Double-click (or enter) to edit

Section 1 – Python + Data Familiarity

1, List: Muttible and ordered, can consist different data type like integers, float, strings, tuples, and lists. Lists be can add with.append(), remove with .remove(), or sort with .sort(). , like a playlis. You can loop through or slice them. Slow for math but great for sequences.
Example: a = [1, 7.5, "cat", (2, 3), [4, 5]]

Dictionary: Muttible, ordered (modern Python), consist of key and value, values can be different data type like integers, float, strings, tuples, lists, but keys unique. Be can add with dict[key] = value, remove with del dict[key], or get with dict[key]. , like a phonebook. Fast for lookups, no math support. Example: c = {"name": "abu", 12: 87, "list": [1, 2]}

NumPy Array: Fixed size, ordered, consist same data type like integers or float. Be can multiply with *, add with +, or reshape with .reshape(). , like a calculator grid. Super fast for math, used in data or games, needs NumPy. Mixed types convert to one. Example: import numpy as np; arr = np.array([1, 2, 3])

2,

```
def f():
    a = [10, 15, 20, 25, 30]
    b = []
    for c in a:
        if not c % 2:
            b.append(c * c)
    return b

print(f())
```

```
[100, 400, 900]
```

3,

```
[1, 2, 3, 4]
```

```
[1, 2, 3, 4]
```

y = x makes both y and x point to the same list. So when y.append(4), it also changes x. That's why print(x) shows [1, 2, 3, 4]

4,

.shape provides the size of the DataFrame by returning a tuple with the number of rows and columns. It helps you quickly understand the structure of your dataset, such as whether it has many or few rows and columns.

```
import pandas as pd

data = {'Age': [22, 25, 30, 35, 40], 'Salary': [30000, 40000, 50000, 60000, 70000]}
df = pd.DataFrame(data)
```

```
print(df.shape)
```

```
(5, 2)
```

4,

.describe() gives a statistical summary of the numeric columns, showing the count of non-null values, mean, standard deviation, min, max, and the 25th, 50th, and 75th percentiles. This method is useful for understanding the distribution and range of your data at a glance.

```
print(df.describe())
```

```
             Age        Salary
count   5.000000      5.000000
mean   30.400000  50000.000000
std     7.300685  15811.388301
min    22.000000  30000.000000
25%    25.000000  40000.000000
50%    30.000000  50000.000000
75%    35.000000  60000.000000
max    40.000000  70000.000000
```

```
# 5
import pandas as pd

data = {
    "Month": ["JAN", "FEB", "MAR", "APR", "MAY"],
    "1958": [340, 318, 362, 348, 363],
    "1959": [360, 342, 406, 396, 420],
    "1960": [417, 391, 419, 461, 472]
}

df = pd.DataFrame(data)
print(df)
```

```
   Month  1958  1959  1960
0   JAN   340   360   417
1   FEB   318   342   391
2   MAR   362   406   419
3   APR   348   396   461
4   MAY   363   420   472
```

1,The values increase from 1958 to 1960 for each month,in air travel on these years. 2,The data is seasonal, with higher numbers in (JUN, JUL, AUG) and lower numbers in (JAN, FEB, NOV, DEC).

```
# 6  Air travel data
months = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC']
year_1958 = [340, 318, 362, 348, 363, 435, 491, 505, 404, 359, 310, 337]
year_1959 = [360, 342, 406, 396, 420, 472, 548, 559, 463, 407, 362, 405]
year_1960 = [417, 391, 419, 461, 472, 535, 622, 606, 508, 461, 390, 432]

# 1st Calculate the totals for each month
totals = [a+b+c for a,b,c in zip(year_1958, year_1959, year_1960)]

# 2nd Find highest of  all years
max_total = max(totals)
```

```
max_month = months[totals.index(max_total)]

# 3rd find lowest in 1958
min_1958 = min(year_1958)
min_month = months[year_1958.index(min_1958)]

# Print results
print(f"The highest is {max_month} with {max_total} passenger")
print(f"The lowest is {min_month} with {min_1958} passenger")
```

```
The highest is AUG with 1670 passenger
The lowest is NOV with 310 passenger
```

1. **Highest Total (1958-1960):**
   Sum passengers for each month across all 3 years.
   Pick the month with the largest sum (July: 1,661).
2. **Lowest in 1958:**
   Find the smallest value in 1958 data (November: 310).

7,

.groupby() function in panda splits data into groups (by column value), lets you apply operations (like sum/mean/count) to each group, combines results into a new DataFrame/Series, mimics SQL's GROUP BY but more powerful, enables category comparisons (e.g., sales by product), and works with .agg() for multi-stat outputs example: df.groupby('Product')['Sales'].sum()gives total sales per product.

```
#example
import pandas as pd

# Dummy sales data
df = pd.DataFrame({
    'Product': ['Apple', 'Banana', 'Apple', 'Banana'],
    'Sales': [100, 50, 120, 80]
})

# Group by 'Product' and sum sales
result = df.groupby('Product')['Sales'].sum()
print(result)
```

```
Product
Apple     220
Banana    130
Name: Sales, dtype: int64
```

```
#8
import seaborn as sns

# Load dataset and count missing values
titanic = sns.load_dataset('titanic')
missing_counts = titanic.isna().sum()

print(missing_counts)
```

```
survived         0
pclass           0
```

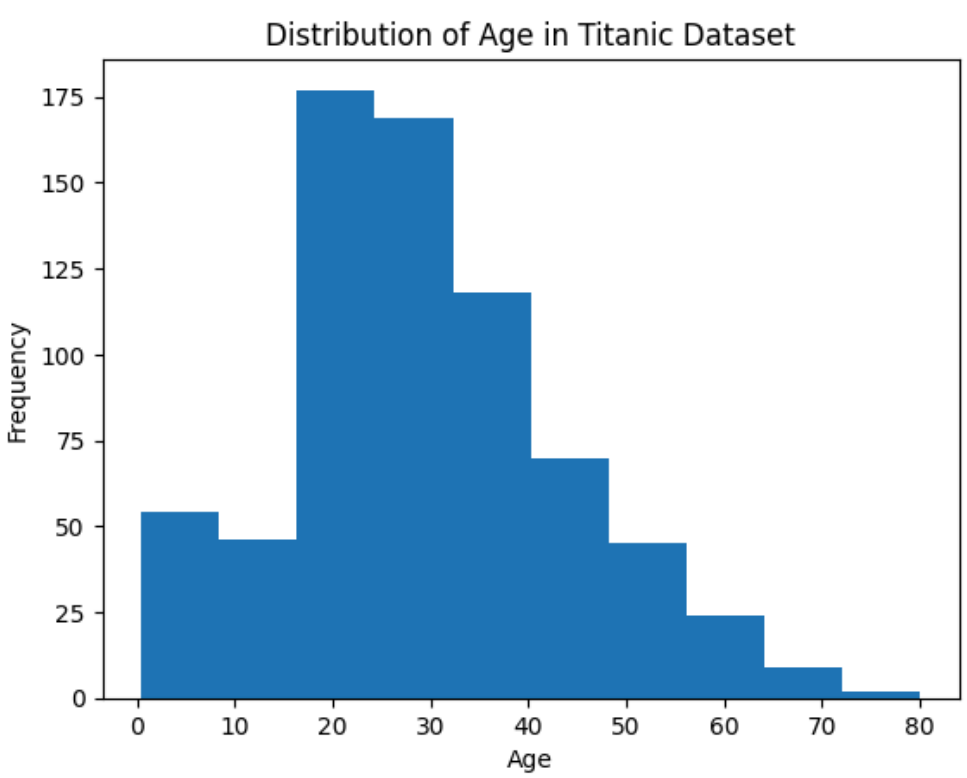
```
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

1. isna() (or isnull()) detects missing values
2. sum() counts them per column
3. Major gaps appear in:
    - deck (688 missing)
    - age (177 missing)
    - embarked/embark_town (2 missing each)

```
#9
import seaborn as sns
import matplotlib.pyplot as plt  # Import matplotlib for plotting

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

# Create a histogram of the 'age' column in the 'titanic' DataFrame
plt.hist(titanic['age'])  # Use plt.hist() for creating the histogram
plt.xlabel('Age')  # Set x-axis label
plt.ylabel('Frequency')  # Set y-axis label
plt.title('Distribution of Age in Titanic Dataset')  # Set title
plt.show()  # Display the histogram
```



What seen to me

When plot passenger ages, most people are clustered between 20-40 years old There are fewer babies/children Very few elderly passengers (age 60+) The graph has a "tail" stretching to the right (this is called right-skew)

Why This Pattern Exists:

- Ships carried many working-age adults (sailors, merchants, immigrants)
- Families traveled with some children, but not as many as adults
- In 1912:
    - Fewer elderly people existed in population
    - Older people rarely traveled by ship
    - Those who did had lower survival chances

10,

Standard Deviation (σ) in Simple Terms:
Low σ : Data points are close together (consistent).
Example: [10, 11, 9, 10] → All values near 10.
High σ : Data points are spread out (variable).
Example: [1, 50, 100] → Values range widely.

```
import numpy as np
low_std_data = [10, 10, 11, 9, 10]  # Tightly clustered values
high_std_data = [1, 5, 20, 50, 100]  # Widely spread values
print("Low STD:", np.std(low_std_data))   # Output: ~0.7
print("High STD:", np.std(high_std_data)) # Output: ~39.6
print("Difference:", round(np.std(high_std_data)/np.std(low_std_data), 1), "times")
```

```
Low STD: 0.6324555320336759
High STD: 36.69005314795824
Difference: 58.0 times
```

11,

1. **Medical Research**
    - *Problem*: If patient allergy records are incomplete in a drug trial, researchers might miss dangerous side effects.
    - *Consequence*: Could lead to approval of medications that are harmful to certain populations.
2. **Credit Scoring**
    - *Problem*: When applicants' income data is missing, lenders may rely solely on credit history.
    - *Consequence*: Could unfairly deny loans to qualified applicants or approve risky ones, increasing default