Pro 1

```python
import statistics
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load the iris dataset
data = load_iris()
sepal_lengths = data.data[:, 0]

# Compute central tendency measures
mean = statistics.mean(sepal_lengths)
median = statistics.median(sepal_lengths)

# Calculate mode (handling StatisticsError)
try:
    mode = statistics.mode(sepal_lengths)
except statistics.StatisticsError as e:
    mode = f"No unique mode: {e}"

# Compute measures of dispersion
variance = statistics.variance(sepal_lengths)
std_dev = statistics.stdev(sepal_lengths)

# Plot histogram
plt.hist(sepal_lengths, bins=20, color='skyblue', edgecolor='black')
plt.title('Histogram of Sepal Lengths')
plt.xlabel('Sepal Length')
plt.ylabel('Frequency')
plt.show()


# Print central tendency measures
print("\nCentral Tendency Measures:")
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)

# Print measures of dispersion
print("\nDispersion Measures:")
print("Variance:", variance)
print("Standard Deviation:", std_dev)
```

Pro 2

```python
import numpy as np
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
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the Diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Linear Regression
linear_reg = LinearRegression().fit(X_train, y_train)
linear_pred = linear_reg.predict(X_test)

# Multiple Regression with selected features
selected_features = [2, 3, 6]
multiple_reg = LinearRegression().fit(X_train[:, selected_features], y_train)
multiple_pred = multiple_reg.predict(X_test[:, selected_features])

# Calculate and print error metrics
linear_mae = mean_absolute_error(y_test, linear_pred)
linear_rmse = np.sqrt(mean_squared_error(y_test, linear_pred))
multiple_mae = mean_absolute_error(y_test, multiple_pred)
multiple_rmse = np.sqrt(mean_squared_error(y_test, multiple_pred))

print("Linear Regression Error Metrics:")
print("MAE:", linear_mae)
print("RMSE:", linear_rmse)
print("\nMultiple Regression Error Metrics:")
print("MAE:", multiple_mae)
print("RMSE:", multiple_rmse)

# Plot the actual vs. predicted values
plt.figure(figsize=(10, 5))
plt.scatter(y_test, linear_pred, color='b', label='Linear Regression')
plt.scatter(y_test, multiple_pred, color='r', label='Multiple Regression')
plt.plot([0, 350], [0, 350], 'g--', label='Perfect Prediction')
plt.xlabel('Actual Value')
plt.ylabel('Predicted Value')
plt.title('Linear vs. Multiple Regression')
plt.legend()
plt.show()
```

Pro 3

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the logistic regression model with a higher max_iter
model = LogisticRegression(max_iter=1000)

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:\n",report)

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix using a heatmap
plt.figure(figsize=(10, 5))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False,xticklabels=data.target_names,
yticklabels=data.target_names)

plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Por 4

```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train the logistic regression model with an increased max_iter
model = LogisticRegression(max_iter=1000).fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n",conf_matrix)

# Plot the confusion matrix
plt.figure(figsize=(6, 5))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False,xticklabels=data.target_names,
yticklabels=data.target_names)

plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Pro 5

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

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

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# K-nearest Neighbors
knn = KNeighborsClassifier().fit(X_train, y_train)
knn_accuracy = accuracy_score(y_test, knn.predict(X_test))

# Naive Bayes
nb = GaussianNB().fit(X_train, y_train)
nb_accuracy = accuracy_score(y_test, nb.predict(X_test))

# Print the accuracies
print("KNN Accuracy:", knn_accuracy)
print("Naive Bayes Accuracy:", nb_accuracy)

# Plot true, KNN predicted, and Naive Bayes predicted labels
plt.figure(figsize=(15, 5))  # Adjusted figsize for better
visualization

plt.subplot(1, 3, 1)
plt.scatter(X_test[:, 0], X_test[:, 1], c=knn.predict(X_test),
cmap='viridis')
plt.title('KNN Predicted Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.subplot(1, 3, 2)
plt.scatter(X_test[:, 0], X_test[:, 1], c=nb.predict(X_test),
cmap='viridis')
plt.title('Naive Bayes Predicted Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Pro 6

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

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Decision Tree classifier
dt_classifier = DecisionTreeClassifier(criterion='entropy',
splitter='random', max_depth=10)

# Model fit
dt_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = dt_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print results
print("Accuracy:", accuracy)
print("Classification Report:\n",classification_rep)

# Convert class_names to a list
class_names_list = data.target_names.tolist()

# Visualize the Decision Tree
plt.figure(figsize=(12, 15))
plot_tree(dt_classifier, filled=True,
feature_names=data.feature_names, class_names=class_names_list,
rounded=True)
plt.title("Decision Tree Visualization")
plt.show()
```

Pro 7

```
import os
import warnings
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris

# Set OMP_NUM_THREADS environment variable to avoid KMeans memory
leak on Windows
os.environ["OMP_NUM_THREADS"] = "1"

# Ignore the UserWarning from KMeans
warnings.filterwarnings("ignore", category=UserWarning)

# Load the Iris dataset
data = load_iris()
x = data.data
k = 3

# Initialize and fit the KMeans model
km = KMeans(n_clusters=k, n_init=10)
km.fit(x)

# Get cluster assignments and centroids
cl = km.labels_
ce = km.cluster_centers_

# Print cluster assignments and centroids
print("Cluster Assignment:", cl)
print("Centroid Assignment:", ce)

# Plot clusters and centroids
plt.scatter(x[:, 0], x[:, 1], c=cl, cmap='viridis')
plt.scatter(ce[:, 0], ce[:, 1], c='red', marker='x')
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")
plt.title("K-Means Cluster")
plt.show()
```

Pro 8

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

# Load the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()

# Normalize pixel values to the range [0, 1]
train_images, test_images = train_images / 255.0, test_images /
255.0

# Define a simple CNN architecture
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,
32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

# Compile the model
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
,metrics=['accuracy'])

# Train the model with a smaller number of epochs
history = model.fit(train_images, train_labels, epochs=3,
validation_data=(test_images, test_labels))

# Evaluate the model on the test data
test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test Accuracy: {test_acc}")

# Plot the training and validation accuracy over epochs
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Pro 9

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models

# Generate sample data
data = np.random.rand(1000, 32)

# Normalize the data to range [0, 1]
data = (data - data.min()) / (data.max() - data.min())

# Encoder
inputs = layers.Input(shape=(32,))
encoded = layers.Dense(16, activation='relu')(inputs)

# Decoder
decoded = layers.Dense(32, activation='sigmoid')(encoded)

# Create Autoencoder
autoencoder = models.Model(inputs, decoded)

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mse')

# Train the autoencoder
autoencoder.fit(data, data, epochs=50, batch_size=32, verbose=0)

# Encode and decode the data
decoded_data = autoencoder.predict(data)

print("Orignail Data",data[0])
print("Decode Data",decoded_data[0])

# Visualize the original and decoded data for the first sample
plt.figure(figsize=(8, 4))
plt.plot(data[0], label='Original', marker='o')
plt.plot(decoded_data[0], label='Decoded', marker='x')
plt.title('Original vs Decoded Data')
plt.xlabel('Feature Index')
plt.ylabel('Value')
plt.legend()
plt.show()
```

Pro 10

```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import models, layers

# Generate sample data
data = np.random.rand(100, 1)
target = np.full((100, 1), 100)  # Replicate the target value for each sequence

# Reshape data
data = data.reshape((100, 1, 1))

# Define the model
model = models.Sequential([
    layers.LSTM(units=50, activation='relu', input_shape=(None, 1)),
    layers.Dense(units=1)
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Train the model
model.fit(data, target, epochs=100, verbose=0)

# Make predictions
predictions = model.predict(data)

# Print the results
print("Original Data:\n",target.flatten())
print("Predicted Data:\n",predictions.flatten())

# Visualization
plt.plot(target, label='Original Data', marker='o')
plt.plot(predictions, label='Predicted Data', marker='x')
plt.title('Original Data vs Predicted Data')
plt.xlabel('Sample Index')
plt.ylabel('Value')
plt.legend()
plt.show()
```