Regression :

```
import numpy as np
from sklearn.linear_model import LinearRegression
# Features: [size (m²), number of rooms]
X = np.array([
    [50, 2],
    [70, 3],
    [90, 3],
    [120, 4],
    [150, 5]
])
# Target: price (in thousands)
y = np.array([100, 150, 180, 250, 300])
# Create model
model = LinearRegression()
# Train model
model.fit(X, y)
# Predict price for a new house
# Example: size = 100 m², rooms = 3
new_house = np.array([[100, 3]])
predicted_price = model.predict(new_house)
print(f"Predicted price: {predicted_price[0]:.2f}")

#price=w1·size+w2·rooms+b     (b shifts (up/down), w tilts (rotate))
print("Weights:", model.coef_)
print("Bias:", model.intercept_)
```

#Visualize Actual vs Predicted Price

```
# Plot
import matplotlib.pyplot as plt
y_pred = model.predict(X)
plt.figure()
plt.scatter(y, y_pred)  # Actual vs Predicted
# Line y = x (perfect prediction)
plt.plot([y.min(), y.max()], [y.min(), y.max()])
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted House Prices")
plt.show()
```

#3D visualization Features vs Price

```
# Create grid for surface
size_range = np.linspace(50, 150, 10)
rooms_range = np.linspace(2, 5, 10)
size_grid, rooms_grid = np.meshgrid(size_range, rooms_range)
# Predict over grid
grid_points = np.c_[size_grid.ravel(), rooms_grid.ravel()]
price_grid = model.predict(grid_points).reshape(size_grid.shape)
# Plot
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
# Scatter (real data)
ax.scatter(X[:, 0], X[:, 1], y)
# Surface (model)
ax.plot_surface(size_grid, rooms_grid, price_grid)
ax.set_xlabel("Size (m²)")
ax.set_ylabel("Rooms")
ax.set_zlabel("Price")
ax.set_title("Linear Regression Plane")
plt.show()
```

#Visualize 2D

```
# Fix rooms = 3
sizes = np.linspace(50, 150, 50)
rooms_fixed = 3

X_line = np.c_[sizes, np.full_like(sizes, rooms_fixed)]
y_line = model.predict(X_line)

# Plot
plt.figure()
plt.scatter(X[:, 0], y) # real data (size vs price)
plt.plot(sizes, y_line) # model line

plt.xlabel("Size (m²)")
plt.ylabel("Price")
plt.title("Price vs Size (rooms = 3)")
plt.show()
```

#MSE calculation

```
from sklearn.metrics import mean_squared_error

y_pred = model.predict(X)
mse = mean_squared_error(y, y_pred)

print("MSE:", mse)
```

Binary Classification: (k-NN) – Red/Blue Color classification from RGB

Class	R	G	B
RED	179	26	52
	192	12	80
	..	..	..
BLUE	21	13	191
	57	49	155
	…	…	..

```
# Import all required libraries
import json
import pandas as pd
import joblib
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Create color dictionary and save it as json file (fill the list with more examples)

color = [{"R":179, "G":26, "B":52, "class": "red"},

    {"R":192, "G":12, "B":80, "class": "red"},

    {"R":21, "G":13, "B":191, "class": "blue"},

    {"R":57, "G":49, "B":155, "class": "blue"},]

with open("dataset.json", "w") as data:

    json.dump(color, data, indent = 4)

# Load the dataset from the JSON file and put it into dataframe

with open("dataset.json", "r") as data:

    content = json.load(data)

df = pd.DataFrame(content)

# Separate the RGB features from the class labels

X = df.drop(columns = ["class"])

y = df["class"]

# Partition the dataset into training and testing subsets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Build and train the K-NN classifier

knn = KNeighborsClassifier(n_neighbors = 3)

knn.fit(X_train, y_train)
```

```
# Evaluate model accuracy on the test set

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_pred, y_test)

print(accuracy)

# Save the trained model to disk

model = joblib.dump(knn, "my_model.pkl")

# Reload the saved model and use it to predict the class of a new color

my_model = joblib.load("my_model.pkl")

sample = [[153, 41,25]]

print(sample, my_model.predict(sample) )
```

#Binary Cross-Entropy (BCE) calculation

```
# Convert labels to binary (red=1, blue=0)
y_test_bin = y_test.map({"red": 1, "blue": 0}).values

# Get probabilities for class "red"
probs = knn.predict_proba(X_test)[:, list(knn.classes_).index("red")]

# Avoid log(0)
epsilon = 1e-15
probs = np.clip(probs, epsilon, 1 - epsilon)

# Compute BCE
bce = -np.mean( y_test_bin * np.log(probs) + (1 - y_test_bin) * np.log(1 - probs))

print("Binary Cross-Entropy:", bce)
```