_sex.update_layout(
    xaxis_title='Age',
    yaxis_title='Frequency',
    title_x=0.5,  # Center the title
    bargap=0.1,   # Add gap between bars
    plot_bgcolor='rgba(0,0,0,0)',  # Set background to transparent
    paper_bgcolor='white',
    font=dict(family="Arial", size=14)
)
```

```
# Display the plot
fig_age_sex.show()
```

When splitting the histograms by sex, I notice two things.

1. The distribution is more or less the same for male and female Olympians. Maybe there are more younger female athletes than younger male athletes, but it is not a huge difference.
2. There are *significantly* more male athletes than female athletes.

[13]:
```
# Create the histogram for the 'height' column
fig_height = px.histogram(df, x='height', nbins=70, title='Histogram of Height')

# Customize the layout for a cleaner and more aesthetic look
fig_height.update_layout(
    xaxis_title='Height',
    yaxis_title='Count',
    title_x=0.5,  # Center the title
    bargap=0.1,   # Add gap between bars
    plot_bgcolor='rgba(0,0,0,0)',  # Set background to transparent
    paper_bgcolor='white',
    font=dict(family="Arial", size=14)
)

# Set bar color to red
fig_height.update_traces(marker_color='#ff7f0e')  # Red color

# Display the plot
fig_height.show()
```

This histogram is much less normally distributed. However, there doesn't appear (at first) to be a skew. It does seem to be multimodal.

I'll change the bin size below to see if that shows us anything more.

[14]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Create subplots (2 rows, 1 column)
fig = make_subplots(rows=2, cols=1, subplot_titles=("Histogram of Height␣
 ↪(Bins=20)", "Histogram of Height (Bins=100)"))

# Create the first histogram with 20 bins
hist_20 = go.Histogram(
    x=df['height'],
    nbinsx=20,
    marker_color='#ff7f0e',
    name='Height (20 Bins)',
)
```

```
# Create the second histogram with 100 bins
hist_100 = go.Histogram(
    x=df['height'],
    nbinsx=100,
    marker_color='#1f77b4',
    name='Height (100 Bins)',
)

# Add histograms to the subplots
fig.add_trace(hist_20, row=1, col=1)
fig.add_trace(hist_100, row=2, col=1)

# Update layout with increased height
fig.update_layout(
    title_text='Histograms of Height with Different Bin Sizes',
    xaxis_title='Height',
    yaxis_title='Count',
    plot_bgcolor='rgba(0,0,0,0)',  # Set background to transparent
    paper_bgcolor='white',
    font=dict(family="Arial", size=14),
    height=800  # Increase height of the figure
)

# Update x-axis and y-axis titles for each subplot
fig.update_xaxes(title_text='Height', row=1, col=1)
fig.update_xaxes(title_text='Height', row=2, col=1)
fig.update_yaxes(title_text='Count', row=1, col=1)
fig.update_yaxes(title_text='Count', row=2, col=1)

# Display the plot
fig.show()
```

I will overlay the histogram by sex.

[15]:
```
# Create the histogram, separating by 'sex'
fig_height_sex = px.histogram(df, x='height', color='sex', nbins=80,
                              title='Histogram of Height by Sex',
                              labels={'sex': 'Sex', 'height': 'Height'},
                              barmode='overlay',
                              color_discrete_sequence=['#1f77b4', '#ff7f0e'])␣
 ↪ # Blue for Male, Red for Female

# Customize layout
fig_height_sex.update_layout(
    xaxis_title='Height',
    yaxis_title='Frequency',
```

```
    title_x=0.5,  # Center the title
    bargap=0.1,   # Add gap between bars
    plot_bgcolor='rgba(0,0,0,0)',  # Set background to transparent
    paper_bgcolor='white',
    font=dict(family="Arial", size=14)
)

# Display the plot
fig_height_sex.show()
```

This makes it somewhat better. However, each histogram now appears to be bimodal.

I spent some time messing with different columns, but couldn't figure out exactly what it was. If I had to make a guess, I would say that there are athletes of average heights for sports where height doesn't matter, and athletes of sports where height is an advantage (eg. basketball), and not many athletes in between.

[16]:
```
import plotly.express as px
import plotly.subplots as sp
import pandas as pd

# Example DataFrame
# df = pd.DataFrame({'height': [...], 'sport': [...], 'sex': [...]})  # Use␣
 ↪your actual DataFrame

# Get unique sports
unique_sports = df['sport'].unique()

# Create subplots with more columns
columns = 6  # Set number of columns to 6
rows = (len(unique_sports) + columns - 1) // columns  # Calculate number of rows
fig = sp.make_subplots(rows=rows, cols=columns, subplot_titles=unique_sports)

# Loop through each unique sport and create a histogram
for i, sport in enumerate(unique_sports):
    row = i // columns + 1  # Row index
    col = i % columns + 1    # Column index

    # Filter DataFrame for current sport
    df_sport = df[df['sport'] == sport]

    # Create histogram
    hist = px.histogram(df_sport, x='height', nbins=80,
                        title=sport,
                        labels={'height': 'Height'},
                        color_discrete_sequence=['#1f77b4'])  # Blue for all␣
 ↪sports
```

```
    # Hide number labels
    hist.update_layout(showlegend=False)

    # Add histogram trace to the subplot
    for trace in hist.data:
        fig.add_trace(trace, row=row, col=col)

# Update layout for better display
fig.update_layout(height=150 * rows,  # Adjust height based on number of rows
                  width=150 * columns,  # Set width for the number of columns
                  title_text='Histograms of Height by Sport',
                  title_x=0.5,          # Center the title
                  showlegend=False)

# Display the plot
fig.show()
```

[17]:
```
import plotly.express as px
import plotly.subplots as sp
import pandas as pd

# Example DataFrame
# df = pd.DataFrame({'height': [...], 'sport': [...], 'sex': [...]})  # Use␣
 ↪your actual DataFrame

# Get unique sports
unique_sports = df['sport'].unique()

# Create subplots with more columns
columns = 6  # Set number of columns to 6
rows = (len(unique_sports) + columns - 1) // columns  # Calculate number of rows
fig = sp.make_subplots(rows=rows, cols=columns, subplot_titles=unique_sports)

# Loop through each unique sport and create a histogram
for i, sport in enumerate(unique_sports):
    row = i // columns + 1  # Row index
    col = i % columns + 1    # Column index

    # Filter DataFrame for current sport
    df_sport = df[df['sport'] == sport]

    # Create histogram, split by sex
    hist = px.histogram(df_sport, x='height', nbins=80,
                        title=sport,
                        labels={'height': 'Height'},
                        color='sex',  # Split by sex
```

```
                          color_discrete_sequence=['#1f77b4', '#ff7f0e'])  #␣
 ↪Blue for Male, Red for Female

    # Hide number labels
    hist.update_layout(showlegend=False)

    # Add histogram traces to the subplot
    for trace in hist.data:
        fig.add_trace(trace, row=row, col=col)

# Update layout for better display
fig.update_layout(height=150 * rows,  # Adjust height based on number of rows
                  width=150 * columns,  # Set width for the number of columns
                  title_text='Histograms of Height by Sport, Split by Sex',
                  title_x=0.5,          # Center the title
                  showlegend=False)

# Display the plot
fig.show()
```

Huh, interesting. I guess my guess about the sports thing wasn't right.

[18]:
```
import pandas as pd
import plotly.express as px

# Assuming df is already loaded
# Filter out rows where the medal is not specified (optional)
df_medal = df[df['medal'].isin(['Gold', 'Silver', 'Bronze'])]

# Create the histogram, separating by 'medal'
fig_height_medal = px.histogram(df_medal, x='height', color='medal', nbins=80,
                                title='Histogram of Height by Medal Type',
                                labels={'medal': 'Medal Type', 'height':␣
 ↪'Height'},
                                barmode='overlay',
                                color_discrete_sequence=['#FFD700',␣
 ↪'#C0C0C0', '#CD7F32'])  # Gold, Silver, Bronze colors

# Customize layout
fig_height_medal.update_layout(
    xaxis_title='Height',
    yaxis_title='Frequency',
    title_x=0.5,  # Center the title
    bargap=0.1,   # Add gap between bars
    plot_bgcolor='rgba(0,0,0,0)',  # Set background to transparent
    paper_bgcolor='white',
    font=dict(family="Arial", size=14)
```

```
)

# Display the plot
fig_height_medal.show()
```

Here's a fun one.

Interestingly, the height doesn't seem to impact the medal type.

1.0.7 7.

[19]:
```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x="gdpPercap", y="lifeExp", animation_frame="year",␣
 ↪animation_group="country",
           size="pop", color="continent", hover_name="country",
           log_x=True, size_max=55, range_x=[100,100000], range_y=[25,90])
fig.show()
```

1.0.8 8.

[20]:
```
bn = pd.read_csv('https://raw.githubusercontent.com/hadley/data-baby-names/
 ↪master/baby-names.csv')
bn['name'] = bn['name']+" "+bn['sex'] # make identical boy and girl names␣
 ↪distinct
bn['rank'] = bn.groupby('year')['percent'].rank(ascending=False)
bn = bn.sort_values(['name','year'])
# the next three lines create the increaes or decrease in name prevalence from␣
 ↪the last year
bn['percent change'] = bn['percent'].diff()
new_name = [True]+list(bn.name[:-1].values!=bn.name[1:].values)
bn.loc[new_name,'percentage change'] = bn.loc[new_name,'percent']
bn = bn.sort_values('year')
bn = bn[bn.percent>0.001] # restrict to "common" names

fig = px.scatter(bn,
                 x="percent change",
                 y="rank",
                 animation_frame="year",
                 animation_group="name",
                 size="percent",
                 color="sex",
                 hover_name="name",
                 size_max=50,
                 range_x=[-0.005, 0.005])

fig.update_yaxes(autorange='reversed') # Keep rank 1 at the top
```

```
fig.show()
```

1.1 ChatGPT transcripts and summaries

1.1.1 Transcripts

Part 1: https://chatgpt.com/share/66f484a8-05ec-8010-a311-dce2bf7d92cb

Part 2: https://chatgpt.com/share/66f48576-9214-8010-9460-b2cd787e303d

Part 3: https://chatgpt.com/share/66f4858c-5a28-8010-a8a5-236ed03ce053

Part 4: https://chatgpt.com/share/66f48597-3030-8010-b636-ae8f13caa516

Part 5: https://chatgpt.com/share/66f485a1-6bf4-8010-902c-50ca77e31293

1.1.2 Summaries

Part 1: Summary of Interactions

1. **Handling Overlapping Elements in Plotly Histograms**:
   - You initially provided a dataset and requested that vertical lines and rectangles be added to mark the mean, median, and scale (range, interquartile range, and standard deviations) on histograms of `flipper_length_mm` for different penguin species.
   - There were issues with overlapping elements on the graph, leading to difficulties in distinguishing the lines and rectangles.
   - After troubleshooting, we ensured better visual clarity by separating the species into subplots and improving labels for the mean, median, and scale.
2. **Transition to KDE Plots**:
   - You later requested the same analysis but using Seaborn KDE plots, arranged in a row of three plots for each species. The goal was to visually compare the distributions.
   - We faced some technical issues (e.g., `IndexError`), which we resolved, and we adjusted the axes to maintain consistent scales for easier side-by-side comparison.
3. **Choosing a Visualization Method**:
   - You described the pros and cons of box plots, histograms, and KDEs, noting that each has different strengths and weaknesses.
   - I provided further elaboration on the benefits and limitations of each method to help you choose the best visualization for your needs. I highlighted the simplicity of box plots, the clear depiction of distribution in histograms, and the smooth, flexible representation of KDEs.
4. **Preference for a Visualization Method**:
   - We discussed the benefits of each method. Histograms are great for showing sample size and distribution with fewer assumptions, while KDEs provide a smoother, continuous distribution but can sometimes oversmooth the data depending on bandwidth selection.
5. **Code for Summary Statistics**:
   - You requested help with printing the sample mean, variance, and standard deviation for four generated datasets (`data1`, `data2`, `data3`, and `data4`).
   - I provided a Python code snippet using NumPy to calculate and display these statistics.

Key Takeaways:

- **Visual Clarity**: Ensuring clear labeling and layout of visual elements (lines, rectangles, plots) is crucial for readability.
- **Consistency Across Plots**: When comparing multiple plots (e.g., histograms or KDEs), having consistent axes and scales helps with visual interpretation.
- **Understanding Visualization Methods**: Histograms are useful for raw frequency representation, while KDEs provide a smoother view of the data distribution. Box plots are good for summarizing key statistics (median, quartiles, outliers).
- **Statistical Calculation**: Use of NumPy to compute basic statistical measures like mean, variance, and standard deviation.

This summary covers our discussions, problem-solving, and code solutions for your homework assignment. Let me know if you'd like any further details!

Key Takeaways:

```
Visual Clarity: Ensuring clear labeling and layout of visual elements (lines, rectangles, plots
Consistency Across Plots: When comparing multiple plots (e.g., histograms or KDEs), having cons
Understanding Visualization Methods: Histograms are useful for raw frequency representation, wh
Statistical Calculation: Use of NumPy to compute basic statistical measures like mean, variance
```

Part 2: Here's a summary of our exchanges for your homework assignment:

Interaction Summary:

1. **Relationship Between Mean and Median**:
   - We discussed how the **mean** and **median** are measures of central tendency:
     - In **symmetric distributions**, the mean and median are the same.
     - In **right-skewed distributions** (tail on the right), the **mean** is greater than the **median**.
     - In **left-skewed distributions** (tail on the left), the **mean** is less than the **median**.
2. **Code Debugging**:
   - You shared some code that was producing errors. We fixed it by importing `plotly.express` (required for plotting histograms) and ensuring the renderer `png` is used when showing plots for submission.
   - The code generates random samples from a **gamma distribution**, creates a histogram, and computes the mean and median. We also explained the difference between a **normal gamma distribution** (right-skewed) and a **negative gamma distribution** (left-skewed by multiplying by `-1`).
3. **Simplification of Gamma Distribution**:
   - We simplified the explanation of the **gamma distribution**:
     - It generates mostly **small numbers**, but a few **large numbers** create a long tail.
     - A **negative gamma distribution** (created by multiplying by `-1`) flips the tail to the **left** (i.e., creates a left-skewed distribution).
4. **Explanation of Left and Right Skew**:
   - We defined **right-skewed** and **left-skewed** distributions:
     - **Right-skewed**: Tail stretches to the right (more small numbers, a few big ones).
     - **Left-skewed**: Tail stretches to the left (more big numbers, a few small ones).
   - We discussed how the **mean** and **median** behave differently in skewed distributions.
5. **Extending the Code**:

- You asked to extend the code to generate histograms for **normal**, **left-skewed**, and **right-skewed** distributions, with **mean** and **median** labeled.
- We modified the code to create and display these histograms using `plotly.express` and added dashed lines for the **mean** (red) and **median** (blue).

6. **Adjusting Line Display**:
   - We updated the code to ensure the lines for the **mean** and **median** extend all the way to the top of each histogram by calculating the maximum y-value (`y_max`) for the histogram and adjusting the lines accordingly.

Let me know if you'd like to make any changes or add more details to this summary!

Part 3: Summary of Interactions for Homework Assignment

1. **Initial Request: Pandas Histogram of Age**
   - You initially requested help to create a histogram using the `age` column from the dataset. After importing the dataset, we discussed switching from pandas to Plotly for visualization.
2. **Histogram of Age by Sex**
   - I helped you create histograms separating the data by the `sex` column, using blue for males and red for females, ensuring no opacity and consistent color.
3. **Improving Visual Aesthetic**
   - We focused on making the histograms more visually appealing by centering titles, adjusting layout features (such as background transparency), and using specific color codes.
4. **Custom Histogram for Height**
   - You requested a histogram for the `height` column, and I provided the code to generate one with a red color scheme. We later adjusted the bin sizes and experimented with subplots showing different binning (e.g., 20 and 100 bins).
5. **Stacked Histogram with Increased Vertical Space**
   - You wanted two histograms (20 and 100 bins) stacked vertically. I provided the code using Plotly's `make_subplots` function and adjusted the figure height to give more vertical space.
6. **Switching Age Group Binning**
   - We created a histogram for `height` by specific age groups (`<20`, `20-30`, `30-50`, `>50`). I used the `pd.cut` function to bin the ages, then plotted the resulting histogram, ensuring a clear color distinction between the groups.
7. **Height Histogram by Medal Type**
   - You asked to replace the age grouping with medal types (`Gold`, `Silver`, `Bronze`). I provided a new histogram for height, categorized by medals, with respective colors (gold, silver, bronze).
8. **Unique Sports and Height Histogram**
   - Finally, we extracted the unique sports from the dataset and discussed creating a histogram of height by sport. I provided the list of unique sports from the dataset.

Key Features Implemented:

- **Use of Plotly** for creating aesthetically appealing histograms.
- **Custom binning** of data based on age groups, medal types, and sports.

- **Consistent color scheme** using blue and red for various categories, avoiding opacity, and ensuring clarity in overlapping data.
- **Subplots** and layout adjustments for better visual representation of the data.

This record covers the technical steps we took during the assignment and the visual improvements made to the histograms.

Part 4:

1.1.3 Summary of Interactions:

1. **Original Histogram Plot**:
   - You provided a Plotly code snippet to generate a histogram of height data, split by sex, using blue for males and red for females. This version included customization for layout and styling (e.g., no opacity, center-aligned title, and transparent background).
2. **Switch from 'sex' to 'sport'**:
   - You requested a modification to create histograms grouped by the 'sport' column instead of 'sex'. Additionally, you wanted the histograms to be small to fit within your Jupyter notebook.
   - We initially tried setting the size of the histograms using the `Marker.size` property, which led to an error because it's not applicable for histograms.
   - We resolved this by adjusting the subplot layout and sizing parameters to make the histograms smaller and fit more compactly.
3. **Uniform X and Y Axis Scales**:
   - You requested that all the histograms have uniform x and y axis scales. We attempted to achieve this by:
     - First, setting the axis ranges explicitly using Plotly's `update_xaxes` and `update_yaxes` methods.
     - Although this approach didn't fully meet expectations, we tried setting fixed axis limits (e.g., `range=[120, 220]` for the x-axis), but this also didn't yield the desired result.
4. **Reverting to a Previous Version**:
   - You decided to revert to a simpler version of the code that created small subplots of histograms grouped by sport but without modifying the axis ranges.
   - We restored this previous version successfully, with the subplots sized as needed and grouped by sport.
5. **Adding Sex Split to Sports Histograms**:
   - Finally, you asked to modify the previous version to split each sport's histogram by sex (overlaying male and female distributions). This was accomplished by:
     - Adding `color='sex'` to the Plotly `px.histogram` call, which split the bars by sex and used consistent blue and red colors for male and female categories.

If you need any specific changes or additional details for submission, feel free to ask!

Part 5: Here's a summary of our interactions:

Session Summary: Scatter Plot for Baby Names Dataset

1. **Initial Dataset Setup**:
   - The baby names dataset was read from a CSV file, and identical boy and girl names were distinguished by concatenating the sex column to the names. A ranking for each name was calculated based on the percentage of babies given the name in a given year, and the data was sorted. A new column was created to calculate the year-to-year percentage change in name prevalence, with adjustments for new names. Finally, the dataset was restricted to include only names with a prevalence greater than 0.001%.
2. **Visualization Request**:
   - A `plotly.express.scatter` plot was set up to visualize the dataset, where you specified the following requirements for the scatter plot:
     - **x-axis**: `"percent change"`
     - **y-axis**: `"rank"`
     - **size**: `"percent"`
     - **color**: `"sex"`
     - **animation_frame**: `"year"`
     - **animation_group** and **hover_name**: `"name"`
     - Additional parameters included `size_max=50` and `range_x=[-0.005, 0.005]`. The `log_x` and `range_y` parameters were removed, and the y-axis was set to auto-reverse, so that rank 1 appeared at the top.
3. **Final Code**:
   - The final `px.scatter` plot setup used the following code:

```
fig = px.scatter(bn,
                 x="percent change",
                 y="rank",
                 animation_frame="year",
                 animation_group="name",
                 size="percent",
                 color="sex",
                 hover_name="name",
                 size_max=50,
                 range_x=[-0.005, 0.005])

fig.update_yaxes(autorange='reversed') # Keep rank 1 at the top
fig.show(renderer="png")
```

This record captures the key steps in processing and visualizing the dataset, addressing both the data manipulation and the plot configuration required for the assignment.

[ ]: