23CSE301 Machine Learning

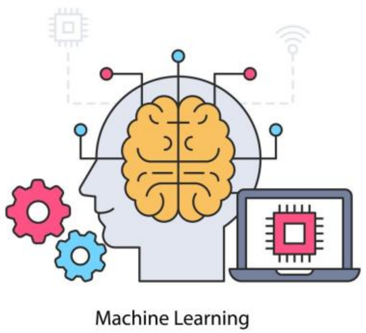
V Sem. CSE B

Practical – Week 2

Course Instructor: Dr. M. Anbazhagan

Pract. #	Experiment Title
P1-P3	• Introduction: Python, Pandas (scikit learn and other libraries) • Pre-processing: Dataset selection, Exploratory Data analysis and Feature engineering; Introduction to Colab/Jupyter Notebook, Pandas(Data Frames); Data Selection (iloc, loc); Sorting, Grouping merge, join, concat; Crosstab; Missing data treatment(fillna, dropna), Converting categorical values, Visualization(Line chart, Bar Chart, Pie chart, Scatter plot, Box plot); Distributions; Summary statistics.
Lab 1 Evaluation (P1 to P3)	
P4	Dimensionality Reduction Technique: PCA
P5	Feature Selection
P6	Regression Algorithms: Linear Regression
P7	Regression Algorithms: Logistic Regression
P8	Classification Algorithms: Decision Tree Classifier
	Classification Algorithms: K-Nearest Neighbor Classifier
Lab 2 Mid-Term exam (P1 to P8)	
P9	Classification Algorithms: Random Forest Classifier, ensemble learning.
P10	Classification Algorithms: Support Vector Machines
P11	Classification Algorithms: Perceptron
P12	Clustering: 1. K-Means Clustering 2. Agglomerative Clustering
Lab 3 Evaluation (P1 to P12)	

The Essential Python Libraries



numpy

Foundation for numerical computing, ML models process data as arrays/matrices.

matplotlib

Basic data visualization, helps understand data trends, distributions, model results.

seaborn

Built on matplotlib, for attractive, statistical plots, useful for data exploration.

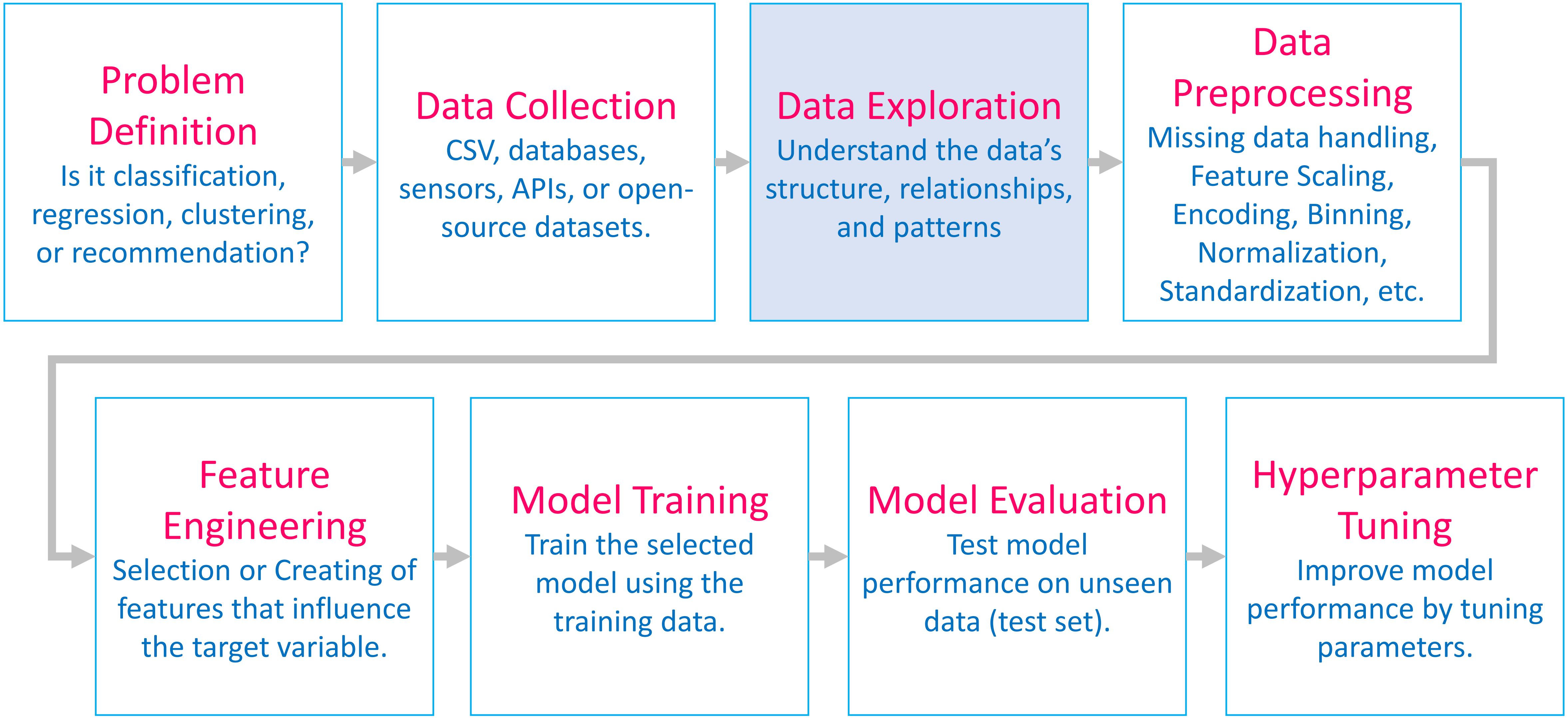
pandas

Primary tool for data loading, manipulation, and preprocessing before ML.

sklearn

Primary ML library for classical ML models, data preprocessing, model evaluation.

End-to-End Machine Learning Pipeline



Data Exploration

1. Reading and viewing a CSV file through a dataframe

```
import pandas as pd
df = pd.read_csv('/content/sample_data/california_housing_test.csv')
df.head()
df.tail(3)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	344700.0
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	176500.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	270500.0
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	330000.0
4	-119.67	36.33	19.0	1241.0	244.0	850.0	237.0	2.9375	81700.0

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
2997	-119.70	36.30	10.0	956.0	201.0	693.0	220.0	2.2895	62000.0
2998	-117.12	34.10	40.0	96.0	14.0	46.0	14.0	3.2708	162500.0
2999	-119.63	34.42	42.0	1765.0	263.0	753.0	260.0	8.5608	500001.0

Data Exploration

2. Knowing the shape/info/columns

```
df.shape
df.info()
df.describe()
```

```
(3000, 9)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           3000 non-null   float64
 1   latitude            3000 non-null   float64
 2   housing_median_age  3000 non-null   float64
 3   total_rooms         3000 non-null   float64
 4   total_bedrooms      3000 non-null   float64
 5   population          3000 non-null   float64
 6   households          3000 non-null   float64
 7   median_income       3000 non-null   float64
 8   median_house_value  3000 non-null   float64
dtypes: float64(9)
memory usage: 211.1 KB
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	3000.000000	3000.00000	3000.000000	3000.000000	3000.000000	3000.000000	3000.00000	3000.000000	3000.00000
mean	-119.589200	35.63539	28.845333	2599.578667	529.950667	1402.798667	489.91200	3.807272	205846.27500
std	1.994936	2.12967	12.555396	2155.593332	415.654368	1030.543012	365.42271	1.854512	113119.68747
min	-124.180000	32.56000	1.000000	6.000000	2.000000	5.000000	2.00000	0.499900	22500.00000
25%	-121.810000	33.93000	18.000000	1401.000000	291.000000	780.000000	273.00000	2.544000	121200.00000
50%	-118.485000	34.27000	29.000000	2106.000000	437.000000	1155.000000	409.50000	3.487150	177650.00000
75%	-118.020000	37.69000	37.000000	3129.000000	636.000000	1742.750000	597.25000	4.656475	263975.00000
max	-114.490000	41.92000	52.000000	30450.000000	5419.000000	11935.000000	4930.00000	15.000100	500001.00000

Data Exploration

3. Knowing columns/index/dtypes

```
df.columns
df.index
df.dtypes
```

```
RangeIndex(start=0, stop=3000, step=1)
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

	0
longitude	float64
latitude	float64
housing_median_age	float64
total_rooms	float64
total_bedrooms	float64
population	float64
households	float64
median_income	float64
median_house_value	float64

dtype: object

Data Exploration

4. Count missing values

```
df.isnull()
df.isnull().sum()
```

	0
longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	0
population	0
households	0
median_income	0
median_house_value	0

dtype: int64

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
2995	False	False	False	False	False	False	False	False	False
2996	False	False	False	False	False	False	False	False	False
2997	False	False	False	False	False	False	False	False	False
2998	False	False	False	False	False	False	False	False	False
2999	False	False	False	False	False	False	False	False	False

3000 rows × 9 columns

Data Exploration

5. Selecting Data

```
df['longitude']  # Single Column
df[['longitude', 'total_bedrooms']] # Multiple Columns
```

	longitude
0	-122.05
1	-118.30
2	-117.81
3	-118.36
4	-119.67
...	...
2995	-119.86
2996	-118.14
2997	-119.70
2998	-117.12
2999	-119.63

3000 rows × 1 columns

dtype: float64

	longitude	total_bedrooms
0	-122.05	661.0
1	-118.30	310.0
2	-117.81	507.0
3	-118.36	15.0
4	-119.67	244.0
...	...	...
2995	-119.86	642.0
2996	-118.14	1082.0
2997	-119.70	201.0
2998	-117.12	14.0
2999	-119.63	263.0

3000 rows × 2 columns

Data Exploration

6. loc[] Vs. iloc[]

- loc[] - Label-based indexing
  - Accesses data using row and column labels (names)
  - Includes the end value in slicing
  - Supports boolean indexing and label ranges
  - df.loc['row_label', 'column_label']
- iloc[] - Integer position-based indexing
  - Accesses data using integer positions (like list indexing)
  - Follows Python-style slicing (end index is exclusive)
  - Does not support label-based access or boolean Series with labels
  - df.iloc[0, 1]

Data Exploration

7. Filtering / Conditional Selection

```
df[df['longitude'] > -120]
df[(df['longitude'] > -120) & (df['latitude'] < 35)]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	176500.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	270500.0
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	330000.0
4	-119.67	36.33							
5	-119.56	36.51							
...	...	...							
2995	-119.86	34.42							
2996	-118.14	34.06							
2997	-119.70	36.30							
2998	-117.12	34.10							
2999	-119.63	34.42							

1805 rows × 9 columns

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	176500.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	270500.0
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	330000.0
9	-118.02	34.08	31.0	2402.0	632.0	2830.0	603.0	2.3333	164200.0
10	-118.24	33.98	45.0	972.0	249.0	1288.0	261.0	2.2054	125000.0
...	...	...	...	...	...	...	...	...	...
2994	-117.93	33.86	35.0	931.0	181.0	516.0	174.0	5.5867	182500.0
2995	-119.86	34.42	23.0	1450.0	642.0	1258.0	607.0	1.1790	225000.0
2996	-118.14	34.06	27.0	5257.0	1082.0	3496.0	1036.0	3.3906	237200.0
2998	-117.12	34.10	40.0	96.0	14.0	46.0	14.0	3.2708	162500.0
2999	-119.63	34.42	42.0	1765.0	263.0	753.0	260.0	8.5608	500001.0

1611 rows × 9 columns

Data Exploration

8. Sorting Data

```
df.sort_values('total_rooms')
df.sort_values('total_bedrooms', ascending=False)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
1115	-116.95	33.86	1.0	6.0	2.0	8.0	2.0	1.6250	55000.0
740	-117.12	32.66	52.0	16.0	4.0	8.0	3.0	1.1250	60000.0
2640	-114.62	33.62	26.0	18.0	3.0	5.0	3.0	0.5360	275000.0
641	-121.04	37.67	16.0	19.0	19.0	166.0	9.0	0.5360	162500.0
2690	-118.06	34.03	36.0	21.0	7.0	21.0	9.0	2.3750	175000.0
...	...	...	...	...	...	...	...	...	...
1597	-117.12	33.49	4.0						
1146	-117.27	33.15	4.0						
292	-116.36	33.78	6.0						
978	-121.53	38.48	5.0						
2429	-117.20	33.58	2.0						

3000 rows × 9 columns

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
1563	-118.44	33.98	21.0	18132.0	5419.0	7431.0	4930.0	5.3359	500001.0
2429	-117.20	33.58	2.0	30450.0	5033.0	9419.0	3197.0	4.5936	174300.0
978	-121.53	38.48	5.0	27870.0	5027.0	11935.0	4855.0	4.8811	212200.0
2014	-117.22	32.86	4.0	16289.0	4585.0	7604.0	4176.0	3.6287	280800.0
292	-116.36	33.78	6.0	24121.0	4522.0	4176.0	2221.0	3.3799	239300.0
...	...	...	...	...	...	...	...	...	...
2690	-118.06	34.03	36.0	21.0	7.0	21.0	9.0	2.3750	175000.0
1355	-117.11	32.66	52.0	25.0	5.0	14.0	9.0	1.6250	118800.0
740	-117.12	32.66	52.0	16.0	4.0	8.0	3.0	1.1250	60000.0
2640	-114.62	33.62	26.0	18.0	3.0	5.0	3.0	0.5360	275000.0
1115	-116.95	33.86	1.0	6.0	2.0	8.0	2.0	1.6250	55000.0

3000 rows × 9 columns

Data Exploration

9. Adding / Modifying Columns & Dropping Rows / Columns

```
df['house_value_IRS'] = df['median_house_value'] * 85
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	house_value_IRS
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	344700.0	29299500.0
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	176500.0	15002500.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	270500.0	22992500.0
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	330000.0	28050000.0
4	-119.67	36.33	19.0	1241.0	244.0	850.0	237.0	2.9375	81700.0	6944500.0

```
df.drop('house_value_IRS', axis=1, inplace=True)
df.drop(1, axis=0, inplace=True)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	344700.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	270500.0
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	330000.0
4	-119.67	36.33	19.0	1241.0	244.0	850.0	237.0	2.9375	81700.0
5	-119.56	36.51	37.0	1018.0	213.0	663.0	204.0	1.6635	67000.0

Data Exploration

10. Group By

```
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
df.groupby('target').mean()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
target				
0	5.006	3.428	1.462	0.246
1	5.936	2.770	4.260	1.326
2	6.588	2.974	5.552	2.026

Data Exploration

11. Merging Dataframes

```
df1 = pd.read_csv('/content/sample_data/california_housing_train.csv')
df2 = pd.read_csv('/content/sample_data/california_housing_test.csv')


pd.merge(df1, df2, on='latitude')
```

	longitude_x	latitude	housing_median_age_x	total_rooms_x	total_bedrooms_x	population_x	households_x	median_income_x	median_house_value_x	longitude_y	housing_media
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0	-119.18	
1	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0	-118.41	
2	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0	-118.86	
3	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0	-118.30	
4	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0	-118.45	
...	...	...	...	...	...	...	...	...	...	...	
209003	-124.19	40.78	37.0	1371.0	319.0	640.0	260.0	1.8242	70000.0	-124.15	
209004	-124.19	40.77	30.0	2975.0	634.0	1367.0	583.0	2.4420	69000.0	-124.16	
209005	-124.19	40.77	30.0	2975.0	634.0	1367.0	583.0	2.4420	69000.0	-123.28	
209006	-124.21	40.75	32.0	1218.0	331.0	620.0	268.0	1.6528	58100.0	-122.31	
209007	-124.30	41.80	19.0	2672.0	552.0	1298.0	478.0	1.9797	85800.0	-124.17	

209008 rows × 17 columns

Data Exploration

12. Renaming columns and Aggregate Functions

```
df.rename(columns={'sepal length (cm)': 'SepLen'}, inplace=True)


df['SepLen'].mean()
df['SepLen'].sum()
df['SepLen'].max()
df['SepLen'].min()
df['SepLen'].count()
df['SepLen'].std()
df['SepLen'].var()
```

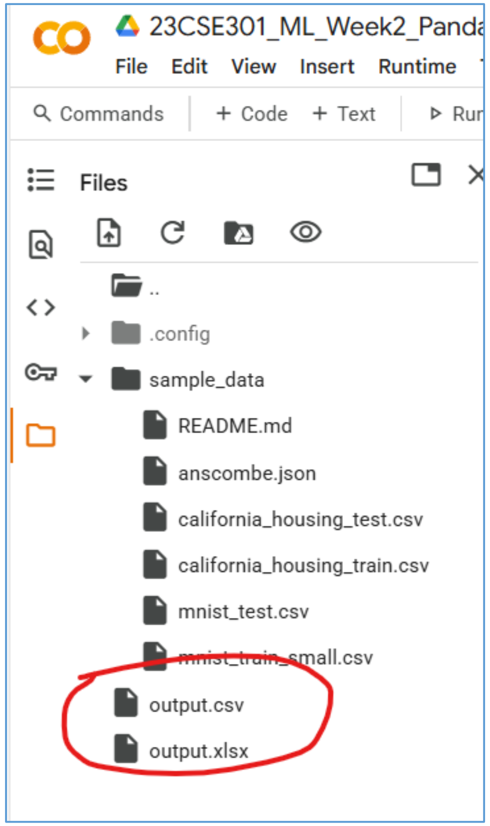
```
Index(['SepLen', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)',
       'target'],
      dtype='object')
```

Data Exploration

14. Exporting Data

```
df.to_csv('output.csv', index=False)


df.to_excel('output.xlsx', index=False)
```



Data Visualization

matplotlib Vs. seaborn

Feature	Matplotlib	Seaborn
Type	Low-level plotting library	High-level wrapper over Matplotlib
Code Complexity	More verbose, manual settings	Simpler, cleaner, with sensible defaults
Style/Look	Basic by default	Beautiful, attractive, publication-quality visualizations
Customization	Highly customizable (every axis, label, etc.)	Less control than Matplotlib, but most common customizations are easy
Built-in Plots	Line, bar, scatter, histogram, pie, etc.	Advanced statistical plots (box, violin, pairplot, heatmap, etc.)
Statistical Plots	Not built-in, must compute stats yourself	Built-in statistical visualizations (with grouping, hue, confidence intervals)
Integration	Pure Python plotting library	Built on Matplotlib — integrates easily with Pandas and NumPy
When to Use	When you need absolute control over every element	When you need quick, aesthetically pleasing exploratory or statistical plots

Why is Data Visualization Important?

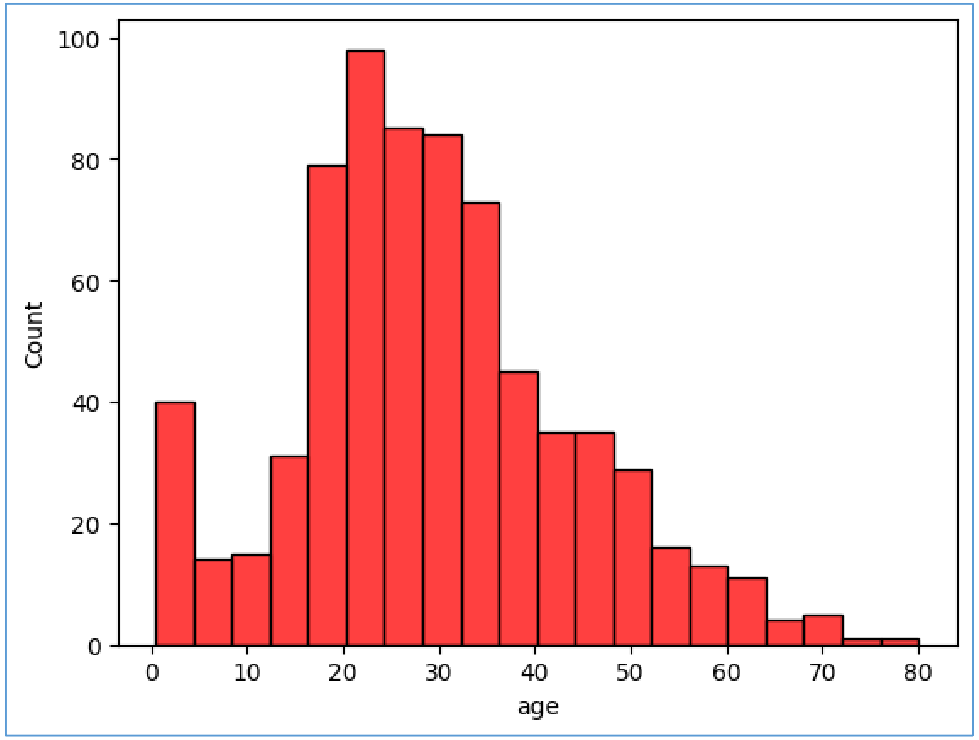
A picture is worth a thousand words!

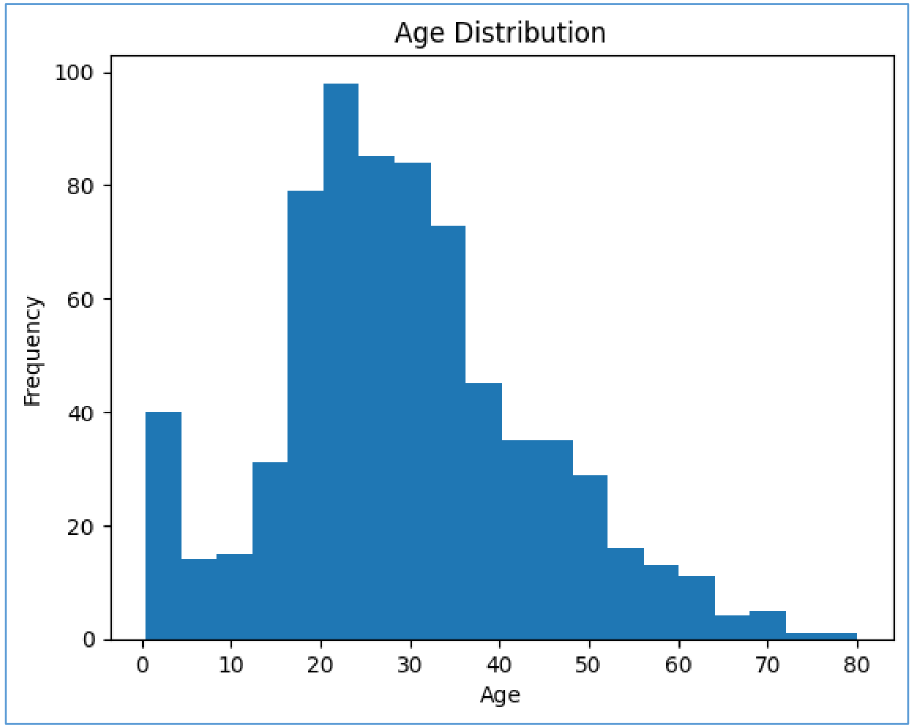
Data Visualization

1. Histogram: Shows distribution of a numerical variable

```
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset('titanic')
sns.histplot(data=df, x='age', bins=20)
plt.show()
```

```
plt.hist(df['age'], bins=20)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution')
plt.show()
```

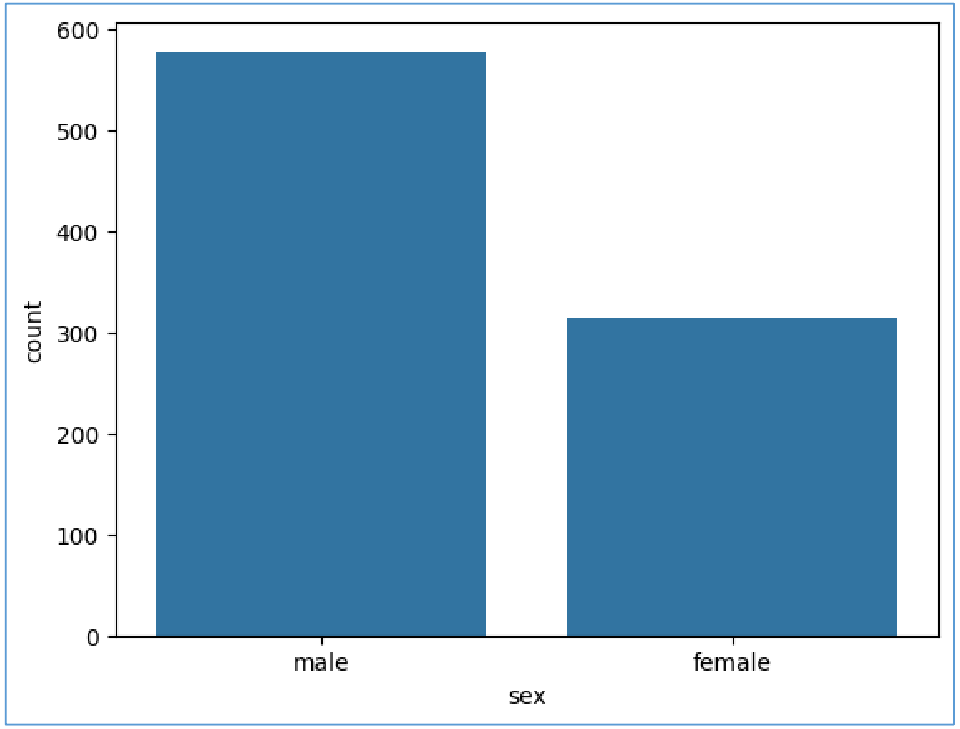




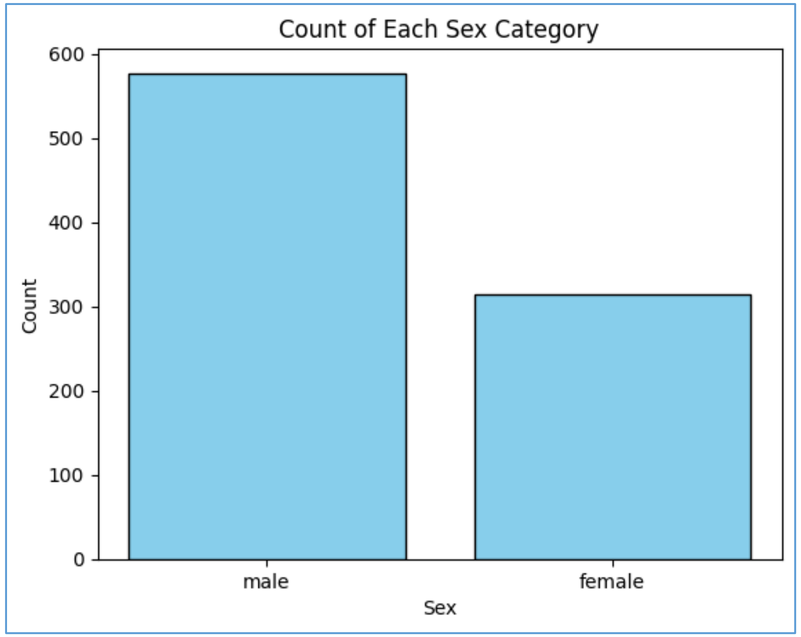
Data Visualization

2. Bar Plot: Compares categories (like survival counts)

```
sns.countplot(x='sex', data=df)
plt.show()
```



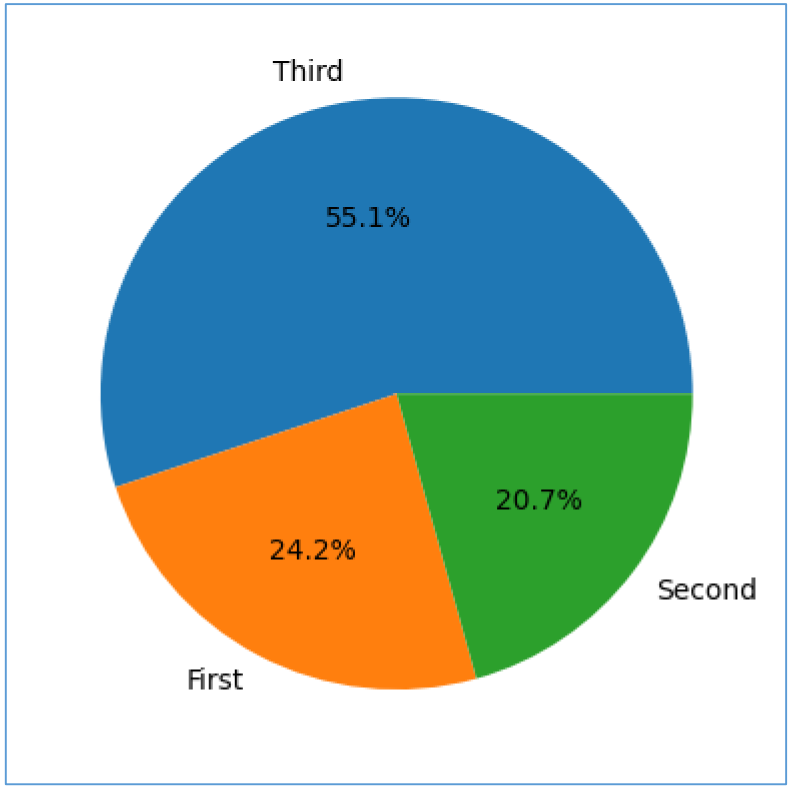

```
counts = df['sex'].value_counts()
plt.bar(counts.index, counts.values)
plt.xlabel('Sex')
plt.ylabel('Count')
plt.title('Count of Each Sex Category')
plt.show()
```



Data Visualization

3. Pie Chart: Proportional representation of a categorical column

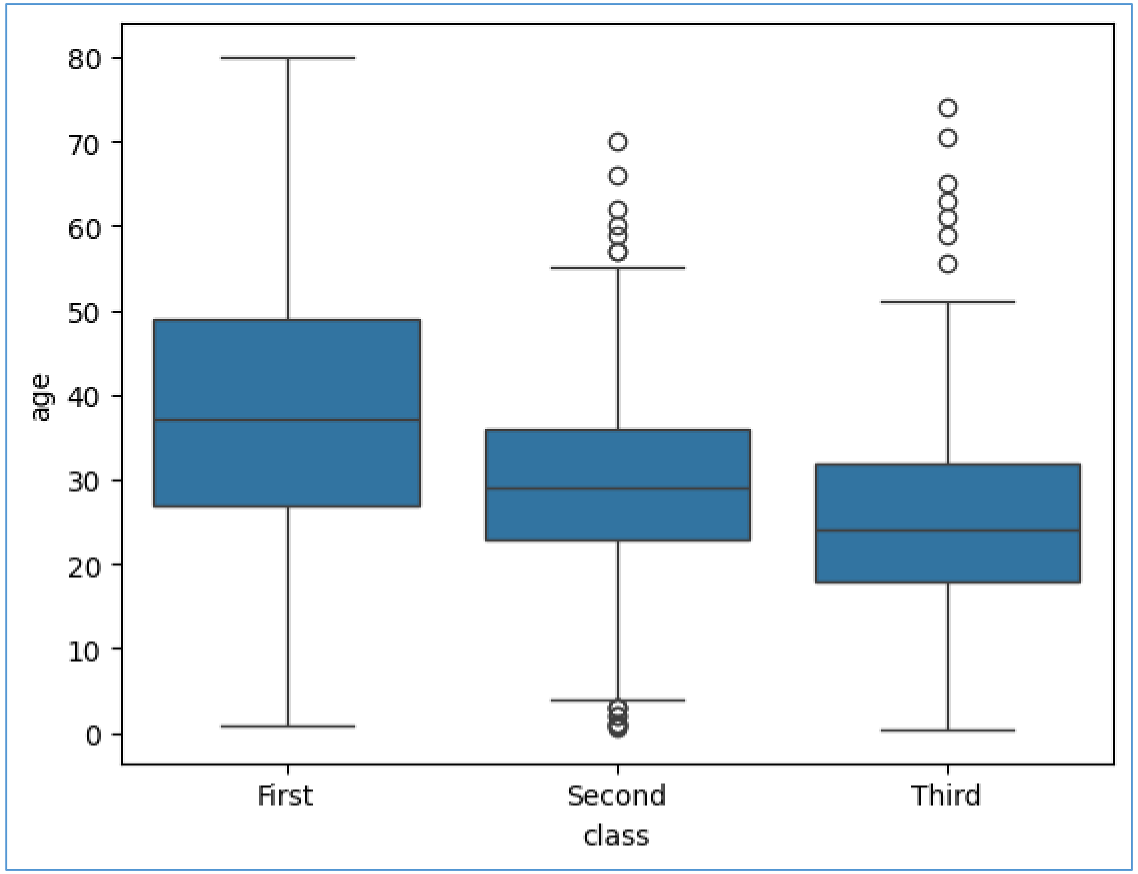
```
df['class'].value_counts().plot.pie(autopct='%1.1f%%')
plt.ylabel('')
plt.show()
```



Data Visualization

4. Box Plot: Shows distribution, median, and outliers

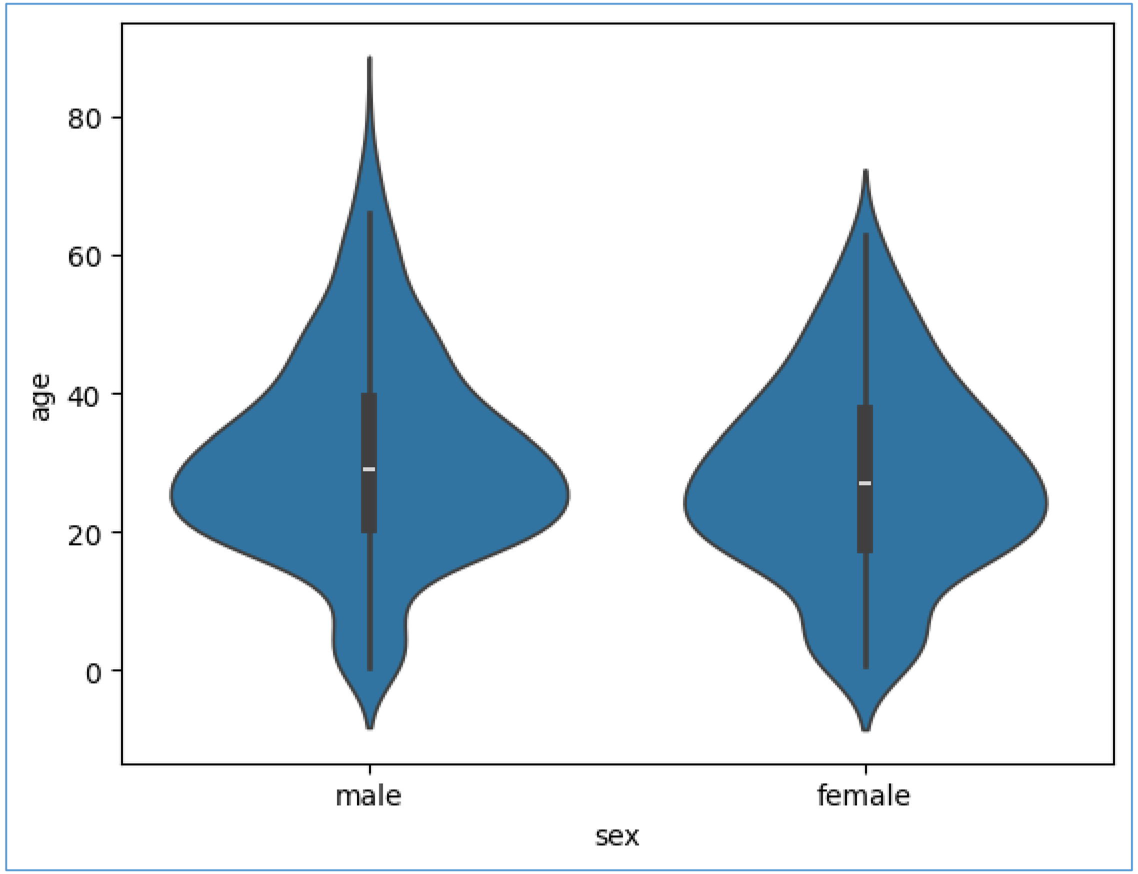
```
sns.boxplot(x='class', y='age', data=df)
plt.show()
```



Data Visualization

5. Violin Plot: Combines boxplot and KDE (density) for better shape visualization

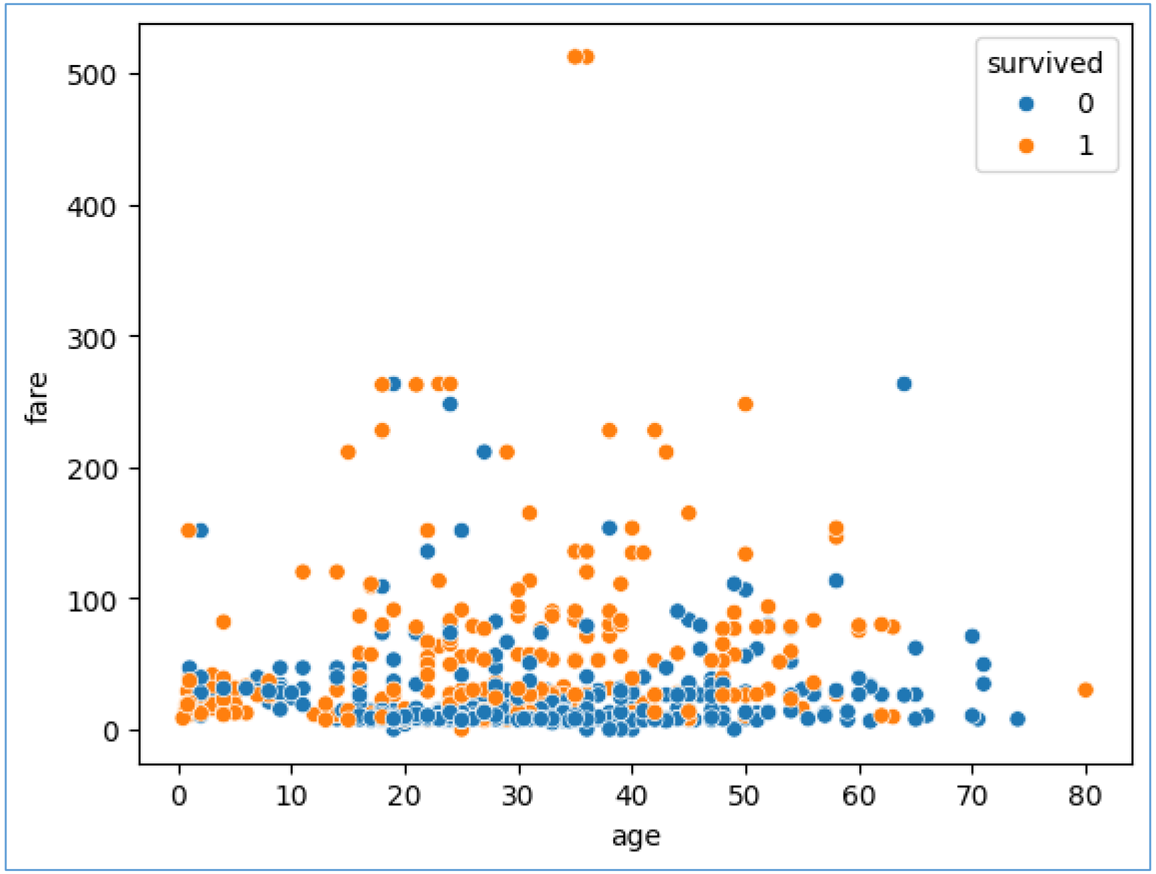
```
sns.violinplot(x='sex', y='age', data=df)
plt.show()
```



Data Visualization

6. Scatter Plot: Shows distribution, median, and outliers

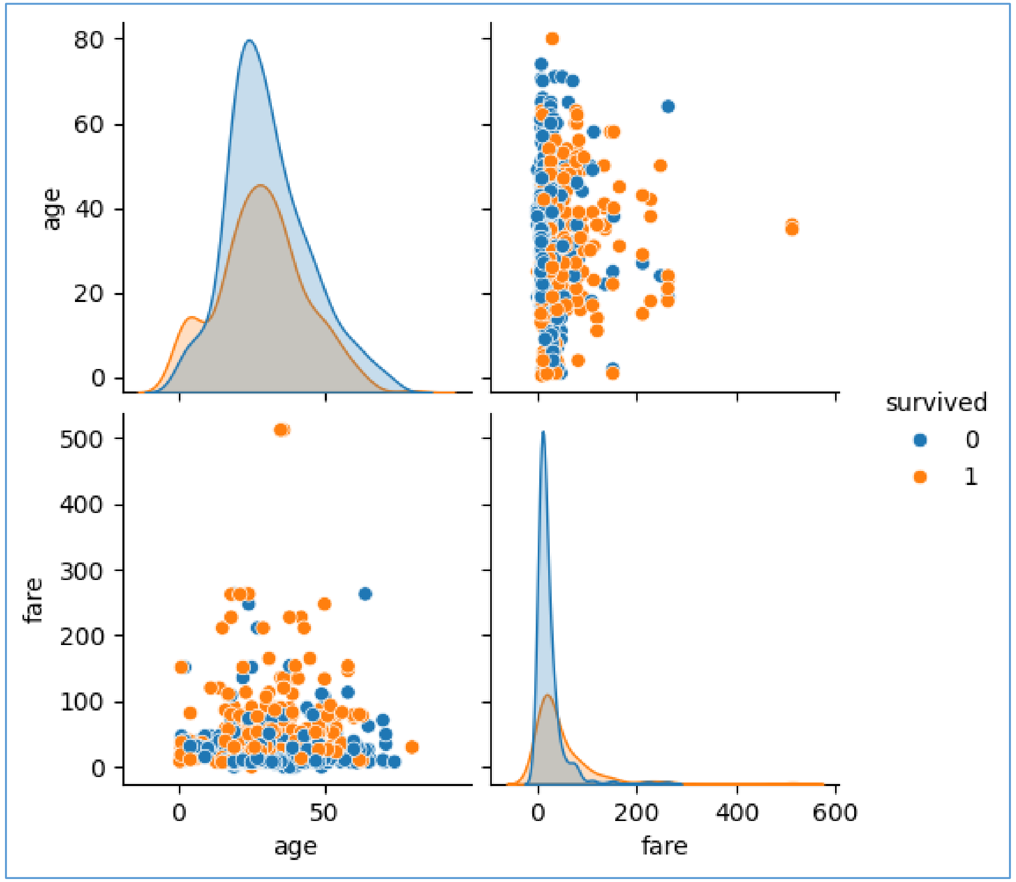
```
sns.scatterplot(x='age', y='fare', hue='survived', data=df)
plt.show()
```



Data Visualization

7. Pair Plot: Plots pairwise scatterplots + histograms for multiple numerical features

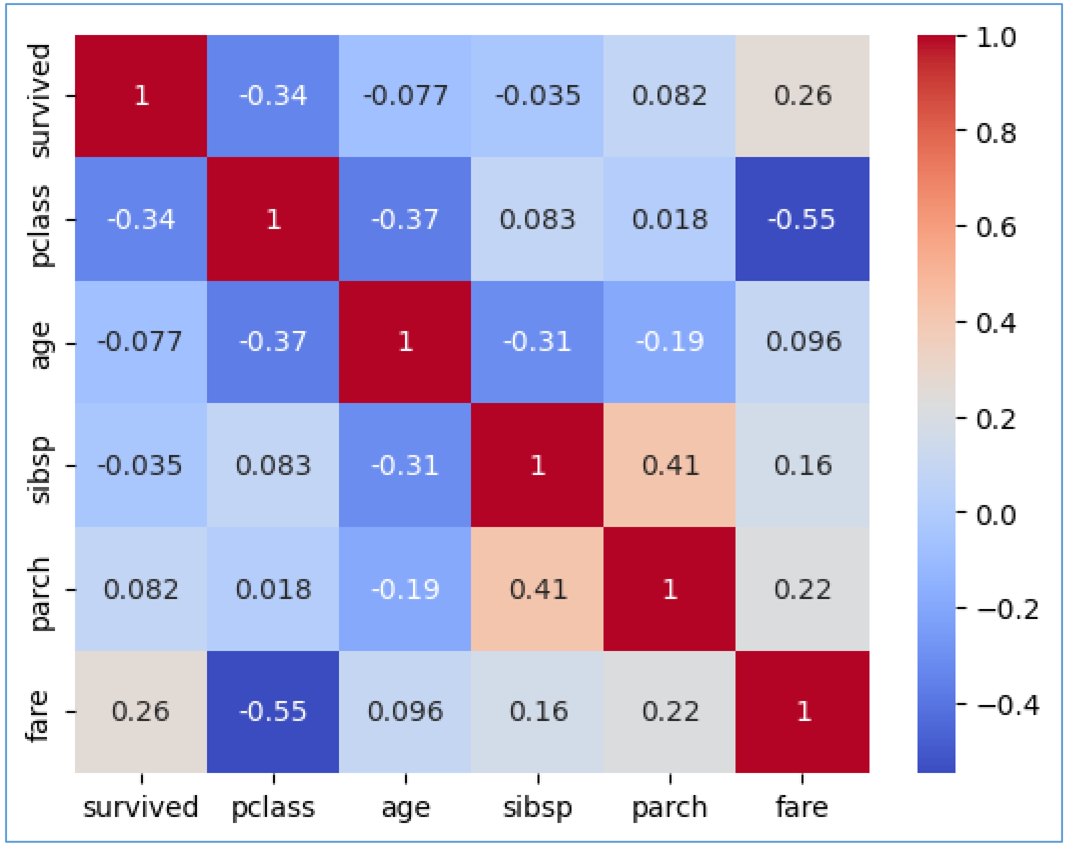
```
sns.pairplot(df[['age', 'fare', 'survived']], hue='survived')
plt.show()
```



Data Visualization

8. Heatmap (Correlation Matrix): Shows distribution, median, and outliers

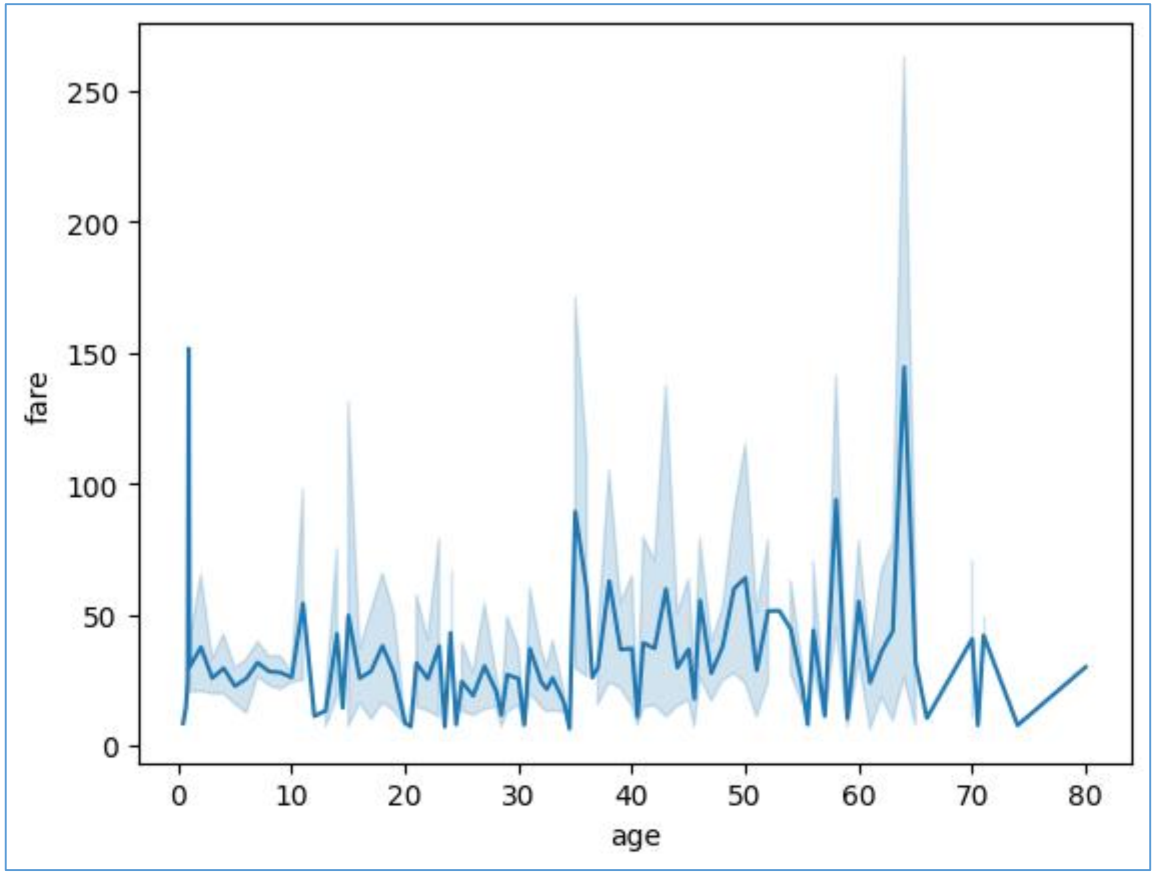
```
numeric_df = df.select_dtypes(include=['number'])
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



Data Visualization

9. Line Plot: Line chart of a variable against its index or another variable

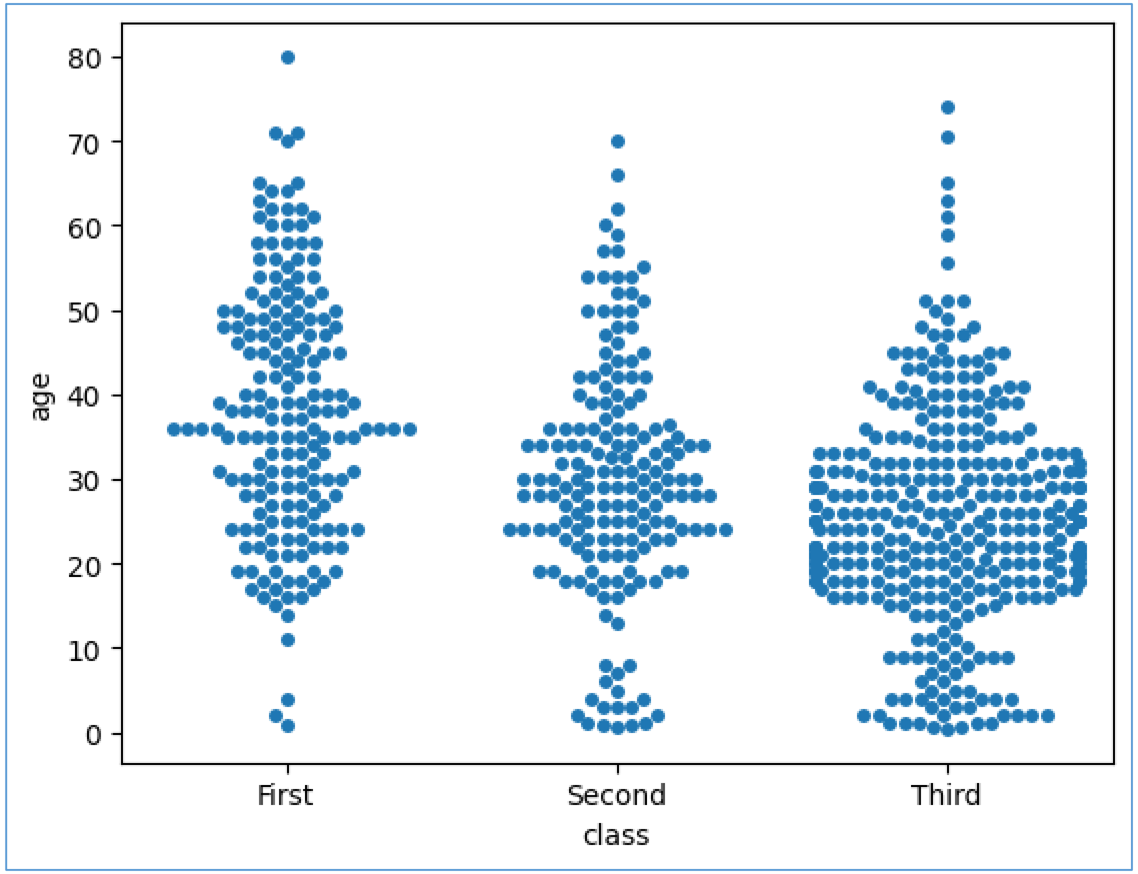
```
sns.lineplot(x='age', y='fare', data=df)
plt.show()
```



Data Visualization

10. Swarm Plot: Points distribution within categories (no overlap like scatter)

```
sns.swarmplot(x='class', y='age', data=df)
plt.show()
```



Exercise 1 - Week 2

- Pandas (Data Handling and Manipulation)
  - Load the dataset into a DataFrame.
  - Display the first 10 and last 5 rows.
  - Show summary statistics and data types of each column.
  - Filter records based on a condition (e.g., values greater than a threshold).
  - Add a new derived column using existing columns.
  - Group the data by a categorical column and compute mean/median for another numeric column.
  - Sort the DataFrame based on one or more columns.
  - Handle missing values by either dropping or filling them.
  - Export the final DataFrame to a new CSV file.
- Use any dataset of your choice (CSV/Excel/JSON)

Exercise 2 - Week 2

- Seaborn Visualization
  - One distribution plot (histogram, KDE, or violin plot).
  - One categorical plot (barplot, countplot, or boxplot).
  - One relationship plot (scatterplot or lineplot).
  - A correlation heatmap of numerical columns.
  - A pairplot to compare multiple relationships.
- Use any dataset of your choice (CSV/Excel/JSON)