```
import pandas as pd
df = pd.read_csv("AUTOMPG.csv")
print(df)

print("\nCylinders greater than 8:")
cyl_list = df['cylinders'].tolist()
cyl_above_8 = list(filter(lambda x: x > 8, cyl_list))
print(cyl_above_8)

print("d. Get the number of car manufacturing each year")
print(df.groupby(['model_year']).count()[['mpg']])

print("Calculate the mean value of horsepower")
df['horsepower'] = pd.to_numeric(df['horsepower'],  errors='coerce')
print(df['horsepower'].mean())

print("Calculate the Standard Deviation value of acceleration",df['acceleration'].std())

print("Retrieve details of all the cars built in year 72")
print(df.loc[df['model_year'] == 72 ].head())

print("Data sorted according to the number of cylinders")
print(df.sort_values(by = 'cylinders'))

print(" XYZ custom cars wants to know about minimum and maximum of "
"all the numerical columns")
num_cols = df.select_dtypes(include=['number']).columns
print(df[num_cols].agg(['min', 'max']))
```

```python
import pandas as pd

A = pd.Series([10,20,30,40,50])
B = pd.Series([40,50,60,70,80])
not_common = pd.concat([A, B]).drop_duplicates(keep=False)
common = pd.Series(list(set(A) & set(B)))

smallest_A = A.min()
largest_A = A.max()

smallest_B = B.min()
largest_B = B.max()

sum_B = B.sum()
average_A = A.mean()
median_B = B.median()

print("Not common:", not_common)
print("Common:", common)
print("Smallest A:", smallest_A)
print("Largest A:", largest_A)
print("Smallest B:", smallest_B)
print("Largest B:", largest_B)
print("Sum B:", sum_B)
print("Average A:", average_A)
print("Median B:", median_B)
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv('Iris.csv')
print(df.head())

print("statistical summary:/n",df.describe())
print("size of the dataset:/n",df.shape)

sns.scatterplot(x='PetalLengthCm',y='PetalWidthCm',data=df)
print("missing values:/n",df.isnull().sum())
print("replace missing values with mean:/n",df.isnull().mean())
subset = df.loc[0:5, ["Id", "Species"]]
print(subset)

sns.histplot(df['SepalLengthCm'])
plt.show()
sns.scatterplot(data=df,x='SepalLengthCm',y='SepalWidthCm')
plt.show()
sns.heatmap(df[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm',]].corr(),annot=True)
plt.show()
Q1, Q3 = df['PetalWidthCm'].quantile([0.25, 0.75])
IQR = Q3 - Q1
lower, upper = Q1 - 1.5*IQR, Q3 + 1.5*IQR

outliers = df[(df['PetalWidthCm'] < lower) | (df['PetalWidthCm'] > upper)]
print("\nOutliers detected:", len(outliers))
data_no_outliers = df[(df['PetalWidthCm'] >= lower) & (df['PetalWidthCm'] <= upper)]
print("Shape before:", df.shape, " &  After:", data_no_outliers.shape)
sns.boxplot(data_no_outliers['PetalWidthCm'])
plt.show()
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.DataFrame({'number':[1,2,3,4,5],
                 'pencil':[300,350,400,500,520],
                 'textbooks':[250,350,400,420,500],
                 'drawing_sheets':[100,125,190,210,250],
                 'total_units':[700,1075,1320,1510,np.nan],
                 'profit':[80000,9500,10256,12000,15000]})
print(df)

print("statistical summary:/n",df.describe())

sns.lineplot(data=df,x='number',y='profit')
plt.show()

sns.barplot(data=df,x='number',y='pencil')
plt.show()

sns.scatterplot(data=df,x='pencil',y='textbooks')
plt.show()

print("missing values:/n",df.isnull().sum())
print("replace missing values:/n",df.isnull().mean())
print("sum of profit:/n",df['profit'].sum())
print("max of drawing sheets:/n",df['drawing_sheets'].max())

print("replace with median:/n",df.isnull().median())
print("replace with mode:/n",df.isnull().mode())
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv('AUTOMPG.csv')
print(df.head())

print("information of dataset:/n",df.info())
print("missing values:/n",df.isnull().sum())
print("replaced with mean all columns:/n",df.fillna(df.mean(numeric_only=True)))
print("verifying values:/n",df.isnull().sum())
sns.histplot(df['horsepower'])
plt.show()
sns.scatterplot(data=df,x='cylinders',y='acceleration')
plt.show()
sns.heatmap(df[['cylinders','horsepower','acceleration']].corr(),annot=True)
plt.show()
Q1, Q3 = df['horsepower'].quantile([0.25, 0.75])
IQR = Q3 - Q1
lower, upper = Q1 - 1.5*IQR, Q3 + 1.5*IQR

outliers = df[(df['horsepower'] < lower) | (df['horsepower'] > upper)]
print("\nOutliers detected:", len(outliers))
data_no_outliers = df[(df['horsepower'] >= lower) & (df['horsepower'] <= upper)]
print("Shape before:", df.shape, " &  After:", data_no_outliers.shape)
sns.boxplot(data_no_outliers['horsepower'])
plt.show()
```

(variable) df: DataFrame

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

df = pd.DataFrame({'salary': [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,11000],
    'experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]})

df['experience'] = df['experience'].fillna(df['experience'].mean())

X = df[['salary']]
y = df['experience']

X_train, X_test, y_train, y_test = train_test_split(X, y,
    train_size=0.8, test_size=0.2, random_state=42)
print('X_train:\n', X_train)
print('y_train:\n', y_train)

model = LinearRegression()
model.fit(X_train, y_train)
print('Slope (Coefficient):', model.coef_.item())
print('Intercept:', model.intercept_.item())


y_pred = model.predict(X_test)
accuracy = model.score(X_test, y_test)
print("\nX_test:\n", X_test)
print("y_test:\n", y_test.values)
print("y_pred:\n", y_pred)
print(f'Accuracy (R² score): {accuracy:.2f}')

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'MSE: {mse:.2f}, R²: {r2:.2f}')

plt.scatter(df['salary'], df['experience'], color='blue', label='Actual Data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel('salary')
plt.ylabel('experience')
plt.title('salary vs experience (Simple Linear Regression)')
plt.legend()
plt.show()
```

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

data = pd.read_csv('tips.csv')
sns.histplot(data['total_bill'])
plt.show()

sns.scatterplot(data=data, x='day', y='tip',
                hue='tip')
plt.show()

sns.barplot(data=data, x='day', y='tip',hue='day')
plt.show()

sns.lineplot(data=data,x='size',y='tip')
plt.show()

print(" Create a list containing tip values greater than 5 using filter function")
tips = data['tip'].tolist()
tips_above_5 = list(filter(lambda x: x > 5, tips))
print(tips_above_5)

print("Find out the sum of the total_bill of each day")
total_bill_per_day = data.groupby('day')['total_bill'].sum()
print(total_bill_per_day)

print("iii. Find the sum of the tip using reduce function")
from functools import reduce

tip_sum = reduce(lambda x, y: x + y, data['tip'])
print(tip_sum)
```

```python
import numpy as np
import pandas as pd
Tracker = pd.DataFrame({
    'Day' : [1,2,3,4,5,6,7,8,9,10],
    'steps' : [7079,8576,6586,4508,5564,6971,7733,8520,9753,9569]})

print(" 10*2 array (Day,steps):\n",Tracker)

Tracker['updated_steps'] =Tracker['steps'] + 1000
print("\n  updated array (after adding 2000 steps):\n",Tracker[['Day','updated_steps']])

above_7000 = Tracker[Tracker['updated_steps'] > 7000]
print("\n steps walked more than 9000:\n",above_7000[['Day','updated_steps']])

sorted_steps = np.sort(Tracker['updated_steps'])
print(":\n  steps walked in sorted order:\n",sorted_steps)
```

```
import pandas as pd
marks_A = {'chemistry': [67,90,66,32], 'physics' : [45,92,72,40], 'english': [67,90,66,32], 'maths': [67,90,66,32]}

marks_A_df = pd.DataFrame(marks_A,index=['subodh','ram','abdul','john'])

marks_B = {'chemistry': [72,45,60,98], 'physics': [78,34,72,95], 'english': [67,90,66,32], 'maths': [67,90,66,32]}
marks_B_df = pd.DataFrame(marks_B,index=['nandini','zoya','shivam','james'])

print(marks_A_df)
print(marks_B_df)

print("the teeacher wants to combine the marks of these students")
marks_df=pd.concat([marks_A_df,marks_B_df],sort=False)
print(marks_df)

print("stident has scored marks >= 33, then have passed;otherwise,they have failed")
f = marks_df < 33
result=marks_df.mask(f,'fail')
print(result)

print("5 bonus marks awarded to each student")
new_marks=marks_df+5
print(new_marks)

print("the teacher wants to increse the marks of all the students as follows")
print("chemistry: +5 physics: +10 maths: +10 english: +2")
custom_marks =marks_df+[5,10,10,2]
print(custom_marks)

import numpy as np
print("the teacher wants to get the total marks scored in each subject ")
print(marks_df.apply(np.sum,axis=0))
```

```python
import numpy as np
import pandas as pd
from scipy import stats

data = [5, 7, 7, 8, 9, 10, 10, 10, 12, 15]

series = pd.Series(data)

mean_val = series.mean()

median_val = series.median()

mode_val = series.mode().iloc[0]

print("Data:", data)
print("Mean:", mean_val)
print("Median:", median_val)
print("Mode:", mode_val)
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

iris = load_iris()
X = iris.data
y = iris.target
print(f"Original dataset shape: {X.shape}")

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print(f"Shape after PCA: {X_pca.shape}")
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y,  cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.colorbar()
plt.show()
```

```python
import numpy as np
import pandas as pd
from sklearn.metrics import (accuracy_score, classification_report,confusion_matrix)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC

df = pd.read_csv("Iris.csv")
print(df.head())

print("Missing values before handling:\n", df.isnull().sum())

le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
r1 = pd.get_dummies(df, columns=['Species'],drop_first=False)
print(r1)

X = df.drop("Species", axis=1)
y = df["Species"]
print("X shape:", X.shape, " | y shape:", y.shape)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split( X_scaled, y, test_size=0.2, random_state=42, stratify=y)

svm_model = SVC(kernel="rbf", C=1, gamma="scale")  # You can try 'linear' or 'poly' too
svm_model.fit(X_train, y_train)

# Predictions
y_pred = svm_model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=le.classes_))
```

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.metrics import (accuracy_score, classification_report, confusion_matrix)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
df=pd.read_csv("Iris.csv")
print(df.head())

print("Missing values before handling:\n", df.isnull().sum())

le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
r1 = pd.get_dummies(df, columns=['Species'],drop_first=False)
print(r1)

X=df.drop("Species",axis=1)
y=df["Species"]

print("x shape;",X.shape,"|y shape",y.shape)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split( X_scaled, y, test_size=0.2, random_state=42, stratify=y)

log_reg = LogisticRegression(max_iter=500, solver="lbfgs")
log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,target_names=le.classes_))
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('Mall_Customers.csv')
print("print top 5 record of dataset\n",df.head())

x = df.iloc[:, [3, 4]].values
print(x.shape)

from sklearn.cluster import KMeans

wcss_list = []
print(wcss_list)

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
y_pred = kmeans.fit_predict(x)
print(y_pred)

plt.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s=100, c='blue', label='Cluster 1')
plt.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s=100, c='green', label='Cluster 2')
plt.scatter(x[y_pred == 2, 0], x[y_pred == 2, 1], s=100, c='red', label='Cluster 3')
plt.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s=100, c='cyan', label='Cluster 4')
plt.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s=100, c='magenta', label='Cluster 5')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='yellow', label='Centroids')
plt.show()
```

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler

data = {"Name": ["John", "John", "Camela"],"Surname": ["Wick", "Constantine", "Anderson"],
"Age": [35, 27, 30],"Smokes": [3, 20, 0],"AreaQ": [5, 2, 5],"Alkhol": [4.0, 5.0, 2.0],
"Result": [1, 1, 0], "Gender": ["F", "M", "M"],"Location": ["United states", "florida", "canada"]}
df = pd.DataFrame(data)

print("\n-ataset Info",df.info())
print("\n-Dataset Description",df.describe())

print("\n--- Missing Values",df.isnull().sum())

sns.histplot(df["Age"], kde=True)
plt.show()
sns.countplot(x="Result", data=df)
plt.show()

sns.scatterplot(x="Result", y="Age", data=df)
plt.show()
sns.pairplot(df[["Age","Smokes","AreaQ","Alkhol","Result"]],
hue="Result")
plt.show()
sns.heatmap(df[["Age","Smokes","AreaQ","Alkhol","Result"]].corr(),
annot=True, cmap="coolwarm")
plt.show()

Q1 = df['Smokes'].quantile(0.25)
Q3 = df['Smokes'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
df_clean = df[(df['Smokes'] >= lower) & (df['Smokes'] <= upper)]

numeric_cols = ["Age", "Smokes", "AreaQ", "Alkhol"]

le = LabelEncoder()
df["Gender"] = le.fit_transform(df["Gender"]) # F=0, M=1
df = pd.get_dummies(df, columns=["Location"], drop_first=False)

print("\n after Dummies\n",df)
print("\n--- Encoded Dataset ---",df.head())

scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
print("\n--- Normalized Dataset ---",df.head())

sns.countplot(x="Gender", hue="Gender", data=df)
plt.title("Cancer Result by Gender")
plt.show()
sns.countplot(x="Result", hue="Location_canada", data=df)
plt.title("Cancer Result by Location (Canada vs Others)")
plt.show()
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset("iris")

print(df.head())

label = LabelEncoder()
df["Species"] = label.fit_transform(df["species"])  # Convert species names to numbers
df.drop("species", axis=1, inplace=True)

X = df.drop("Species", axis=1)
y = df["Species"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

model = DecisionTreeClassifier(criterion="entropy", random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))

print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt="d",
            xticklabels=label.classes_, yticklabels=label.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

from sklearn import tree
plt.figure(figsize=(14,8))
tree.plot_tree(model, feature_names=X.columns, class_names=label.classes_, filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```

```
import nltk
from nltk.tokenize import sent_tokenize,word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

text = "Machine Learning is best platform. To group over selves."
print("sent_tokenize:\n", sent_tokenize(text))
print("word tokenization \n", word_tokenize(text))

from nltk.corpus import stopwords
words = ["Machine", "Learning", "is", "best", "platform", ".", "To", "group", "over", "selves"]
stop_words = set(stopwords.words('english'))
filtered_words = [w for w in words if w.lower() not in stop_words]
print("filtered_words:", filtered_words)

from nltk.stem import PorterStemmer,WordNetLemmatizer
ps = PorterStemmer()
stemmed_words = [ps.stem(w) for w in filtered_words]
print("stemmed_words:", stemmed_words)

lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(w) for w in filtered_words]
print("lemmatized_words:", lemmatized_words)
```

```
import datetime
import 
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

mnist = tf.keras.datasets.mnist
(x_train , y_train), (x_test, y_test) = mnist.load_data()
x_train,x_test=x_train / 255.0, x_test/255.0

print("third digit in y_train is:",y_train[20])
plt.imshow(x_train[20],cmap='coolwarm')
plt.show()

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
log_dir="logs/"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir=log_dir)
checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(filepath="model checkpoint.h5",save_best_only=True)
history = model.fit(x_train,y_train,epochs=6,validation_data=(x_test,y_test),callbacks=[tensorboard_callback,checkpoint_callback])
loss,acc=model.evaluate(x_test,y_test,verbose=1)
print("original model,accuracy:{:5.2f}%.format(100*acc)")
```

```
import spacy
from nltk import ngrams
nlp = spacy.load("en_core_web_sm")
sentence4 = """The students are studying hard to pass the exams"""
doc = nlp(sentence4)
unigrams = ngrams (doc, 1)
print("\n\n unigrams\n", list(unigrams))
bigrams = ngrams (doc, 2)
print("\n\n bigrams \n", list(bigrams))
Trigrams = ngrams (doc, 3)
print("\n\n trigrams \n", list(Trigrams))
```