Experiment No.2

Title A company wants to analyze the relationship between an employee's years of experience and their salary. Develop a simple linear regression model that predicts the salary of an employee based on the number of years they have worked. Additionally, evaluate the performance of the model using standard regression metric.

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
dataset = pd.read_csv('Salary_data.csv')
print("Dataset Preview:")
print(dataset.head())
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
print("\nTraining and Test Split Completed.")
regressor = LinearRegression()
regressor.fit(X_train, Y_train)
print("\nModel Training Completed.")
Y_train_pred = regressor.predict(X_train)
Y_test_pred = regressor.predict(X_test)
train_mse = mean_squared_error(Y_train, Y_train_pred)
test_mse = mean_squared_error(Y_test, Y_test_pred)
train_r2 = r2_score(Y_train, Y_train_pred)
test_r2 = r2_score(Y_test, Y_test_pred)
plt.scatter(X_train, Y_train, color='blue', label='Training Data')
plt.plot(X_train, Y_train_pred, color='red', label='Regression Line')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
plt.scatter(X_test, Y_test, color='green', label='Test Data')
plt.plot(X_train, Y_train_pred, color='red', label='Regression Line')
plt.title('Salary vs Experience (Test Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
print(f"\nRegression Coefficients:")
print(f"Intercept (b0): {regressor.intercept_:.2f}")
print(f"Slope (b1): {regressor.coef_[0]:.2f}")
```

Experiment No. 3
Title: Study and Implement Multiple Linear Regression
Code

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

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

df = df.dropna()

X = df[['Duration', 'Pulse', 'Maxpulse']]
y = df['Calories']

X = pd.get_dummies(X, drop_first=True)

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared Value: {r2}")

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.show()
```

Experiment No. 4
Title: Implement Logistic Regression using any dataset.
Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve,
roc_auc_score
np.random.seed(42)
X1 = np.random.rand(100) * 10
X2 = np.random.rand(100) * 5
y = (2 * X1 + 3 * X2 + np.random.randn(100) * 2 > 20).astype(int)
data = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'Y': y
})

X = data[['X1', 'X2']]
y = data['Y']

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

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

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
y_test_proba = model.predict_proba(X_test)[:, 1]

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
conf_matrix = confusion_matrix(y_test, y_test_pred)
roc_auc = roc_auc_score(y_test, y_test_proba)

print(f"Train Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
print(f"ROC-AUC Score: {roc_auc:.2f}")
print("\nConfusion Matrix:")
print(conf_matrix)

fpr, tpr, _ = roc_curve(y_test, y_test_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})", color='blue')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.grid()
plt.show()
```

Experiment No. 5

Title: Study and Implement Decision Tree.

Code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_model = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
tree.plot_tree(dt_model, feature_names=data.feature_names,
class_names=data.target_names, filled=True)
plt.show()
```

Experiment No. 6

Title: Implement Naïve Bayes Classifier on data set of your choice. Test and Compare for Accuracy and Precision.

Code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, classification_report
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
classification_rep = classification_report(y_test, y_pred, target_names=iris.target_names)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print("Classification Report:\n", classification_rep)
```

Experiment No. 7

Title: Implement K Nearest Neighbour (KNN) classification

Code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
data = load_iris()
X, y = data.data, data.target
X_reduced = X[:, :2]
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_reduced, y)
def plot_decision_boundary(X, y, model):
cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF', '#AAFFAA'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF', '#00FF00'])
h = .02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
plt.title("KNN Decision Boundary")
plt.show()
plot_decision_boundary(X_reduced, y, knn)
```

Experiment No. 8
Title: Implement Support Vector Machine algorithm.
Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000,
n_features=2,
n_informative=2,
n_redundant=0,
n_repeated=0,
n_classes=2,
random_state=42)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Dataset Visualization')
plt.show()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
print(classification_report(y_test, y_pred))
```

Experiment No. 9

Title: Given the following data, which specify classifications for nine combinations of VAR1 and VAR2. Predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means (i.e. 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
data = {
    'VAR1': [1.713, 0.180, 0.353, 0.940, 1.486, 1.266, 1.540, 0.459, 0.773],
    'VAR2': [1.586, 1.786, 1.240, 1.566, 0.759, 1.106, 0.419, 1.799, 0.186]
}
df = pd.DataFrame(data)
X = df[['VAR1', 'VAR2']].values
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', label='Centroids')
plt.title('K-Means Clustering (k=3)')
plt.xlabel('VAR1')
plt.ylabel('VAR2')
plt.legend()
plt.grid(True)
plt.show()
new_point = np.array([[0.906, 0.606]])
predicted_cluster = kmeans.predict(new_point)
print("New data point [0.906, 0.606] belongs to cluster:", predicted_cluster[0])
```

Experiment No. 10
Title: Unsupervised Learning: Implement K-Means Clustering and Hierarchical clustering on proper data set of your choice. Compare their Convergence
Code
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.metrics import silhouette_score
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
iris = load_iris()
X = iris.data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
inertia = []
silhouette_scores = []
K_range = range(2, 10)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))
plt.figure(figsize=(10, 4))
plt.plot(K_range, inertia, marker='o', linestyle='--', label='Inertia')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (SSE)')
plt.title('Elbow Method for K-Means')
plt.legend()
plt.show()
plt.figure(figsize=(10, 4))
plt.plot(K_range, silhouette_scores, marker='o', linestyle='--', color='red', label='Silhouette Score')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for K-Means')
plt.legend()
plt.show()
plt.figure(figsize=(10, 6))
linked = linkage(X_scaled, method='ward')
dendrogram(linked, truncate_mode='level', p=5)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
hc_labels = hc.fit_predict(X_scaled)
plt.figure(figsize=(10, 5))
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1], hue=hc_labels, palette='viridis', s=60)
plt.title("Hierarchical Clustering - Cluster Visualization")
plt.show()
```

Experiment No. 11
Title: Study and Implement Bagging/Boosting using Random Forests.
Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import make_classification
data, labels = make_classification(n_samples=1000, n_features=20, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
bagging = BaggingClassifier(estimator=RandomForestClassifier(), n_estimators=50,
random_state=42)
bagging.fit(X_train, y_train)
y_pred_bagging = bagging.predict(X_test)
print("Bagging Accuracy:", accuracy_score(y_test, y_pred_bagging))
boosting = AdaBoostClassifier(estimator=RandomForestClassifier(n_estimators=10,
random_state=42), n_estimators=50, random_state=42)
boosting.fit(X_train, y_train)
y_pred_boosting = boosting.predict(X_test)
print("Boosting Accuracy:", accuracy_score(y_test, y_pred_boosting))
```