



RAJALAKSHMI ENGINEERING COLLEGE

Approved by AICTE | Affiliated to Anna University | Accredited by NAAC

Department of Computer Science and Engineering

CS23334 Fundamentals of Data Science Lab

III semester II Year (2023R)

Name of the Student : Danyl Enoch

Register Number : 2116240701092

ex - 1a

October 31, 2025

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

sns.set(style="whitegrid")

def load_job_data(csv_path=None):
    if csv_path:
        df = pd.read_csv(csv_path, parse_dates=['date_posted'])
        return df
    rng = pd.date_range(start='2015-01-01', end='2024-12-31', freq='D')
    years = rng.year
    base_by_year = {y: 50 + (y - 2015) * 60 for y in range(2015, 2025)}
    counts = [np.random.poisson(lam=max(1, base_by_year[y]/30)) for y in years]
    df = pd.DataFrame({'date_posted': rng, 'postings': counts})
    return df

def aggregate_by_year(df):
    df['date_posted'] = pd.to_datetime(df['date_posted'])
    df['year'] = df['date_posted'].dt.year
    if 'postings' in df.columns:
        yearly = df.groupby('year')['postings'].sum().
↪reset_index(name='num_postings')
    else:
        yearly = df.groupby('year').size().reset_index(name='num_postings')
    return yearly

def plot_trend(yearly_df, title="Data Science Job Postings by Year"):
    plt.figure(figsize=(10,5))
    ax = sns.lineplot(data=yearly_df, x='year', y='num_postings', marker='o')
    ax.set_title(title)
    ax.set_xlabel("Year")
    ax.set_ylabel("Number of Job Postings")
    plt.xticks(yearly_df['year'])
    plt.tight_layout()
```

```
plt.show()

if __name__ == "__main__":
    df = load_job_data(csv_path=None)
    yearly = aggregate_by_year(df)
    print(yearly)
    plot_trend(yearly)
```

ex - 1b

October 31, 2025

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

sns.set(style="whitegrid")

ROLE_KEYWORDS = {
    'Data Scientist': ['data scientist', r'\bds\b', 'machine learning',
↪scientist', 'ml scientist'],
    'Data Engineer': ['data engineer', 'etl engineer', 'pipeline engineer'],
    'Data Analyst': ['data analyst', 'business analyst', 'analyst', 'bi',
↪analyst'],
    'Machine Learning Engineer': ['ml engineer', 'machine learning engineer',
↪'mle'],
    'BI Developer': ['bi developer', 'business intelligence', 'power bi',
↪'tableau developer'],
    'Research Scientist': ['research scientist', 'researcher'],
    'Other': []
}

def map_title_to_role(title):
    t = title.lower()
    for role, keys in ROLE_KEYWORDS.items():
        for key in keys:
            if re.search(r'\b' + re.escape(key) + r'\b', t) or key in t:
                return role
    return 'Other'

def categorize_roles(df, title_col='job_title'):
    df = df.copy()
    df[title_col] = df[title_col].astype(str)
    df['role'] = df[title_col].apply(map_title_to_role)
    return df

def plot_role_distribution(df, title_col='role'):
    counts = df[title_col].value_counts().reset_index()
```

```

counts.columns = ['role', 'count']
plt.figure(figsize=(10,5))
sns.barplot(data=counts, x='role', y='count')
plt.xticks(rotation=45, ha='right')
plt.title("Distribution of Data Science Roles (bar)")
plt.tight_layout()
plt.show()

plt.figure(figsize=(7,7))
plt.pie(counts['count'], labels=counts['role'], autopct='%1.1f%%',
↪startangle=140)
plt.title("Distribution of Data Science Roles (pie)")
plt.tight_layout()
plt.show()

if __name__ == "__main__":
    sample_titles = [
        "Senior Data Scientist", "Junior Data Analyst", "Machine Learning
↪Engineer",
        "Data Engineer", "BI Developer (Power BI)", "Business Analyst - Data",
        "Research Scientist, ML", "Data Scientist / ML", "Analyst", "ETL
↪Engineer",
        "Data Scientist", "Data Analyst", "MLOps Engineer", "Data Engineer -
↪Big Data"
    ]
    df = pd.DataFrame({'job_title': sample_titles})
    df = categorize_roles(df)
    print(df[['job_title', 'role']])
    plot_role_distribution(df, title_col='role')

```

ex - 1c

October 31, 2025

```
[ ]: import pandas as pd
import json
from xml.etree import ElementTree as ET

def structured_example():
    data = {
        'id': [1,2,3],
        'name': ['Alice','Bob','Carol'],
        'age': [29, 34, 23]
    }
    df = pd.DataFrame(data)
    print("Structured data (pandas DataFrame):")
    print(df)
    df.to_csv('structured_example.csv', index=False)
    print("Saved to structured_example.csv")

def unstructured_example():
    docs = [
        "Today I attended a data science meetup and learned about transformers.",
        ↪ "Error: Connection refused at 2025-10-31 10:12:00 - service X failed.",
        "Image: binary data (not text) - e.g. photos, audio transcripts"
    ]
    print("\nUnstructured data (plain text documents):")
    for i, doc in enumerate(docs,1):
        print(f"Doc {i}: {doc}")
    with open('unstructured_example.txt','w',encoding='utf-8') as f:
        for d in docs:
            f.write(d + "\n")

def semi_structured_example():
    items = [
        {"id":1, "name":"Alice", "skills":["python","sql"]},
        {"id":2, "name":"Bob", "contact":{"email":"bob@example.com","phone":
        ↪ "12345"}},
        {"id":3, "name":"Carol", "notes":"Prefers remote"}
    ]
```

```

print("\nSemi-structured data (JSON-like):")
print(json.dumps(items, indent=2))
with open('semi_structured_example.json', 'w', encoding='utf-8') as f:
    json.dump(items, f, indent=2)

def xml_example():
    root = ET.Element('employees')
    e1 = ET.SubElement(root, 'employee', attrib={'id': '1'})
    ET.SubElement(e1, 'name').text = 'Alice'
    ET.SubElement(e1, 'role').text = 'Data Scientist'
    tree = ET.ElementTree(root)
    tree.write('semi_structured_example.xml', encoding='utf-8',
    ↪xml_declaration=True)
    print("\nWrote semi_structured_example.xml (XML is semi-structured)")

if __name__ == "__main__":
    structured_example()
    unstructured_example()
    semi_structured_example()
    xml_example()

    print("\nCharacteristics summary:")
    print("- Structured: rigid schema, easy to query (e.g., SQL tables, CSV).")
    print("- Unstructured: no predefined schema (text, images), needs parsing/
    ↪NLP/vision.")
    print("- Semi-structured: tags/keys but not rigid (JSON, XML, logs with key:
    ↪value).")

```

ex - 1d

October 31, 2025

```
[ ]: from cryptography.fernet import Fernet

def generate_key():
    return Fernet.generate_key()

def encrypt_message(key: bytes, plaintext: str) -> bytes:
    f = Fernet(key)
    token = f.encrypt(plaintext.encode('utf-8'))
    return token

def decrypt_message(key: bytes, token: bytes) -> str:
    f = Fernet(key)
    plaintext = f.decrypt(token)
    return plaintext.decode('utf-8')

if __name__ == "__main__":
    key = generate_key()
    print("Generated key (store securely):", key.decode())

    secret = "MyVerySensitivePassword123!"
    token = encrypt_message(key, secret)
    print("\nEncrypted token (bytes):", token)

    recovered = decrypt_message(key, token)
    print("\nDecrypted plaintext:", recovered)

    with open('secret.key', 'wb') as f:
        f.write(key)
    print("\nKey saved to secret.key (handle securely)")
```


ex - 2

October 31, 2025

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
file_path='C:\sales_data.csv'
df = pd.read_csv(file_path)
print(df.head())
print(df.isnull().sum())
df['Sales'].fillna(df['Sales'].mean(), inplace=True)
df.dropna(subset=['Product', 'Quantity', 'Region'], inplace=True)
print(df.describe())
product_summary = df.groupby('Product').agg({
'Sales': 'sum',
'Quantity': 'sum'
}).reset_index()
print(product_summary)
plt.figure(figsize=(10, 6))
plt.bar(product_summary['Product'], product_summary['Sales'])
plt.xlabel('Product')
plt.ylabel('Total Sales')
plt.title('Total Sales by Product')
plt.show()
df['Date'] = pd.to_datetime(df['Date'])
sales_over_time = df.groupby('Date').agg({'Sales': 'sum'}).reset_index()
plt.figure(figsize=(10, 6))
plt.plot(sales_over_time['Date'], sales_over_time['Sales'])
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('SalesOver Time')
plt.show()
pivot_table = df.pivot_table(values='Sales', index='Region', columns='Product',
aggfunc=np.sum, fill_value=0)
print(pivot_table)
correlation_matrix = df.corr()
print(correlation_matrix)
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

```
plt.title('Correlation Matrix')  
plt.show()
```

ex - 3a

October 31, 2025

```
[ ]: import numpy as np
import pandas as pd
df = pd.read_csv("Hotel_Dataset.csv")
print("Original Data:")
print(df)
print("\nDuplicate rows:", df.duplicated().sum())
df.drop_duplicates(inplace=True)
print("\nAfter removing duplicates:")
print(df)
df.reset_index(drop=True, inplace=True)
print("\nAfter resetting index:")
print(df)
if 'Age_Group.1' in df.columns:
    df.drop(['Age_Group.1'], axis=1, inplace=True)
df.loc[df['CustomerID'] < 0, 'CustomerID'] = np.nan
df.loc[df['Bill'] < 0, 'Bill'] = np.nan
df.loc[df['EstimatedSalary'] < 0, 'EstimatedSalary'] = np.nan
df.loc[(df['NoOfPax'] < 1) | (df['NoOfPax'] > 20), 'NoOfPax'] = np.nan
df['Hotel'] = df['Hotel'].replace({'Ibys': 'Ibis'})
df['FoodPreference'] = df['FoodPreference'].replace(
    {'Vegetarian': 'Veg', 'veg': 'Veg', 'non-Veg': 'Non-Veg'}
)
df['EstimatedSalary'] = df['EstimatedSalary'].
    ↪ fillna(round(df['EstimatedSalary'].mean()))
df['NoOfPax'] = df['NoOfPax'].fillna(round(df['NoOfPax'].median()))
df['Bill'] = df['Bill'].fillna(round(df['Bill'].mean()))
df.loc[(df['Rating(1-5)'] < 1) | (df['Rating(1-5)'] > 5), 'Rating(1-5)'] = np.
    ↪ nan
df['Rating(1-5)'] = df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()))
df = df.astype({
    'CustomerID': 'Int64',
    'NoOfPax': 'Int64',
    'Bill': 'Int64',
    'EstimatedSalary': 'Int64',
    'Rating(1-5)': 'Int64'
})
print("\nFinal Cleaned DataFrame:")
```

```
print(df)
print("\nCleaned DataFrame Info:")
print(df.info())
```

ex - 3b

October 31, 2025

```
[ ]: import numpy as np
import pandas as pd
df = pd.read_csv("/content/pre_process_datasample (1).csv")
print("Original Dataset:\n", df)
df.info()
df.Country.fillna(df.Country.mode()[0], inplace=True)
df.Age.fillna(df.Age.median(), inplace=True)
df.Salary.fillna(round(df.Salary.mean()), inplace=True)
print("\nDataset after filling missing values:\n", df)
country_dummies = pd.get_dummies(df.Country)
updated_dataset = pd.concat([country_dummies, df.iloc[:, [1, 2, 3]]], axis=1)
updated_dataset.Purchased.replace(['No', 'Yes'], [0, 1], inplace=True)
print("\nFinal Processed Dataset:\n", updated_dataset)
```

exe-4

November 2, 2025

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

```
[2]: arr=np.random.randint(50,100,10)
```

```
[3]: arr
```

```
[3]: array([59, 66, 67, 81, 61, 73, 88, 64, 52, 66], dtype=int32)
```

```
[4]: arr.mean()
```

```
[4]: np.float64(67.7)
```

```
[5]: sorted(arr)
```

```
[5]: [np.int32(52),
      np.int32(59),
      np.int32(61),
      np.int32(64),
      np.int32(66),
      np.int32(66),
      np.int32(67),
      np.int32(73),
      np.int32(81),
      np.int32(88)]
```

```
[6]: def out_detec(arr):
      q1,q3=np.percentile(arr,[25,75])
      qr=q3-q1
      n=q1-(1.5*qr)
      m=q3+(1.5*qr)
      return n,m
```

```
[7]: n,m=out_detec(arr)
```

```
[8]: print(n)  
      print(m)
```

```
47.125  
86.125
```

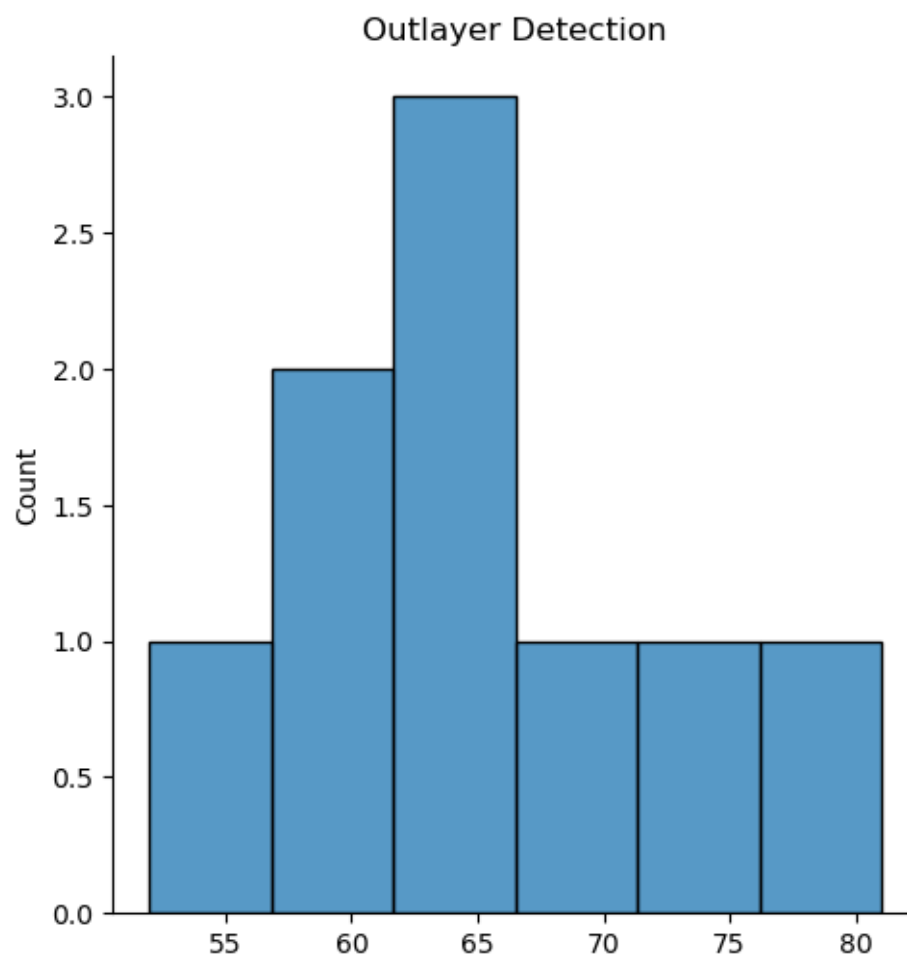
```
[9]: arr1=arr[(arr>n) & (arr<m)]
```

```
[10]: arr1
```

```
[10]: array([59, 66, 67, 81, 61, 73, 64, 52, 66], dtype=int32)
```

```
[11]: sn.displot(arr1)  
      plt.title("Outlayer Detection")
```

```
[11]: Text(0.5, 1.0, 'Outlayer Detection')
```



[]:

Exercise5

November 2, 2025

```
[9]: import pandas as pd
import numpy as np
df=pd.read_csv('pre_process_datasample_outlayers.csv')
```

```
[10]: df.head()
```

```
[10]:   Country  Age  Salary Purchased
0  France  44.0  72000.0         No
1   Spain  27.0  48000.0         Yes
2  Germany  30.0  54000.0         No
3   Spain  38.0  61000.0         No
4  Germany  40.0     NaN         Yes
```

```
[11]: df['Country'] = df['Country'].fillna(df['Country'].mode()[0])
```

```
[12]: val=df.iloc[:, :-1].values
val1=df.iloc[:, -1].values
from sklearn.impute import SimpleImputer
n=SimpleImputer(strategy="mean",missing_values=np.nan)
sa=SimpleImputer(strategy="mean",missing_values=np.nan)
n.fit(val[:, [1]])
```

```
[12]: SimpleImputer()
```

```
[13]: sa.fit(val[:, [2]])
```

```
[13]: SimpleImputer()
```

```
[15]: val[:, [1]]=n.transform(val[:, [1]])
val[:, [2]]=sa.transform(val[:, [2]])
val
```

```
[15]: array([[ 'France', 44.0, 72000.0],
       [ 'Spain', 27.0, 48000.0],
       [ 'Germany', 30.0, 54000.0],
       [ 'Spain', 38.0, 61000.0],
       [ 'Germany', 40.0, 63777.777777777778],
       [ 'France', 35.0, 58000.0],
```

```

['Spain', 38.77777777777778, 52000.0],
['France', 48.0, 79000.0],
['Germany', 50.0, 83000.0],
['France', 37.0, 67000.0]], dtype=object)

```

```

[16]: from sklearn.preprocessing import OneHotEncoder
      m = OneHotEncoder(sparse_output=False)
      m

```

```

[16]: OneHotEncoder(sparse_output=False)

```

```

[17]: c=m.fit_transform(val[:,[0]])
      c

```

```

[17]: array([[1., 0., 0.],
            [0., 0., 1.],
            [0., 1., 0.],
            [0., 0., 1.],
            [0., 1., 0.],
            [1., 0., 0.],
            [0., 0., 1.],
            [1., 0., 0.],
            [0., 1., 0.],
            [1., 0., 0.]])

```

```

[18]: set_final=np.concatenate((c,val[:,[1,2]]),axis=1)

```

```

[19]: from sklearn.preprocessing import StandardScaler

```

```

[20]: sc=StandardScaler()
      sc.fit(set_final)
      feat_standard_scaler=sc.transform(set_final)
      feat_standard_scaler

```

```

[20]: array([[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
              7.58874362e-01,  7.49473254e-01],
            [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
              -1.71150388e+00, -1.43817841e+00],
            [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
              -1.27555478e+00, -8.91265492e-01],
            [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
              -1.13023841e-01, -2.53200424e-01],
            [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
              1.77608893e-01,  6.63219199e-16],
            [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
              -5.48972942e-01, -5.26656882e-01],
            [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
              0.00000000e+00, -1.07356980e+00],

```

```
[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
 1.34013983e+00,  1.38753832e+00],
[-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
 1.63077256e+00,  1.75214693e+00],
[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
-2.58340208e-01,  2.93712492e-01]])
```

```
[21]: from sklearn.preprocessing import MinMaxScaler
mn1=MinMaxScaler(feature_range=(0,1))
mn1.fit(set_final)
f_min=mn1.transform(set_final)
f_min
```

```
[21]: array([[1.          , 0.          , 0.          , 0.73913043, 0.68571429],
 [0.          , 0.          , 1.          , 0.          , 0.          ],
 [0.          , 1.          , 0.          , 0.13043478, 0.17142857],
 [0.          , 0.          , 1.          , 0.47826087, 0.37142857],
 [0.          , 1.          , 0.          , 0.56521739, 0.45079365],
 [1.          , 0.          , 0.          , 0.34782609, 0.28571429],
 [0.          , 0.          , 1.          , 0.51207729, 0.11428571],
 [1.          , 0.          , 0.          , 0.91304348, 0.88571429],
 [0.          , 1.          , 0.          , 1.          , 1.          ],
 [1.          , 0.          , 0.          , 0.43478261, 0.54285714]])
```

```
[ ]:
```

exercise6

November 2, 2025

```
[13]: import pandas as pd
import numpy as np
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
```

```
[14]: df=pd.read_csv('tips.csv')
```

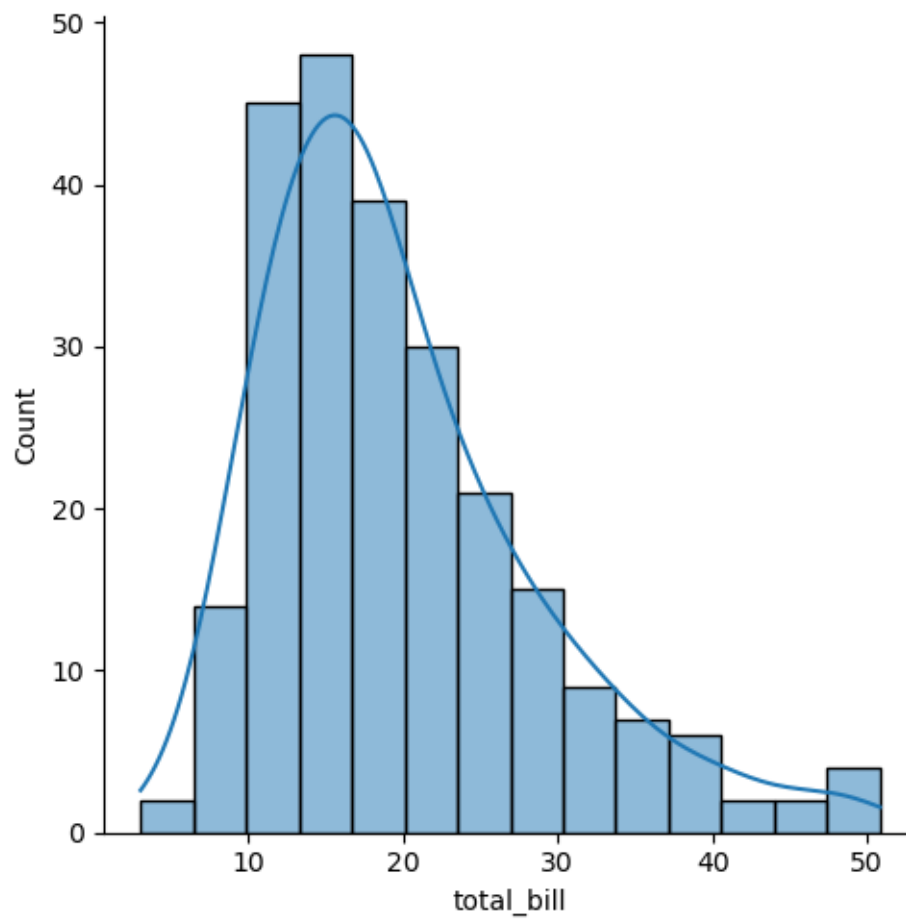
```
[15]: df.head(10)
```

```
[15]:
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2

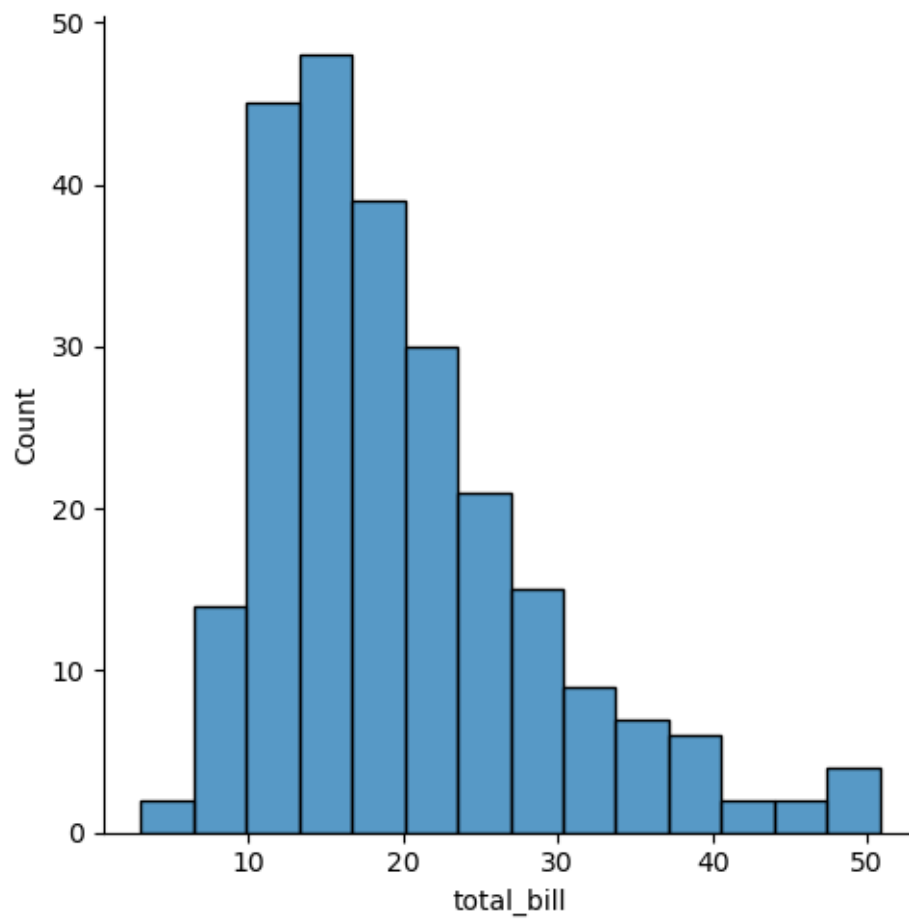
```
[16]: sn.displot(df.total_bill,kde=True)
```

```
[16]: <seaborn.axisgrid.FacetGrid at 0x19285a8a490>
```



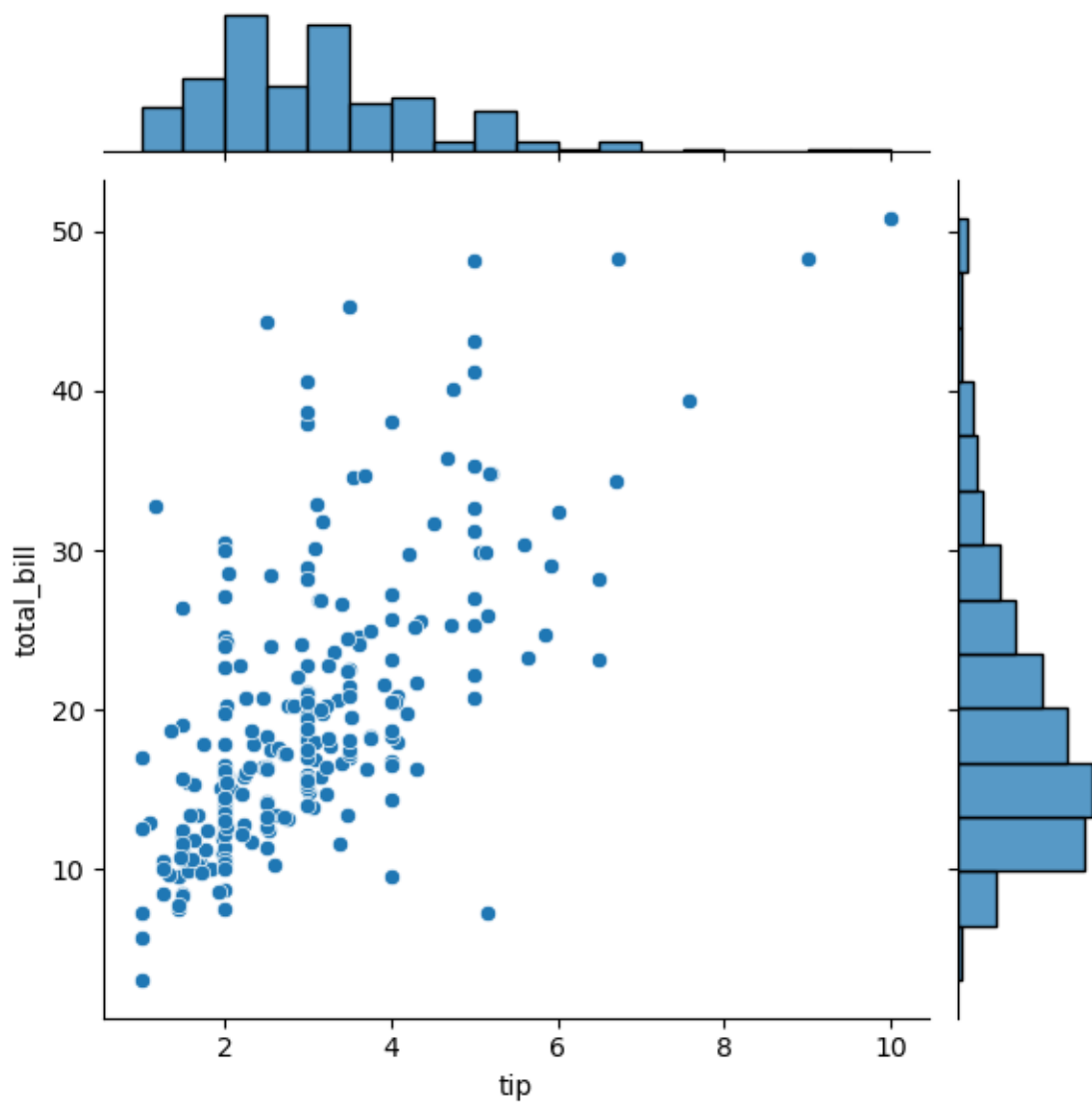
```
[17]: sn.displot(df.total_bill,kde=False)
```

```
[17]: <seaborn.axisgrid.FacetGrid at 0x19285e2c7d0>
```



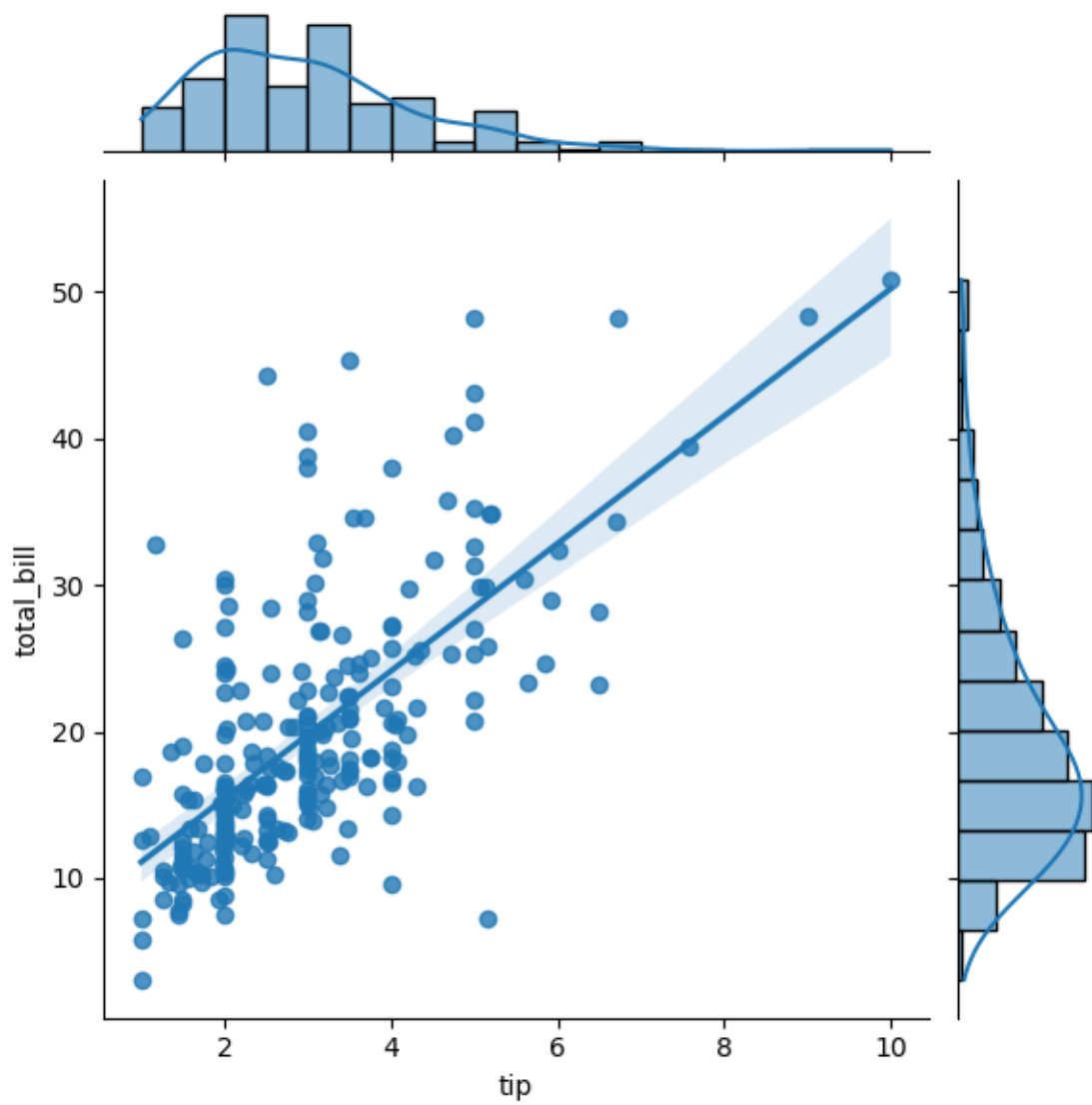
```
[21]: #plt.scatter(x=df.tip,y=df.total_bill)  
sn.jointplot(x=df.tip,y=df.total_bill)
```

```
[21]: <seaborn.axisgrid.JointGrid at 0x19285f33b10>
```



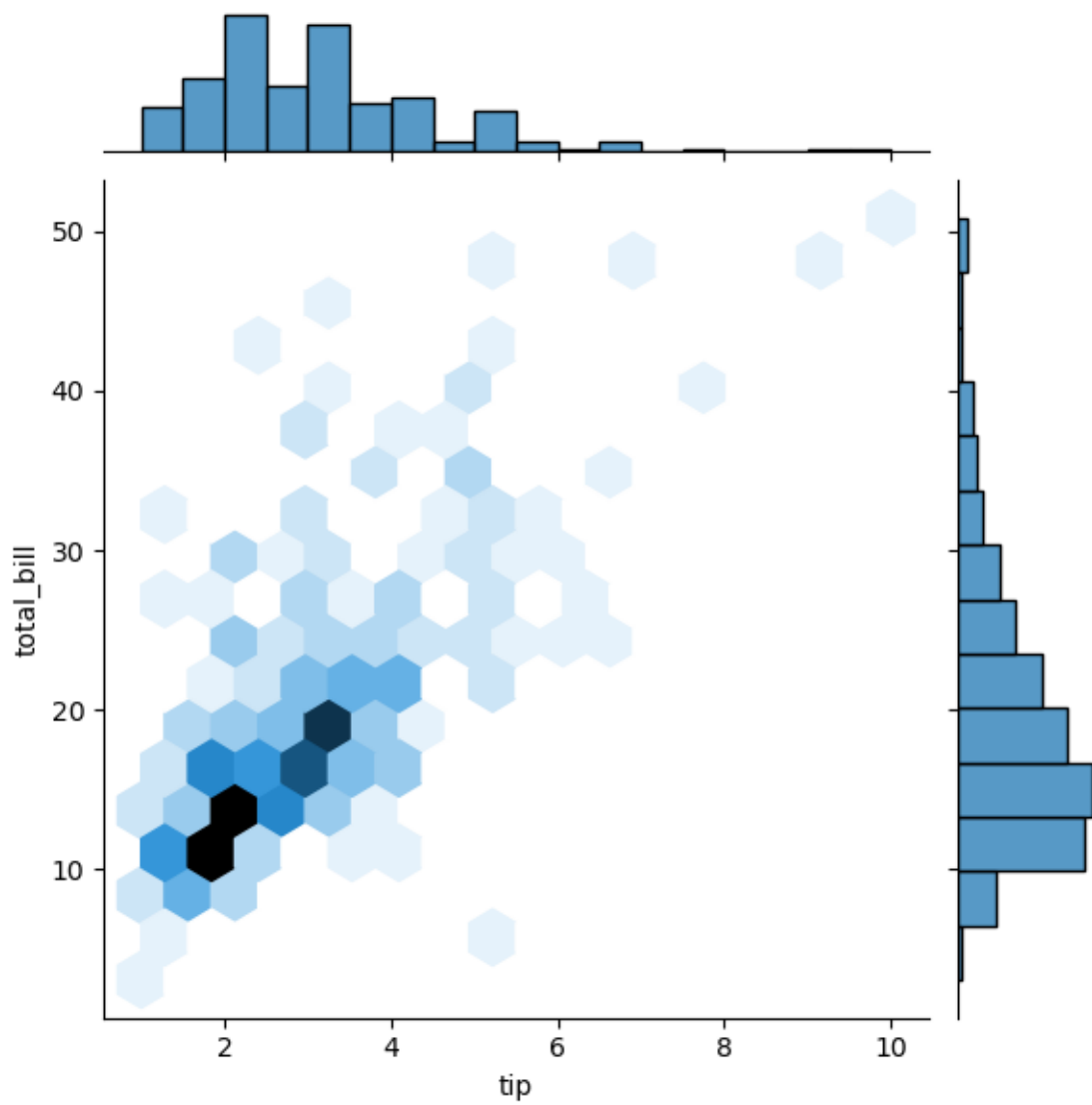
```
[22]: sn.jointplot(x=df.tip,y=df.total_bill,kind="reg")
```

```
[22]: <seaborn.axisgrid.JointGrid at 0x1928606d1d0>
```



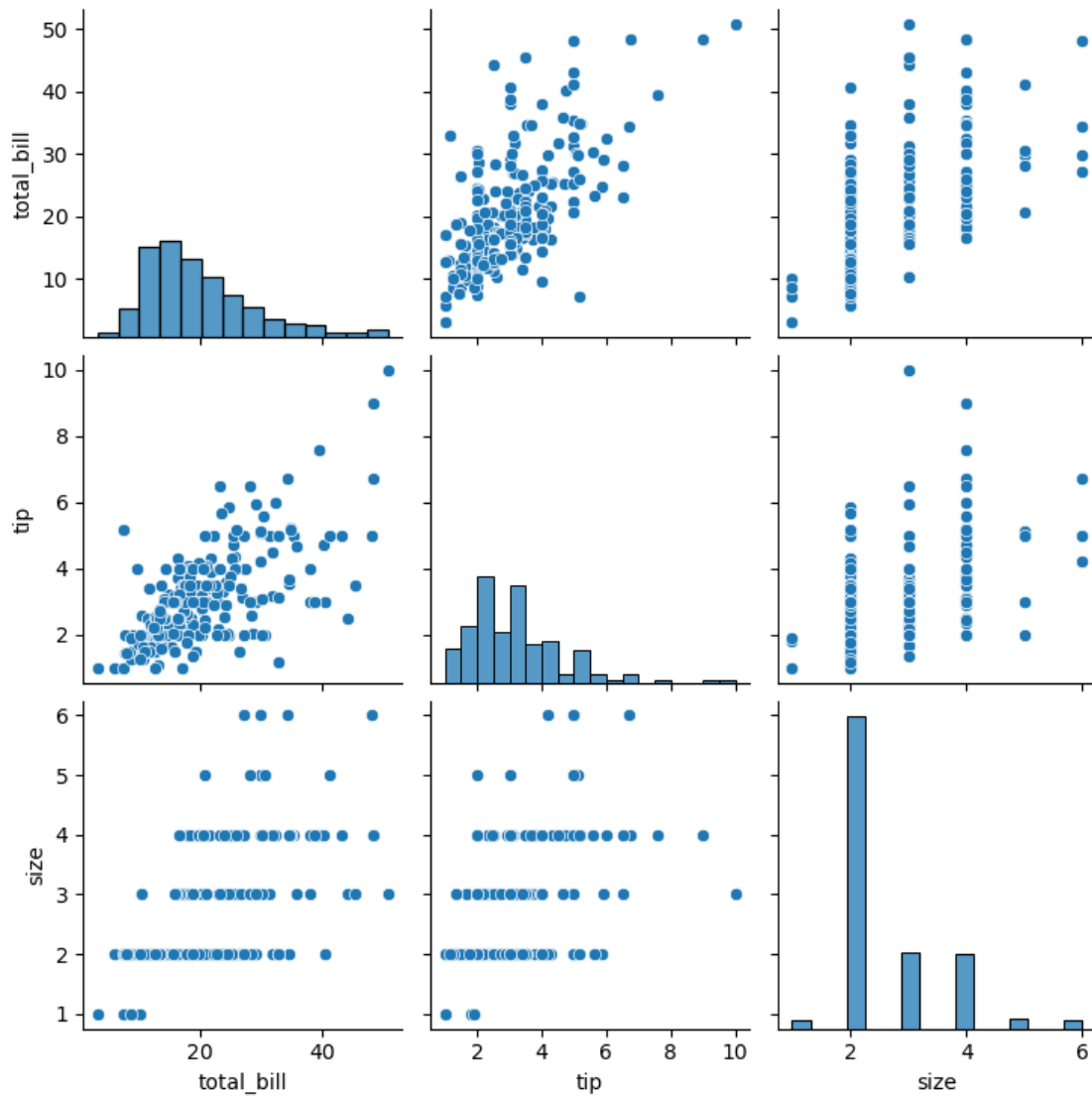
```
[25]: sn.jointplot(x=df.tip,y=df.total_bill,kind="hex")
```

```
[25]: <seaborn.axisgrid.JointGrid at 0x19286a5c050>
```

```
[26]: sn.pairplot(df)
```

```
[26]: <seaborn.axisgrid.PairGrid at 0x192fe45b4d0>
```

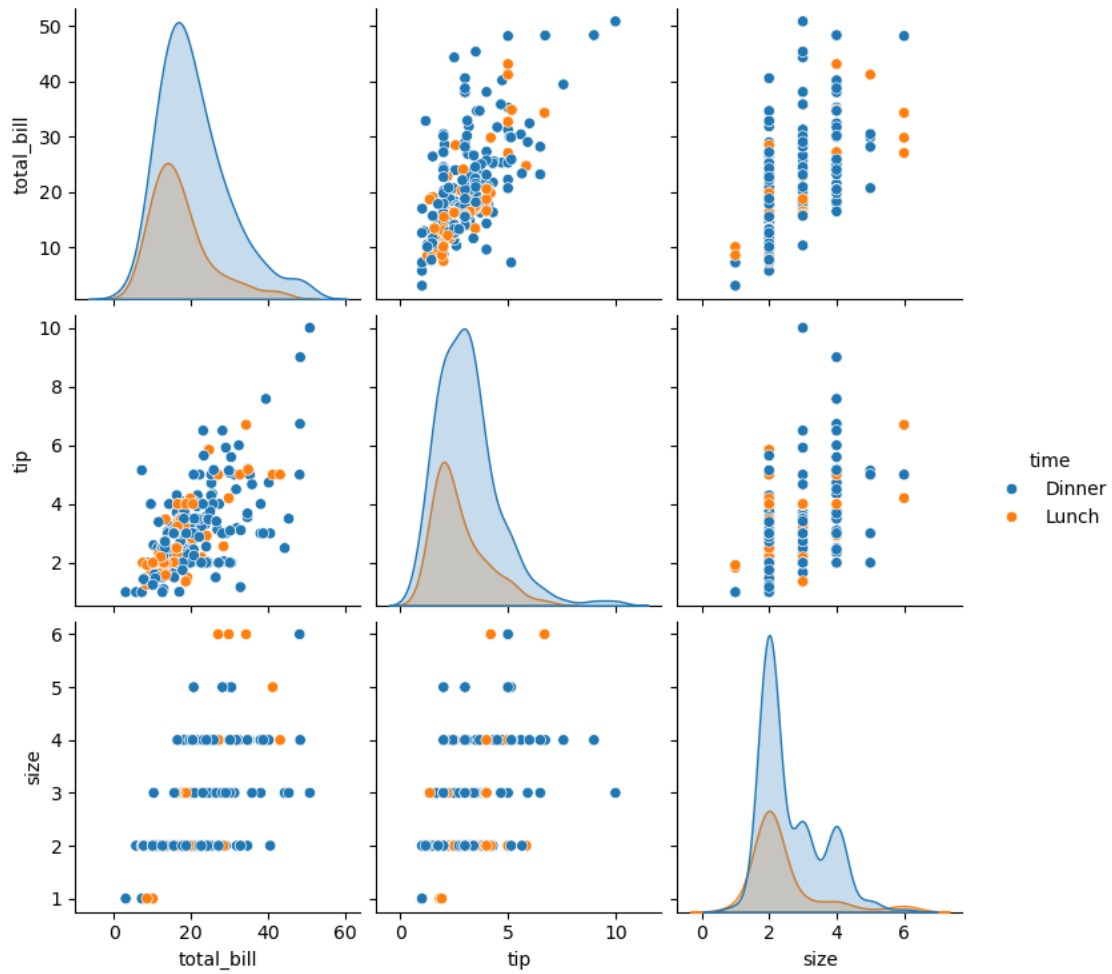


```
[29]: #df.info()
df.time.value_counts()
```

```
[29]: time
Dinner    176
Lunch      68
Name: count, dtype: int64
```

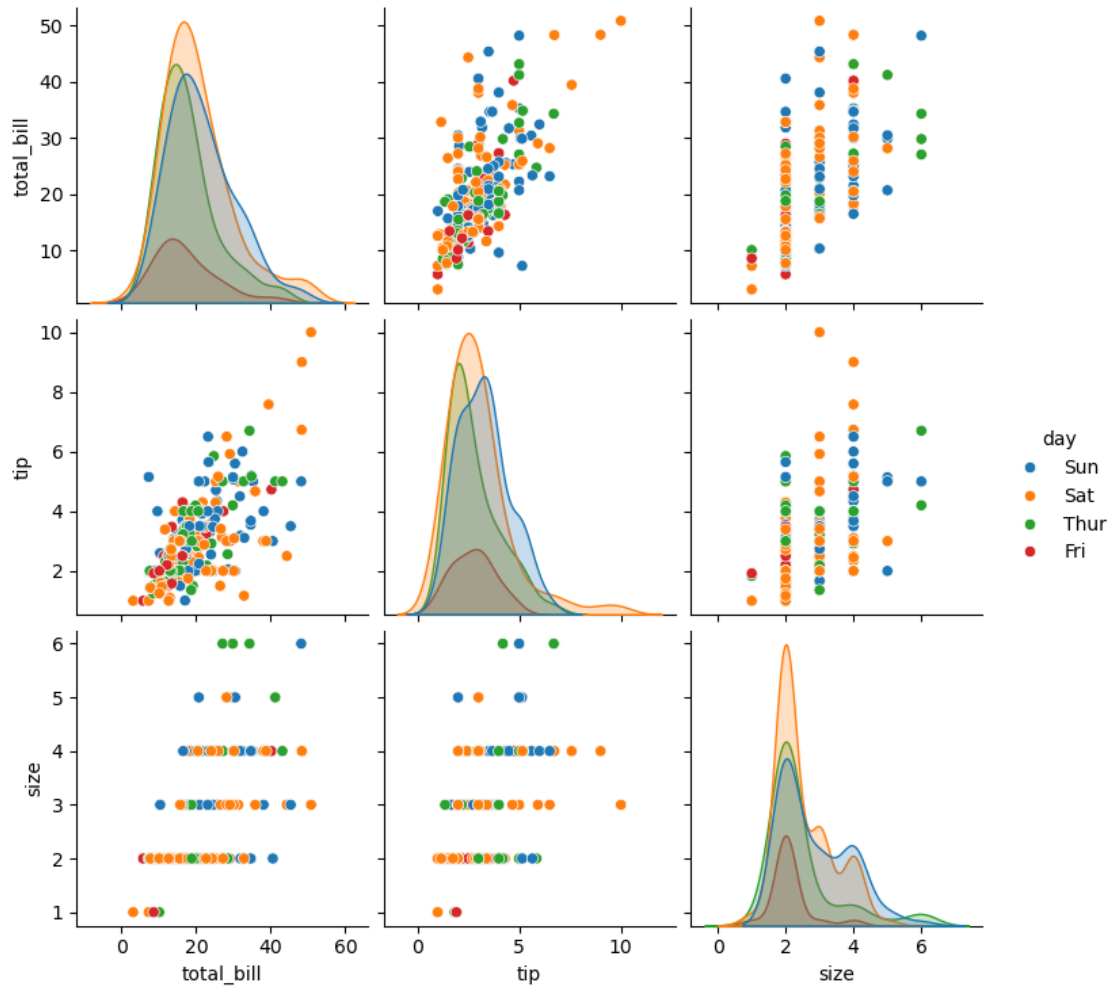
```
[30]: sn.pairplot(df, hue='time')
```

```
[30]: <seaborn.axisgrid.PairGrid at 0x192885e7390>
```



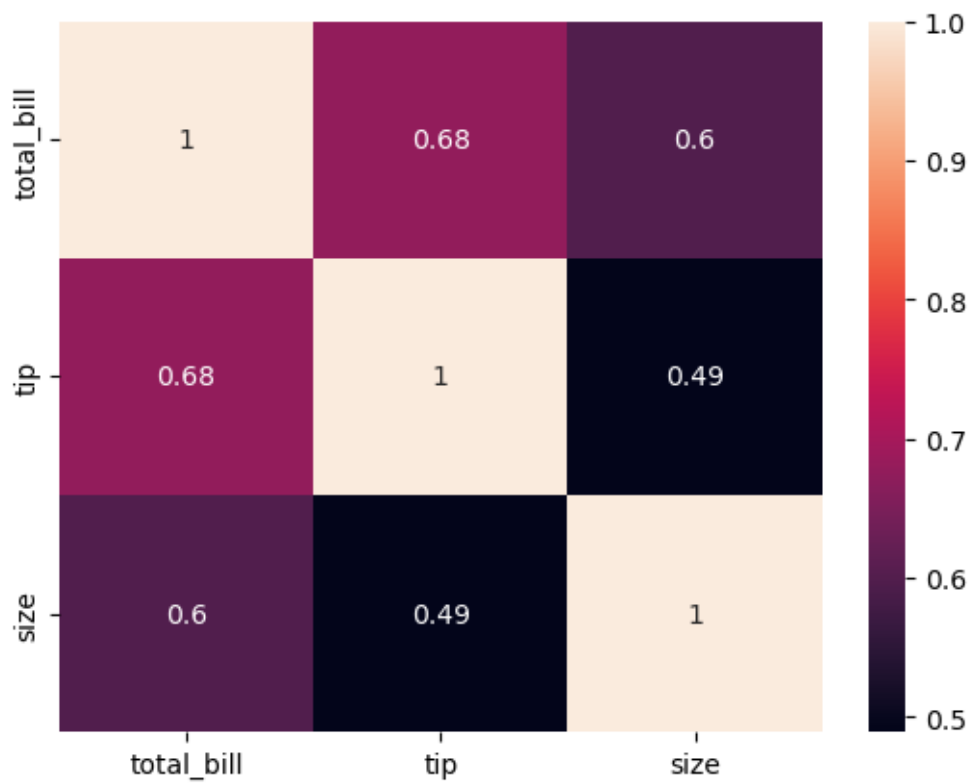
```
[31]: sn.pairplot(df,hue='day')
```

```
[31]: <seaborn.axisgrid.PairGrid at 0x1928939cf50>
```



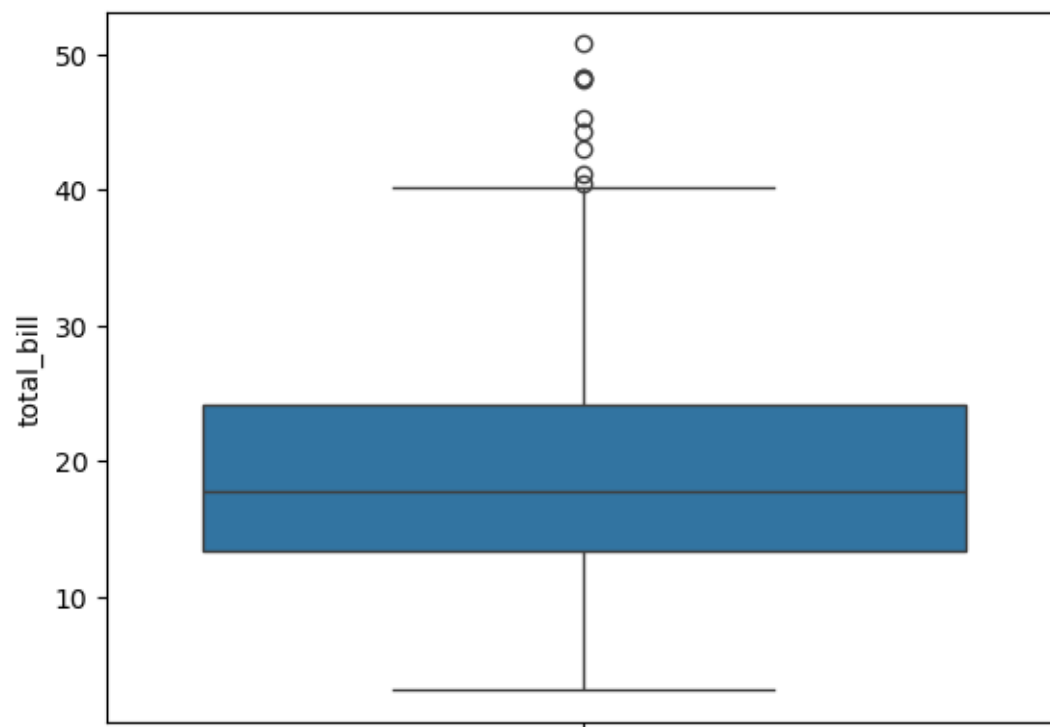
```
[32]: sn.heatmap(df.corr(numeric_only=True),annot=True)
```

```
[32]: <Axes: >
```



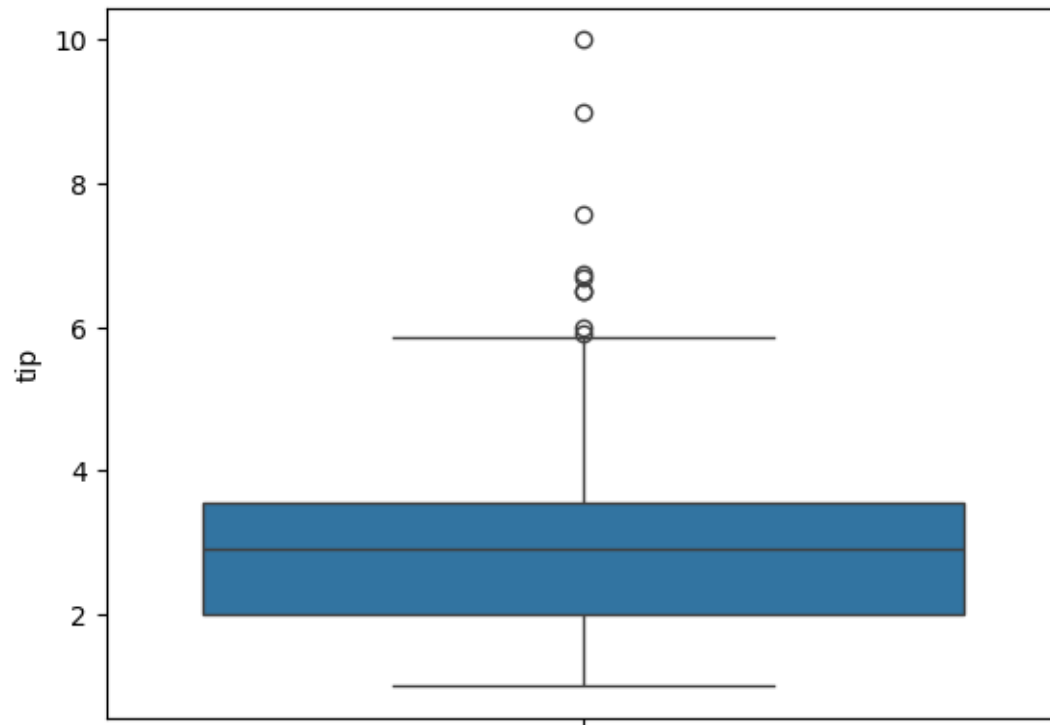
```
[34]: sn.boxplot(df.total_bill)
```

```
[34]: <Axes: ylabel='total_bill'>
```



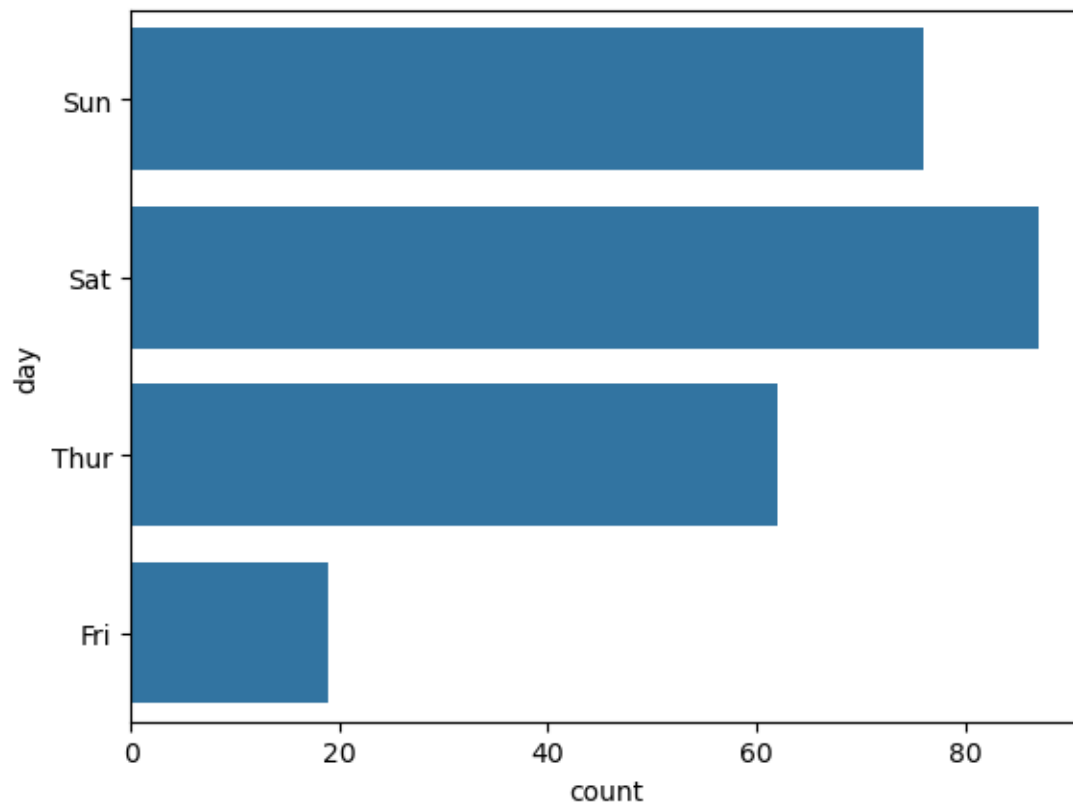
```
[36]: sn.boxplot(df.tip)
```

```
[36]: <Axes: ylabel='tip'>
```



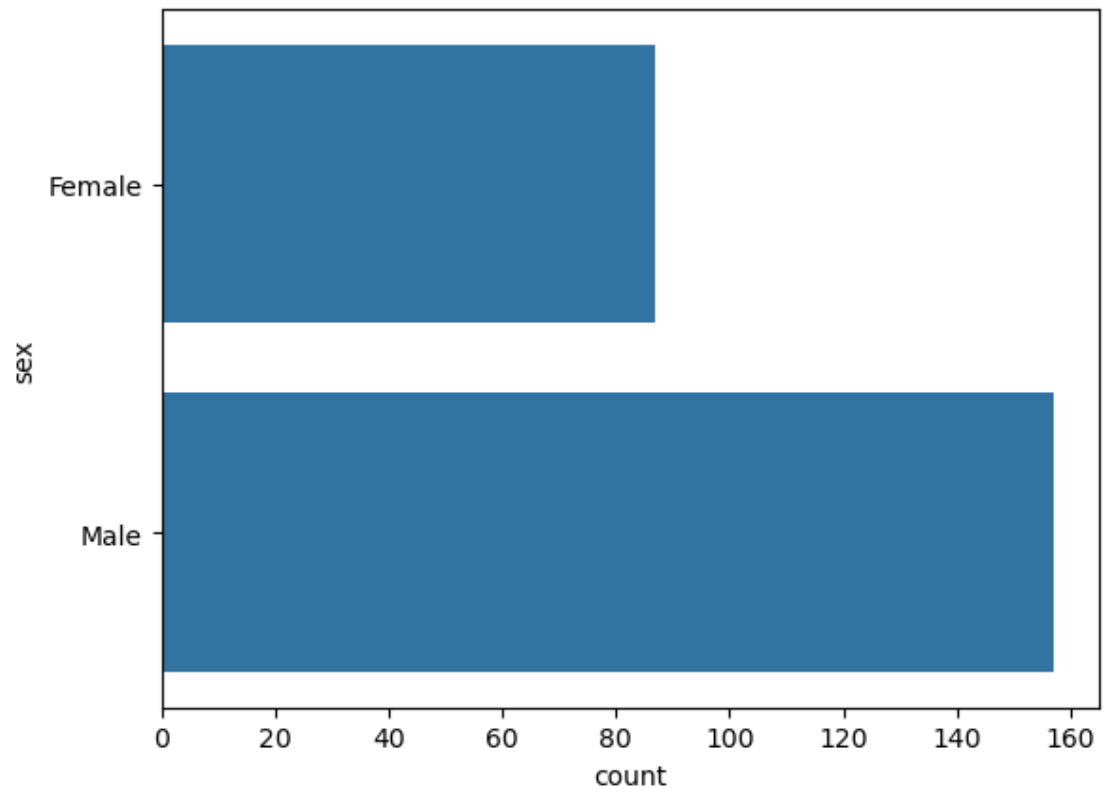
```
[37]: sn.countplot(df.day)
```

```
[37]: <Axes: xlabel='count', ylabel='day'>
```



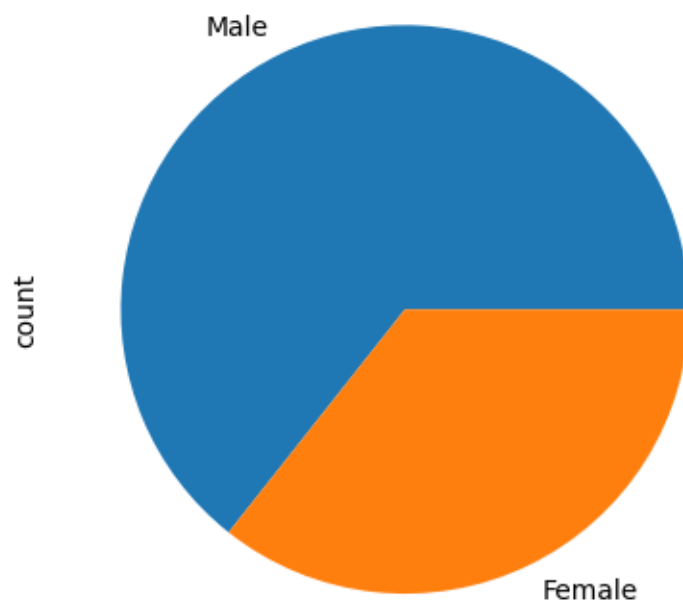
```
[38]: sn.countplot(df.sex)
```

```
[38]: <Axes: xlabel='count', ylabel='sex'>
```

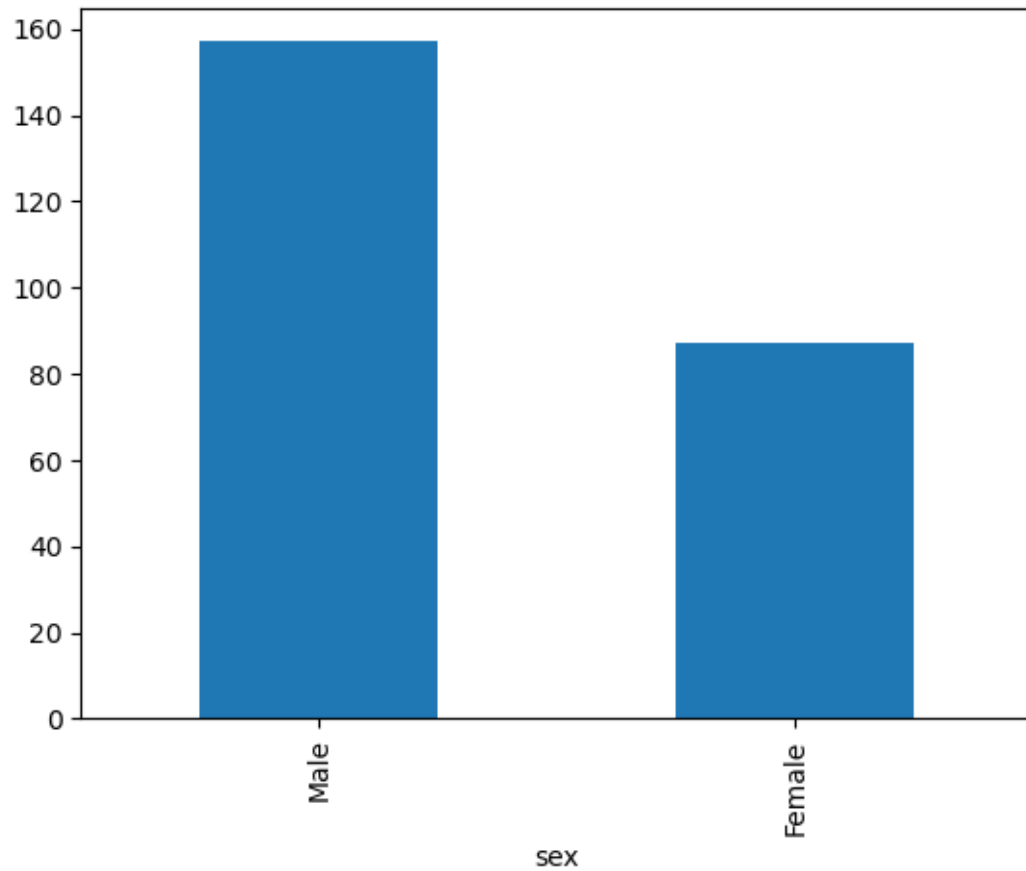
```
[39]: df.sex.value_counts().plot(kind='pie')
```

```
[39]: <Axes: ylabel='count'>
```



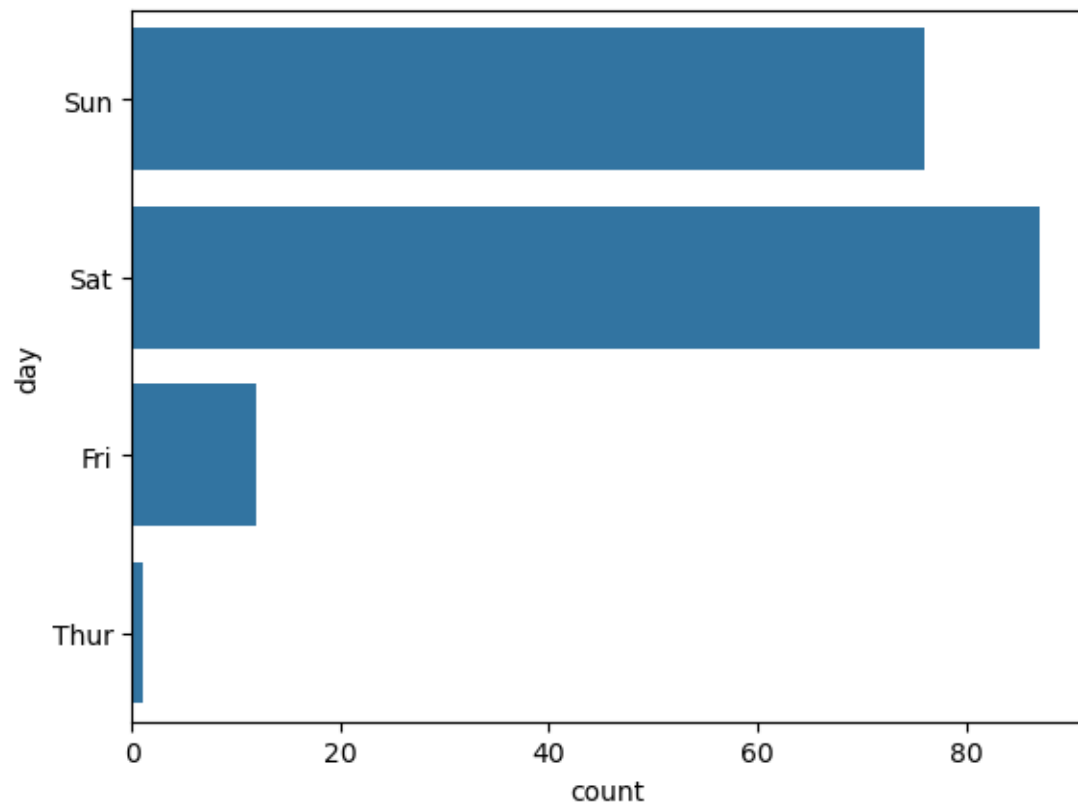
```
[40]: df.sex.value_counts().plot(kind='bar')
```

```
[40]: <Axes: xlabel='sex'>
```



```
[41]: sn.countplot(df[df.time=='Dinner']['day'])
```

```
[41]: <Axes: xlabel='count', ylabel='day'>
```



```
[ ]:
```

regression_and_exercise_7

November 2, 2025

```
[1]: import pandas as pd  
df=pd.read_csv('Salary_data.csv')
```

```
[2]: df.head(5)
```

```
[2]:  YearsExperience  Salary  
0             1.1    39343  
1             1.3    46205  
2             1.5    37731  
3             2.0    43525  
4             2.2    39891
```

```
[3]: df.dropna()
```

```
[3]:  YearsExperience  Salary  
0             1.1    39343  
1             1.3    46205  
2             1.5    37731  
3             2.0    43525  
4             2.2    39891  
5             2.9    56642  
6             3.0    60150  
7             3.2    54445  
8             3.2    64445  
9             3.7    57189  
10            3.9    63218  
11            4.0    55794  
12            4.0    56957  
13            4.1    57081  
14            4.5    61111  
15            4.9    67938  
16            5.1    66029  
17            5.3    83088  
18            5.9    81363  
19            6.0    93940  
20            6.8    91738  
21            7.1    98273  
22            7.9   101302
```

```

23          8.2  113812
24          8.7  109431
25          9.0  105582
26          9.5  116969
27          9.6  112635
28         10.3  122391
29         10.5  121872

```

```
[4]: x=df.iloc[:,[0]].values
     y=df.iloc[:,[1]].values
```

```
[5]: from sklearn.model_selection import train_test_split
```

```
[6]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
[7]: from sklearn.linear_model import LinearRegression
```

```
[8]: model=LinearRegression()#this is the stage where i create a model which has no
     ↳data an empty model with no knowledge
```

```
[9]: model
```

```
[9]: LinearRegression()
```

```
[10]: model.fit(x_train,y_train)#model is trained with the data of x and y
```

```
[10]: LinearRegression()
```

```
[11]: model.predict([[5]])
```

```
[11]: array([[73342.97478427]])
```

```
[12]: y_pred=model.predict(x_test)
```

```
[13]: y_pred
```

```
[13]: array([[ 40748.96184072],
             [122699.62295594],
             [ 64961.65717022],
             [ 63099.14214487],
             [115249.56285456],
             [107799.50275317]])
```

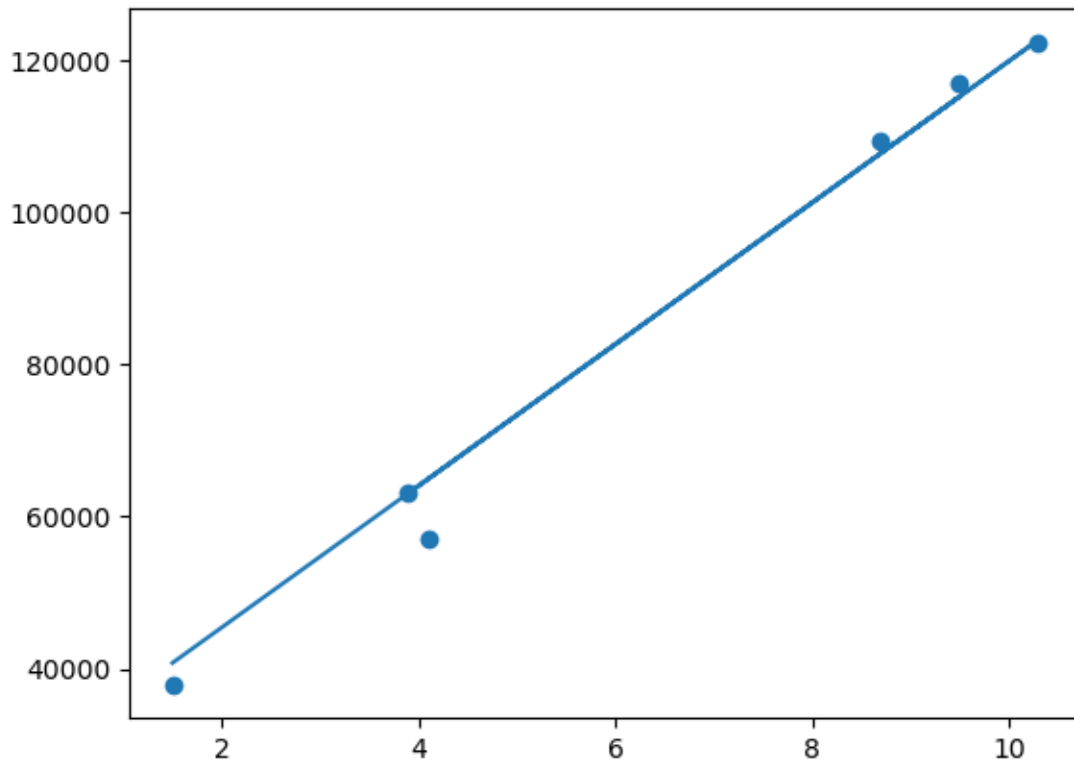
```
[14]: errors=y_pred-y_test
     errors
```

```
[14]: array([[ 3017.96184072],
             [ 308.62295594],
```

```
[ 7880.65717022],  
[ -118.85785513],  
[-1719.43714544],  
[-1631.49724683]])
```

```
[15]: import matplotlib.pyplot as plt  
plt.scatter(x_test,y_test)  
plt.plot(x_test,y_pred)
```

```
[15]: [<matplotlib.lines.Line2D at 0x26ea0de2fd0>]
```



```
[16]: from sklearn.metrics import r2_score  
accuracy=r2_score(y_test,y_pred)
```

```
[17]: accuracy
```

```
[17]: 0.988169515729126
```

```
[18]: model.predict([[44]])
```

```
[18]: array([[436533.40472671]])
```

```
[19]: model.score(x_train,y_train)#This tells how the model regression fits this model
```

```
[19]: 0.9411949620562126
```

```
[20]: model.score(x_test,y_test)
```

```
[20]: 0.988169515729126
```

```
[21]: model.coef_#the coefficient is the slope of the best-fit line.
```

```
[21]: array([[9312.57512673]])
```

```
[22]: model.intercept_
```

```
[22]: array([26780.09915063])
```

```
[23]: model.predict([[55]])
```

```
[23]: array([[538971.73112073]])
```

```
[ ]:
```


Exercise8

November 2, 2025

```
[1]: import pandas as pd
df=pd.read_csv('Iris (1).csv')
```

```
[8]: df.head(5)
```

```
[8]:   sepal.length  sepal.width  petal.length  petal.width  variety
0           5.1           3.5           1.4           0.2   Setosa
1           4.9           3.0           1.4           0.2   Setosa
2           4.7           3.2           1.3           0.2   Setosa
3           4.6           3.1           1.5           0.2   Setosa
4           5.0           3.6           1.4           0.2   Setosa
```

```
[9]: df.variety.value_counts()
```

```
[9]: variety
Setosa      50
Versicolor  50
Virginica   50
Name: count, dtype: int64
```

```
[10]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   sepal.length    150 non-null   float64
1   sepal.width     150 non-null   float64
2   petal.length    150 non-null   float64
3   petal.width     150 non-null   float64
4   variety         150 non-null   object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
[16]: df["output"] = 0
```

```
[42]: df.loc[:49, "output"] = 1
```

```
[43]: df.loc[50:99,"output"]=2
```

```
[44]: df.loc[100:149,"output"]=3
```

```
[45]: df.head(5)
```

```
[45]:
```

	sepal.length	sepal.width	petal.length	petal.width	variety	output
0	5.1	3.5	1.4	0.2	Setosa	1
1	4.9	3.0	1.4	0.2	Setosa	1
2	4.7	3.2	1.3	0.2	Setosa	1
3	4.6	3.1	1.5	0.2	Setosa	1
4	5.0	3.6	1.4	0.2	Setosa	1

```
[46]: df.tail(5)
```

```
[46]:
```

	sepal.length	sepal.width	petal.length	petal.width	variety	output
145	6.7	3.0	5.2	2.3	Virginica	3
146	6.3	2.5	5.0	1.9	Virginica	3
147	6.5	3.0	5.2	2.0	Virginica	3
148	6.2	3.4	5.4	2.3	Virginica	3
149	5.9	3.0	5.1	1.8	Virginica	3

```
[47]: from sklearn.model_selection import train_test_split
```

```
[48]: feature=df  
label=df
```

```
[56]: feature=df.drop("output",axis=1)  
for col in feature.columns:  
    if feature[col].dtype == 'object':  
        le = LabelEncoder()  
        feature[col] = le.fit_transform(feature[col])  
label=df["output"]
```

```
[57]: feature
```

```
[57]:
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
..
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

[150 rows x 5 columns]

```
[58]: label
```

```
[58]: 0      1
      1      1
      2      1
      3      1
      4      1
      ..
     145     3
     146     3
     147     3
     148     3
     149     3
      Name: output, Length: 150, dtype: int64
```

```
[59]: X_train,X_test,Y_train,y_test=train_test_split(feature,label,test_size=0.
      ↪2,random_state=1)
```

```
[60]: from sklearn.neighbors import KNeighborsClassifier
```

```
[61]: op=KNeighborsClassifier(n_neighbors=5)
```

```
[62]: op.fit(X_train,Y_train)
```

```
[62]: KNeighborsClassifier()
```

```
[64]: print(op.score(X_test,y_test))
```

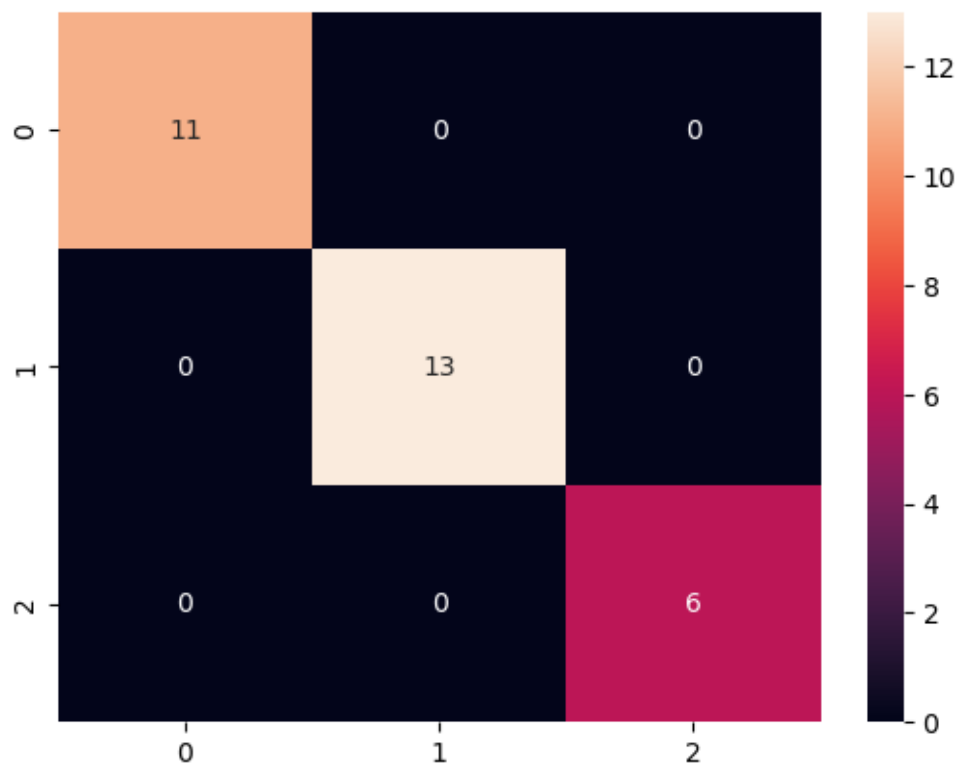
1.0

```
[65]: from sklearn.metrics import confusion_matrix
      y_pred=op.predict(X_test)
```

```
[69]: c_n_m=confusion_matrix(y_test,y_pred)
```

```
[70]: import seaborn as sn
      sn.heatmap(c_n_m,annot=True)
```

```
[70]: <Axes: >
```



```
[72]: from sklearn.metrics import classification_report
print(classification_report(label,op.predict(feature)))
```

	precision	recall	f1-score	support
1	1.00	1.00	1.00	50
2	1.00	1.00	1.00	50
3	1.00	1.00	1.00	50
accuracy			1.00	150
macro avg	1.00	1.00	1.00	150
weighted avg	1.00	1.00	1.00	150

```
[ ]:
```

Exercise-9

November 2, 2025

```
[1]: import pandas as pd
df=pd.read_csv('Social_Network_Ads.csv')
```

```
[3]: import numpy as np
import pandas as pd
```

```
[4]: features=df.iloc[:,[2,3]].values
label=df.iloc[:,4].values
features
```

```
[4]: array([[ 19, 19000],
 [ 35, 20000],
 [ 26, 43000],
 [ 27, 57000],
 [ 19, 76000],
 [ 27, 58000],
 [ 27, 84000],
 [ 32, 150000],
 [ 25, 33000],
 [ 35, 65000],
 [ 26, 80000],
 [ 26, 52000],
 [ 20, 86000],
 [ 32, 18000],
 [ 18, 82000],
 [ 29, 80000],
 [ 47, 25000],
 [ 45, 26000],
 [ 46, 28000],
 [ 48, 29000],
 [ 45, 22000],
 [ 47, 49000],
 [ 48, 41000],
 [ 45, 22000],
 [ 46, 23000],
 [ 47, 20000],
 [ 49, 28000],
 [ 47, 30000],
```

```

[ 29, 43000],
[ 31, 18000],
[ 31, 74000],
[ 27, 137000],
[ 21, 16000],
[ 28, 44000],
[ 27, 90000],
[ 35, 27000],
[ 33, 28000],
[ 30, 49000],
[ 26, 72000],
[ 27, 31000],
[ 27, 17000],
[ 33, 51000],
[ 35, 108000],
[ 30, 15000],
[ 28, 84000],
[ 23, 20000],
[ 25, 79000],
[ 27, 54000],
[ 30, 135000],
[ 31, 89000],
[ 24, 32000],
[ 18, 44000],
[ 29, 83000],
[ 35, 23000],
[ 27, 58000],
[ 24, 55000],
[ 23, 48000],
[ 28, 79000],
[ 22, 18000],
[ 32, 117000],
[ 27, 20000],
[ 25, 87000],
[ 23, 66000],
[ 32, 120000],
[ 59, 83000],
[ 24, 58000],
[ 24, 19000],
[ 23, 82000],
[ 22, 63000],
[ 31, 68000],
[ 25, 80000],
[ 24, 27000],
[ 20, 23000],
[ 33, 113000],
[ 32, 18000],

```

```

[ 34, 112000],
[ 18, 52000],
[ 22, 27000],
[ 28, 87000],
[ 26, 17000],
[ 30, 80000],
[ 39, 42000],
[ 20, 49000],
[ 35, 88000],
[ 30, 62000],
[ 31, 118000],
[ 24, 55000],
[ 28, 85000],
[ 26, 81000],
[ 35, 50000],
[ 22, 81000],
[ 30, 116000],
[ 26, 15000],
[ 29, 28000],
[ 29, 83000],
[ 35, 44000],
[ 35, 25000],
[ 28, 123000],
[ 35, 73000],
[ 28, 37000],
[ 27, 88000],
[ 28, 59000],
[ 32, 86000],
[ 33, 149000],
[ 19, 21000],
[ 21, 72000],
[ 26, 35000],
[ 27, 89000],
[ 26, 86000],
[ 38, 80000],
[ 39, 71000],
[ 37, 71000],
[ 38, 61000],
[ 37, 55000],
[ 42, 80000],
[ 40, 57000],
[ 35, 75000],
[ 36, 52000],
[ 40, 59000],
[ 41, 59000],
[ 36, 75000],
[ 37, 72000],

```

```

[ 40, 75000],
[ 35, 53000],
[ 41, 51000],
[ 39, 61000],
[ 42, 65000],
[ 26, 32000],
[ 30, 17000],
[ 26, 84000],
[ 31, 58000],
[ 33, 31000],
[ 30, 87000],
[ 21, 68000],
[ 28, 55000],
[ 23, 63000],
[ 20, 82000],
[ 30, 107000],
[ 28, 59000],
[ 19, 25000],
[ 19, 85000],
[ 18, 68000],
[ 35, 59000],
[ 30, 89000],
[ 34, 25000],
[ 24, 89000],
[ 27, 96000],
[ 41, 30000],
[ 29, 61000],
[ 20, 74000],
[ 26, 15000],
[ 41, 45000],
[ 31, 76000],
[ 36, 50000],
[ 40, 47000],
[ 31, 15000],
[ 46, 59000],
[ 29, 75000],
[ 26, 30000],
[ 32, 135000],
[ 32, 100000],
[ 25, 90000],
[ 37, 33000],
[ 35, 38000],
[ 33, 69000],
[ 18, 86000],
[ 22, 55000],
[ 35, 71000],
[ 29, 148000],

```



```

[ 29, 47000],
[ 21, 88000],
[ 34, 115000],
[ 26, 118000],
[ 34, 43000],
[ 34, 72000],
[ 23, 28000],
[ 35, 47000],
[ 25, 22000],
[ 24, 23000],
[ 31, 34000],
[ 26, 16000],
[ 31, 71000],
[ 32, 117000],
[ 33, 43000],
[ 33, 60000],
[ 31, 66000],
[ 20, 82000],
[ 33, 41000],
[ 35, 72000],
[ 28, 32000],
[ 24, 84000],
[ 19, 26000],
[ 29, 43000],
[ 19, 70000],
[ 28, 89000],
[ 34, 43000],
[ 30, 79000],
[ 20, 36000],
[ 26, 80000],
[ 35, 22000],
[ 35, 39000],
[ 49, 74000],
[ 39, 134000],
[ 41, 71000],
[ 58, 101000],
[ 47, 47000],
[ 55, 130000],
[ 52, 114000],
[ 40, 142000],
[ 46, 22000],
[ 48, 96000],
[ 52, 150000],
[ 59, 42000],
[ 35, 58000],
[ 47, 43000],
[ 60, 108000],

```

[49, 65000],
[40, 78000],
[46, 96000],
[59, 143000],
[41, 80000],
[35, 91000],
[37, 144000],
[60, 102000],
[35, 60000],
[37, 53000],
[36, 126000],
[56, 133000],
[40, 72000],
[42, 80000],
[35, 147000],
[39, 42000],
[40, 107000],
[49, 86000],
[38, 112000],
[46, 79000],
[40, 57000],
[37, 80000],
[46, 82000],
[53, 143000],
[42, 149000],
[38, 59000],
[50, 88000],
[56, 104000],
[41, 72000],
[51, 146000],
[35, 50000],
[57, 122000],
[41, 52000],
[35, 97000],
[44, 39000],
[37, 52000],
[48, 134000],
[37, 146000],
[50, 44000],
[52, 90000],
[41, 72000],
[40, 57000],
[58, 95000],
[45, 131000],
[35, 77000],
[36, 144000],
[55, 125000],

[35, 72000],
[48, 90000],
[42, 108000],
[40, 75000],
[37, 74000],
[47, 144000],
[40, 61000],
[43, 133000],
[59, 76000],
[60, 42000],
[39, 106000],
[57, 26000],
[57, 74000],
[38, 71000],
[49, 88000],
[52, 38000],
[50, 36000],
[59, 88000],
[35, 61000],
[37, 70000],
[52, 21000],
[48, 141000],
[37, 93000],
[37, 62000],
[48, 138000],
[41, 79000],
[37, 78000],
[39, 134000],
[49, 89000],
[55, 39000],
[37, 77000],
[35, 57000],
[36, 63000],
[42, 73000],
[43, 112000],
[45, 79000],
[46, 117000],
[58, 38000],
[48, 74000],
[37, 137000],
[37, 79000],
[40, 60000],
[42, 54000],
[51, 134000],
[47, 113000],
[36, 125000],
[38, 50000],

[42, 70000],
[39, 96000],
[38, 50000],
[49, 141000],
[39, 79000],
[39, 75000],
[54, 104000],
[35, 55000],
[45, 32000],
[36, 60000],
[52, 138000],
[53, 82000],
[41, 52000],
[48, 30000],
[48, 131000],
[41, 60000],
[41, 72000],
[42, 75000],
[36, 118000],
[47, 107000],
[38, 51000],
[48, 119000],
[42, 65000],
[40, 65000],
[57, 60000],
[36, 54000],
[58, 144000],
[35, 79000],
[38, 55000],
[39, 122000],
[53, 104000],
[35, 75000],
[38, 65000],
[47, 51000],
[47, 105000],
[41, 63000],
[53, 72000],
[54, 108000],
[39, 77000],
[38, 61000],
[38, 113000],
[37, 75000],
[42, 90000],
[37, 57000],
[36, 99000],
[60, 34000],
[54, 70000],

```
[ 41, 72000],
[ 40, 71000],
[ 42, 54000],
[ 43, 129000],
[ 53, 34000],
[ 47, 50000],
[ 42, 79000],
[ 42, 104000],
[ 59, 29000],
[ 58, 47000],
[ 46, 88000],
[ 38, 71000],
[ 54, 26000],
[ 60, 46000],
[ 60, 83000],
[ 39, 73000],
[ 59, 130000],
[ 37, 80000],
[ 46, 32000],
[ 46, 74000],
[ 42, 53000],
[ 41, 87000],
[ 58, 23000],
[ 42, 64000],
[ 48, 33000],
[ 44, 139000],
[ 49, 28000],
[ 57, 33000],
[ 56, 60000],
[ 49, 39000],
[ 39, 71000],
[ 47, 34000],
[ 48, 35000],
[ 48, 33000],
[ 47, 23000],
[ 45, 45000],
[ 60, 42000],
[ 39, 59000],
[ 46, 41000],
[ 51, 23000],
[ 50, 20000],
[ 36, 33000],
[ 49, 36000]])
```

```
[5]: from sklearn.model_selection import train_test_split
      from sklearn.linear_model import LogisticRegression
```

```
[6]: for i in range(1, 401):
    x_train, x_test, y_train, y_test = train_test_split(features, label,
    ↪test_size=0.2, random_state=i)

    model = LogisticRegression()
    model.fit(x_train, y_train)

    train_score = model.score(x_train, y_train)
    test_score = model.score(x_test, y_test)

    if test_score > train_score:
        print("Test {:.3f} Train {:.3f} Random State {}".format(test_score,
    ↪train_score, i))
```

```
Test 0.900 Train 0.841 Random State 4
Test 0.863 Train 0.850 Random State 5
Test 0.863 Train 0.859 Random State 6
Test 0.887 Train 0.838 Random State 7
Test 0.863 Train 0.838 Random State 9
Test 0.900 Train 0.841 Random State 10
Test 0.863 Train 0.856 Random State 14
Test 0.850 Train 0.844 Random State 15
Test 0.863 Train 0.856 Random State 16
Test 0.875 Train 0.834 Random State 18
Test 0.850 Train 0.844 Random State 19
Test 0.875 Train 0.844 Random State 20
Test 0.863 Train 0.834 Random State 21
Test 0.875 Train 0.841 Random State 22
Test 0.875 Train 0.841 Random State 24
Test 0.850 Train 0.834 Random State 26
Test 0.850 Train 0.841 Random State 27
Test 0.863 Train 0.834 Random State 30
Test 0.863 Train 0.856 Random State 31
Test 0.875 Train 0.853 Random State 32
Test 0.863 Train 0.844 Random State 33
Test 0.875 Train 0.831 Random State 35
Test 0.863 Train 0.853 Random State 36
Test 0.887 Train 0.841 Random State 38
Test 0.875 Train 0.838 Random State 39
Test 0.887 Train 0.838 Random State 42
Test 0.875 Train 0.847 Random State 46
Test 0.912 Train 0.831 Random State 47
Test 0.875 Train 0.831 Random State 51
Test 0.900 Train 0.844 Random State 54
Test 0.850 Train 0.844 Random State 57
Test 0.875 Train 0.844 Random State 58
Test 0.925 Train 0.838 Random State 61
Test 0.887 Train 0.834 Random State 65
```

Test 0.887 Train 0.841 Random State 68
Test 0.900 Train 0.831 Random State 72
Test 0.887 Train 0.838 Random State 75
Test 0.925 Train 0.825 Random State 76
Test 0.863 Train 0.841 Random State 77
Test 0.863 Train 0.859 Random State 81
Test 0.875 Train 0.838 Random State 82
Test 0.887 Train 0.838 Random State 83
Test 0.863 Train 0.853 Random State 84
Test 0.863 Train 0.841 Random State 85
Test 0.863 Train 0.841 Random State 87
Test 0.875 Train 0.847 Random State 88
Test 0.912 Train 0.838 Random State 90
Test 0.863 Train 0.850 Random State 95
Test 0.875 Train 0.850 Random State 99
Test 0.850 Train 0.841 Random State 101
Test 0.850 Train 0.841 Random State 102
Test 0.900 Train 0.825 Random State 106
Test 0.863 Train 0.841 Random State 107
Test 0.850 Train 0.834 Random State 109
Test 0.850 Train 0.841 Random State 111
Test 0.912 Train 0.841 Random State 112
Test 0.863 Train 0.850 Random State 115
Test 0.863 Train 0.841 Random State 116
Test 0.875 Train 0.834 Random State 119
Test 0.912 Train 0.828 Random State 120
Test 0.863 Train 0.859 Random State 125
Test 0.850 Train 0.847 Random State 128
Test 0.875 Train 0.850 Random State 130
Test 0.900 Train 0.844 Random State 133
Test 0.925 Train 0.834 Random State 134
Test 0.863 Train 0.850 Random State 135
Test 0.875 Train 0.831 Random State 138
Test 0.863 Train 0.850 Random State 141
Test 0.850 Train 0.847 Random State 143
Test 0.850 Train 0.847 Random State 146
Test 0.850 Train 0.844 Random State 147
Test 0.863 Train 0.850 Random State 148
Test 0.875 Train 0.838 Random State 150
Test 0.887 Train 0.831 Random State 151
Test 0.925 Train 0.844 Random State 152
Test 0.850 Train 0.841 Random State 153
Test 0.900 Train 0.844 Random State 154
Test 0.900 Train 0.841 Random State 155
Test 0.887 Train 0.847 Random State 156
Test 0.887 Train 0.834 Random State 158
Test 0.875 Train 0.828 Random State 159
Test 0.900 Train 0.831 Random State 161

Test 0.850 Train 0.838 Random State 163
Test 0.875 Train 0.831 Random State 164
Test 0.863 Train 0.850 Random State 169
Test 0.875 Train 0.841 Random State 171
Test 0.850 Train 0.841 Random State 172
Test 0.900 Train 0.825 Random State 180
Test 0.850 Train 0.834 Random State 184
Test 0.925 Train 0.822 Random State 186
Test 0.900 Train 0.831 Random State 193
Test 0.863 Train 0.850 Random State 195
Test 0.863 Train 0.841 Random State 196
Test 0.863 Train 0.838 Random State 197
Test 0.875 Train 0.841 Random State 198
Test 0.887 Train 0.838 Random State 199
Test 0.887 Train 0.844 Random State 200
Test 0.863 Train 0.838 Random State 202
Test 0.863 Train 0.841 Random State 203
Test 0.887 Train 0.831 Random State 206
Test 0.863 Train 0.834 Random State 211
Test 0.850 Train 0.844 Random State 212
Test 0.863 Train 0.834 Random State 214
Test 0.875 Train 0.831 Random State 217
Test 0.963 Train 0.819 Random State 220
Test 0.875 Train 0.844 Random State 221
Test 0.850 Train 0.841 Random State 222
Test 0.900 Train 0.844 Random State 223
Test 0.863 Train 0.853 Random State 227
Test 0.863 Train 0.834 Random State 228
Test 0.900 Train 0.841 Random State 229
Test 0.850 Train 0.844 Random State 232
Test 0.875 Train 0.847 Random State 233
Test 0.912 Train 0.841 Random State 234
Test 0.863 Train 0.841 Random State 235
Test 0.850 Train 0.847 Random State 236
Test 0.875 Train 0.847 Random State 239
Test 0.850 Train 0.844 Random State 241
Test 0.887 Train 0.850 Random State 242
Test 0.887 Train 0.825 Random State 243
Test 0.875 Train 0.847 Random State 244
Test 0.875 Train 0.841 Random State 245
Test 0.875 Train 0.847 Random State 246
Test 0.863 Train 0.859 Random State 247
Test 0.887 Train 0.844 Random State 248
Test 0.863 Train 0.850 Random State 250
Test 0.875 Train 0.831 Random State 251
Test 0.887 Train 0.844 Random State 252
Test 0.863 Train 0.847 Random State 255
Test 0.900 Train 0.841 Random State 257

Test 0.863 Train 0.856 Random State 260
Test 0.863 Train 0.841 Random State 266
Test 0.863 Train 0.838 Random State 268
Test 0.875 Train 0.841 Random State 275
Test 0.863 Train 0.850 Random State 276
Test 0.925 Train 0.838 Random State 277
Test 0.875 Train 0.847 Random State 282
Test 0.850 Train 0.847 Random State 283
Test 0.850 Train 0.844 Random State 285
Test 0.912 Train 0.834 Random State 286
Test 0.850 Train 0.841 Random State 290
Test 0.850 Train 0.841 Random State 291
Test 0.850 Train 0.847 Random State 292
Test 0.863 Train 0.838 Random State 294
Test 0.887 Train 0.828 Random State 297
Test 0.863 Train 0.834 Random State 300
Test 0.863 Train 0.850 Random State 301
Test 0.887 Train 0.850 Random State 302
Test 0.875 Train 0.847 Random State 303
Test 0.863 Train 0.834 Random State 305
Test 0.912 Train 0.838 Random State 306
Test 0.875 Train 0.847 Random State 308
Test 0.900 Train 0.844 Random State 311
Test 0.863 Train 0.834 Random State 313
Test 0.912 Train 0.834 Random State 314
Test 0.875 Train 0.838 Random State 315
Test 0.900 Train 0.847 Random State 317
Test 0.912 Train 0.822 Random State 319
Test 0.863 Train 0.850 Random State 321
Test 0.912 Train 0.828 Random State 322
Test 0.850 Train 0.847 Random State 328
Test 0.850 Train 0.838 Random State 332
Test 0.887 Train 0.853 Random State 336
Test 0.850 Train 0.838 Random State 337
Test 0.875 Train 0.841 Random State 343
Test 0.863 Train 0.844 Random State 346
Test 0.887 Train 0.831 Random State 351
Test 0.863 Train 0.850 Random State 352
Test 0.950 Train 0.819 Random State 354
Test 0.863 Train 0.850 Random State 356
Test 0.912 Train 0.841 Random State 357
Test 0.863 Train 0.838 Random State 358
Test 0.850 Train 0.841 Random State 362
Test 0.900 Train 0.844 Random State 363
Test 0.863 Train 0.853 Random State 364
Test 0.938 Train 0.822 Random State 366
Test 0.912 Train 0.841 Random State 369
Test 0.863 Train 0.853 Random State 371

```

Test 0.925 Train 0.834 Random State 376
Test 0.912 Train 0.828 Random State 377
Test 0.887 Train 0.850 Random State 378
Test 0.887 Train 0.850 Random State 379
Test 0.863 Train 0.841 Random State 382
Test 0.863 Train 0.859 Random State 386
Test 0.850 Train 0.838 Random State 387
Test 0.875 Train 0.828 Random State 388
Test 0.850 Train 0.844 Random State 394
Test 0.863 Train 0.838 Random State 395
Test 0.900 Train 0.844 Random State 397
Test 0.863 Train 0.844 Random State 400

```

```

[7]: x_train, x_test, y_train, y_test = train_test_split(features, label,
↳test_size=0.2, random_state=42)
finalModel = LogisticRegression()
finalModel.fit(x_train, y_train)

```

```

[7]: LogisticRegression()

```

```

[8]: print(finalModel.score(x_train,y_train))
print(finalModel.score(x_test,y_test))

```

```

0.8375
0.8875

```

```

[9]: from sklearn.metrics import classification_report
print(classification_report(label,finalModel.predict(features)))

```

	precision	recall	f1-score	support
0	0.85	0.93	0.89	257
1	0.85	0.70	0.77	143
accuracy			0.85	400
macro avg	0.85	0.81	0.83	400
weighted avg	0.85	0.85	0.84	400

```

[ ]:

```

Experiment-10

November 2, 2025

```
[34]: import pandas as pd  
df=pd.read_csv('Mall_Customers.csv')
```

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

```
[36]: feature=df.iloc[:,[3,4]].values
```

```
[37]: feature
```

```
[37]: array([[ 15,  39],  
            [ 15,  81],  
            [ 16,   6],  
            [ 16,  77],  
            [ 17,  40],  
            [ 17,  76],  
            [ 18,   6],  
            [ 18,  94],  
            [ 19,   3],  
            [ 19,  72],  
            [ 19,  14],  
            [ 19,  99],  
            [ 20,  15],  
            [ 20,  77],  
            [ 20,  13],  
            [ 20,  79],  
            [ 21,  35],  
            [ 21,  66],  
            [ 23,  29],  
            [ 23,  98],  
            [ 24,  35],  
            [ 24,  73],  
            [ 25,   5],  
            [ 25,  73],  
            [ 28,  14],
```

[28, 82],
[28, 32],
[28, 61],
[29, 31],
[29, 87],
[30, 4],
[30, 73],
[33, 4],
[33, 92],
[33, 14],
[33, 81],
[34, 17],
[34, 73],
[37, 26],
[37, 75],
[38, 35],
[38, 92],
[39, 36],
[39, 61],
[39, 28],
[39, 65],
[40, 55],
[40, 47],
[40, 42],
[40, 42],
[42, 52],
[42, 60],
[43, 54],
[43, 60],
[43, 45],
[43, 41],
[44, 50],
[44, 46],
[46, 51],
[46, 46],
[46, 56],
[46, 55],
[47, 52],
[47, 59],
[48, 51],
[48, 59],
[48, 50],
[48, 48],
[48, 59],
[48, 47],
[49, 55],
[49, 42],

[50, 49],
[50, 56],
[54, 47],
[54, 54],
[54, 53],
[54, 48],
[54, 52],
[54, 42],
[54, 51],
[54, 55],
[54, 41],
[54, 44],
[54, 57],
[54, 46],
[57, 58],
[57, 55],
[58, 60],
[58, 46],
[59, 55],
[59, 41],
[60, 49],
[60, 40],
[60, 42],
[60, 52],
[60, 47],
[60, 50],
[61, 42],
[61, 49],
[62, 41],
[62, 48],
[62, 59],
[62, 55],
[62, 56],
[62, 42],
[63, 50],
[63, 46],
[63, 43],
[63, 48],
[63, 52],
[63, 54],
[64, 42],
[64, 46],
[65, 48],
[65, 50],
[65, 43],
[65, 59],
[67, 43],

[67, 57],
[67, 56],
[67, 40],
[69, 58],
[69, 91],
[70, 29],
[70, 77],
[71, 35],
[71, 95],
[71, 11],
[71, 75],
[71, 9],
[71, 75],
[72, 34],
[72, 71],
[73, 5],
[73, 88],
[73, 7],
[73, 73],
[74, 10],
[74, 72],
[75, 5],
[75, 93],
[76, 40],
[76, 87],
[77, 12],
[77, 97],
[77, 36],
[77, 74],
[78, 22],
[78, 90],
[78, 17],
[78, 88],
[78, 20],
[78, 76],
[78, 16],
[78, 89],
[78, 1],
[78, 78],
[78, 1],
[78, 73],
[79, 35],
[79, 83],
[81, 5],
[81, 93],
[85, 26],
[85, 75],

```
[ 86, 20],
[ 86, 95],
[ 87, 27],
[ 87, 63],
[ 87, 13],
[ 87, 75],
[ 87, 10],
[ 87, 92],
[ 88, 13],
[ 88, 86],
[ 88, 15],
[ 88, 69],
[ 93, 14],
[ 93, 90],
[ 97, 32],
[ 97, 86],
[ 98, 15],
[ 98, 88],
[ 99, 39],
[ 99, 97],
[101, 24],
[101, 68],
[103, 17],
[103, 85],
[103, 23],
[103, 69],
[113, 8],
[113, 91],
[120, 16],
[120, 79],
[126, 28],
[126, 74],
[137, 18],
[137, 83]])
```

```
[38]: import os
os.environ["OMP_NUM_THREADS"] = "1"
```

```
[39]: from sklearn.cluster import KMeans
model=KMeans(n_clusters=5)
model.fit(feature)
KMeans(n_clusters=5)
```

D:\Ashvanthan\anaconda3\python\Lib\site-packages\sklearn\cluster_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.

```
warnings.warn(
```

```
[39]: KMeans(n_clusters=5)
```

```
[40]: Final=df.iloc[:,[3,4]]
Final['label']=model.predict(feature)
Final
```

C:\Users\Lenovo\AppData\Local\Temp\ipykernel_9408\551092936.py:2:

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

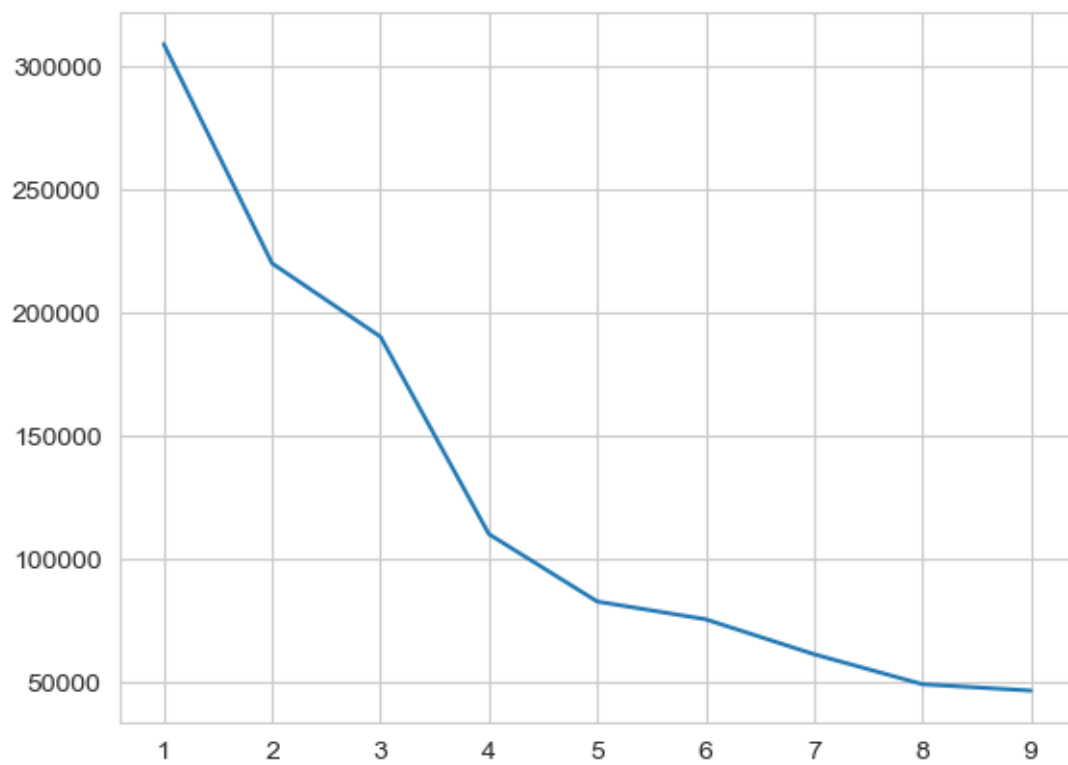
```
Final['label']=model.predict(feature)
```

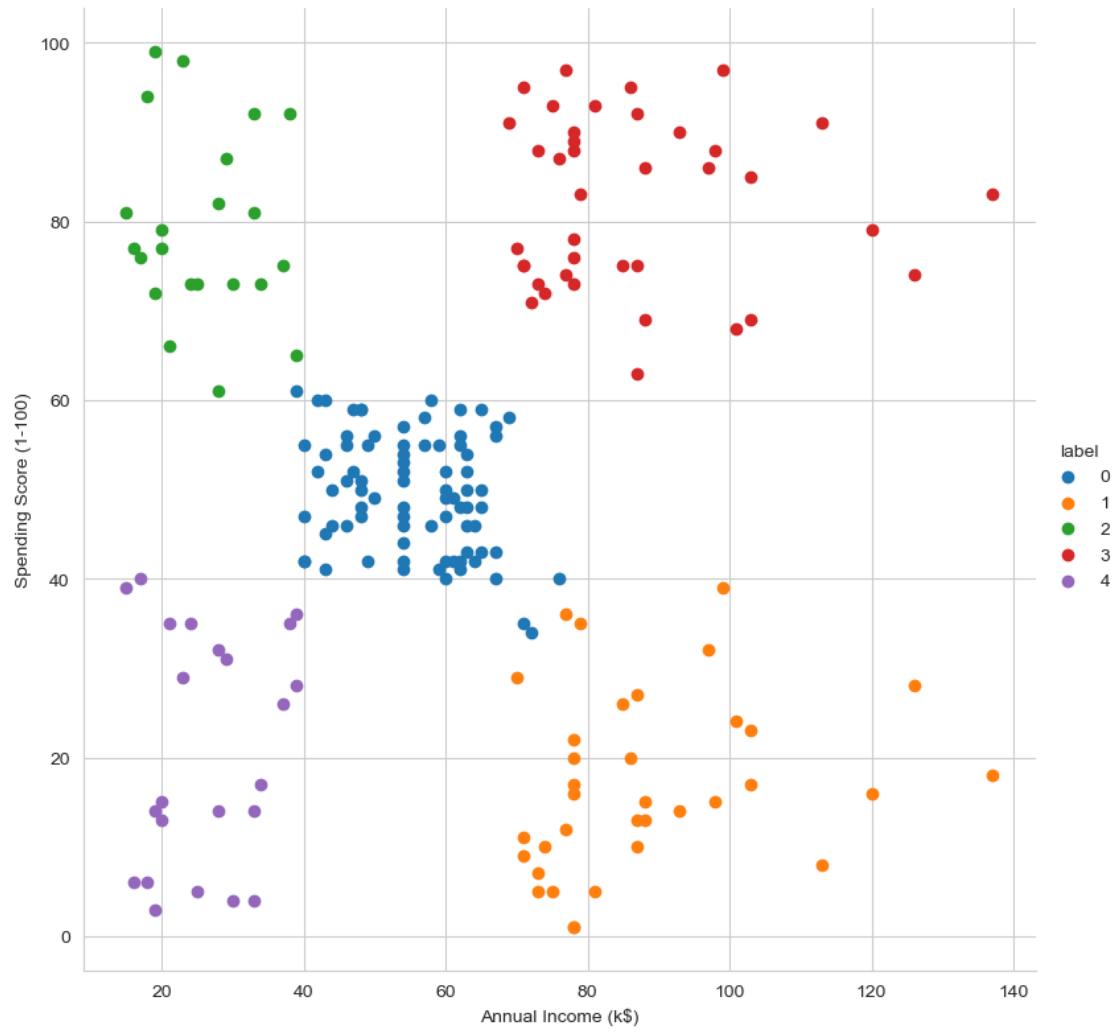
```
[40]:
```

	Annual Income (k\$)	Spending Score (1-100)	label
0	15	39	4
1	15	81	2
2	16	6	4
3	16	77	2
4	17	40	4
..
195	120	79	3
196	126	28	1
197	126	74	3
198	137	18	1
199	137	83	3

[200 rows x 3 columns]

```
[41]: sns.set_style("whitegrid")
sns.FacetGrid(Final,hue="label",height=8).map(plt.scatter,"Annual Income (k$)",
↪ "Spending Score (1-100)").add_legend();
plt.show()
```



Exercie-11

November 2, 2025

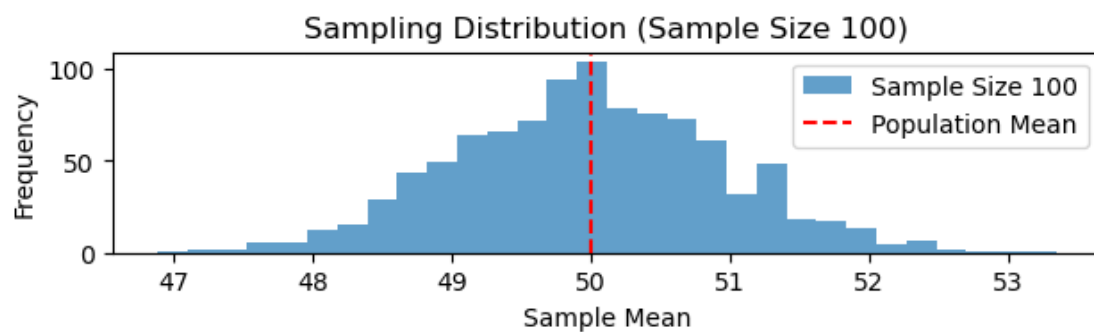
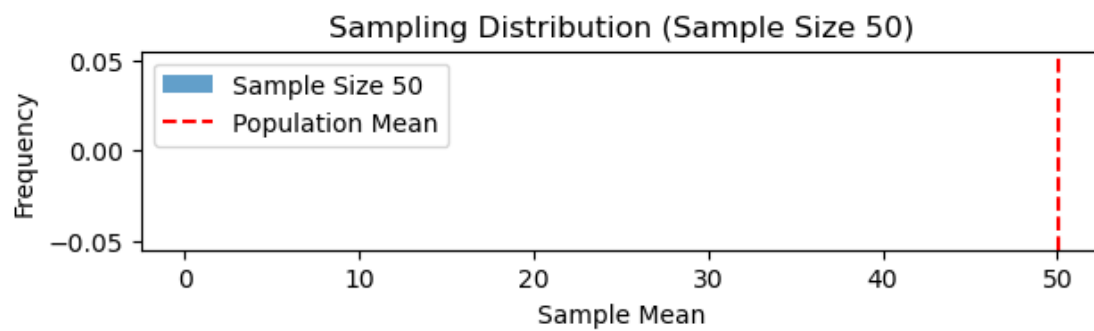
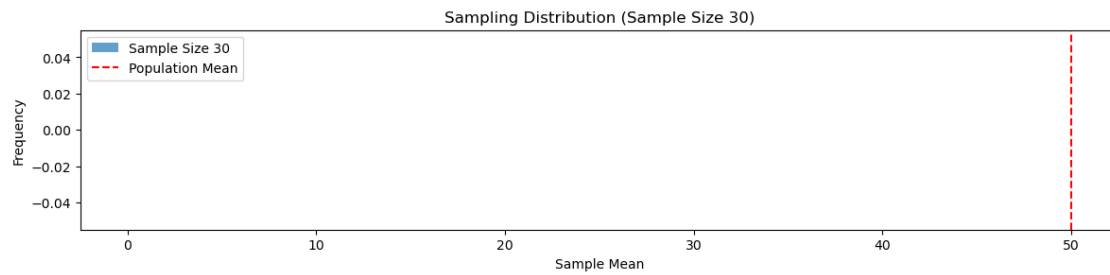
```
[8]: import numpy as np
import matplotlib.pyplot as plt
```

```
[9]: population_mean = 50
population_std = 10
population_size = 100000
population = np.random.normal(population_mean, population_std, population_size)
```

```
[10]: sample_sizes = [30, 50, 100]
num_samples = 1000
```

```
[11]: sample_means = {}
for size in sample_sizes:
    sample_means[size] = []
for _ in range(num_samples):
    sample = np.random.choice(population, size=size, replace=False)
    sample_means[size].append(np.mean(sample))
```

```
[12]: plt.figure(figsize=(12, 8))
for i, size in enumerate(sample_sizes):
    plt.subplot(len(sample_sizes), 1, i+1)
    plt.hist(sample_means[size], bins=30, alpha=0.7, label=f'Sample Size_{size}')
    plt.axvline(np.mean(population), color='red', linestyle='dashed', linewidth=1.5, label='Population Mean')
    plt.title(f'Sampling Distribution (Sample Size {size})')
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.legend()
    plt.tight_layout()
    plt.show()
```



[]:

Experiment-12

November 2, 2025

```
[1]: import numpy as np
import scipy.stats as stats
```

```
[2]: sample_data = np.array([152, 148, 151, 149, 147, 153, 150, 148, 152,
149,151, 150, 149, 152, 151, 148, 150, 152, 149, 150,148, 153, 151,
150, 149, 152, 148, 151, 150, 153])
```

```
[3]: population_mean = 150
```

```
[4]: sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
```

```
[5]: n = len(sample_data)
```

```
[6]: z_statistic = (sample_mean - population_mean) / (sample_std /np.sqrt(n))
```

```
[7]: p_value = 2 * (1 - stats.norm.cdf(np.abs(z_statistic)))
```

```
[8]: print(f"Sample Mean: {sample_mean:.2f}")
print(f"Z-Statistic: {z_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
```

Sample Mean: 150.20

Z-Statistic: 0.6406

P-Value: 0.5218

```
[9]: alpha = 0.05
if p_value<alpha:
    print("Reject the null hypothesis: The average weight is significantly_
    ↪different from 150 grams")
else:
    print("Fail to reject the null hypothesis: There is nosignificant_
    ↪difference in average weight from 150 grams")
```

Fail to reject the null hypothesis: There is nosignificant difference in average weight from 150 grams

```
[ ]:
```

Exercise-13

November 2, 2025

```
[1]: import numpy as np
import scipy.stats as stats
```

```
[2]: np.random.seed(42)
```

```
[3]: sample_size = 25
sample_data = np.random.normal(loc=102, scale=15, size=sample_size)
```

```
[4]: population_mean = 100
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
```

```
[5]: n = len(sample_data)
```

```
[6]: t_statistic, p_value = stats.ttest_1samp(sample_data, population_mean)
print(f"Sample Mean: {sample_mean:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
```

Sample Mean: 99.55

T-Statistic: -0.1577

P-Value: 0.8760

```
[8]: alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average IQ score is significantly_
↪different from 100")
else:
    print("Fail to reject the null hypothesis: There is no significant_
↪difference in average IQ score from 100")
```

Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100

```
[ ]:
```

Exercise-14

November 2, 2025

```
[1]: import numpy as np
import scipy.stats as stats
```

```
[2]: np.random.seed(42)
n_plants = 25
growth_A = np.random.normal(loc=10, scale=2, size=n_plants)
growth_B = np.random.normal(loc=12, scale=3, size=n_plants)
growth_C = np.random.normal(loc=15, scale=2.5, size=n_plants)
```

```
[3]: all_data = np.concatenate([growth_A, growth_B, growth_C])
treatment_labels = ['A'] * n_plants + ['B'] * n_plants + ['C'] * n_plants
```

```
[4]: f_statistic, p_value = stats.f_oneway(growth_A, growth_B, growth_C)
```

```
[5]: print("Treatment A Mean Growth:", np.mean(growth_A))
print("Treatment B Mean Growth:", np.mean(growth_B))
print("Treatment C Mean Growth:", np.mean(growth_C))
print()
print(f"F-Statistic: {f_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
```

Treatment A Mean Growth: 9.672983882683818

Treatment B Mean Growth: 11.137680744437432

Treatment C Mean Growth: 15.265234904828972

F-Statistic: 36.1214

P-Value: 0.0000

```
[8]: alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference in_
    ↪ mean growth rates among the three treatments")
else:
    print("Fail to reject the null hypothesis: There is no significant_
    ↪ difference in mean growth rates among the three treatments")
if p_value < alpha:
    from statsmodels.stats.multicomp import pairwise_tukeyhsd
    tukey_results = pairwise_tukeyhsd(all_data, treatment_labels, alpha=0.05)
```

```
print("\nTukey's HSD Post-hoc Test:")
print(tukey_results)
```

Reject the null hypothesis: There is a significant difference in mean growth rates among the three treatments

Tukey's HSD Post-hoc Test:

Multiple Comparison of Means - Tukey HSD, FWER=0.05

```
=====
group1 group2 meandiff p-adj  lower  upper  reject
-----
      A      B   1.4647 0.0877 -0.1683 3.0977  False
      A      C   5.5923   0.0  3.9593 7.2252   True
      B      C   4.1276   0.0  2.4946 5.7605   True
-----
```

```
[ ]:
```