```
# -*- coding: utf-8 -*-
"""ExperimentML.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1EzNUyF4HnkaeeXdjXaxBB0_rZPZeB9JQ

### **CODE FOR LINEAR REGRESSION**
"""

import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

diabetes = load_diabetes()

X = diabetes.data[:,2].reshape(-1,1)
y = diabetes.target

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)

slope = model.coef_[0]
intercept = model.intercept_
print(f"Equation of the line: y = {slope:.2f}x + {intercept:.2f}")

plt.scatter(X_test, y_test , color = "blue")
plt.plot(X_test, model.predict(X_test), color = "red")
plt.show()

"""### **CODE FOR LOGISTICS REGRESSION**"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()

X = cancer.data[:, :2]
y = cancer.target

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

model = LogisticRegression()
```

```
model.fit(X_train , y_train)


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, 0.1),
np.arange(y_min,y_max,0.1))


Z = model.predict(np.c_[xx.ravel(),yy.ravel()]).reshape(xx.shape)


plt.contourf(xx,yy,Z,alpha=0.8)


plt.scatter(X_test[:,0] ,X_test[:,0], c= y_test , edgecolors = 'k', marker = 'o' )


plt.xlabel(cancer.feature_names[0])
plt.ylabel(cancer.feature_names[1])
plt.title('Logistic Regression Decision Boundary')


y_pred = model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")


plt.show()



"""### **CODE FOR ID3 DECISION TREE**"""
```

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = DecisionTreeClassifier(criterion='entropy')
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Display accuracy, precision, recall, F1-score, and support
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=iris.target_names))
```

```
# Plotting the tree


plt.figure(figsize=(10, 8))

plot_tree(model, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)

plt.show()


"""### **CODE FOR CONFUSION MATRIX**"""


import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion_matrix, classification_report


# Load dataset

iris = load_iris()

X = iris.data

y = (iris.target == 0).astype(int)  # Binary classification: Setosa vs non-Setosa


# Split data

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


# Train a model

model = DecisionTreeClassifier()
```

```
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

# Calculate specificity, F1-score, Type 1 & Type 2 errors
specificity = tn / (tn + fp)
f1 = 2 * tp / (2 * tp + fp + fn)
type_1_error = fp / (fp + tn)  # False Positive Rate
type_2_error = fn / (fn + tp)  # False Negative Rate

# Display results
print(f"F1-score: {f1:.2f}")
print(f"Specificity: {specificity:.2f}")
print(f"Type 1 Error (False Positive Rate): {type_1_error:.2f}")
print(f"Type 2 Error (False Negative Rate): {type_2_error:.2f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Setosa', 'Setosa'], yticklabels=['Non-Setosa', 'Setosa'])
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

"""### **CODE FOR RANDOM FOREST ALGORITHM**"""

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target
df = pd.DataFrame(data=X, columns=iris.feature_names)

print("\nSample of the dataset (5 rows):")
print(df.head())

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

# Train Random Forest model
```

```
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\nClassification Accuracy: {accuracy:.2f}")

# Display predicted species
predicted_species = iris.target_names[y_pred]
print("\nPredicted species from the model:")
print(predicted_species)

# Plotting feature importance
plt.figure(figsize=(10, 6))
plt.barh(iris.feature_names, model.feature_importances_, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.show()



"""### **CODE FOR SUPPORT VECTOR MACHINE**"""
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load dataset
iris = load_iris()
X = iris.data[:, :2]  # Use the first two features for 2D plotting
y = iris.target

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

# Train SVM model
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
# Create a mesh grid for plotting decision boundary
xx, yy = np.meshgrid(np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1, 100),
                     np.linspace(X[:, 1].min() - 1, X[:, 1].max() + 1, 100))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

# Plotting
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap='coolwarm')
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Support Vector Machine Decision Boundary')
plt.show()

"""### **CODE FOR GRAPH BASED CLUSTERING**"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import SpectralClustering

# Load dataset
iris = load_iris()
X = iris.data[:, :2]  # Use the first two features for 2D plotting

# Apply Spectral Clustering
```

```
clustering = SpectralClustering(n_clusters=3, affinity='nearest_neighbors',
random_state=42)
labels = clustering.fit_predict(X)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Graph-Based Clustering (Spectral Clustering) on Iris Dataset')
plt.show()

"""### **CODE FOR DB-SCAN AND CLUSTERING ALGORITHM**"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score, adjusted_rand_score

# Generate synthetic dataset (make_moons)
X, _ = make_moons(n_samples=300, noise=0.1, random_state=42)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.2, min_samples=5)
labels = dbscan.fit_predict(X)
```

```
# Calculate silhouette coefficient and adjusted Rand index
if len(set(labels)) > 1:  # Check if there is more than one cluster
    silhouette_avg = silhouette_score(X, labels)
    rand_index = adjusted_rand_score(_, labels)
else:
    silhouette_avg = -1  # If there's only one cluster
    rand_index = -1

# Display results
print(f"Silhouette Coefficient: {silhouette_avg:.2f}")
print(f"Adjusted Rand Index: {rand_index:.2f}")

# Plotting the density-based clustering
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Density-Based Clustering (DBSCAN) on Synthetic Data')
plt.colorbar(label='Cluster Label')
plt.show()

"""### **CODE FOR PRINCIPAL COMPONENT ANALYSIS(PCA)**"""

import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
```

```
# Load dataset
iris = load_iris()
X = iris.data

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=iris.target, cmap='viridis', marker='o')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.colorbar(ticks=[0, 1, 2], label='Species', format='%(value)s')
plt.show()

import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

# Load dataset
iris = load_iris()
X = iris.data
```

```
# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_

# Plotting PCA results
plt.figure(figsize=(12, 6))

# Scatter plot of the first two principal components
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=iris.target, cmap='viridis', marker='o')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.colorbar(ticks=[0, 1, 2], label='Species', format='%(value)s')

# Plot explained variance ratio
plt.subplot(1, 2, 2)
plt.bar(range(1, len(explained_variance) + 1), explained_variance, alpha=0.7,
color='blue')
plt.xticks(range(1, len(explained_variance) + 1))
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Principal Components')
plt.title('Explained Variance Ratio of Principal Components')
```

```
plt.tight_layout()
plt.show()
```