STA130 HW3

September 25, 2024

Q1

[3]:
```
import pandas as pd
import plotly.express as px
import seaborn as sns

# Load the penguins dataset
penguins = sns.load_dataset('penguins')

# Drop rows with missing 'flipper_length_mm' values
penguins = penguins.dropna(subset=['flipper_length_mm'])

# Compute statistics for each species using 'agg'
stats = penguins.groupby('species')['flipper_length_mm'].agg(
    mean='mean',
    median='median',
    std='std',
    min_='min',
    max_='max',
    q1=lambda x: x.quantile(0.25),
    q3=lambda x: x.quantile(0.75)
).reset_index()

# Verify the columns of 'stats'
print("Stats DataFrame columns:", stats.columns.tolist())
print(stats)

# Create a histogram faceted by species
fig = px.histogram(penguins, x='flipper_length_mm', facet_col='species',␣
 ↪opacity=0.7, nbins=30)

# Map species to subplot columns
species_order = penguins['species'].unique()
species_to_col = {species: i+1 for i, species in enumerate(species_order)}
print("Species to column mapping:", species_to_col)

# Add lines and rectangles to the plot
for index, row in stats.iterrows():
```

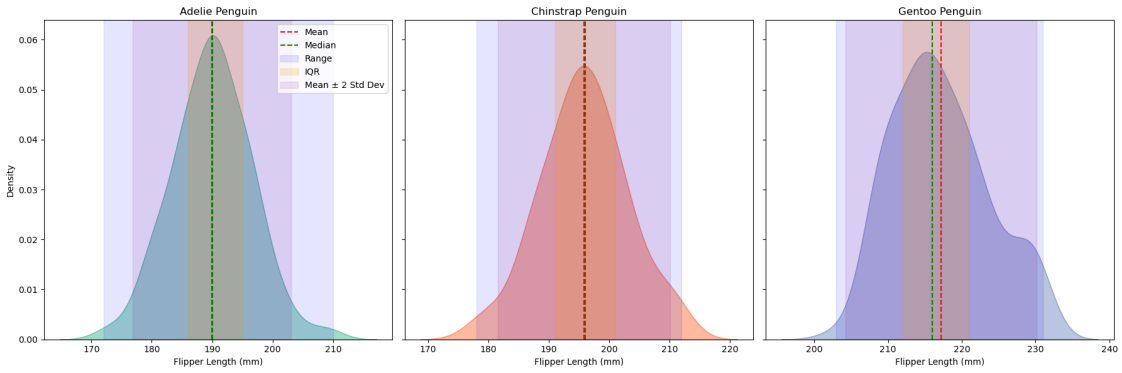
```
    species = row['species']
    mean = row['mean']
    median = row['median']
    std = row['std']
    min_ = row['min_']
    max_ = row['max_']
    q1 = row['q1']
    q3 = row['q3']
    col = species_to_col[species]

    # Add vertical lines for mean and median
    fig.add_vline(x=mean, line_color='red', line_dash='dash', row=1, col=col)
    fig.add_vline(x=median, line_color='green', line_dash='dash', row=1,␣
↪col=col)

    # Add rectangles for range, IQR, and mean ± 2 std
    fig.add_vrect(x0=min_, x1=max_, fillcolor='blue', opacity=0.1,␣
↪line_width=0, row=1, col=col)
    fig.add_vrect(x0=q1, x1=q3, fillcolor='orange', opacity=0.2, line_width=0,␣
↪row=1, col=col)
    x0 = max(mean - 2*std, min_)
    x1 = min(mean + 2*std, max_)
    fig.add_vrect(x0=x0, x1=x1, fillcolor='purple', opacity=0.1, line_width=0,␣
↪row=1, col=col)

# Update layout for better visualization
fig.update_layout(
    title_text='Flipper Length Distribution by Penguin Species',
    xaxis_title='Flipper Length (mm)',
    yaxis_title='Count',
    showlegend=False
)

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

```
Stats DataFrame columns: ['species', 'mean', 'median', 'std', 'min_', 'max_',
'q1', 'q3']
     species        mean  median       std   min_   max_     q1     q3
0     Adelie  189.953642   190.0  6.539457  172.0  210.0  186.0  195.0
1  Chinstrap  195.823529   196.0  7.131894  178.0  212.0  191.0  201.0
2     Gentoo  217.186992   216.0  6.484976  203.0  231.0  212.0  221.0
Species to column mapping: {'Adelie': 1, 'Chinstrap': 2, 'Gentoo': 3}
```

Q2

[4]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the penguins dataset
penguins = sns.load_dataset('penguins')

# Drop rows with missing 'flipper_length_mm' values
penguins = penguins.dropna(subset=['flipper_length_mm'])

# Compute statistics for each species
stats = penguins.groupby('species')['flipper_length_mm'].agg(
    mean='mean',
    median='median',
    std='std',
    min_='min',
    max_='max',
    q1=lambda x: x.quantile(0.25),
    q3=lambda x: x.quantile(0.75)
).reset_index()

# Set up the matplotlib figure with three subplots in a row
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)

# Set a color palette
palette = sns.color_palette("Set2", n_colors=3)

# Iterate over species and plot KDEs
for i, (species, group) in enumerate(penguins.groupby('species')):
    ax = axes[i]

    # Plot KDE for flipper_length_mm
    sns.kdeplot(data=group, x='flipper_length_mm', fill=True, ax=ax,␣
 ↪color=palette[i], alpha=0.6)

    # Get statistics for the current species
    species_stats = stats[stats['species'] == species].iloc[0]
    mean = species_stats['mean']
    median = species_stats['median']
    std = species_stats['std']
    min_ = species_stats['min_']
    max_ = species_stats['max_']
    q1 = species_stats['q1']
    q3 = species_stats['q3']
    x0 = max(mean - 2*std, min_)
    x1 = min(mean + 2*std, max_)
```

```
    # Add vertical lines for mean and median
    ax.axvline(mean, color='red', linestyle='--', label='Mean')
    ax.axvline(median, color='green', linestyle='--', label='Median')

    # Add shaded areas for range, IQR, and mean ± 2 std
    ax.axvspan(min_, max_, color='blue', alpha=0.1, label='Range')
    ax.axvspan(q1, q3, color='orange', alpha=0.2, label='IQR')
    ax.axvspan(x0, x1, color='purple', alpha=0.1, label='Mean ± 2 Std Dev')

    # Set titles and labels
    ax.set_title(f'{species} Penguin')
    ax.set_xlabel('Flipper Length (mm)')
    if i == 0:
        ax.set_ylabel('Density')
    else:
        ax.set_ylabel('')

    # Add legend only to the first subplot
    if i == 0:
        ax.legend()
    else:
        ax.legend().remove()

# Adjust layout
plt.tight_layout()
plt.show()
```



Q3

I prefer Kernel Density Estimators for visualizing data distributions because they provide a smooth and detailed representation of the underlying probability density.

Q4

[5]:
```
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
```

1. Which datasets have similar means and similar variances? Datasets A (data1) and C (data3): Both have a mean of approximately 5. Their variances are similar: Dataset A: Variance 8.33 Dataset C: Variance 9.16
2. Which datasets have similar means but quite different variances? Datasets A (data1) and B (data2):

Both have a mean of 5. Their variances are quite different: Dataset A: Variance 8.33 Dataset B:

Variance = 2.25 Datasets B (data2) and C (data3):

Both have a mean of 5. Their variances are quite different: Dataset B: Variance = 2.25 Dataset C: Variance 9.16

3. Which datasets have similar variances but quite different means? No datasets have both similar variances and different means based on the provided data.
4. Which datasets have quite different means and quite different variances? Datasets D (data4) and A (data1):

Means: Dataset D: Mean = 6 Dataset A: Mean = 5 Variances: Dataset D: Variance = 0.25 Dataset A: Variance 8.33 Datasets D (data4) and C (data3):

Means: Dataset D: Mean = 6 Dataset C: Mean = 5 Variances: Dataset D: Variance = 0.25 Dataset C: Variance 9.16 Datasets D (data4) and B (data2):

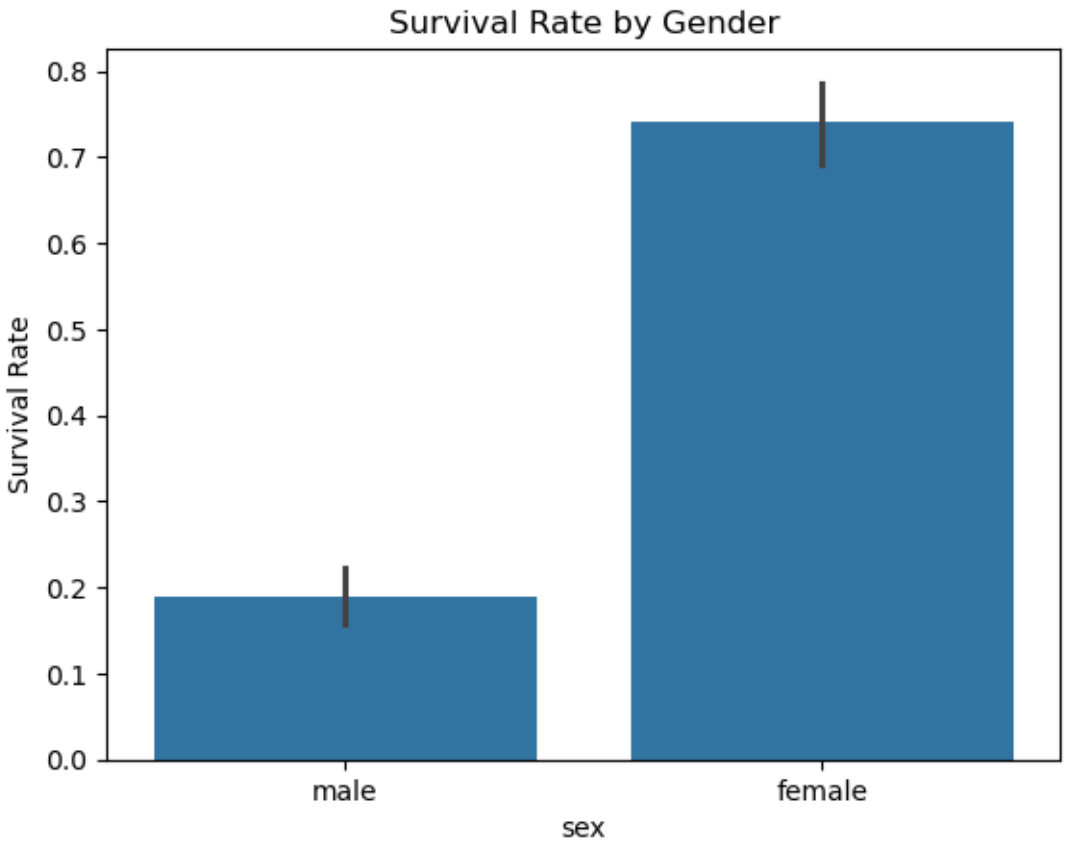
Means: Dataset D: Mean = 6 Dataset B: Mean = 5 Variances: Dataset D: Variance = 0.25 Dataset B: Variance = 2.25

Q6

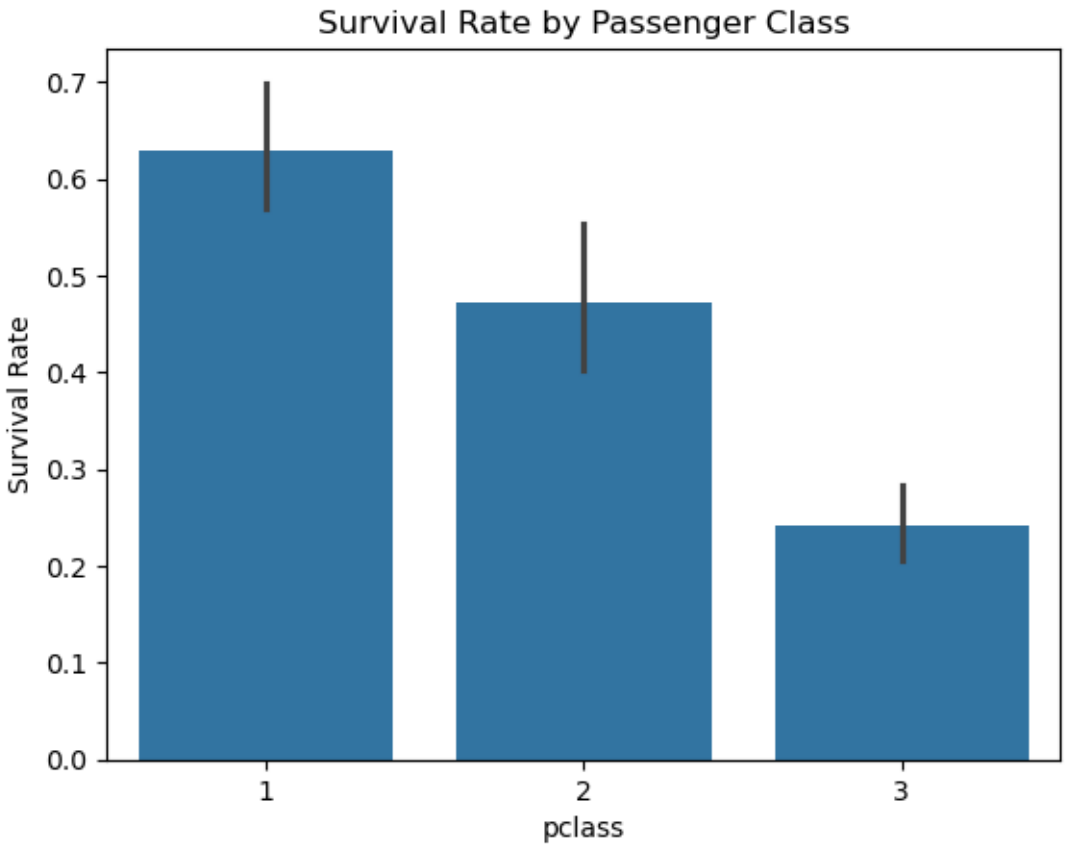
[7]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset and handle missing values
titanic = sns.load_dataset('titanic')
titanic['age'].fillna(titanic['age'].median(), inplace=True)

# Overall survival rate
print(f"Overall Survival Rate: {titanic['survived'].mean() * 100:.2f}%")

# Survival rate by gender
print(titanic.groupby('sex')['survived'].mean() * 100)
sns.barplot(x='sex', y='survived', data=titanic)
plt.title('Survival Rate by Gender')
plt.ylabel('Survival Rate')
plt.show()

# Survival rate by passenger class
print(titanic.groupby('pclass')['survived'].mean() * 100)
sns.barplot(x='pclass', y='survived', data=titanic)
plt.title('Survival Rate by Passenger Class')
plt.ylabel('Survival Rate')
plt.show()

# Survival rate by age group
titanic['age_group'] = pd.cut(titanic['age'], bins=[0,12,18,60,80],␣
 ↪labels=['Child','Teen','Adult','Senior'])
print(titanic.groupby('age_group')['survived'].mean() * 100)
```

```
sns.barplot(x='age_group', y='survived', data=titanic,␣
 ↪order=['Child','Teen','Adult','Senior'])
plt.title('Survival Rate by Age Group')
plt.ylabel('Survival Rate')
plt.show()

# Fare distribution and analysis
sns.histplot(titanic['fare'], kde=True, bins=30)
plt.title('Fare Distribution')
plt.xlabel('Fare')
plt.ylabel('Number of Passengers')
plt.show()

sns.boxplot(x='pclass', y='fare', data=titanic)
plt.title('Fare by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Fare')
plt.show()

sns.boxplot(x='survived', y='fare', data=titanic)
plt.title('Fare vs. Survival')
plt.xlabel('Survived (0=No, 1=Yes)')
plt.ylabel('Fare')
plt.show()

# Family size analysis
titanic['family_size'] = titanic['sibsp'] + titanic['parch'] + 1
print(titanic.groupby('family_size')['survived'].mean() * 100)
sns.barplot(x='family_size', y='survived', data=titanic)
plt.title('Survival Rate by Family Size')
plt.ylabel('Survival Rate')
plt.show()

# Correlation matrix (Fixed)
numeric_cols = titanic.select_dtypes(include=['number']).columns
plt.figure(figsize=(10, 8))
sns.heatmap(titanic[numeric_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
Overall Survival Rate: 38.38%
sex
female    74.203822
male      18.890815
Name: survived, dtype: float64
```

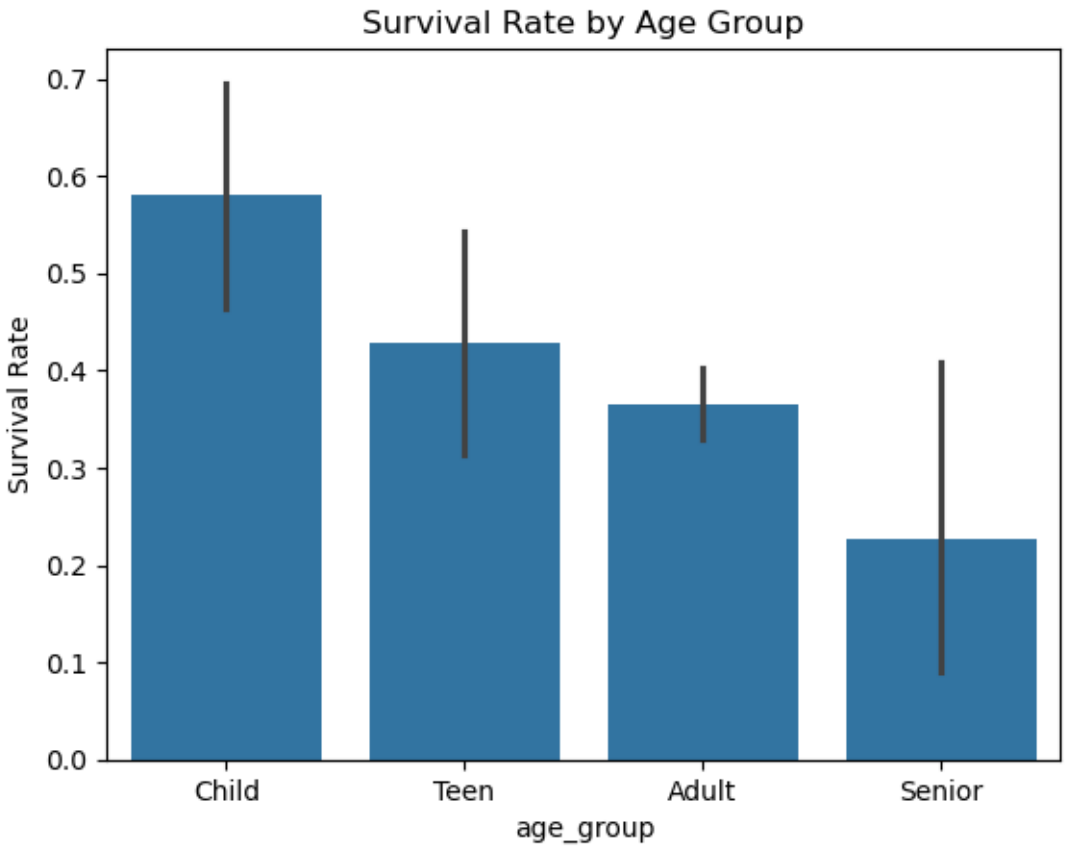


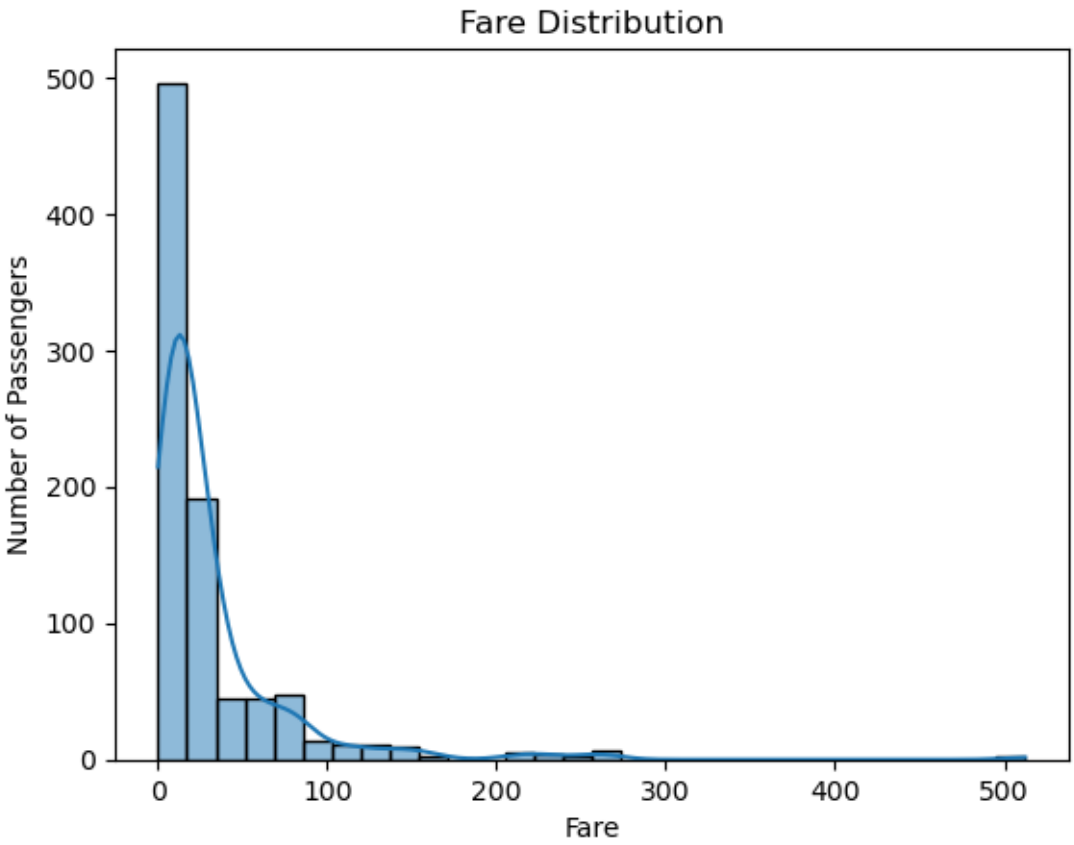

```
pclass
1    62.962963
2    47.282609
3    24.236253
Name: survived, dtype: float64
```

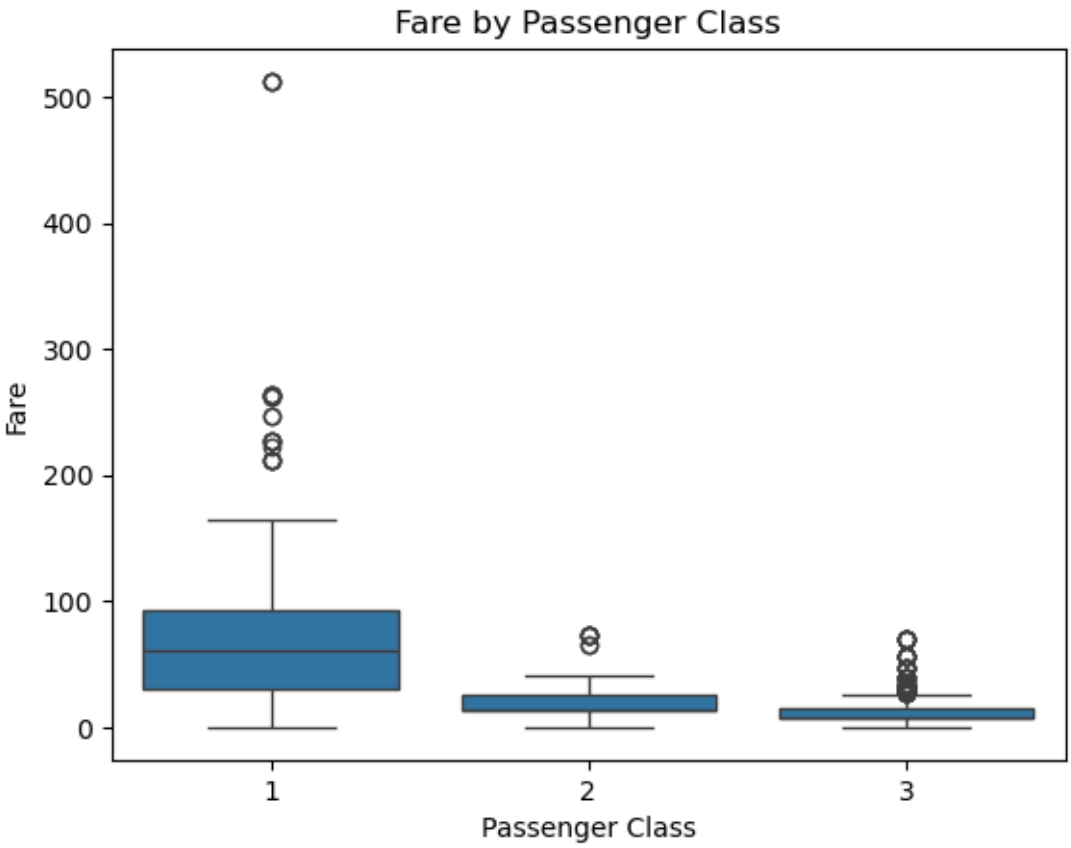


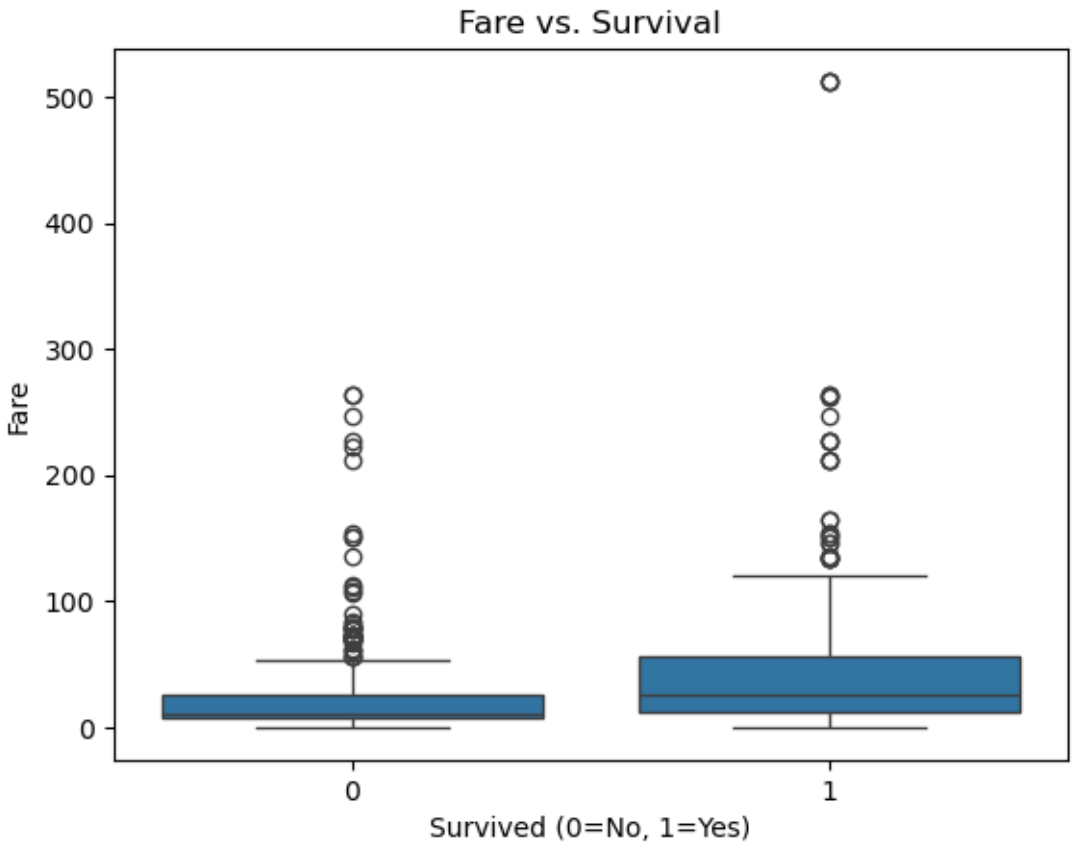

```
age_group
Child     57.971014
Teen      42.857143
Adult     36.575342
Senior    22.727273
Name: survived, dtype: float64
```

```
/tmp/ipykernel_52/1617876369.py:28: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
```

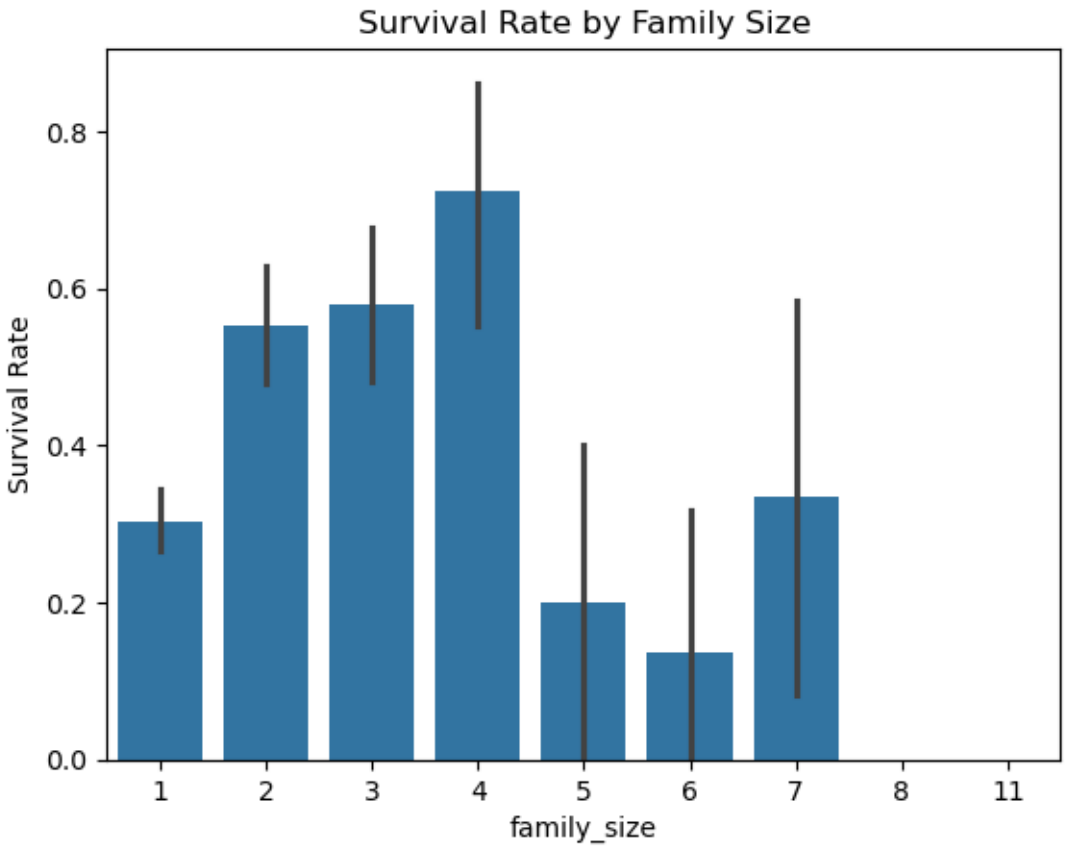

Survival Rate by Age Group
Survival Rate
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Child
Teen
Adult
Senior
age_group

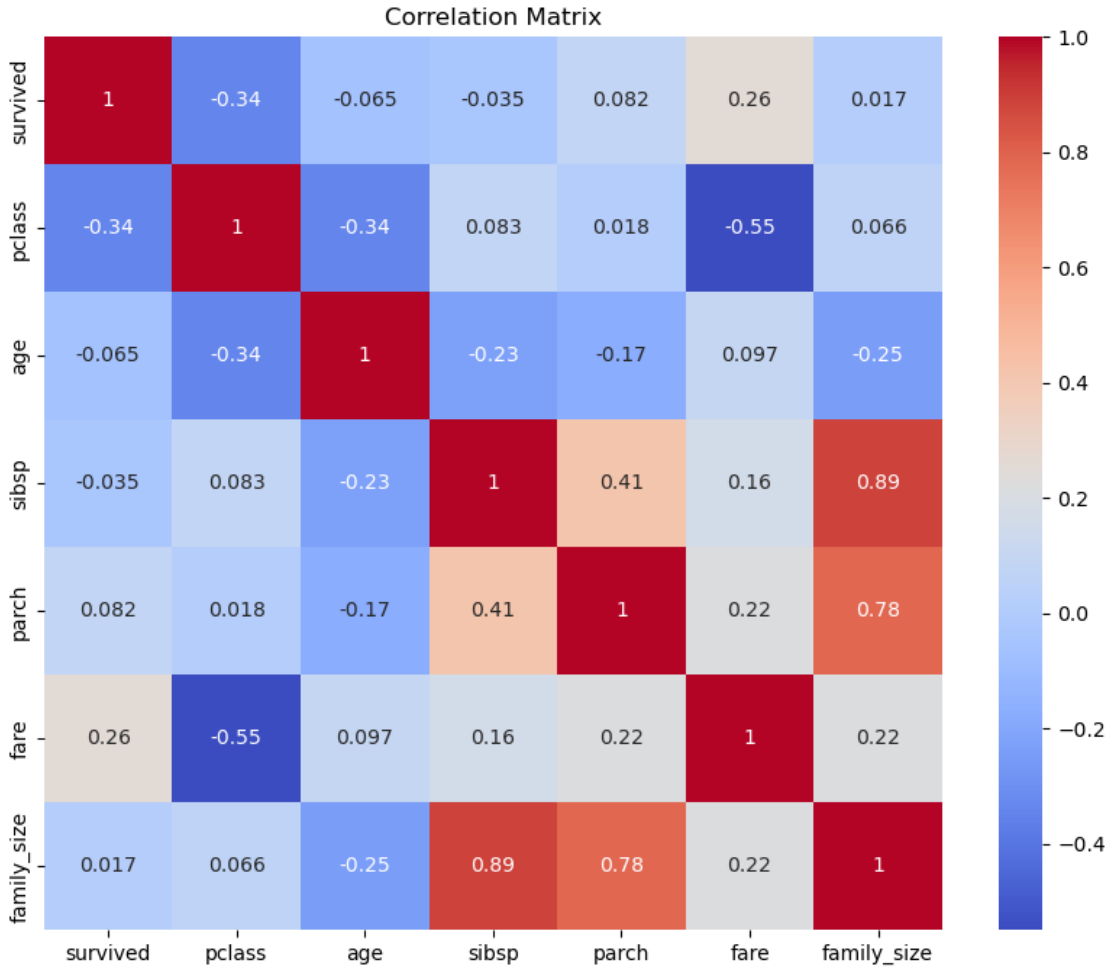

Fare Distribution
Number of Passengers
Fare
0
100
200
300
400
500


Fare by Passenger Class
Fare
500
400
300
200
100
0
1
2
3
Passenger Class




```
family_size
1     30.353818
2     55.279503
3     57.843137
4     72.413793
5     20.000000
6     13.636364
7     33.333333
8      0.000000
11     0.000000
Name: survived, dtype: float64
```


Survival Rate by Family Size
Survival Rate
0.0
0.2
0.4
0.6
0.8
1
2
3
4
5
6
7
8
11
family_size



This code performs the following actions:

Data Loading and Preparation:

Imports necessary libraries. Loads the Titanic dataset. Fills missing age values with the median age. Statistical Calculations and Visualizations:

Calculates and prints overall survival rate. Calculates survival rates by gender, class, age group, and family size. Creates bar plots to visualize survival rates across different categories. Analyzes fare distribution and its relation to class and survival using histograms and box plots. Computes and visualizes the correlation matrix.

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

# Load the dataset
bn = pd.read_csv('https://raw.githubusercontent.com/hadley/data-baby-names/
  ↪master/baby-names.csv')
```

```
# Make identical boy and girl names distinct
bn['name'] = bn['name'] + " " + bn['sex']

# Calculate rank
bn['rank'] = bn.groupby('year')['percent'].rank(ascending=False)
bn = bn.sort_values(['name', 'year'])

# Create the increases or decreases 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, 'percent change'] = bn.loc[new_name, 'percent']
bn = bn.sort_values('year')

# Restrict to common names
bn = bn[bn.percent > 0.001]

# Create the scatter plot
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]
)

# Reverse the y-axis to put rank 1 at the top
fig.update_yaxes(autorange='reversed')

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

[ ]: