Open in Colab YouTube Video Lecture محاظرة مسجلة

Introduction To Multiple Linear Regression

Topics and Outcomes

- Introduce Multiple Regression.

What is multiple Linear Regression?

Multiple Linear Regression is an **extension** of simple linear regression that allows for predicting a dependent variable based on multiple independent variables. The general form of the model is expressed as:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

Where:

- $y \in Y$ is the dependent variable (the value we aim to predict).
- β_0 is the intercept (constant term).
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for the independent variables.
- $X_1, X_2, \ldots, X_n$ are the independent variables (features that influence y).

The model gets the best regression fit line by finding the best values for β_0, β_1 and so on.

Process Steps:

1. Import the required Libraries
2. Import the Dataset
3. Scrub the Dataset, correct any errors in the data, make need transformations
4. Split the Dataset into training and testing datasets
5. Select a Machine Learning algorithm and configure its parameters
6. Fit/Train the Model
7. Make Predictions
8. Evaluate the results and the accuracy of the model

Python Example: Predicting Apartment Price based on Appartment Features

In this example, we will train a model to predict the **price** of an apartment based on its area size, number of rooms, age of the building, floor number.

Step 1: Import the Required Libraries

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

Step 2: Open the Dataset

We will use the house/apartment prices dataset to demonstrate multiple linear regression

In [2]:
```
df = pd.read_csv("https://raw.githubusercontent.com/msfasha/307304-Data-Mining/refs
df
```

Out[2]:

	Square_Area	Num_Rooms	Age_of_Building	Floor_Level	City	Price
0	162	1	15	12	Amman	74900.0
1	152	5	8	8	Aqaba	79720.0
2	74	3	2	8	Irbid	43200.0
3	166	1	3	18	Irbid	69800.0
4	131	3	14	15	Aqaba	63160.0
...	...	...	...	...	...	...
495	177	1	6	12	Irbid	64100.0
496	79	5	9	13	Irbid	52700.0
497	106	3	7	14	Aqaba	60160.0
498	108	3	9	18	Amman	72600.0
499	73	1	18	6	Aqaba	19280.0

500 rows × 6 columns

Step 3: Define Input Features and Target Column

We use **Square_Area**, **Num_Rooms**, **Age_of_Building** and **Floor_Level** as features and **Price** as the target.

In [3]:
```
# Features and Target
X = df[['Square_Area', 'Num_Rooms', 'Age_of_Building','Floor_Level']]  # Independen
y = df['Price']  # Dependent variable (Sales)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Step 4: Train the Model

In [4]:
```
# Initialize the Linear Regression model
model = LinearRegression()

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

# Coefficients and Intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [ 364.36926791 5064.2938487  -925.54218137  951.56065025]
Intercept: 461.2702038561838
```

Step 5: Make Predictions

In [5]:
```
# Predict the target variable for the test set
y_pred = model.predict(X_test)

# Display the predictions alongside the actual values
df = pd.DataFrame({'Actual': np.round(y_test,2), 'Predicted': np.round(y_pred,2),'R
df
```

Out[5]:

	Actual	Predicted	Residual
361	102550.0	89530.89	13019.11
73	54200.0	62947.21	8747.21
374	44000.0	44989.04	989.04
155	67000.0	77635.12	10635.12
104	63700.0	57098.21	6601.79
...	...	...	...
347	73360.0	75233.49	1873.49
86	36800.0	32110.44	4689.56
75	85500.0	75366.09	10133.91
438	89200.0	74516.43	14683.57
15	52960.0	53594.27	634.27

100 rows × 3 columns

Step 6: Evaluate the Model using R-Squared

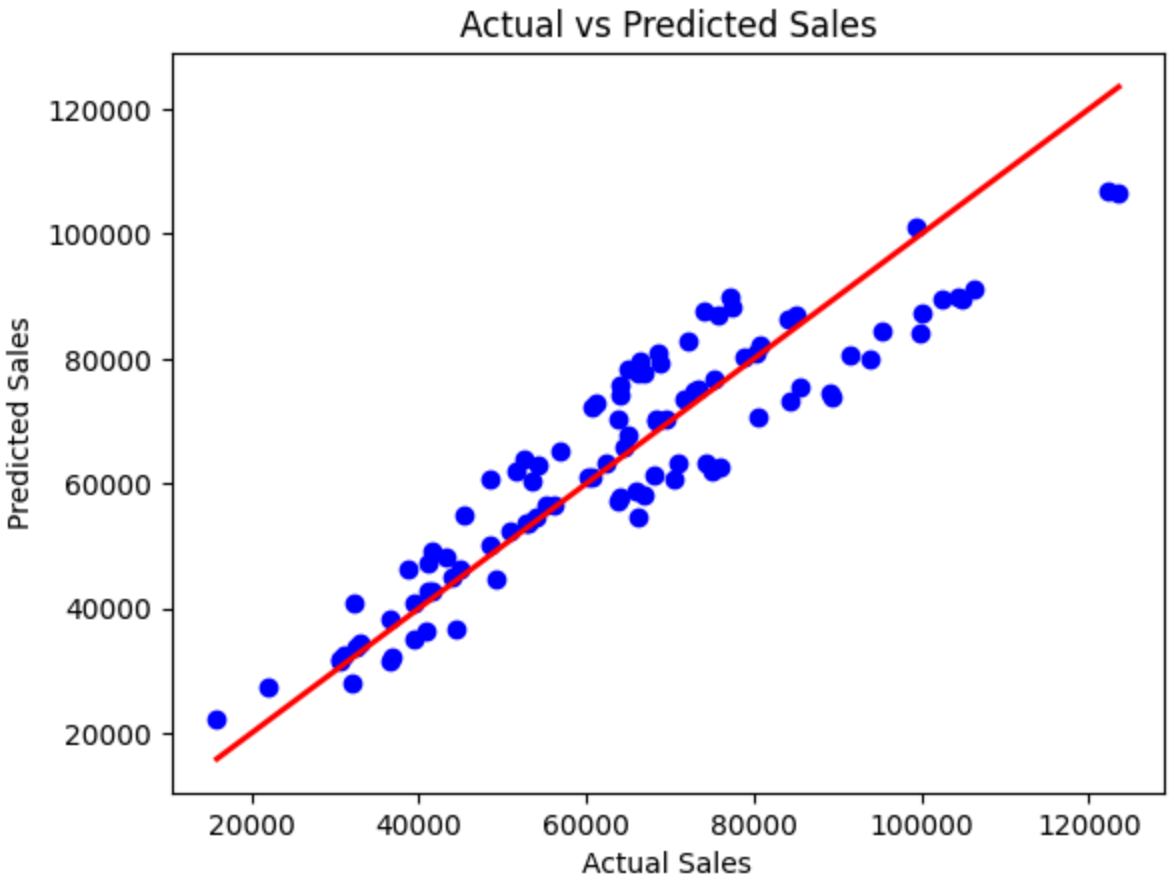
In [6]:
```
# R-squared (R²)
print("R-squared (R²):", round(model.score(X_test,y_test),2))
```

```
R-squared (R²): 0.84
```

Step 7: Plot Actual vs. Predicted Sales

We can visualize the predicted vs actual sales values to assess model performance.

In [7]:
```
# Plot actual vs predicted values
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linew
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.title('Actual vs Predicted Sales')
plt.show()
```



The Actual vs. Predicted plot shows how closely the model's predictions match the true sales values. Each point represents one observation, with actual values on the x-axis and predicted values on the y-axis. The red diagonal line represents perfect prediction, meaning a point on this line would have zero error.

In this plot, the blue points fall close to the red line, indicating that the model's predictions generally align well with the actual values. There is no obvious curvature or systematic pattern in the deviations, suggesting that the linearity assumption is reasonable. The spread of the points around the line appears moderate and fairly consistent across the range, which implies that the model does not exhibit strong bias or major heteroscedasticity.

Overall, the plot suggests that the multiple regression model is performing well, capturing the relationship between the predictors and the target variable with relatively low error. Numerical metrics such as R^2, RMSE, or MAE would further confirm this, but visually, the model fits the data reasonably accurately.

Check the Variance Heteroscedasticity

To check for heteroscedasticity (non-constant variance of residuals) in our linear regression model, we can create a residual plot. In this plot, we plot the residuals (differences between actual and predicted values) against the predicted values. If the residuals show a random

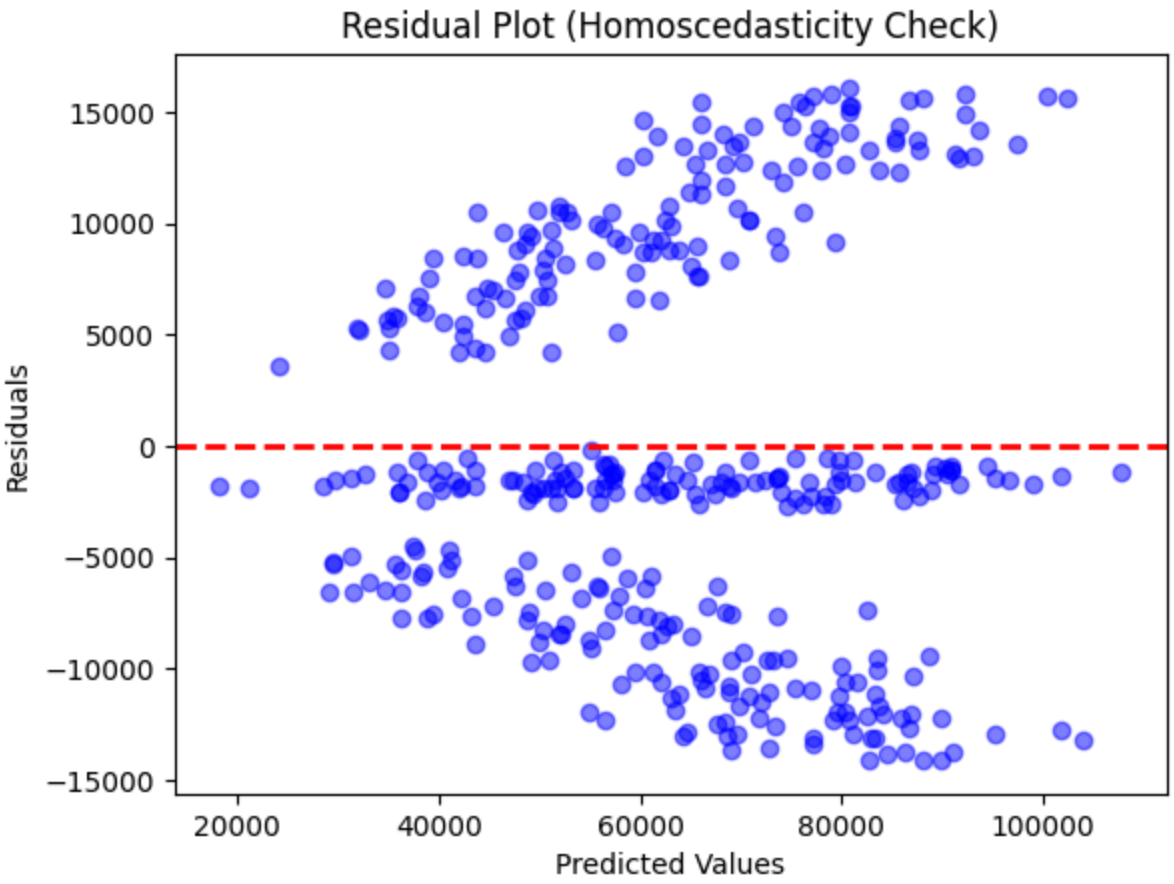
pattern with constant spread, it suggests homoscedasticity. However, if the spread of the residuals increases or decreases with the predicted values, it suggests heteroscedasticity.

In [8]:
```
import matplotlib.pyplot as plt
import numpy as np

# Predict the target variable for the training set
y_train_pred = model.predict(X_train)

# Calculate residuals
residuals = y_train - y_train_pred

# Plot residuals vs. predicted values
plt.scatter(y_train_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--', linewidth=2)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot (Homoscedasticity Check)")
plt.show()
```



From the residual plot, it appears that there is a **fanning-out pattern** in the residuals, where the variance of the residuals increases as the predicted values increase. This pattern is indicative of **heteroscedasticity**.

In the next lesson, we will try to enhance the performance by introducing more features.

Complete Code

In [9]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the dataset
df = pd.read_csv('https://raw.githubusercontent.com/msfasha/307304-Data-Mining/refs

# Features and Target
X = df[['Square_Area', 'Num_Rooms', 'Age_of_Building','Floor_Level']]  # Independen
y = df['Price']  # Dependent variable (Sales)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize the Linear Regression model
model = LinearRegression()

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

# Coefficients and Intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("R^2 Score:", round(model.score(X_test, y_test),2))  # Corrected
```

```
Coefficients: [ 364.36926791 5064.2938487  -925.54218137  951.56065025]
Intercept: 461.2702038561838
R^2 Score: 0.84
```