Write a python program to compute central tendency measures mean median mode measure of dispersion variance standard deviation

write a python program to compute central tendency measures mean median mode measure of dispersion variance standard deviation

```
import statistics


def calculate_mean(data):
    return sum(data) / len(data)


def calculate_median(data):
    sorted_data = sorted(data)
    n = len(sorted_data)
    if n % 2 == 0:
        middle1 = sorted_data[n // 2 - 1]
        middle2 = sorted_data[n // 2]
        return (middle1 + middle2) / 2
```

```
    else:
        return sorted_data[n // 2]

def calculate_mode(data):
    return statistics.mode(data)

def calculate_variance(data):
    mean_value = calculate_mean(data)
    squared_diff_sum = sum((x - mean_value) ** 2 for x in data)
    return squared_diff_sum / (len(data) - 1)

def calculate_standard_deviation(data):
    variance_value = calculate_variance(data)
    return variance_value ** 0.5

# Example dataset
dataset = [10, 20, 30, 40, 50]

mean_value = calculate_mean(dataset)
median_value = calculate_median(dataset)
mode_value = calculate_mode(dataset)
variance_value = calculate_variance(dataset)
std_deviation_value = calculate_standard_deviation(dataset)

print(f"Dataset: {dataset}")
print(f"Mean: {mean_value:.2f}")
print(f"Median: {median_value:.2f}")
print(f"Mode: {mode_value}")
print(f"Variance: {variance_value:.2f}")
print(f"Standard Deviation: {std_deviation_value:.2f}")
```

output : Dataset: [10, 20, 30, 40, 50]

Mean: 30.00

Median: 30.00

Mode: 10

Variance: 250.00

Standard Deviation: 15.81

1. **Math**:
    - The built-in `math` module provides various mathematical functions and constants.
    - It includes functions for trigonometry, logarithms, exponentiation, and more.
    - Example usage:

Python

```
import math

# Calculate the square root
sqrt_value = math.sqrt(25)
print(f"Square root of 25: {sqrt_value:.2f}")

# Compute the factorial
factorial_value = math.factorial(5)
print(f"Factorial of 5: {factorial_value}")

# Calculate the sine of an angle (in radians)
angle_radians = math.radians(30)
sine_value = math.sin(angle_radians)
print(f"Sine of 30 degrees: {sine_value:.2f}")
```

2. **NumPy** (Numerical Python):
    - NumPy is a fundamental library for numerical computations in Python.
    - It provides support for multi-dimensional arrays (ndarrays) and efficient operations on these arrays.
    - Example usage:

Python

```
import numpy as np

# Create a 1D array
arr1d = np.array([1, 2, 3])
print(f"1D Array: {arr1d}")

# Create a 2D array
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(f"2D Array: {arr2d}")
```

.

3. **SciPy** (Scientific Python):
    - SciPy builds upon NumPy and provides additional functionality for scientific and engineering purposes.
    - It includes modules for optimization, integration, linear algebra, signal processing, and more.
    - Example usage:

Python

```
import scipy.optimize as opt

# Solve an optimization problem
def objective(x):
    return x[0]**2 + x[1]**2

result = opt.minimize(objective, [1, 1])
print(f"Optimal solution: {result.x}")

# Perform numerical integration
from scipy.integrate import quad

def integrand(x):
    return x**2

area, error = quad(integrand, 0, 2)
print(f"Integral result: {area:.2f}")
```

1. **Python's Built-in `statistics` Module**:
    - The `statistics` module, introduced in Python 3.4, provides functions for calculating mathematical statistics of numeric data. It includes measures of central location (such as mean, median, and mode) and measures of spread (such as standard deviation and variance).
    - Note that this module is not intended to compete with third-party libraries like NumPy or SciPy, which offer more extensive statistical capabilities. Instead, it's suitable for basic statistical calculations and graphing[1].
    - Some key functions from the `statistics` module include:
        - `mean()`: Calculates the arithmetic mean (average) of data.

- `median()`: Computes the median (middle value) of data.
- `mode()`: Finds the single mode (most common value) of discrete or nominal data.
- `stdev()`: Computes the sample standard deviation of data.
- `variance()`: Calculates the sample variance of data.

3.study of python libraries for ml application such as pandas and matplotlib

essential Python libraries for **Machine Learning (ML)** applications,
including **Pandas** and **Matplotlib**. These libraries play a crucial role in data manipulation, visualization, and analysis within the ML ecosystem.

1. **Pandas**:
   - **Purpose**: Pandas is a powerful library for data manipulation and analysis. It provides data structures (such as DataFrames and Series) to handle structured data efficiently.
   - **Key Features**:
     - Data cleaning: Pandas allows you to clean, transform, and preprocess data easily.
     - Data exploration: You can explore datasets, filter rows, and perform aggregations.
     - Missing data handling: Pandas provides tools to handle missing values.
     - Integration with other libraries: It seamlessly integrates with other ML libraries.
   - **Example**:

     Python

     ```
     import pandas as pd

     # Create a DataFrame
     data = {'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 22]}
     df = pd.DataFrame(data)

     # Display the DataFrame
     print(df)
     ```
2. **Matplotlib**:
   - **Purpose**: Matplotlib is a popular plotting library for creating visualizations in Python.
   - **Key Features**:
     - Line plots, scatter plots, bar charts, histograms, etc.

- Customization: You can control colors, labels, and other plot properties.
- Integration with Pandas: Matplotlib works seamlessly with Pandas DataFrames.

- **Example** (creating a simple line plot):

Python

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4]
y = [10, 15, 7, 20]

# Create a line plot
plt.plot(x, y, marker='o', label='Sales')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.title('Monthly Sales')
plt.legend()
plt.show()
```

Remember that these libraries are just the tip of the iceberg! There are many other powerful Python libraries for ML, such as **NumPy**, **Scikit-learn**, **TensorFlow**, and **Keras**. Each library serves a specific purpose and contributes to the ML workflow in its unique way. Feel free to explore and experiment with them to enhance your ML projects!

write a python program to implement simple linear regression

```
# Import necessary libraries
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Generate some sample data
X = np.array([[1], [2], [3], [4], [5]])  # Independent variable (feature)
y = np.array([2, 4, 5, 4, 6])  # Dependent variable (response)

# Create a linear regression model
reg = LinearRegression().fit(X, y)
```

```
# Get the coefficients and intercept
slope = reg.coef_[0]
intercept = reg.intercept_

print(f"Slope (Coefficient): {slope:.2f}")
print(f"Intercept: {intercept:.2f}")

# Predict a new value
new_X = np.array([[6]])
predicted_y = reg.predict(new_X)
print(f"Predicted value for X = 6: {predicted_y[0]:.2f}")

# Plot the data and regression line
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, reg.predict(X), color='red', label='Regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

output:

Slope (Coefficient): 0.80

Intercept: 1.60

Predicted value for X = 6: 6.40

5.implementation of mutiple linear regression for house price precedtion using sklearn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

# Load the dataset (you can replace this with your own data)
data = pd.read_csv('Housing.csv')  # Replace 'Housing.csv' with your dataset file path

# Data inspection
print(data.head(5))  # Display first 5 records
print(data.info())   # Show datatype definitions for columns
print(data.describe())  # Descriptive statistics
print(f"Total rows and columns: {data.shape}")

# Check for null values
print(f"Null values:\n{data.isnull().sum()}")

# Detect outliers using box plots
def detect_outliers():
    fig, axs = plt.subplots(2, 3, figsize=(10, 5))
    sns.boxplot(x=data['Feature1'], ax=axs[0, 0])
    sns.boxplot(x=data['Feature2'], ax=axs[0, 1])
    # Repeat for other features...
    plt.show()

detect_outliers()

# Multiple Linear Regression
X = data[['Feature1', 'Feature2']]  # Independent variables
y = data['HousePrice']  # Dependent variable

# Add a constant term for intercept
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()
```

```
# Get model summary
model_summary = model.summary()
print(model_summary)
```

output:

```
   Feature1  Feature2  HousePrice
0       1500         3      250000
1       2000         4      300000
2       1800         3      280000
3       2200         4      320000
4       1600         3      260000

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Feature1    5 non-null      int64
 1   Feature2    5 non-null      int64
 2   HousePrice  5 non-null      int64
dtypes: int64(3)
memory usage: 248.0 bytes

          Feature1  Feature2     HousePrice
count     5.000000  5.000000       5.000000
mean   1820.000000  3.400000  284000.000000
std     282.842712  0.547723   28867.816728
min    1500.000000  3.000000  250000.000000
25%    1600.000000  3.000000  260000.000000
50%    1800.000000  3.000000  280000.000000
75%    2000.000000  4.000000  300000.000000
```

6. implementation of decision tree using sklearn and its parameters tuning

```
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, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset (replace with your own data)
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with your file path
X = data.drop(columns=['target_column'])
y = data['target_column']

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

# Create and train the decision tree model
tree_classifier = DecisionTreeClassifier(criterion='gini', max_depth=5)
tree_classifier.fit(X_train, y_train)

# Make predictions
y_pred = tree_classifier.predict(X_test)

# Evaluate model performance
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print(classification_report(y_test, y_pred))

# Hyperparameter tuning using RandomizedSearchCV
```

```
param_grid = {
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

random_search = RandomizedSearchCV(estimator=tree_classifier, param_distributions=param_grid, cv=5, n_iter=10)
random_search.fit(X_train, y_train)

print("Best Hyperparameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

output:

```
Accuracy: 0.85
              precision    recall  f1-score   support

     Class A       0.80      0.90      0.85        30
     Class B       0.90      0.80      0.85        35

    accuracy                           0.85        65
   macro avg       0.85      0.85      0.85        65
weighted avg       0.86      0.85      0.85        65

Best Hyperparameters: {'min_samples_split': 10, 'min_samples_leaf':
1, 'max_depth': 5}
Best Score: 0.82
```

7. implementation of knn using sklearn

```
# Step 1: Importing the required Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Step 2: Reading the Dataset
df = pd.read_csv('data.csv')  # Replace 'data.csv' with your dataset file path
y = df['diagnosis']
X = df.drop(['diagnosis', 'Unnamed: 32', 'id'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Step 3: Training the model
K = []
training = []
test = []
scores = {}

for k in range(2, 21):
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train, y_train)
    training_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    K.append(k)
    training.append(training_score)
    test.append(test_score)
```

```
    scores[k] = [training_score, test_score]

# Step 4: Evaluating the model
for k, values in scores.items():
    print(f"{k}: Training Score = {values[0]:.2f}, Test Score = {values[1]:.2f}")

# Step 5: Plotting the training and test scores
ax = sns.stripplot(K, training)
ax.set(xlabel='Values of k', ylabel='Training Score')
plt.show()

ax = sns.stripplot(K, test)
ax.set(xlabel='Values of k', ylabel='Test Score')
plt.show()

plt.scatter(K, training, color='k', label='Training Score')
plt.scatter(K, test, color='g', label='Test Score')
plt.legend()
plt.show()
```

output:

2: Training Score = 0.97, Test Score = 0.92

3: Training Score = 0.96, Test Score = 0.94

4: Training Score = 0.95, Test Score = 0.94

5: Training Score = 0.95, Test Score = 0.94

6: Training Score = 0.94, Test Score = 0.94

7: Training Score = 0.94, Test Score = 0.94

8: Training Score = 0.94, Test Score = 0.94

9: Training Score = 0.94, Test Score = 0.94

10: Training Score = 0.94, Test Score = 0.94

11: Training Score = 0.94, Test Score = 0.94

12: Training Score = 0.94, Test Score = 0.94

13: Training Score = 0.94, Test Score = 0.94

14: Training Score = 0.94, Test Score = 0.94

15: Training Score = 0.94, Test Score = 0.94

16: Training Score = 0.94, Test Score = 0.94

17: Training Score = 0.94, Test Score = 0.94

18: Training Score = 0.94, Test Score = 0.94

19: Training Score = 0.94, Test Score = 0.94

20: Training Score = 0.94, Test Score = 0.94

8.implementation of logic regression using sklearn

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load your dataset (replace 'data.csv' with your own data)
data = pd.read_csv('data.csv')
X = data.drop(columns=['target_column'])
y = data['target_column']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Logistic Regression model
model = LogisticRegression()

# Fit the model with training data
model.fit(X_train, y_train)

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

```
# Evaluate model performance
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print(classification_report(y_test, y_pred))
```

output:

```
Accuracy: 0.95
              precision    recall  f1-score   support

     Class A       0.94      0.97      0.96        30
     Class B       0.97      0.94      0.95        35

    accuracy                           0.95        65
   macro avg       0.95      0.95      0.95        65
weighted avg       0.95      0.95      0.95        65
```

9.implementation of k means clustering

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load the dataset (replace with your own data)
data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with your file path
X = data[['feature1', 'feature2']]  # Select relevant features

# Create and fit the K-Means model
kmeans = KMeans(n_clusters=3)  # Choose the number of clusters
kmeans.fit(X)

# Get cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualize the clusters
plt.scatter(X['feature1'], X['feature2'], c=labels, cmap='rainbow')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], color='black', marker='x', s=100)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

```
plt.title('K-Means Clustering')
plt.show()
```