Experiment 0: Extract the data from the database using Python. Perform preprocessing of data. Calculate covariance and correlation of attributes.

python Collapse Wrap Run Copy

```
import sqlite3
import pandas as pd
import numpy as np

# Create in-memory database
conn = sqlite3.connect(':memory:')

# Create table and insert data
data = {
    'A': [1, 2, 3, 4],
    'B': [5, 6, 7, 8],
    'C': [9, 10, 11, 12]
}
df = pd.DataFrame(data)
df.to_sql('table1', conn, index=False)

# Extract
query = "SELECT * FROM table1"
extracted = pd.read_sql(query, conn)
print("Extracted data:\n", extracted)

# Preprocess: normalize
normalized = (extracted - extracted.min()) / (extracted.max() - extracted.min
print("Normalized:\n", normalized)

# Covariance
cov = normalized.cov()
print("Covariance:\n", cov)

# Correlation
corr = normalized.corr()
print("Correlation:\n", corr)
```

Experiment 1: (i) Implement Linear Regression. (ii) Implement

Multivariate Linear Regression.

python Collapse Wrap Run Copy

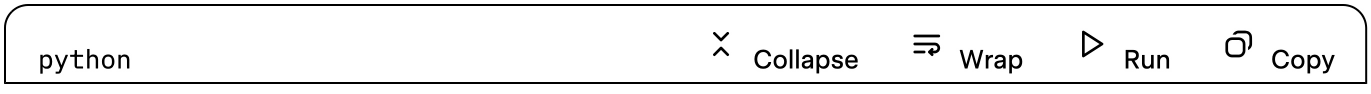
```
import numpy as np

# Sample data for linear regression
x = np.array([1,2,3,4,5])
y = np.array([2,4,6,8,10])

# Linear regression: y = mx + c
X = np.vstack([x, np.ones(len(x))]).T
m, c = np.linalg.lstsq(X, y, rcond=None)[0]
print(f"Slope: {m}, Intercept: {c}")

# Predictions
pred = m * x + c
print("Predictions:", pred)

# Multivariate
x1 = np.array([1,2,3,4,5])
x2 = np.array([5,4,3,2,1])
X_multi = np.vstack([x1, x2, np.ones(5)]).T
betas = np.linalg.lstsq(X_multi, y, rcond=None)[0]
print("Betas:", betas)

# Predictions
pred_multi = np.dot(X_multi, betas)
print("Multi Predictions:", pred_multi)
```

Experiment 2: Implement Logistic Regression.

python Collapse Wrap Run Copy

```
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Sample data
X = np.array([[0.5, 1.5], [1,1], [1.5,0.5], [3, 0.5], [2, 2], [1,2.5]])
y = np.array([0,0,0,1,1,1])

# Add intercept
X = np.c_[np.ones(X.shape[0]), X]

# Init weights
w = np.zeros(X.shape[1])
lr = 0.1
for _ in range(1000):
    preds = sigmoid(np.dot(X, w))
    grad = np.dot(X.T, (preds - y)) / len(y)
    w -= lr * grad

print("Weights:", w)

# Probs
probs = sigmoid(np.dot(X, w))
print("Probabilities:", probs)
classes = (probs > 0.5).astype(int)
print("Classes:", classes)
```

Experiment 3: Implement Ensemble Learning (Bagging/Boosting).


python
Collapse
Wrap
Run
Copy

```
import numpy as np
import statsmodels.api as sm

# Sample data for ensemble (bagging with linear regression)
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2,4,5,8,10])  # slightly noisy

X = sm.add_constant(X)

# Bagging: 3 models
models = []
np.random.seed(42)
for i in range(3):
    idx = np.random.choice(len(y), len(y), replace=True)
    X_b = X[idx]
    y_b = y[idx]
    model = sm.OLS(y_b, X_b).fit()
    models.append(model)

# Predict on new x=6
x_new = np.array([[1, 6]])  # manually add constant
preds = [m.predict(x_new)[0] for m in models]
avg_pred = np.mean(preds)
print("Bagging prediction for x=6:", avg_pred)

# Simple boosting (AdaBoost like for regression, manual)
# Base model1
model1 = sm.OLS(y, X).fit()
preds1 = model1.predict(X)
errors1 = np.abs(y - preds1)
weights = errors1 / errors1.sum()

# Model2 on weighted
model2 = sm.WLS(y, X, weights=1/(errors1+1e-6)).fit()  # approximate
preds2 = model2.predict(X)

# Combine, simple average for demo
combined = (preds1 + preds2) / 2
print("Boosting combined preds:", combined)
```

Experiment 4: Implement CART Algorithm.

python Collapse Wrap Run Copy

```python
import numpy as np

class DecisionNode:
    def __init__(self, feature=None, threshold=None, left=None, right=None, v
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

def gini(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    return 1 - np.sum(probabilities**2)

def best_split(X, y):
    best_gini = np.inf
    best_feature, best_threshold = None, None
    for feature in range(X.shape[1]):
        thresholds = np.unique(X[:, feature])
        for threshold in thresholds:
            left = y[X[:, feature] <= threshold]
            right = y[X[:, feature] > threshold]
            if len(left) == 0 or len(right) == 0:
                continue
            weighted_gini = (len(left) * gini(left) + len(right) * gini(right
            if weighted_gini < best_gini:
                best_gini = weighted_gini
                best_feature = feature
                best_threshold = threshold
    return best_feature, best_threshold

def build_tree(X, y, depth=0, max_depth=3):
    if len(np.unique(y)) == 1 or depth == max_depth:
        return DecisionNode(value=np.argmax(np.bincount(y)))
    feature, threshold = best_split(X, y)
    if feature is None:
        return DecisionNode(value=np.argmax(np.bincount(y)))
    left_idx = X[:, feature] <= threshold
```

```
    right_idx = X[:, feature] > threshold
    left = build_tree(X[left_idx], y[left_idx], depth+1, max_depth)
    right = build_tree(X[right_idx], y[right_idx], depth+1, max_depth)
    return DecisionNode(feature, threshold, left, right)

def predict(node, x):
    if node.value is not None:
        return node.value
    if x[node.feature] <= node.threshold:
        return predict(node.left, x)
    else:
        return predict(node.right, x)

# Sample data (iris like, binary for simple)
X = np.array([[5.1, 3.5], [4.9, 3.0], [7.0, 3.2], [6.4, 3.2]])
y = np.array([0, 0, 1, 1])

tree = build_tree(X, y)
preds = [predict(tree, x) for x in X]
print("Predictions:", preds)
```

Experiment 5: Implement Support Vector Machine (SVM).

```python
import numpy as np
from scipy.optimize import minimize

# Simple linear SVM using minimize for hard margin
def svm_train(X, y):
    n_samples, n_features = X.shape
    def objective(alpha):
        return 0.5 * np.dot(alpha, np.dot(np.outer(y, y) * np.dot(X, X.T), al
    constraints = ({'type': 'eq', 'fun': lambda alpha: np.dot(alpha, y)})
    bounds = [(0, None) for _ in range(n_samples)]
    res = minimize(objective, np.zeros(n_samples), bounds=bounds, constraints
    alpha = res.x
    w = np.dot(alpha * y, X)
    b = np.mean(y - np.dot(X, w))
    return w, b

def svm_predict(X, w, b):
    return np.sign(np.dot(X, w) + b)

# Data
X = np.array([ [1,1], [2,2], [3,3], [6,6], [7,7], [8,8] ])
y = np.array([-1,-1,-1,1,1,1])

w, b = svm_train(X, y)
preds = svm_predict(X, w, b)
print("Weights:", w, "Bias:", b)
print("Predictions:", preds)
```

Experiment 6: Implement Graph-Based Clustering.

```python
import numpy as np
import networkx as nx
from scipy.cluster.vq import kmeans2

# Simple graph-based clustering using spectral clustering approximation
def spectral_clustering(X, k=2):
    # Affinity matrix (Gaussian similarity)
    sigma = 1.0
    affinity = np.exp(-np.linalg.norm(X[:, np.newaxis] - X, axis=2)**2 / (2 *
    # Laplacian
    degree = np.diag(affinity.sum(axis=1))
    laplacian = degree - affinity
    # Eigen
    eigenvalues, eigenvectors = np.linalg.eig(laplacian)
    idx = np.argsort(eigenvalues)[:k]
    U = eigenvectors[:, idx]
    # Kmeans on U
    _, labels = kmeans2(U, k, minit='points')
    return labels

# Sample data
X = np.array([[1,1], [1.5,1.5], [5,5], [5.5,5.5]])

labels = spectral_clustering(X, 2)
print("Labels:", labels)
```

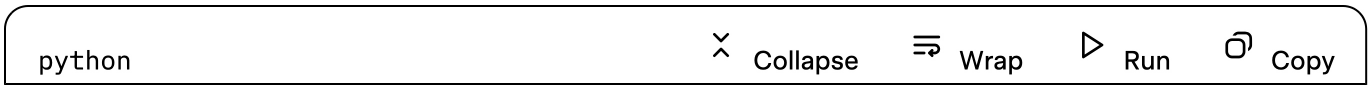
Experiment 7: Implement DBSCAN Algorithm.

python Collapse Wrap Run Copy

```python
import numpy as np

def region_query(X, idx, eps):
    dists = np.linalg.norm(X - X[idx], axis=1)
    return np.where(dists <= eps)[0]

def dbscan(X, eps, min_pts):
    n = X.shape[0]
    labels = np.full(n, -1)  # -1 noise
    cluster_id = 0
```

```
    for i in range(n):
        if labels[i] != -1:
            continue
        neighbors = region_query(X, i, eps)
        if len(neighbors) < min_pts:
            labels[i] = -1  # noise
        else:
            labels[i] = cluster_id
            expand_cluster(i, neighbors, cluster_id, eps, min_pts, X, labels)
            cluster_id += 1

    return labels

def expand_cluster(i, neighbors, cluster_id, eps, min_pts, X, labels):
    k = 0
    while k < len(neighbors):
        j = neighbors[k]
        if labels[j] == -1:
            labels[j] = cluster_id
        if labels[j] != cluster_id:
            k += 1
            continue
        new_neighbors = region_query(X, j, eps)
        if len(new_neighbors) >= min_pts:
            neighbors = np.unique(np.concatenate((neighbors, new_neighbors)))
        k += 1

# Sample data
X = np.array([[1,1], [1.2,1.2], [1.1,1.3], [5,5], [5.1,5.1], [10,10]])

labels = dbscan(X, eps=0.5, min_pts=2)
print("Labels:", labels)
```

Experiment 8: Implement Dimensionality Reduction using PCA, SVD, or LDA.



```
import numpy as np
import statsmodels.multivariate.pca as pca

# Sample data, less collinear
X = np.array([[2.5, 2.4, 3.5], [1.0, 0.5, 1.2], [2.2, 2.9, 3.1], [1.9, 2.2, 2

# PCA
model = pca.PCA(X, ncomp=2)
print("Principal components:\n", model.factors)

# SVD
U, S, V = np.linalg.svd(X, full_matrices=False)
print("SVD singular values:", S)

# For LDA, need classes
classes = np.array([0,0,0,1,1])
means = [np.mean(X[classes==c], axis=0) for c in np.unique(classes)]
S_w = sum([np.cov(X[classes==c].T) * (sum(classes==c)-1) for c in np.unique(c
diff = means[0] - means[1]
w = np.dot(np.linalg.inv(S_w), diff)
print("LDA direction:", w)
```

↳ Explain ensemble boosting details

↳ Implement Random Forest