The Titanic dataset contains the following columns:

* **PassengerId**: Unique ID for each passenger
* **Survived**: Survival (0 = No, 1 = Yes)
* **Pclass**: Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
* **Name**, **Sex**, **Age**: Passenger’s personal information
* **SibSp**: # of siblings/spouses aboard
* **Parch**: # of parents/children aboard
* **Ticket**, **Fare**, **Cabin**, **Embarked**: Ticket details, fare paid, cabin, and port of embarkation

Let’s now perform some **statistical summaries** and then visualize the data using **Matplotlib**, **Seaborn**, and **Plotly**. We'll cover:

1. Basic statistics
2. Missing values
3. Correlation matrix
4. Distribution of age, fare
5. Survival rate by class, sex, and embarkation port
6. Interactive visuals with Plotly

Starting with statistical summary and missing values analysis. ​​

**📊 Summary Insights:**

**✅ General Stats:**

* Total passengers: **891**
* Survival rate: **~38.4%**
* Mean age: **29.7 years**, with a wide range (0.42 to 80).
* Ticket fare ranges from **0 to 512**, with a high standard deviation indicating outliers.
* **Pclass 3** had the most passengers.

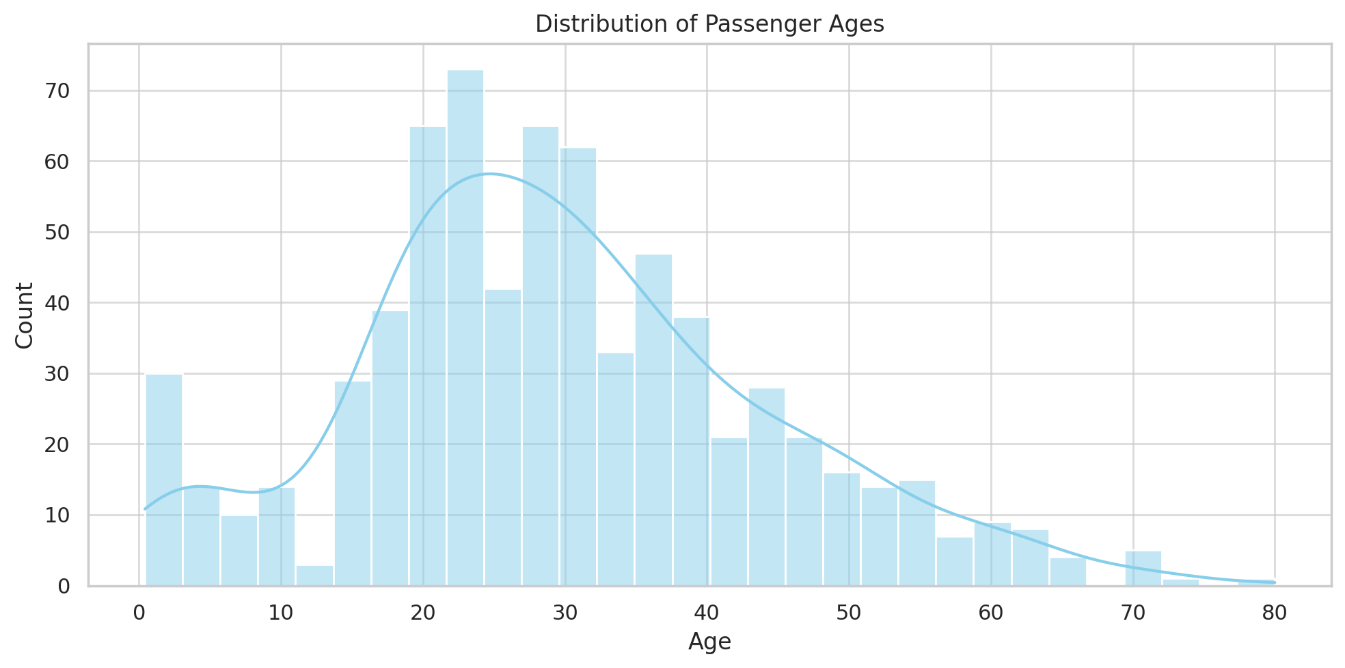
**❗ Missing Values:**

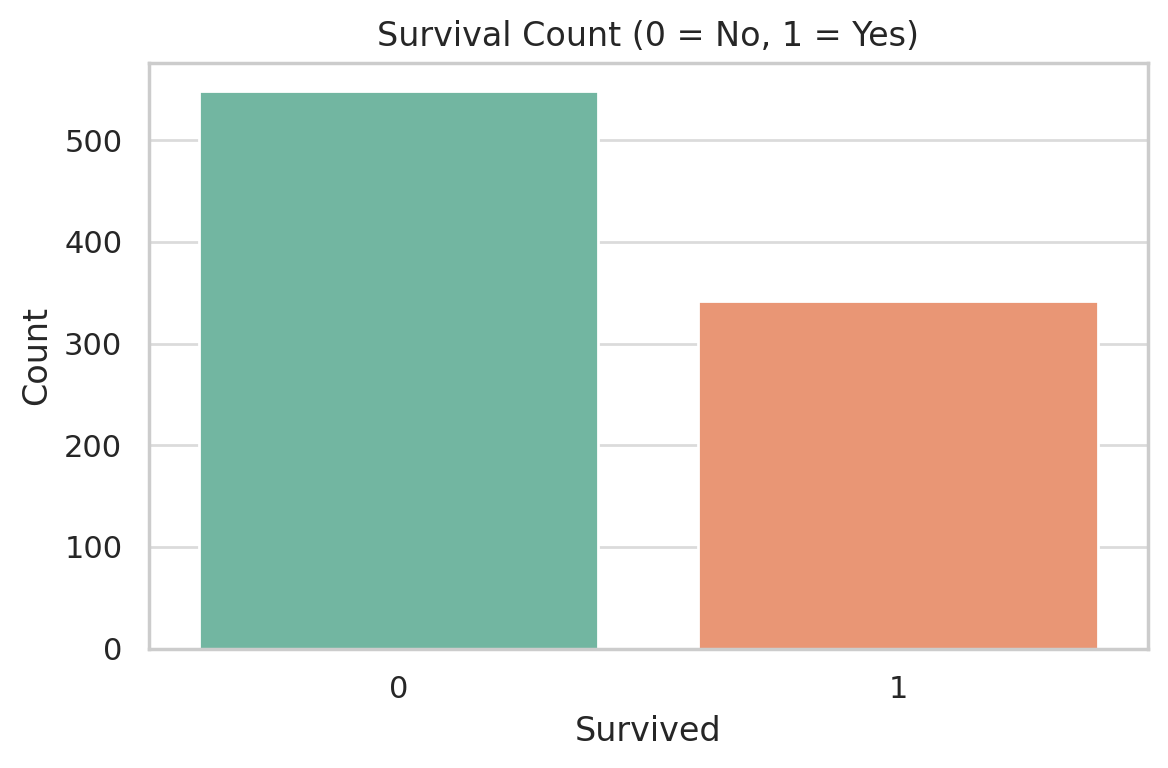
* **Age**: Missing for 177 entries.
* **Cabin**: Missing for 687 entries (most passengers).
* **Embarked**: Missing for 2 entries.

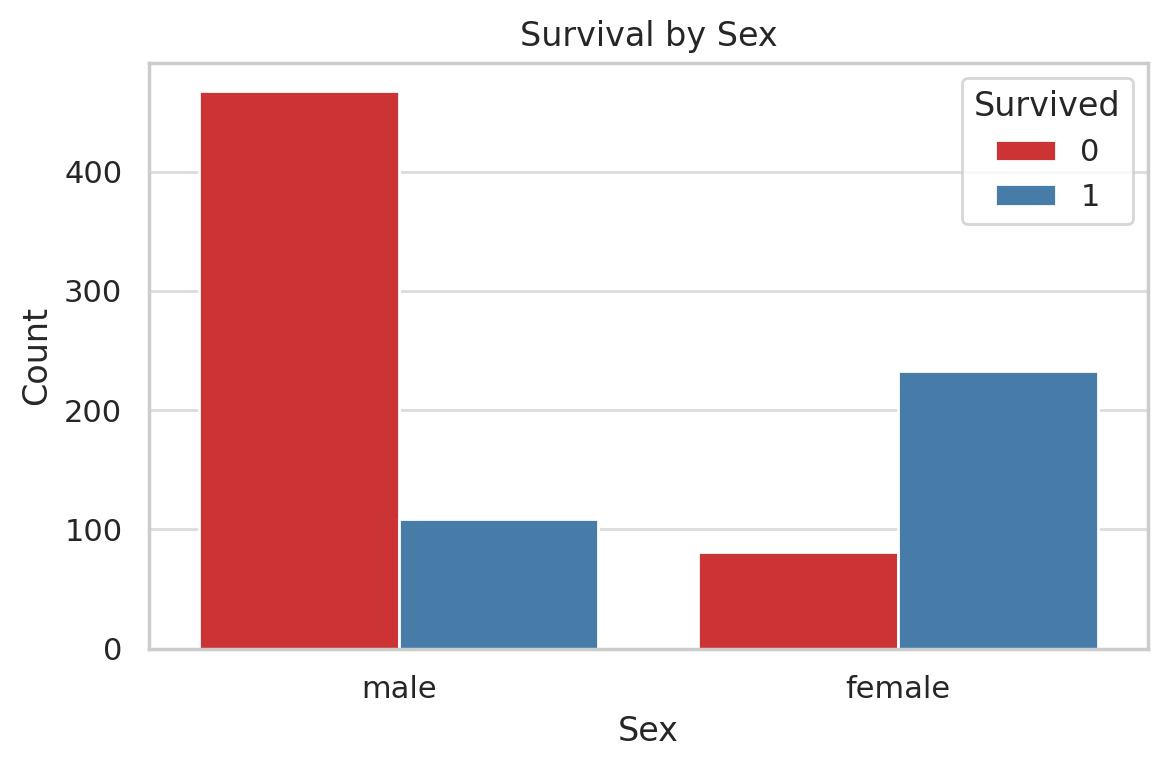
Next up: Let’s create visualizations using **Seaborn**, **Matplotlib**, and **Plotly**, including:

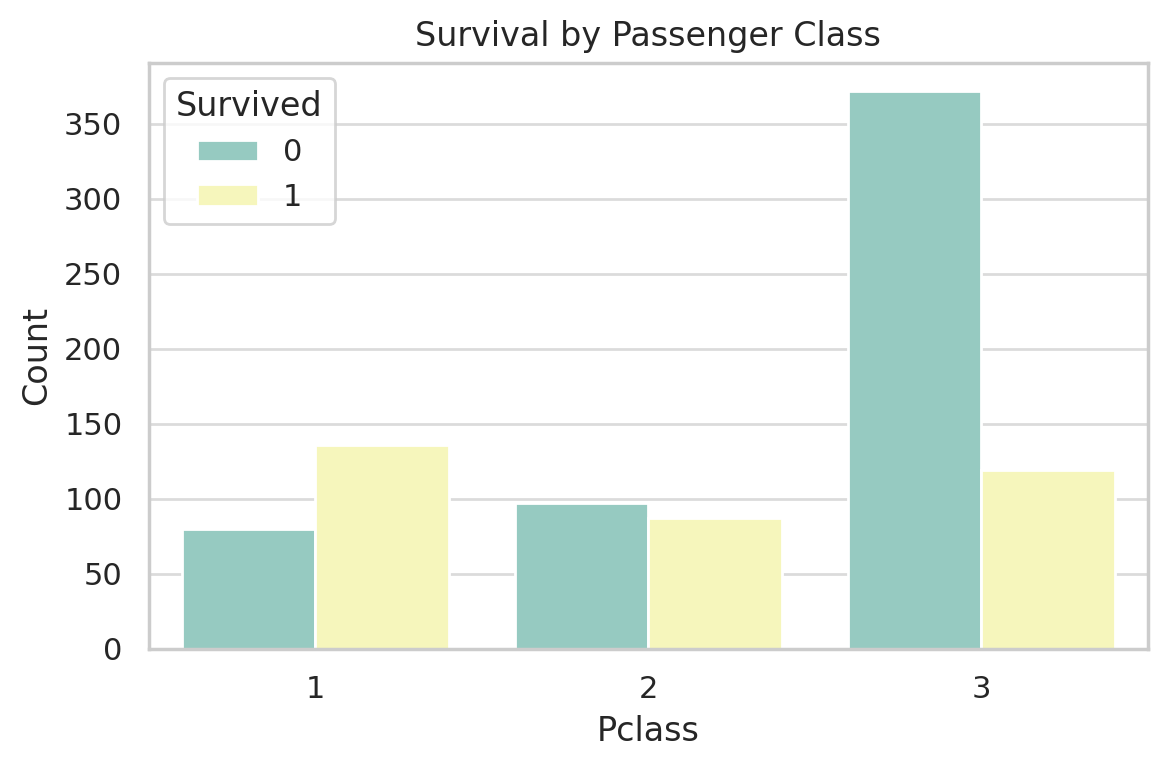
* Age & Fare distributions
* Survival rate by sex, class, and port
* Correlation heatmap
* Interactive survival chart (Plotly)

Starting with Seaborn and Matplotlib plots. ​​









Here’s what we can observe from the visualizations:

1. **Age Distribution**: Most passengers were between 20–40 years old. There's a noticeable dip in very young and elderly passengers.
2. **Survival Count**: Majority of passengers **did not survive** (as expected historically).
3. **Survival by Sex**: Females had a significantly higher survival rate than males.
4. **Survival by Class**: Passengers in **1st class** had the highest survival rate, while those in **3rd class** were least likely to survive.

 **Correlation Heatmap**: Shows the relationships between numeric columns. You’d likely see:

* Strong negative correlation between **Pclass** and **Survived** (lower class, lower survival)
* Positive correlation between **Fare** and **Survived**
* Minor correlations with **Age**, **SibSp**, and **Parch**

 **Plotly Interactive Histogram** (Survival by Class and Sex):

* **Female 1st Class**: Highest survival count
* **Male 3rd Class**: Lowest survival count
* Interactivity lets you hover and filter data dynamically

**1. Correlation Heatmap (Seaborn)**

python

CopyEdit

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv("Titanic-Dataset.csv") # Adjust the path if needed

# Compute correlation matrix (numerical features only)

corr = df.corr(numeric\_only=True)

# Plot heatmap

plt.figure(figsize=(10, 6))

sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.title("Correlation Heatmap of Titanic Dataset")

plt.tight\_layout()

plt.show()

**✅ 2. Interactive Plotly Visualization: Survival by Class and Sex**

python

CopyEdit

import plotly.express as px

# Plotly grouped histogram

fig = px.histogram(

df,

x="Pclass",

color="Survived",

barmode="group",

facet\_col="Sex",

category\_orders={"Pclass": [1, 2, 3]},

color\_discrete\_map={0: "crimson", 1: "mediumseagreen"},

title="Survival Distribution by Class and Sex"

)

fig.update\_layout(

xaxis\_title="Passenger Class",

yaxis\_title="Passenger Count",

bargap=0.2

)

fig.show()

**More Visualization Ideas:**

1. **Age vs Survival (Violin Plot)** – Understand how survival varies across age.
2. **Fare vs Survival (Box Plot)** – Explore fare trends among survivors and non-survivors.
3. **Age vs Fare (Scatter Plot)** – View relationship with a survival hue overlay.

**✅ 1. Violin Plot: Age vs Survival**

python

CopyEdit

plt.figure(figsize=(8, 6))

sns.violinplot(data=df, x="Survived", y="Age", palette="Set2")

plt.title("Age Distribution by Survival")

plt.xlabel("Survived (0 = No, 1 = Yes)")

plt.ylabel("Age")

plt.tight\_layout()

plt.show()

**✅ 2. Box Plot: Fare by Survival**

python

CopyEdit

plt.figure(figsize=(8, 6))

sns.boxplot(data=df, x="Survived", y="Fare", palette="Set1")

plt.title("Fare Distribution by Survival")

plt.xlabel("Survived (0 = No, 1 = Yes)")

plt.ylabel("Fare")

plt.ylim(0, 100) # limit outliers for better visual

plt.tight\_layout()

plt.show()

**✅ 3. Scatter Plot: Age vs Fare, colored by Survival**

python

CopyEdit

plt.figure(figsize=(10, 6))

sns.scatterplot(data=df, x="Age", y="Fare", hue="Survived", palette="coolwarm", alpha=0.7)

plt.title("Fare vs Age colored by Survival")

plt.xlabel("Age")

plt.ylabel("Fare")

plt.tight\_layout()

plt.show()

Here’s a set of **fancy, interactive Plotly visualizations** — great for storytelling, dashboards, or presentations.

**⚡ What You'll Get:**

1. **Animated scatter plot**: Age vs Fare, animated by Pclass
2. **Interactive Sunburst chart**: Class → Sex → Survival breakdown
3. **Treemap**: Passenger hierarchy visualization
4. **Survival rate dashboard layout** (bonus suggestion if you're into Dash)

**✅ 1. Animated Scatter Plot: Age vs Fare by Pclass**

python

CopyEdit

import plotly.express as px

fig = px.scatter(

df,

x="Age",

y="Fare",

animation\_frame="Pclass",

animation\_group="PassengerId",

color="Survived",

hover\_name="Name",

size="Fare",

size\_max=20,

color\_continuous\_scale=["red", "green"],

title="Passenger Distribution: Age vs Fare by Class"

)

fig.update\_layout(

xaxis\_title="Age",

yaxis\_title="Fare",

coloraxis\_colorbar=dict(title="Survival"),

)

fig.show()

**✅ 2. Sunburst Chart: Class → Sex → Survived**

python

CopyEdit

fig = px.sunburst(

df,

path=["Pclass", "Sex", "Survived"],

values=None,

color="Survived",

color\_continuous\_scale=["#e74c3c", "#27ae60"],

title="Survival Breakdown by Class and Sex"

)

fig.show()

**✅ 3. Treemap: Class + Sex + Survival**

python

CopyEdit

fig = px.treemap(

df,

path=["Pclass", "Sex", "Survived"],

color="Survived",

color\_continuous\_scale=["#d63031", "#00b894"],

title="Treemap of Titanic Passengers by Survival"

)

fig.show()

**💡 Bonus: Dash Dashboard Idea**

If you’re into web apps, you could turn this into a mini-dashboard using Dash. It can show:

* Filters for class/sex
* Interactive charts
* Summary statistics + survival KPIs

Here's a **basic Dash dashboard** that visualizes Titanic survival data interactively with dropdown filters and graphs

**Titanic Dashboard Code (Dash + Plotly)**

python

CopyEdit

import dash

from dash import dcc, html, Input, Output

import pandas as pd

import plotly.express as px

# Load the Titanic dataset

df = pd.read\_csv("Titanic-Dataset.csv")

# Initialize Dash app

app = dash.Dash(\_\_name\_\_)

app.title = "Titanic Survival Dashboard"

# App layout

app.layout = html.Div([

html.H1("🛳️ Titanic Survival Dashboard", style={'textAlign': 'center'}),

html.Div([

html.Label("Select Passenger Class"),

dcc.Dropdown(

options=[{'label': f'Class {cls}', 'value': cls} for cls in sorted(df['Pclass'].dropna().unique())],

value=None,

id='class-filter',

placeholder="Filter by Class (optional)"

),

html.Label("Select Sex"),

dcc.Dropdown(

options=[{'label': sex.capitalize(), 'value': sex} for sex in df['Sex'].dropna().unique()],

value=None,

id='sex-filter',

placeholder="Filter by Sex (optional)"

)

], style={'width': '50%', 'margin': 'auto'}),

dcc.Graph(id='survival-bar'),

dcc.Graph(id='fare-age-scatter')

])

# Callback to update graphs

@app.callback(

Output('survival-bar', 'figure'),

Output('fare-age-scatter', 'figure'),

Input('class-filter', 'value'),

Input('sex-filter', 'value')

)

def update\_graphs(selected\_class, selected\_sex):

filtered\_df = df.copy()

if selected\_class:

filtered\_df = filtered\_df[filtered\_df['Pclass'] == selected\_class]

if selected\_sex:

filtered\_df = filtered\_df[filtered\_df['Sex'] == selected\_sex]

bar\_fig = px.histogram(

filtered\_df,

x="Survived",

color="Survived",

color\_discrete\_map={0: "crimson", 1: "mediumseagreen"},

title="Survival Count",

labels={"Survived": "Survived (0 = No, 1 = Yes)"}

)

scatter\_fig = px.scatter(

filtered\_df,

x="Age",

y="Fare",

color="Survived",

hover\_name="Name",

color\_continuous\_scale=["red", "green"],

title="Fare vs Age Scatter Plot"

)

return bar\_fig, scatter\_fig

# Run server

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

**✅ Features:**

* **Dropdowns** to filter by Class and Sex
* **Bar Chart** for survival counts
* **Scatter Plot** to explore Age vs Fare by survival

Let’s upgrade your Titanic dashboard with the following new features:

**🔥 New Features to Add:**

1. **🎯 Pie Chart** – Survival distribution
2. **🎬 Animated Plot** – Fare vs Age by Passenger Class over time
3. **📊 KPI Summary (Optional Bonus)** – Total passengers, survival rate, etc.

**✅ 1. Pie Chart of Survival**

We'll show percentage of survived vs not survived.

**✅ 2. Animated Scatter Plot (Fare vs Age)**

Animated by Pclass to show changes across passenger classes dynamically.

**🔧 Here’s the UPDATED Dash App Code:**

Just replace your app.py code with the one below:

python

CopyEdit

import dash

from dash import dcc, html, Input, Output

import pandas as pd

import plotly.express as px

# Load the Titanic dataset

df = pd.read\_csv("Titanic-Dataset.csv")

# Initialize Dash app

app = dash.Dash(\_\_name\_\_)

app.title = "Titanic Survival Dashboard"

# App layout

app.layout = html.Div([

html.H1("🛳️ Titanic Survival Dashboard", style={'textAlign': 'center'}),

html.Div([

html.Label("Select Passenger Class"),

dcc.Dropdown(

options=[{'label': f'Class {cls}', 'value': cls} for cls in sorted(df['Pclass'].dropna().unique())],

value=None,

id='class-filter',

placeholder="Filter by Class (optional)"

),

html.Label("Select Sex"),

dcc.Dropdown(

options=[{'label': sex.capitalize(), 'value': sex} for sex in df['Sex'].dropna().unique()],

value=None,

id='sex-filter',

placeholder="Filter by Sex (optional)"

)

], style={'width': '50%', 'margin': 'auto'}),

dcc.Graph(id='survival-bar'),

dcc.Graph(id='fare-age-scatter'),

dcc.Graph(id='survival-pie'),

dcc.Graph(id='animated-plot')

])

# Callback to update graphs

@app.callback(

Output('survival-bar', 'figure'),

Output('fare-age-scatter', 'figure'),

Output('survival-pie', 'figure'),

Output('animated-plot', 'figure'),

Input('class-filter', 'value'),

Input('sex-filter', 'value')

)

def update\_graphs(selected\_class, selected\_sex):

filtered\_df = df.copy()

if selected\_class:

filtered\_df = filtered\_df[filtered\_df['Pclass'] == selected\_class]

if selected\_sex:

filtered\_df = filtered\_df[filtered\_df['Sex'] == selected\_sex]

# Bar Chart

bar\_fig = px.histogram(

filtered\_df,

x="Survived",

color="Survived",

color\_discrete\_map={0: "crimson", 1: "mediumseagreen"},

title="Survival Count",

labels={"Survived": "Survived (0 = No, 1 = Yes)"}

)

# Scatter Plot

scatter\_fig = px.scatter(

filtered\_df,

x="Age",

y="Fare",

color="Survived",

hover\_name="Name",

color\_continuous\_scale=["red", "green"],

title="Fare vs Age Scatter Plot"

)

# Pie Chart

pie\_fig = px.pie(

filtered\_df,

names="Survived",

title="Survival Distribution",

color="Survived",

color\_discrete\_map={0: "crimson", 1: "mediumseagreen"},

hole=0.4

)

# Animated Scatter

animated\_fig = px.scatter(

filtered\_df,

x="Age",

y="Fare",

animation\_frame="Pclass",

animation\_group="PassengerId",

color="Survived",

hover\_name="Name",

size="Fare",

size\_max=15,

color\_continuous\_scale=["red", "green"],

title="Animated Scatter: Age vs Fare by Class"

)

return bar\_fig, scatter\_fig, pie\_fig, animated\_fig

# Run server

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

**✅ New Graphs Added:**

* ✅ **Pie Chart**: Percentage of survivors vs non-survivors
* ✅ **Animated Scatter**: Watch how fare vs age spreads across classes!

let’s build a truly **powerful interactive Titanic Dashboard** with:

**🔥 New Features Coming Up:**

1. ✅ **KPI Cards**: Total Passengers, Survival Rate, Average Age
2. ✅ **Multi-page Layout using Tabs**: Clean separation for graphs
3. ✅ **Export Filtered Data to CSV** (Button to download current view)

Let’s start with the upgraded layout & KPIs!

**✅ STEP 1: Install these packages (if not already)**

bash

CopyEdit

pip install dash dash-bootstrap-components pandas plotly

**✅ STEP 2: Full Enhanced Dashboard Code (Copy into app.py)**

python

CopyEdit

import dash

import dash\_bootstrap\_components as dbc

from dash import dcc, html, Input, Output

import pandas as pd

import plotly.express as px

# Load Titanic data

df = pd.read\_csv("Titanic-Dataset.csv")

# Initialize Dash app with Bootstrap theme

app = dash.Dash(\_\_name\_\_, external\_stylesheets=[dbc.themes.BOOTSTRAP])

app.title = "Titanic Dashboard"

# --- KPI Component ---

def generate\_kpis(data):

total = len(data)

survived = data['Survived'].sum()

avg\_age = round(data['Age'].mean(), 1)

survival\_rate = round((survived / total) \* 100, 1) if total else 0

return dbc.Row([

dbc.Col(dbc.Card([

dbc.CardBody([html.H5("Total Passengers", className="card-title"),

html.H2(f"{total}")])

], color="primary", inverse=True)),

dbc.Col(dbc.Card([

dbc.CardBody([html.H5("Survival Rate", className="card-title"),

html.H2(f"{survival\_rate}%")])

], color="success", inverse=True)),

dbc.Col(dbc.Card([

dbc.CardBody([html.H5("Average Age", className="card-title"),

html.H2(f"{avg\_age} yrs")])

], color="info", inverse=True)),

], className="mb-4")

# --- Layout ---

app.layout = dbc.Container([

html.H1("🛳️ Titanic Survival Dashboard", className="text-center my-4"),

dbc.Row([

dbc.Col([

html.Label("Filter by Class"),

dcc.Dropdown(

options=[{'label': f'Class {c}', 'value': c} for c in sorted(df['Pclass'].dropna().unique())],

value=None, id='class-filter', placeholder="Select Class"

)

]),

dbc.Col([

html.Label("Filter by Sex"),

dcc.Dropdown(

options=[{'label': s.capitalize(), 'value': s} for s in df['Sex'].dropna().unique()],

value=None, id='sex-filter', placeholder="Select Sex"

)

])

], className="mb-4"),

html.Div(id='kpi-cards'),

dcc.Tabs(id='tabs', value='tab1', children=[

dcc.Tab(label='Survival & Scatter', value='tab1'),

dcc.Tab(label='Pie & Animated Scatter', value='tab2')

]),

html.Div(id='tab-content'),

html.Br(),

html.A("Download Filtered Data as CSV", id="download-link", download="filtered\_titanic.csv", href="", target="\_blank", className="btn btn-outline-secondary")

], fluid=True)

# --- Callbacks ---

@app.callback(

Output('kpi-cards', 'children'),

Output('tab-content', 'children'),

Output('download-link', 'href'),

Input('class-filter', 'value'),

Input('sex-filter', 'value'),

Input('tabs', 'value')

)

def update\_dashboard(selected\_class, selected\_sex, selected\_tab):

# Filter data

filtered\_df = df.copy()

if selected\_class:

filtered\_df = filtered\_df[filtered\_df['Pclass'] == selected\_class]

if selected\_sex:

filtered\_df = filtered\_df[filtered\_df['Sex'] == selected\_sex]

# Generate KPIs

kpis = generate\_kpis(filtered\_df)

# Create graphs

bar\_fig = px.histogram(filtered\_df, x="Survived", color="Survived", title="Survival Count",

color\_discrete\_map={0: "crimson", 1: "mediumseagreen"})

scatter\_fig = px.scatter(filtered\_df, x="Age", y="Fare", color="Survived",

hover\_name="Name", title="Fare vs Age Scatter Plot",

color\_continuous\_scale=["red", "green"])

pie\_fig = px.pie(filtered\_df, names="Survived", title="Survival Pie Chart",

color="Survived", hole=0.4, color\_discrete\_map={0: "crimson", 1: "mediumseagreen"})

anim\_fig = px.scatter(filtered\_df, x="Age", y="Fare", animation\_frame="Pclass",

animation\_group="PassengerId", size="Fare", color="Survived",

title="Animated Scatter: Age vs Fare", size\_max=15,

color\_continuous\_scale=["red", "green"])

# Download link (in-memory CSV)

csv\_string = filtered\_df.to\_csv(index=False, encoding='utf-8')

csv\_href = "data:text/csv;charset=utf-8," + csv\_string

# Tabs content

if selected\_tab == 'tab1':

tab\_content = html.Div([

dcc.Graph(figure=bar\_fig),

dcc.Graph(figure=scatter\_fig)

])

else:

tab\_content = html.Div([

dcc.Graph(figure=pie\_fig),

dcc.Graph(figure=anim\_fig)

])

return kpis, tab\_content, csv\_href

# --- Run App ---

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

**💥 Features Recap:**

| **Feature** | **Status ✅** |
| --- | --- |
| Dropdown filters | ✅ |
| KPI Cards (stats) | ✅ |
| Tabs for clean layout | ✅ |
| Interactive graphs | ✅ |
| Pie & Animated plots | ✅ |
| CSV Export (filtered) | ✅ |

Let's take this dashboard **to the next level** with:

**🚀 Upcoming Advanced Features:**

1. **📄 Multi-page Navigation** using Dash pages
2. **📋 Paginated Data Table** (easy viewing of passengers)
3. **🤖 Model-Based Survival Prediction** – Input passenger features, get survival probability

**🔧 Here's How We’ll Build It:**

**✅ Page 1 – Dashboard**

* All the great stuff we’ve built so far: filters, KPIs, graphs, CSV export

**✅ Page 2 – Data Table**

* Interactive paginated view
* Search, sort, and filter capabilities

**✅ Page 3 – Predict Survival**

* Input form: Age, Sex, Pclass, Fare, SibSp, Parch
* Predict button → shows survival probability using a trained ML model

Let’s start with **Step 1: Multi-Page Layout** using **Dash Pages**.

**🧱 Step 1: Multi-Page App Setup (3 Pages)**

We’ll structure the app like this:

bash

CopyEdit

titanic\_app/

│

├── app.py # Main entry point

├── pages/

│ ├── dashboard.py # Dashboard page (graphs, KPIs)

│ ├── table.py # Data table page

│ └── predict.py # ML model prediction page

**✅ 1. Install Dash Pages (comes with Dash 2.7+)**

bash

CopyEdit

pip install dash pandas plotly scikit-learn

**✅ 2. Create app.py (Main File)**

python

CopyEdit

import dash

from dash import html, dcc

import dash\_bootstrap\_components as dbc

import os

app = dash.Dash(

\_\_name\_\_,

use\_pages=True,

suppress\_callback\_exceptions=True,

external\_stylesheets=[dbc.themes.BOOTSTRAP]

)

app.title = "Titanic Survival App"

# Navigation Layout

app.layout = dbc.Container([

html.H1("🛳️ Titanic Explorer", className="my-3 text-center"),

dbc.Nav([

dbc.NavLink("Dashboard", href="/", active="exact"),

dbc.NavLink("Data Table", href="/table", active="exact"),

dbc.NavLink("Predict Survival", href="/predict", active="exact"),

], pills=True, justified=True, className="mb-4"),

dash.page\_container # This renders the current page

], fluid=True)

if \_\_name\_\_ == "\_\_main\_\_":

app.run\_server(debug=True)

**✅ 3. Create pages/dashboard.py**

python

CopyEdit

import dash

from dash import html, dcc

import dash\_bootstrap\_components as dbc

dash.register\_page(\_\_name\_\_, path="/", name="Dashboard")

layout = html.Div([

html.H3("📊 Dashboard Page (coming from dashboard.py)"),

html.P("This will host KPIs, filters, graphs, etc.")

])

**✅ 4. Create pages/table.py**

python

CopyEdit

import dash

from dash import html

dash.register\_page(\_\_name\_\_, path="/table", name="Data Table")

layout = html.Div([

html.H3("📋 Data Table Page"),

html.P("This will show the paginated table.")

])

**✅ 5. Create pages/predict.py**

python

CopyEdit

import dash

from dash import html

dash.register\_page(\_\_name\_\_, path="/predict", name="Predict Survival")

layout = html.Div([

html.H3("🔮 Predict Survival Page"),

html.P("This will have a form and ML model to predict survival.")

])

**✅ 6. Run It!**

Make sure you're in the titanic\_app/ folder and then:

bash

CopyEdit

python app.py

You’ll see a top navigation bar with 3 working pages.

**Step 2 – Model-Based Survival Prediction Page**

We'll do 3 things:

**✅ 1. Train a simple ML model (Random Forest)**

**✅ 2. Build an input form in predict.py**

**✅ 3. Predict survival and display the result interactively**

**📁 Folder Structure Reminder:**

cpp

CopyEdit

titanic\_app/

├── app.py

├── Titanic-Dataset.csv

└── pages/

├── dashboard.py

├── table.py

└── predict.py ← We'll edit this

**✅ 1. Train & Save the Model**

**Create model.py** in the main folder:

python

CopyEdit

# model.py

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

import joblib

df = pd.read\_csv("Titanic-Dataset.csv")

# Feature selection and preprocessing

df = df[["Survived", "Pclass", "Sex", "Age", "Fare", "SibSp", "Parch"]].dropna()

df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})

X = df.drop("Survived", axis=1)

y = df["Survived"]

# Train model

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X, y)

# Save model

joblib.dump(model, "titanic\_model.pkl")

➡️ Run it **once** to create titanic\_model.pkl:

bash

CopyEdit

python model.py

**✅ 2. Add Input Form in pages/predict.py**

Now replace predict.py with this:

python

CopyEdit

import dash

from dash import html, dcc, Input, Output, State

import dash\_bootstrap\_components as dbc

import joblib

import numpy as np

dash.register\_page(\_\_name\_\_, path="/predict", name="Predict Survival")

# Load trained model

model = joblib.load("titanic\_model.pkl")

layout = dbc.Container([

html.H3("🔮 Predict Titanic Survival", className="mb-4"),

dbc.Row([

dbc.Col([

dbc.Label("Sex"),

dcc.Dropdown(

options=[

{'label': 'Male', 'value': 0},

{'label': 'Female', 'value': 1}

],

id='sex', value=0

),

dbc.Label("Pclass"),

dcc.Dropdown(

options=[

{'label': '1st Class', 'value': 1},

{'label': '2nd Class', 'value': 2},

{'label': '3rd Class', 'value': 3}

],

id='pclass', value=3

),

dbc.Label("Age"),

dcc.Input(id='age', type='number', value=30, min=0, max=100, className="form-control"),

dbc.Label("Fare"),

dcc.Input(id='fare', type='number', value=32.2, min=0, className="form-control"),

dbc.Label("Siblings/Spouses (SibSp)"),

dcc.Input(id='sibsp', type='number', value=0, min=0, max=10, className="form-control"),

dbc.Label("Parents/Children (Parch)"),

dcc.Input(id='parch', type='number', value=0, min=0, max=10, className="form-control"),

html.Br(),

dbc.Button("Predict", id='predict-btn', color="primary", className="mt-2"),

], width=6)

]),

html.Br(),

html.Div(id='prediction-result', className="mt-4")

])

# Callback for prediction

@dash.callback(

Output('prediction-result', 'children'),

Input('predict-btn', 'n\_clicks'),

State('pclass', 'value'),

State('sex', 'value'),

State('age', 'value'),

State('fare', 'value'),

State('sibsp', 'value'),

State('parch', 'value')

)

def predict\_survival(n\_clicks, pclass, sex, age, fare, sibsp, parch):

if n\_clicks is None:

return ""

features = np.array([[pclass, sex, age, fare, sibsp, parch]])

prob = model.predict\_proba(features)[0][1]

outcome = "💚 Likely Survived" if prob > 0.5 else "💔 Likely Did Not Survive"

return dbc.Alert(f"{outcome} — Probability: {prob:.2%}", color="success" if prob > 0.5 else "danger")

**✅ Final Step: Run the App**

Make sure you're in titanic\_app/ and run:

bash

CopyEdit

python app.py

**Step 3 – Paginated, Searchable Data Table**

We’ll add a professional-grade table to the /table page using dash\_table.

**✅ 1. Install (if not already):**

bash

CopyEdit

pip install dash dash-bootstrap-components pandas

**✅ 2. Replace pages/table.py with the following:**

python

CopyEdit

import dash

from dash import html, dcc, Input, Output

import dash\_table

import pandas as pd

dash.register\_page(\_\_name\_\_, path="/table", name="Data Table")

# Load data

df = pd.read\_csv("Titanic-Dataset.csv")

layout = html.Div([

html.H3("📋 Titanic Passenger Table", className="mb-3"),

dcc.Input(

id='search-input',

type='text',

placeholder='🔍 Search Name...',

debounce=True,

style={"marginBottom": "10px", "width": "300px"}

),

dash\_table.DataTable(

id='passenger-table',

columns=[{"name": col, "id": col} for col in df.columns],

data=df.to\_dict("records"),

page\_size=10,

page\_current=0,

style\_table={'overflowX': 'auto'},

filter\_action="native",

sort\_action="native",

style\_cell={

'textAlign': 'left',

'minWidth': '100px',

'whiteSpace': 'normal'

},

style\_header={

'backgroundColor': '#007BFF',

'color': 'white',

'fontWeight': 'bold'

}

)

])

# Optional: Filter by search

@dash.callback(

Output('passenger-table', 'data'),

Input('search-input', 'value')

)

def filter\_table(search\_value):

if not search\_value:

return df.to\_dict("records")

filtered = df[df['Name'].str.contains(search\_value, case=False, na=False)]

return filtered.to\_dict("records")

**🧠 What You Get:**

| **Feature** | **Included ✅** |
| --- | --- |
| 🔄 Pagination | ✅ 10 rows per page |
| 🔍 Name Search | ✅ Real-time filter |
| 🧮 Sorting | ✅ Click column headers |
| 🎯 Filtering | ✅ Native filter\_action |

**✅ Final Step: Run your app again**

bash

CopyEdit

python app.py