Simple Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
# Sample data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])
# Calculate means
x_mean = np.mean(x)
y_mean = np.mean(y)
# Calculate coefficients
numerator = np.sum((x - x_mean) * (y - y_mean))
denominator = np.sum((x - x_mean)**2)
slope = numerator / denominator
intercept = y_mean - slope * x_mean
# Predict values
y_pred = slope * x + intercept
# Output model parameters
print(f"Slope (m): {slope}")
print(f"Intercept (b): {intercept}")
# Visualization
plt.scatter(x, y, color='blue', label='Actual data')
plt.plot(x, y_pred, color='red', label='Regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Simple Linear Regression')
plt.show()
```

Explaination

1. **Your Data**

You start with:

x = [1, 2, 3, 4, 5]

y = [2, 4, 5, 4, 5]

This means: “At x=1, y was 2; at x=2, y was 4…” and so on.

2. **Mean Calculation**

```python

x_mean = np.mean(x)

y_mean = np.mean(y)

You’re asking: “Where’s the center of gravity for x and y?” It helps center our thinking.

#### 3. **Finding the Line's Slope (m)**

numerator = np.sum((x - x_mean) * (y - y_mean))

denominator = np.sum((x - x_mean)**2)

slope = numerator / denominator

This is how you calculate **how steep the line is**. If `m` is positive, the line goes up; if it’s negative, the line goes down.

#### 4. **Finding the Y-Intercept (b)**

intercept = y_mean - slope * x_mean

This tells you where the line touches the y-axis.

#### 5. **Predicting y-values**

y_pred = slope * x + intercept

Now that you’ve got the formula `y = mx + b`, you use it to get predicted y-values for each x.

#### 6. **Visualizing**

plt.scatter(x, y, ...)   # Actual data points (blue dots)

plt.plot(x, y_pred, ...) # Regression line (red line)

**1)**

data=[1,2,3,4,5]

#Mean

mean=sum(data)/len(data)

#Median

sorted_data=sorted(data)

if len(data)%2==0:

    middle=len(data)//2

    median=(sorted_data[middle-1]-sorted_data[middle])/2
```

```
#Mode
counts={}
for num in data:
    if num in counts:
        counts[num]+=1
    else:
        counts[num]=1
highest_count=0
for count in counts.values():
    if count > highest_count:
        highest_count=count
mode=[]
for num in counts:
    if counts[num]==highest_count:
        mode.append(num)
squared_diffs=[(num-mean)**2 for num in data]
var=sum(squared_diffs)/len(data)
std_dev=var * 0.5
print(mean)
```

Explanation

You're manually calculating mean, median, mode, variance, and standard deviation from the list data = [1, 2, 3, 4, 5].

📌 Step-by-Step Breakdown

📍 Mean (Average)

mean = sum(data)/len(data)

You're adding all numbers in data: 1 + 2 + 3 + 4 + 5 = 15, and dividing by the number of elements: 15 / 5 = 3.0

✅ Output: mean = 3.0

⚠️ Median (Middle Value)

python

```
sorted_data = sorted(data)
if len(data) % 2 == 0:
    median = (sorted_data[middle-1] - sorted_data[middle])/2
```

There’s an issue here: you're subtracting instead of adding.

🛠️ Fix this line like so:

```
median = (sorted_data[middle-1] + sorted_data[middle]) / 2
```

But since your list has odd length, you can just do:

```
median = sorted_data[len(data)//2]  # Which is 3
```

✅ Output: median = 3

📍 Mode (Most Frequent Value)

You're counting frequency of numbers using a dictionary. This part is almost correct:

But then this line is wrong:

```
mode.append(counts)
```

You should append the number, not the entire dictionary. Change it to:

```
mode.append(num)
```

✅ Since all values appear only once, mode will include all numbers.

📍 Variance and Standard Deviation

```
squared_diffs = [(num - mean)**2 for num in data]
```

You're squaring how far each value is from the mean.

Then:

```
var = sum(squared_diffs)/len(data)
```

This gives the population variance.

Finally:

```
std_dev = var * 0.5
```

⚠️ This is not the formula for standard deviation. You should take the square root of the variance:

```
import math
std_dev = math.sqrt(var)
```

4) Multilinear Regression

```
import pandas as pd
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
# Load dataset
df = pd.read_csv('housing.csv')  # Replace with your dataset path
# Select features and target
X = df[['area', 'bedrooms', 'bathrooms']]  # Replace with actual feature column names
y = df['price']  # Replace with actual target column name
# Split 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 model
model = LinearRegression()
model.fit(X_train, y_train)
# Predict on the test set
y_pred = model.predict(X_test)
# Display evaluation metrics
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
# Compare actual vs predicted prices
comparison = pd.DataFrame({'Actual Price': y_test.values, 'Predicted Price': y_pred})
print(comparison.head())
# Visualize actual vs predicted
comparison.head(20).plot(kind='bar', figsize=(12, 6))
plt.title("Comparison of Actual and Predicted Prices")
plt.xlabel("Sample Index")
plt.ylabel("Price")
plt.show()
```