Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant

Simple Linear Regression with Sklearn

Example 1 - Predicting electic bill based on average usage

In [ ]:
```
# Import necessary libraries
import numpy as np                    # NumPy is used for handling arrays, which
from sklearn.linear_model import LinearRegression  # Scikit-Learn's LinearRegres

# Define the dataset
# X represents the average daily usage in hours (our input feature)
# y represents the monthly bill in dollars (our target variable)
X = np.array([[3], [4], [6], [7], [8], [10], [12]])  # input data formatted as a
y = np.array([50, 60, 80, 95, 110, 130, 150])        # output labels as a 1D arr

# Initialize the linear regression model
model = LinearRegression()  # This creates an instance of LinearRegression, read

# Train the model on the dataset
model.fit(X, y)             # fit() finds the best-fit line that minimizes error

# Make a prediction
# Now, let's predict the monthly bill for a household that uses electricity for
predicted_bill_5_hours = model.predict(np.array([[5]]))  # predict() applies the
print("Predicted monthly bill for 5 hours of daily usage:", predicted_bill_5_hou

# Extract and print model parameters
# The slope (rate of increase per additional hour) and intercept (baseline month
slope = model.coef_[0]        # model.coef_ gives the slope of the line, showing
intercept = model.intercept_   # model.intercept_ is the y-intercept, showing th
print("Slope (increase per additional hour):", slope)
print("Intercept (base bill):", intercept)
```

In [ ]:
```
import matplotlib.pyplot as plt

# Plotting the data points
plt.scatter(X, y, color='blue', label='Data Points')  # Scatter plot for actual

# Creating predictions for the entire range of X values for plotting the regress
X_range = np.array([[x] for x in range(2, 14)])  # Generate X values for the ran
y_pred = model.predict(X_range)                   # Get predictions for the rang
```

```
# Plotting the regression line
plt.plot(X_range, y_pred, color='red', label='Regression Line')  # Plot the regr

# Highlight the predicted value for 5 hours of usage
plt.scatter(5, predicted_bill_5_hours, color='green', label='Prediction for 5 ho

# Adding labels and title
plt.title('Monthly Electricity Bill Prediction')            # Title of the plot
plt.xlabel('Average Daily Usage (hours)')                   # X-axis label
plt.ylabel('Monthly Bill (dollars)')                        # Y-axis label
plt.legend()                                                # Show the legend
plt.grid(True)                                             # Add a grid for bette

# Show the plot
plt.show()
```

Final answers based on model output:

- Predicted monthly bill for 5 hours of daily usage: $72.00704225352112
- Increase in monthly bill for each additional hour of daily usage

Code Breakdown of Example 1

In [27]:
```
# Import necessary libraries
import numpy as np                      # NumPy is used for handling arrays, which
from sklearn.linear_model import LinearRegression  # Scikit-Learn's LinearRegres
```

in python

`from abc.d import e`

Module and Package:

A **package** is a collection of Python modules. In our example, `abc` is a package that contains a module named `d`. A **module** is a file containing **Python code (functions, classes, variables)**. The file **d.py** contains the definition of **e**

Simulate this example:

my_package/ ├── **init**.py └── d.py

`__init__.py` can be an empty file that tells Python that my_package is a package.

`d.py` will contain a function `e`

Content of `d.py`

```
# d.py

def e():
    return "Hello from function e!"
```

Let's create a script that imports and uses the function e from the package.

```
# main.py
```

```python
# Importing the function e from the module d in the package my_package
from my_package.d import e

# Calling the imported function
result = e()

# Printing the result
print(result)
```

In [26]:
```python
# Define the dataset
# X represents the average daily usage in hours (our input feature)
# y represents the monthly bill in dollars (our target variable)

X = np.array([[3], [4], [6], [7], [8], [10], [12]])  # input data formatted as a

y = np.array([50, 60, 80, 95, 110, 130, 150])        # output labels as a 1D arr
```

DataFrame Creation: The pd.DataFrame(data=X, columns=['Size']) creates a DataFrame from the X array, assigning the column name 'Size'.

Adding Target Values: The line df['Price'] = y adds a new column named 'Price' to the DataFrame, populated with values from the y array.

In [ ]:
```python
import pandas as pd
data = pd.DataFrame(X, columns=['Size'])
data['try'] = 0
```

In [ ]:
```python
data['Prices'] = y
```

In [ ]:
```python
data
```

In [ ]:
```python
# delete the try column
data.drop(columns=['try'])
```

In [28]:
```python
# Initialize the linear regression model
model = LinearRegression()  # This creates an instance of LinearRegression, read
```

`model` : This variable will hold the instance of the `LinearRegression` class.

`LinearRegression()` : This is the constructor of the `LinearRegression` class, which initializes a new object of this class.

In [30]:
```python
# Train the model on the dataset
model.fit(X, y)             # fit() finds the best-fit line that minimizes error
```

Out[30]:
```
▾ LinearRegression
LinearRegression()
```

`.fit()` method is a function of a model object, which is an instance of a class in the scikit-learn library (in this case, the `LinearRegression class`)

`X` is typically a 2D array (or a DataFrame) representing the input data, where each row is a data sample, and each column is a feature.

`y` is a 1D array (or column in a DataFrame) where each entry is the label or target value for a data sample in X.

`fit()`:

- Takes in X and y and analyzes the relationship between them.
- For a linear regression model, it calculates the optimal line that best fits the data by minimizing the error between the predicted values and actual values in y.
- Internally, it stores the calculated **coefficients** and **intercept** so the model can make predictions on new data.

In [35]:
```
# Make a prediction
# Now, let's predict the monthly bill for a household that uses electricity for
predicted_bill_5_hours = model.predict(np.array([[10]]))  # predict() applies th
print("Predicted monthly bill for 5 hours of daily usage:", predicted_bill_5_hou
```

```
Predicted monthly bill for 5 hours of daily usage: 128.99061032863852
```

`model.predict()`:

- `.predict()` is a method associated with the `model` object created after training (in this case, `model`).
- The purpose of `.predict()` is to take in new input data (unseen by the model during training) and generate predictions based on the model's internal parameters (like the coefficients and intercept calculated during fit()).
- `.predict()` expects an input formatted similarly to the X used in `.fit(X, y)`, usually a 2D array with each row as a sample and each column as a feature.

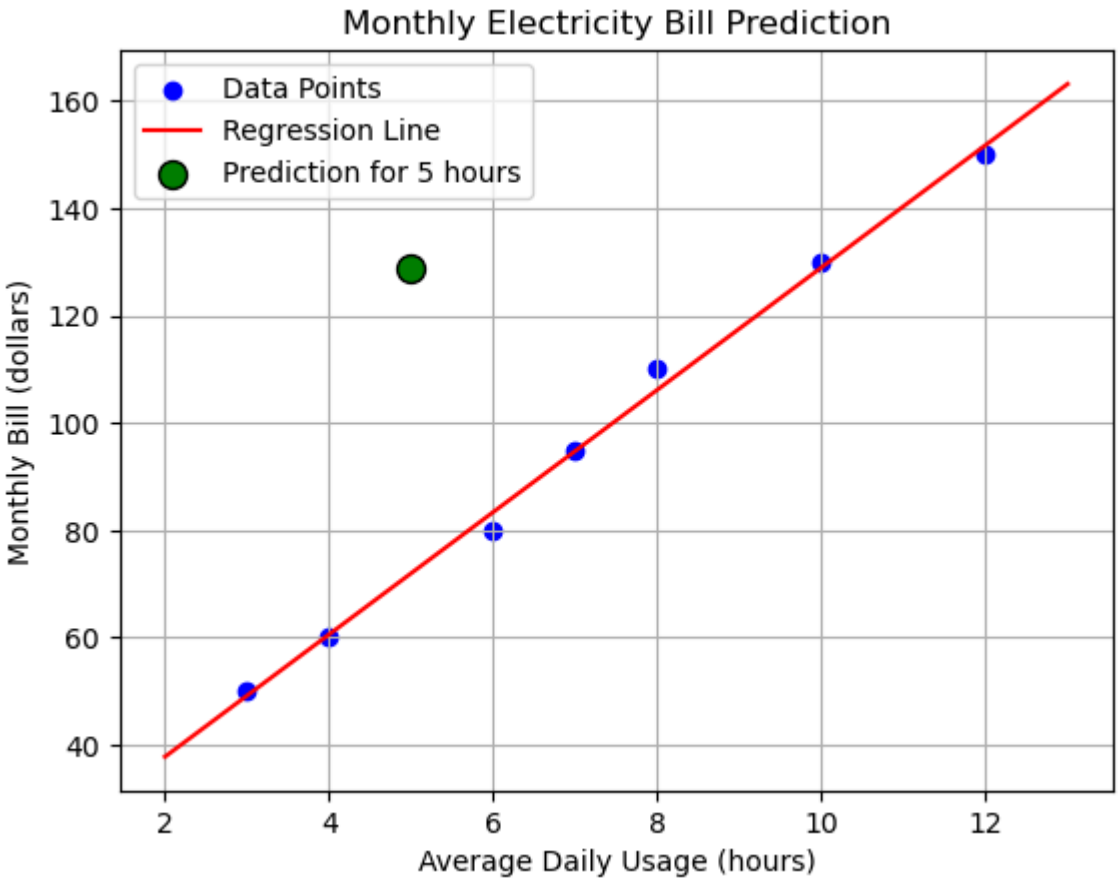
In [37]:
```
# Extract and print model parameters
# The slope (rate of increase per additional hour) and intercept (baseline month
slope = model.coef_[0]        # model.coef_ gives the slope of the line, showing
intercept = model.intercept_   # model.intercept_ is the y-intercept, showing th
print("Slope (increase per additional hour):", slope)
print("Intercept (base bill):", intercept)
```

```
Slope (increase per additional hour): 11.396713615023478
Intercept (base bill): 15.023474178403731
```

In [ ]:
```
```

In [38]:
```
import matplotlib.pyplot as plt

# Plotting the data points
plt.scatter(X, y, color='blue', label='Data Points')  # Scatter plot for actual

# Creating predictions for the entire range of X values for plotting the regress
X_range = np.array([[x] for x in range(2, 14)])  # Generate X values for the ran
y_pred = model.predict(X_range)                   # Get predictions for the rang

# Plotting the regression line
```

```
plt.plot(X_range, y_pred, color='red', label='Regression Line')  # Plot the regr

# Highlight the predicted value for 5 hours of usage
plt.scatter(5, predicted_bill_5_hours, color='green', label='Prediction for 5 ho

# Adding labels and title
plt.title('Monthly Electricity Bill Prediction')            # Title of the plot
plt.xlabel('Average Daily Usage (hours)')                   # X-axis label
plt.ylabel('Monthly Bill (dollars)')                        # Y-axis label
plt.legend()                                                # Show the legend
plt.grid(True)                                             # Add a grid for bette

# Show the plot
plt.show()
```



Another example - for prediction of price based on size

In [6]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

In [7]:
```
# Step 1: Create the Dataset (Size and Price)
# Here, we assume you have a dataset with 'Size' and 'Price' columns
data = {
    'Size': [800, 1000, 1200, 1500, 1800, 2000, 2200, 2500],
    'Price': [150000, 200000, 240000, 290000, 330000, 360000, 400000, 440000]
```

```
}
df = pd.DataFrame(data)
```

Generating data with Random function in python

```
# Step 1: Create a synthetic dataset
# Generate random house sizes (in square feet) between 500 and 2500
np.random.seed(0)  # For reproducibility
house_sizes = np.random.randint(500, 2500, size=100)

# Generate house prices based on sizes with some noise
# Price = 50 * size + some random noise
house_prices = 50 * house_sizes + np.random.normal(0, 10000, size=100)
```

In [25]:
```
np.random.seed(0)  # For reproducibility
house_sizes = np.random.randint(500, 2500, size=100)

# Generate house prices based on sizes with some noise
# Price = 50 * size + some random noise
house_prices = 50 * house_sizes + np.random.normal(0, 10000, size=100)
```

In [27]:
```
house_prices
```

Out[27]:
```
array([ 63253.97784018,  54136.06585234, 120194.14068338,  99991.02040789,
        59311.4391713 ,  37975.62897205,  96479.03976465, 105640.76125382,
        64714.21748056, 123760.42445843,  53944.45082265, 124577.75129464,
       109534.10522807,  55098.73316058,  75943.34103835,  99417.76362555,
        58679.7447415 ,  95552.76299752,  37676.24867481,  38005.96397788,
       107740.06359042,  56140.0276447 ,  40111.21037519,  35621.40834386,
       114729.25957501,  67633.5718261 , 114333.03566508,  61812.77834548,
        62338.50344099,  35726.78165743,  74983.40349621, 100463.25413642,
        26588.88517567,  63091.03008511,  64961.38805129, 100419.42884621,
       108042.42043014, 107870.79053328,  62837.16430472,  63478.851108  ,
        38597.49101136,  82781.30772753,  70861.60270482,  67727.94613106,
        52565.31839815, 102634.62100306,  65742.23999184,  60557.31118318,
        79109.24318983, 132307.14552151, 107011.07030158,  69579.79913805,
        41438.8088836 ,  54111.43477937,  68197.76070726,  46499.74072157,
        12762.17316155,  11986.61449727,  48827.42450066,  60612.85319834,
       113154.40083739,  62550.0831563 ,  82792.09309417,  19256.14390168,
        60091.61234731,  31956.45342156,  26510.38216822, 119000.77554608,
        81447.34653243, 104436.51357165,  94532.05194769,  99407.75899614,
        58337.22719998,  17216.44562768,  41850.3776713 ,  38817.77413129,
       111859.0565129 ,  72558.57988475,  95775.32403129,  62548.46787435,
        69165.20474265,  38744.22289676,  86306.72629248, 121993.03206208,
        35277.95712888,  32000.18473403, 118317.17285194,  60900.79730884,
        53649.81166958,  41325.4813538 ,  78881.99988231,  16785.1978684 ,
       135815.13275381,  61492.7664068 ,  55551.39611962, 119508.38362785,
        71445.20923343,  54289.56849422, 131294.35033869,  33228.56438693])
```

In [8]:
```
# Step 2: Split the Data into Features (X) and Target (y)
X = df[['Size']]  # Selecting 'Size' as the feature
y = df['Price']   # Selecting 'Price' as the target variable
```

In [9]:
```
# Step 3: 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_
```

In [10]:
```
# Step 4: Initialize and Train the Linear Regression Model
model = LinearRegression()
```

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

Out[10]:

```
▾ LinearRegression

LinearRegression()
```

In [11]:

```
# Step 5: Make Predictions on the Test Set
y_pred = model.predict(X_test)
```

purpose of the predict() method is to generate predictions based on the input features provided

takes a single argument, typically a 2D array or a DataFrame, containing the input features for which predictions are to be made.

In [12]:

```
# Step 6: Evaluate the Model
mse = mean_squared_error(y_test, y_pred)  # Calculate Mean Squared Error
r2 = r2_score(y_test, y_pred)             # Calculate R-squared
```

In [13]:

```
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 80837662.97421867
R-squared: 0.9873691151602784
```

MSE is a measure of the average squared differences between predicted and actual values. A high MSE indicates that, on average, predictions deviate significantly from actual values.

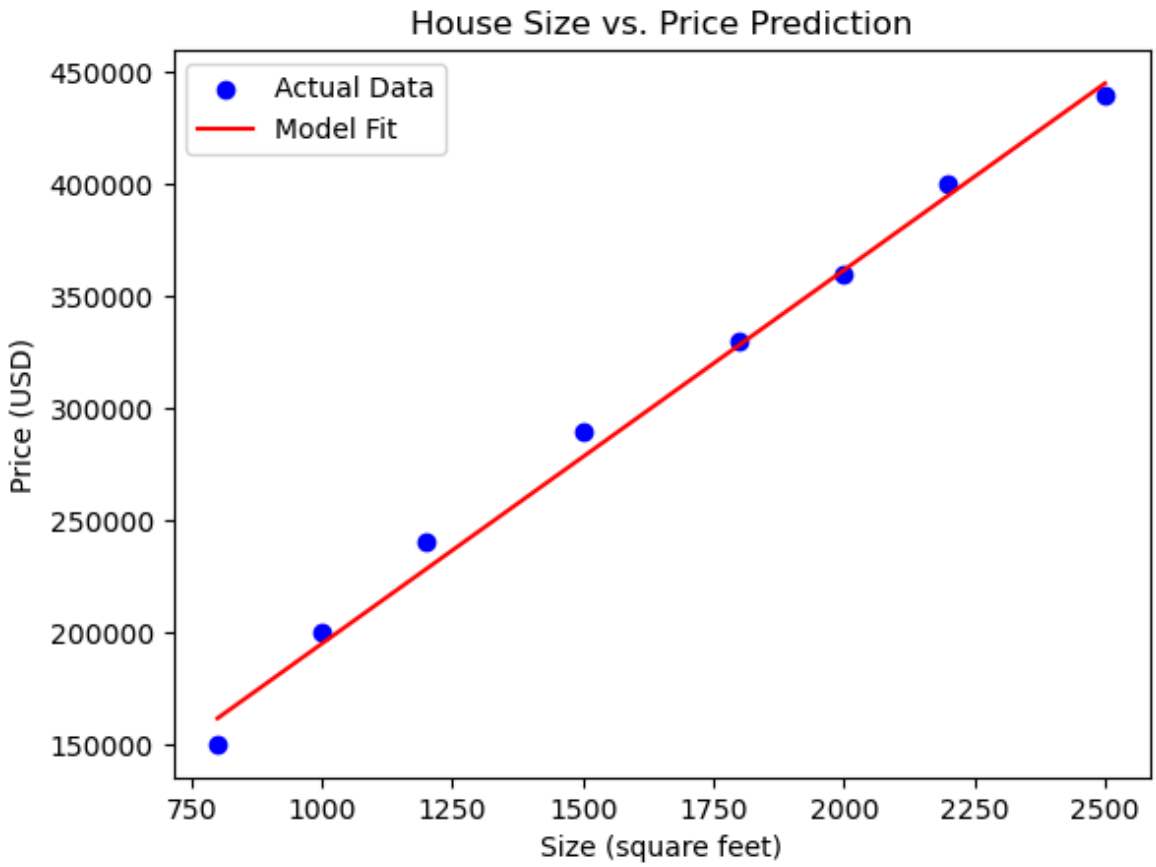
However, whether this MSE is "large" depends on the scale of your data. If the actual values in your dataset are very large (in the millions, for example), then this MSE might be reasonable. But if your data is on a smaller scale, this could indicate higher-than-desired error.

An R2 value close to 1 (like 0.987) suggests that the model explains about 98.7% of the variance in the data, which is generally considered very good.

High R2 values indicate a strong correlation between predicted and actual values.

In [14]:

```
# Step 7: Visualize the Results
plt.scatter(X, y, color='blue', label='Actual Data')            # Plot actual da
plt.plot(X, model.predict(X), color='red', label='Model Fit')    # Plot model's
plt.xlabel("Size (square feet)")
plt.ylabel("Price (USD)")
plt.title("House Size vs. Price Prediction")
plt.legend()
plt.show()
```



In [15]:
```
new_house_size = 1500
new_data_single = np.array([[new_house_size]])
```

In [16]:
```
predicted_price_single = model.predict(new_data_single)
predicted_price_single
```

```
c:\Users\devid\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature nam
es
  warnings.warn(
```

Out[16]:
```
array([278316.83168317])
```

In [ ]:
```
# multiple tests
new_data = np.array([[1100], [1700], [2100]])  # Example sizes for prediction
new_data
```

In [18]:
```
new_predictions = model.predict(new_data)       # Predict prices for new sizes
new_predictions
```

```
c:\Users\devid\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature nam
es
  warnings.warn(
```

In [21]:
```
print(f"Predicted price for a house of size {new_data[0][0]} sq ft: ${new_predic
```

```
Predicted price for a house of size 1100 sq ft: $211584158.42
```

In [22]:
```
# turn 2d array into a df
df= pd.DataFrame(new_data, columns=['Size'])
df
```

Out[22]:

	Size
0	1100
1	1700
2	2100

In [23]:
```
df['predictions']=new_predictions
```

In [24]:
```
df.to_csv('house_price_pred.csv', index= False)
```

another way of doing the same thing

In [ ]:
```

```

magic plot of matplotlib

In [ ]:
```
%matplotlib inline
plt.xlabel('area')
plt.ylabel('price')

plt.scatter(data.Size, data.Price, color='red', marker='+')
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant