Introduction to Regression Analysis in Python

Topics and Outcomes

- Understand what is Regression.
- Main types of linear regression models.
- Introduce the concept of Best-Fit-Line, hypothesis function, OLS method and the cost function.
- Solving simple regressoin using the Closed Form Solution.
- Introduce the Gradient Descent algorithm for solving regression.
- Introduce the assumptions for linear regression.
- Introduce the evaluation metrics for regression.
- Introduce Scikit-Learn library in Python.
- Solving simple linear regression using scitkit-learn.

What is Regressions Analysis?

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to observed data.

When there is only one independent feature, it is known as Simple Linear Regression, and when there are more than one feature, it is known as Multiple Linear Regression.

Why Linear Regression is Important:

1. **Interpretability**: Linear regression provides clear, interpretable coefficients that explain the relationship between independent and dependent variables.
2. **Simplicity**: It is transparent, easy to implement, and serves as a foundational concept for more complex algorithms.
3. **Basis for Advanced Models**: Many advanced techniques (e.g., regularization, support vector machines) are extensions or adaptations of linear regression.
4. **Assumption Testing**: It helps in validating key assumptions about the data, making it crucial in statistical analysis.

Types of Linear Regression

There are two main types of linear regression:

1- Simple Linear Regression

This is the simplest form of linear regression, and it involves only one independent variable and one dependent variable. The equation for simple linear regression is:

$$y = \beta_0 + \beta_1 X$$

where:

- (y) is the dependent variable
- (X) is the independent variable
- β_0 is the intercept
- β_1 is the slope

2- Multiple Linear Regression (Simplified)

In multiple linear regression, there is one dependent variable and more than one independent variable. The equation is:

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

Where:

- y is the dependent variable (what we want to predict)
- $X_1, X_2, \ldots, X_n$ are the independent variables (inputs)
- $beta_0$ is the intercept
- $beta_1, \beta_2, \ldots, \beta_n$ are the slopes (coefficients)

The goal is to find the **best-fit line** that predicts y based on the independent variables $X_1, X_2, \ldots, X_n$.

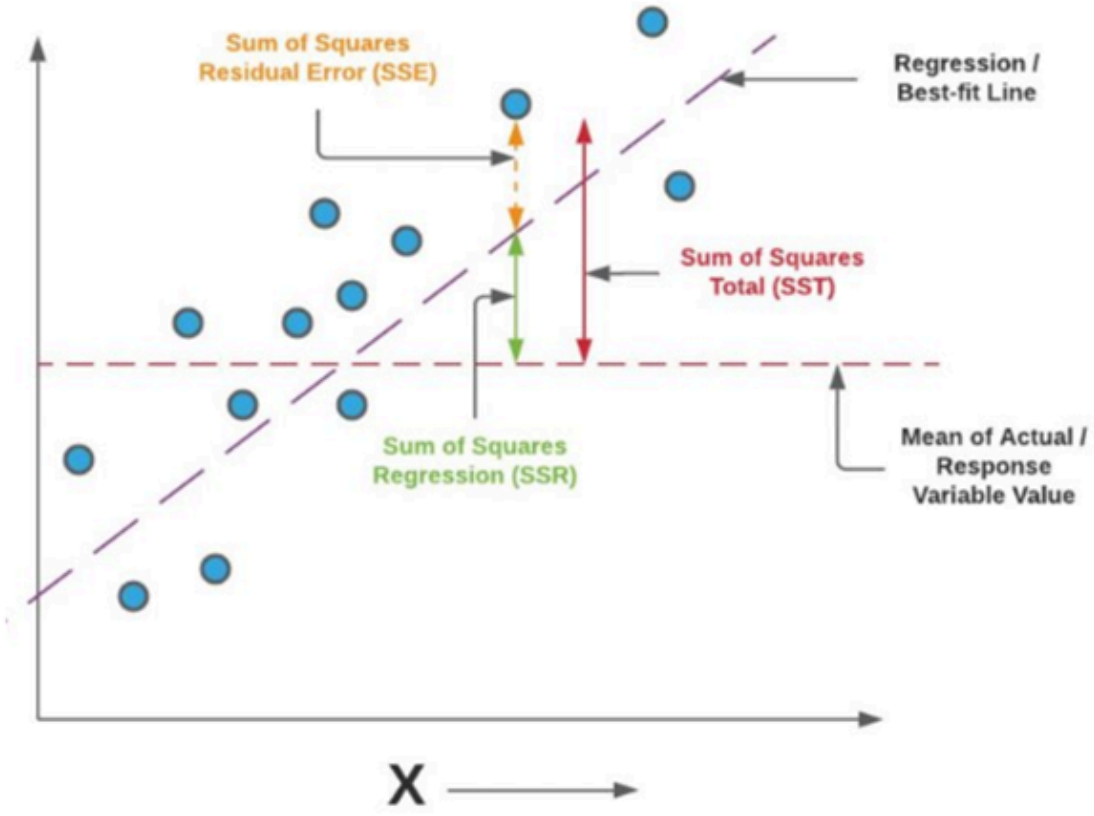
This learned function can then be used to predict y for new X values. The function predicts continuous outcomes based on the input features.

What is the Best Fit Line?

In linear regression, the **best-fit line** is the line that minimizes the error between the predicted and actual values. It represents the relationship between the dependent and independent variables.

The slope of the line shows how much the dependent variable changes for each unit change in the independent variable(s). The goal is to find this line to make the most accurate predictions.

$$R^2 = \frac{\text{Variance explained by the model}}{\text{Total variance}}$$



In regression, Y is the **dependent** variable, and X is the **independent** variable, also called the **predictor**.

Linear regression predicts Y based on X, using a straight line to model the relationship. For example, X could be work experience, and Y could be salary. The goal is to find the **best-fit line** that minimizes the error between predicted and actual values.

To find this line, we use a **cost function**, which helps determine the best values for the line's coefficients.

Hypothesis Function in Linear Regression

In linear regression, the hypothesis function predicts the dependent variable Y (e.g., salary) based on the independent variable X (e.g., experience):

$\hat{Y} = \beta_0 + \beta_1 X$ Where:

- β_0 is the intercept.
- β_1 is the slope.

The goal is to find the best β_0 and β_1 values to get the best-fit line for predicting Y.

OLS Method:

Ordinary Least Squares (OLS) is one of the foundational methods in statistics and machine learning for estimating the parameters of a linear model.
In essence, OLS seeks to find the line (or hyperplane, in the case of multiple predictors) that best fits a set of data points by minimizing the sum of the squared differences between observed values and the predicted values from the model.
This method is widely used in fields such as economics, finance, engineering, and natural sciences because of its simplicity, interpretability, and effectiveness for many types of data.

Cost Function

The **Mean Squared Error (MSE)** is the cost function used to measure the error between the predicted $\hat{Y}$ and the actual Y:

$J = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$

Solving Linear Regression with the Closed-Form Solution

In linear regression, the goal is to find the best-fitting line that predicts the dependent variable (y) from one or more independent variables (X). The relationship between X and y can be expressed as:

$$y = \beta_0 + \beta_1 X + \epsilon$$

Where:

- X is the independent variable (e.g., Years of Experience),
- y is the dependent variable (e.g., Salary),
- β_0 is the intercept (the value of y when $X = 0$),
- β_1 is the slope (the change in y for a one-unit change in X),
- ϵ is the error term.

Formula for Coefficients in Simple Linear Regression

1. **Slope (β_1)** is given by:

$$\beta_1 = \frac{\text{Cov}(X, y)}{\text{Var}(X)}$$

Where:

- $\text{Cov}(X, y)$ is the covariance between X and y,
- $\text{Var}(X)$ is the variance of X.

2. **Intercept (β_0)** is calculated as:

$$\beta_0 = \bar{y} - \beta_1 \bar{X}$$

Where:

- $\bar{y}$ is the mean of the dependent variable y,
- $\bar{X}$ is the mean of the independent variable X.

Python Code to Compute the Coefficients Using This Method

Below is the Python code that computes the slope and intercept using the covariance-variance method.

In [65]:
```
import numpy as np
import pandas as pd

# Data: YearsExperience and Salary
data = {
    "YearsExperience": [1.1, 1.3, 1.5, 2.0, 2.2, 2.9, 3.0, 3.2, 3.2, 3.7, 3.
    "Salary": [39343, 46205, 37731, 43525, 39891, 56642, 60150, 54445, 64445
}

# Create a pandas DataFrame
df = pd.DataFrame(data)

# Extract the independent (X) and dependent (y) variables
X = df["YearsExperience"]
y = df["Salary"]

# Calculate the means of X and y
X_mean = np.mean(X)
y_mean = np.mean(y)

# Calculate the covariance of X and y and the variance of X
cov_X_y = np.sum((X - X_mean) * (y - y_mean))  # Covariance
var_X = np.sum((X - X_mean) ** 2)              # Variance

# Calculate the slope (beta_1) and intercept (beta_0)
slope = cov_X_y / var_X
intercept = y_mean - slope * X_mean

# Print the slope and intercept
print(f"Slope (beta_1): {round(slope,2)}")
```

```
print(f"Intercept (beta_0): {round(intercept,2)}")


# Using the model to predict new values
years_of_experience = 15.5
predicted_salary = slope * years_of_experience + intercept
print(f"The expected/predicted salary of the new employee is: {round(predict
```

```
Slope (beta_1): 9449.96
Intercept (beta_0): 25792.2
The expected/predicted salary of the new employee is: 172266.62
```

When can we use the Closed-Form Solution?

The closed-form solution is efficient for small to medium-sized datasets, but it becomes computationally expensive for large datasets because it requires calculating the inverse of a matrix. It works well when:

- The dataset is not very large (typically up to thousands of rows).
- The number of features (independent variables) is small.
- There are no multicollinearity issues (i.e., features are not highly correlated).

For larger datasets or more complex models, iterative methods like **Gradient Descent** are preferred.

Solving Linear Regression using Gradient Descent Method

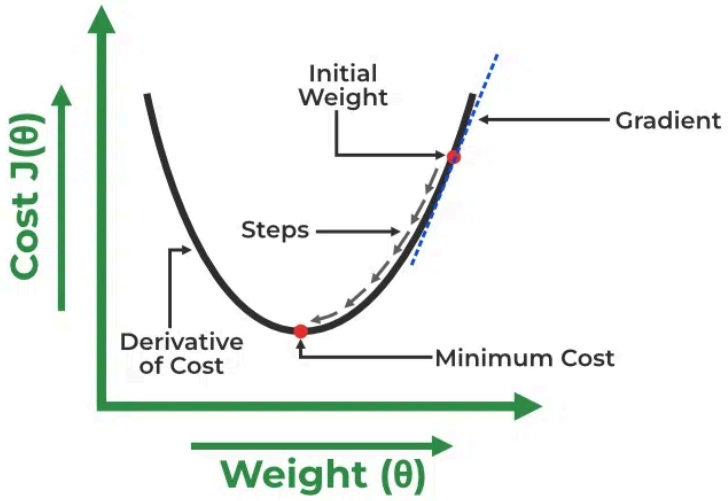
This specific topic (Gradient Descent) is a read only - not included in the exams

هذا الموضوع تحديدا للاطلاع والعلم - غير مطلوب في الامتحان

A linear regression model can be trained using **gradient descent**, which adjusts the model's parameters to minimize the mean squared error (MSE).

To update β_0 and β_1 and reduce the cost function (minimizing the RMSE), gradient descent starts with random values for β_0 and β_1 and iteratively improves them to find the best-fit line.

A gradient is simply the derivative, showing how small changes in inputs affect the output. By moving in the direction of the Mean Squared Error negative gradient with respect to the coefficients, the coefficients can be changed.



Computing β_0 Rate of Change:

$$\beta_0 = \beta_0 - \alpha \left(J'_{\beta_0} \right)$$

$$= \beta_0 - \alpha \left(\frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \right)$$

Computing β_1 Rate of Change:

$$\beta_1 = \beta_1 - \alpha \left(J'_{\beta_1} \right)$$

$$= \beta_1 - \alpha \left(\frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \cdot x_i \right)$$

Evaluation Metrics for Linear Regression

A variety of evaluation measures can be used to determine the strength of any linear regression model. These assessment metrics often give an indication of how well the model is producing the observed outputs.

The most common measurements are:

1. **Mean Squared Error (MSE)**
   Measures the average squared difference between actual and predicted values:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(y_i^{\text{actual}} - y_i^{\text{predicted}} \right)^2$$

2. **Root Mean Squared Error (RMSE)**
   The square root of MSE, providing the error in the same units as the target variable:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i^{\text{actual}} - y_i^{\text{predicted}}\right)^2}$$

3. **R-squared (Coefficient of Determination)**

   The proportion of variance in the dependent variable explained by the independent variables:

$$R^2 = 1 - \frac{SS_{\text{residual}}}{SS_{\text{total}}}$$

   Where:

$$SS_{\text{residual}} = \sum_{i=1}^{n}\left(y_i^{\text{actual}} - y_i^{\text{predicted}}\right)^2$$

$$SS_{\text{total}} = \sum_{i=1}^{n}\left(y_i^{\text{actual}} - \bar{y}\right)^2$$

   Where p is the number of predictors.

In these formulas:

- y_i^{actual} represents the actual observed values,
- $y_i^{\text{predicted}}$ represents the predicted values from the model,
- $\bar{y}$ is the mean of the actual values,
- n is the number of data points, and
- p is the number of estimated parameters in the model.

Solving Regression using Scikit-Learn Library in Python

What is scikit-learn?

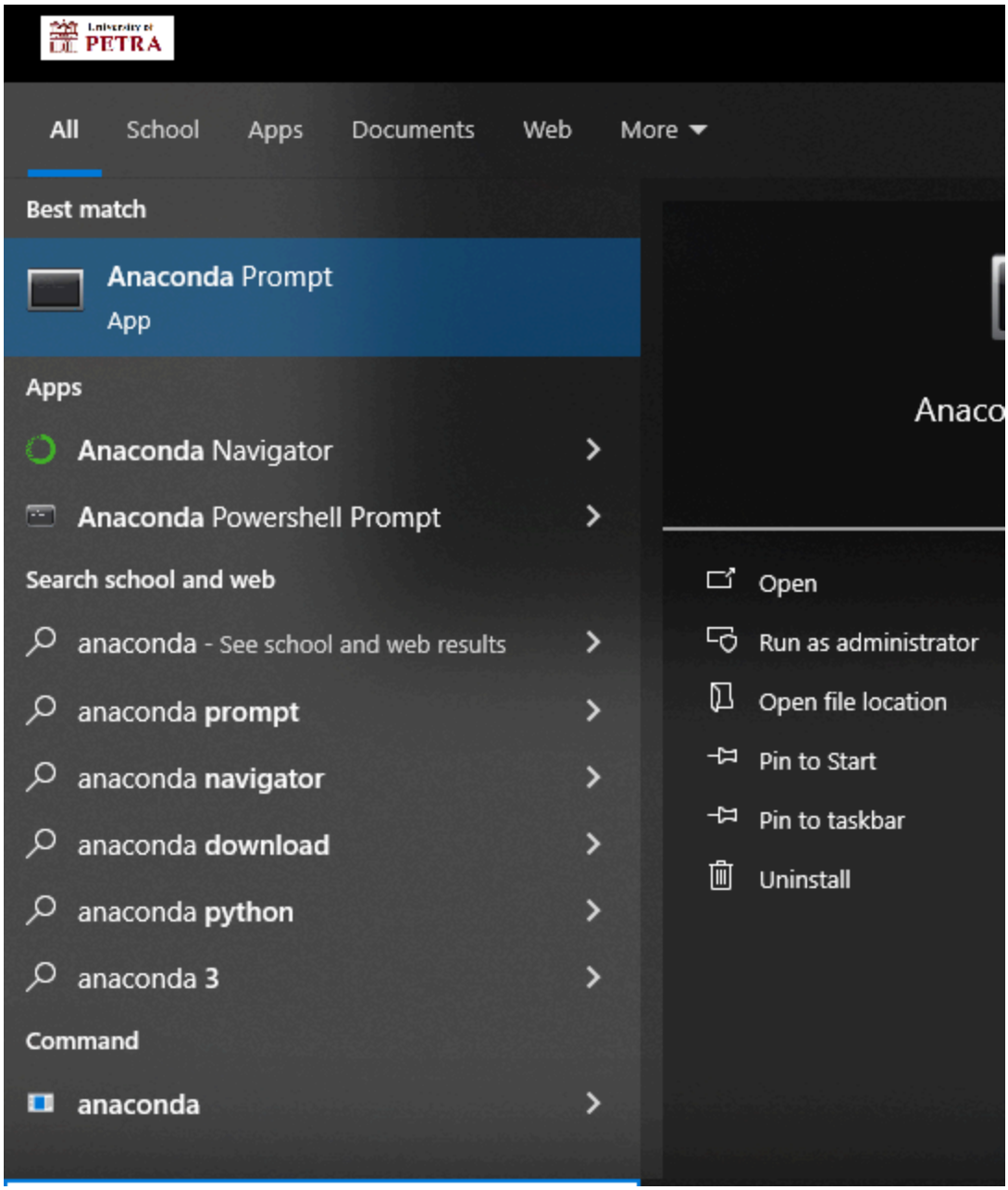
scikit-learn is a very popular tool, and the most prominent Python library for machine learning. It is widely used in industry and academia, and a wealth of tutorials and code snippets are available online.

- scikit-learn is an open source project, meaning that it is free to use and distribute.
- This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.
- The scikit-learn project is constantly being developed and improved, and it has a very active user community.
- It contains a number of state-of-the-art machine learning algorithms, as well as comprehensive documentation about each algorithm.
- Scikit-learn provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a **consistence interface** in Python.

Scikit Learn

Prepare Python environment

1. Install Anaconda Navigator
2. Open Anaconda prompt



3. Execute the following commands in the command line enviroment:

pip install numpy, pandas, matplotlib, scikit-learn

Practical Example - Predicting Apartment Price based on Area

Here we are taking a dataset that has two variables: Price (dependent variable) and Square_Area (Independent variable). The goals of this problem is:

- We want to find out if there is any correlation between these two variables.
- We will find the best fit line for the dataset.
- How the dependent variable is changing by changing the independent variable.

We will create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.

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

In [66]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# import the required machine learning libraries
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

df = pd.read_csv('../datasets/apartment_prices.csv')
print(f"Number of rows in the dataset: {len(df)}")
df.head(3)
```

```
Number of rows in the dataset: 500
```

Out[66]:

	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

Prepare the dataset

Now we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary.

In [67]:
```
x = df.iloc[:, 0].values.reshape(-1, 1)  # Reshape x to be a 2D array
y = df.iloc[:, -1].values  # y can remain as a 1D array

# x = df["Square_Area"]
# y= df["Price"]
```

For x variable, We used -1 value to remove the last column from the dataset.

For y variable, we use 1 to extract the second column (dataframe indexing starts from the

zero).

Split the dataset into training and testing segments

In [68]:
```
# Splitting the dataset into training and test set.
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= .1, rand
```

Create the Machine Learning model and make any required configurations

In [69]:
```
regressor= LinearRegression()
```

Train the model using the training dataset

In [70]:
```
regressor.fit(x_train, y_train)

print(regressor.coef_)
print(regressor.intercept_)
```

```
[362.33639889]
15802.319376714753
```

In the above code, we used the fit() method to fit\train our Simple Linear Regression model using the training set.

We passed the x_train and y_train datasets to the fit() function, which are our training datasets for the dependent and an independent variable.

Make Predictions

Now, our model is ready to predict the output for the new observations.

In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector test_predictions, and train_predictions, which will contain predictions of the test dataset, and prediction of the training set respectively.

In [71]:
```
#Prediction of Test and Training set result
train_predictions= regressor.predict(x_train)
test_predictions= regressor.predict(x_test)
```

On executing the above lines of code, two variables named train_predictions and test_predictions will be generated which contains salary predictions for the training set and the test set.

Evaluate Model Performance

Visualizing the Training set results:

Now in this step, we will visualize the training set result.

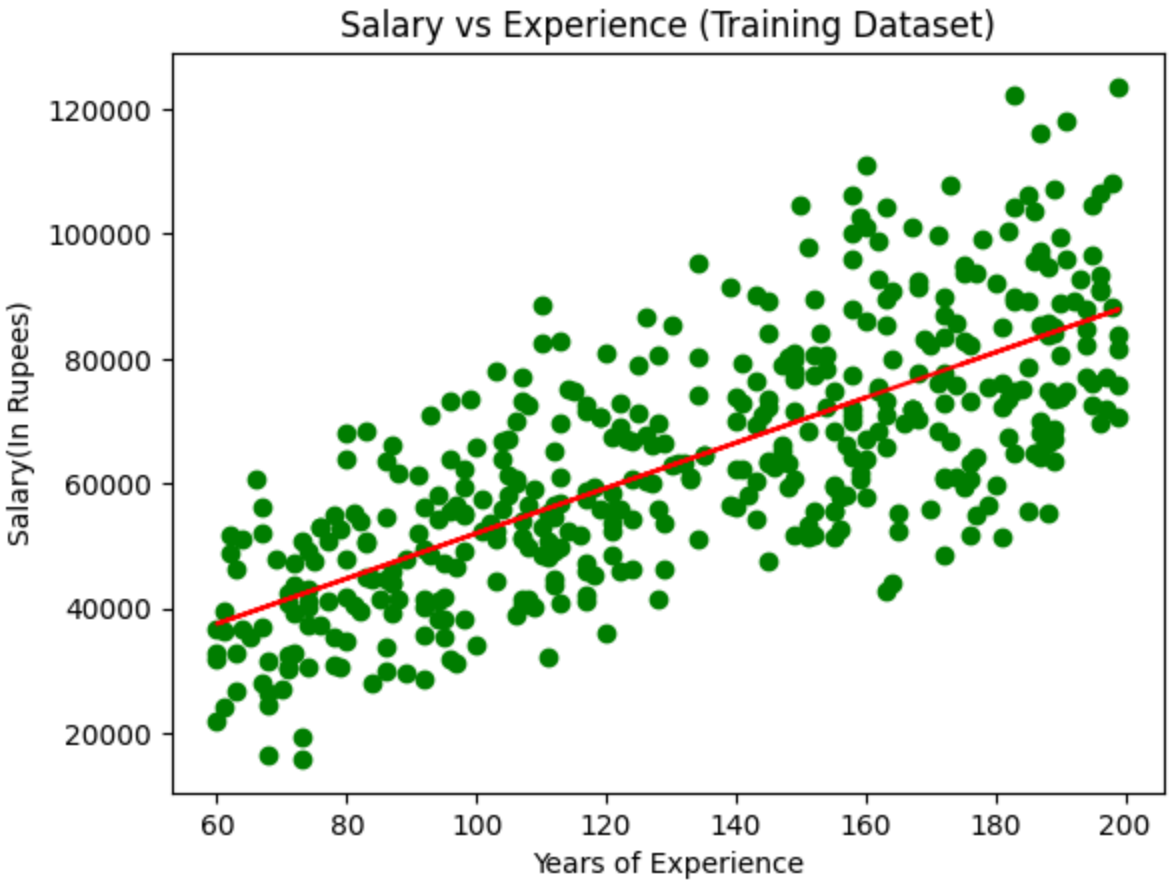
To do so, we will use the scatter() function of the pyplot library, which we have already imported in the pre-processing step. The scatter () function will create a scatter plot of observations.

In the x-axis, we will plot the Years of Experience of employees and on the y-axis, we will plot the salary of the employees.

In the function, we will pass the real values of the training set, which means is the year of experience i.e. x_train, and the depended variable which is the salaries i.e. y_train. We will use a green color for the observation.

We also will plot the regression line using the plot() function of the pyplot library. In this function, we will pass the years of experience for the training set, and the predicted salary for the training set training_predictions.

In [72]:
```
# plot the training data with the original dependent variable
plt.scatter(x_train, y_train, color="green")

# plot the training data with the predictions
plt.plot(x_train, train_predictions, color="red")

plt.title("Salary vs Experience (Training Dataset)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary(In Rupees)")
plt.show()
```



In [73]:
```
r2 = r2_score(y_train, train_predictions)
r2 = round(r2,2)
print("r2 score: ", r2)
```

```
r2 score:  0.54
```

In the above plot, we can see the real values for the observations in green dots and the predicted values are covered by the red regression line. The regression line shows a correlation between the dependent and independent variable.

The good fit of the line can be observed by calculating the difference between the actual values and the predicted values. As we can see in the above plot, most of the observations are close to the regression line, hence our model is good for the training set.

Examining the model on the Test dataset

In the previous step, we have visualized the performance of our model on the training set.

Now, we will do the same for the Test dataset. The complete code will remain the same as the above code, except now we will use x_test, and testing_predictions instead of x_train and y_train.

In [74]:
```
# Plot the original values in the testing set (x_test and y_test)
plt.scatter(x_test, y_test, color="blue")
```

```
# Plot the x_test with the predicted test data
plt.plot(x_test, test_predictions, color="red")

plt.title("Salary vs Experience (Test Dataset)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary(In Rupees)")
plt.show()
```

In the above plot, there are observations given by the blue color, and prediction is given by the red regression line. As we can see, most of the observations are close to the regression line, hence we can say our Simple Linear Regression is a good model and able to make good predictions.

But we can compute the R^2 (R-Squared) which provides us with a systematic numerical value for the accuracy of our model.

In [75]:
```
from sklearn.metrics import r2_score

r2 = round(r2_score(y_test, test_predictions),2)
print("r2 score: ", r2)
```

```
r2 score:  0.55
```

Complete Model Code

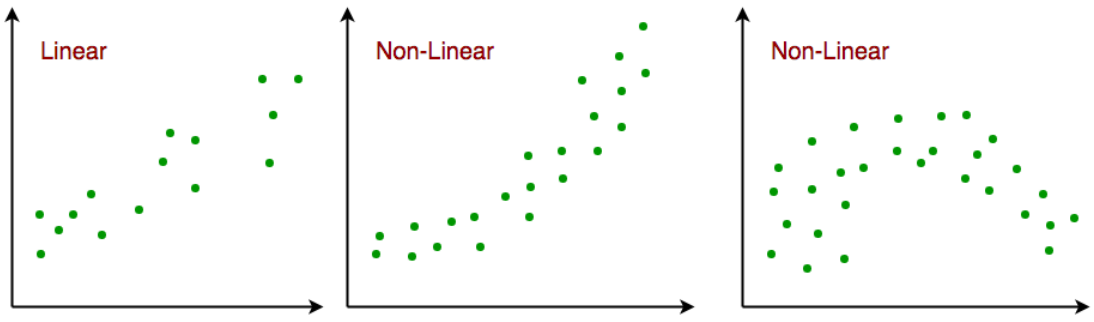
In [76]:
```
df = pd.read_csv('../datasets/apartment_prices.csv')

x = df.iloc[:, 0].values.reshape(-1, 1)  # Reshape x to be a 2D array
y = df.iloc[:, -1].values  # y can remain as a 1D array

# Splitting the dataset into training and test set.
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= .1, rand

# Create the LinearRegression Model
regressor= LinearRegression()

# Train the model
regressor.fit(x_train, y_train)

# Print the results
print("Study Hours Coefficient: ", regressor.coef_)
print("Intercept Value: ", regressor.intercept_)
print("r2 score: ", round(r2_score(y_train, train_predictions),2))
```

```
Study Hours Coefficient:  [362.33639889]
Intercept Value:  15802.319376714753
r2 score:  0.54
```

Assumptions of Simple Linear Regression

Linear regression is a powerful tool for understanding and predicting the behavior of a variable, however, it needs to meet a few conditions in order to be accurate and dependable solutions.

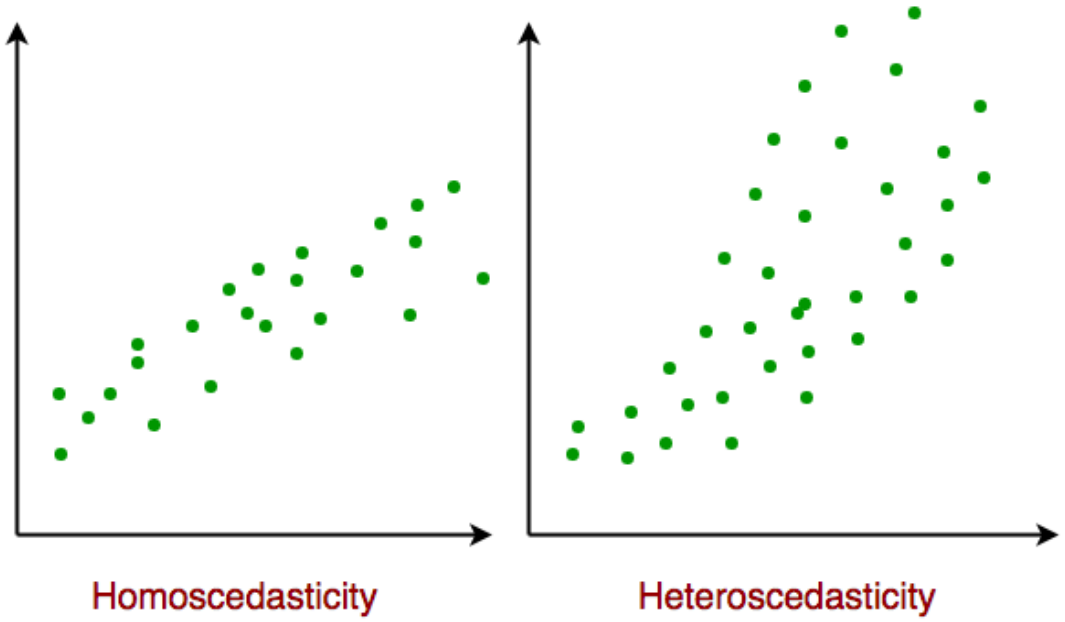
1. Linearity: The independent and dependent variables have a linear relationship with one another. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion. This means that there should be a straight line that can be drawn through the data points. If the relationship is not linear, then linear regression will not be an accurate model.



2. **Independence**: The observations in the dataset must be independent, meaning the value of the dependent variable for one observation should not be influenced by the

value of another. If this condition is violated, linear regression may produce inaccurate results.

3. **Homoscedasticity**: The variance of the errors should be constant across all levels of the independent variable(s). This means that changes in the independent variable(s) should not affect the spread of the errors. If the error variance is not constant (heteroscedasticity), the linear regression model may not be reliable.



4. **Normality**: The residuals should be normally distributed. This means that the residuals should follow a bell-shaped curve. If the residuals are not normally distributed, then linear regression will not be an accurate model.

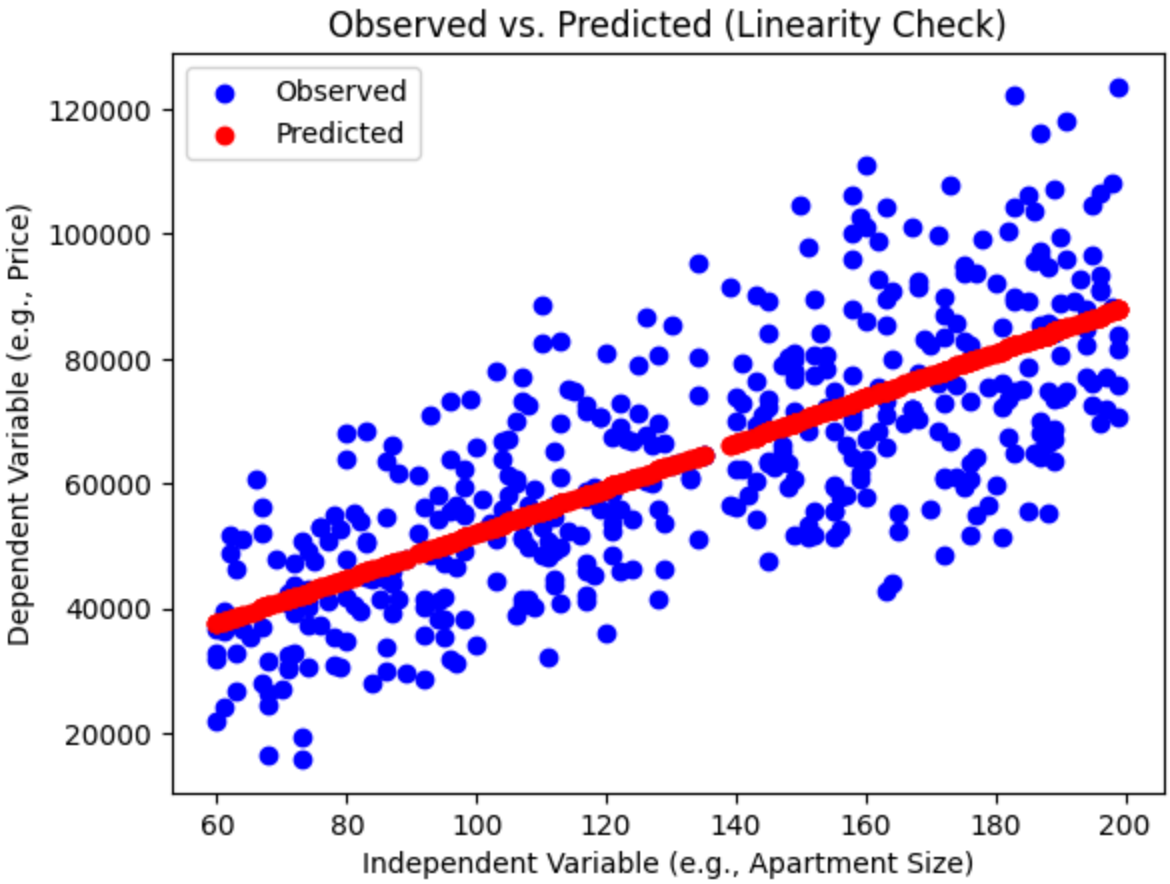
5. **No Multicollinearity**: The independent variables should not be highly correlated with each other. Multicollinearity occurs when two or more independent variables are strongly related, making it difficult to isolate the effect of each variable on the dependent variable. If multicollinearity exists, it can lead to inaccurate results in multiple linear regression.

1. Linearity Check

A scatter plot of the observed vs. predicted values can help us check if there's a linear relationship.

In [77]:
```
# Predictions on the training set
train_predictions = regressor.predict(x_train)

# Plot observed vs. predicted values
```

```
plt.scatter(x_train, y_train, color='blue', label='Observed')
plt.scatter(x_train, train_predictions, color='red', label='Predicted')
plt.xlabel("Independent Variable (e.g., Apartment Size)")
plt.ylabel("Dependent Variable (e.g., Price)")
plt.title("Observed vs. Predicted (Linearity Check)")
plt.legend()
plt.show()
```



If the points form a roughly linear pattern, the linearity assumption is reasonable.

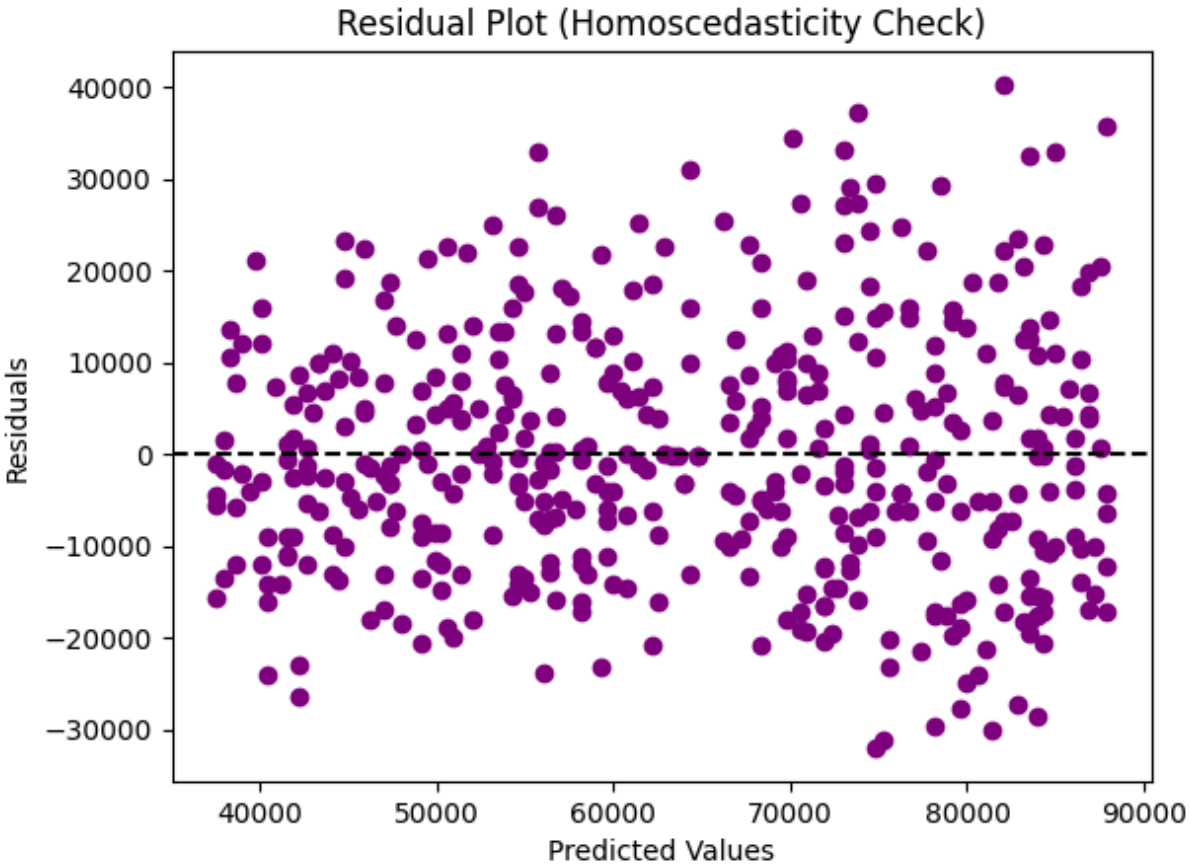
2. Independence of Observations

In typical regression problems, we assume that observations are independent of each other. Testing independence is more challenging without time series data or grouped data but can often be assumed if the data sampling is done correctly. If you're working with time series, you'd look at autocorrelation plots instead.

3. Homoscedasticity Check

We can create a residual plot (plotting residuals vs. predicted values). If the residuals show no clear pattern (they're randomly scattered around zero), then the homoscedasticity assumption holds.

In [78]:
```
# Calculate residuals
residuals = y_train - train_predictions

# Plot residuals vs. predicted values
plt.scatter(train_predictions, residuals, color='purple')
plt.axhline(y=0, color='black', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot (Homoscedasticity Check)")
plt.show()
```



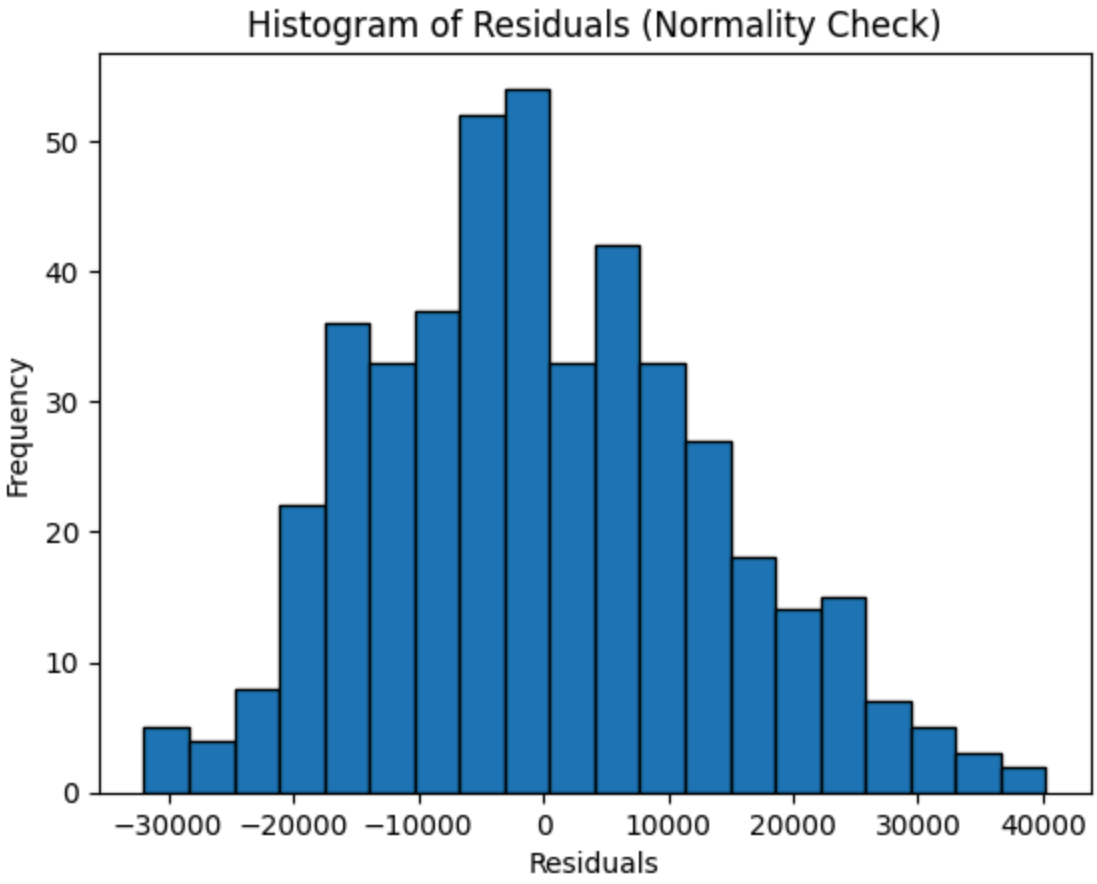
If residuals are randomly scattered around zero with no visible pattern, this suggests homoscedasticity. Patterns (such as a funnel shape) would indicate heteroscedasticity

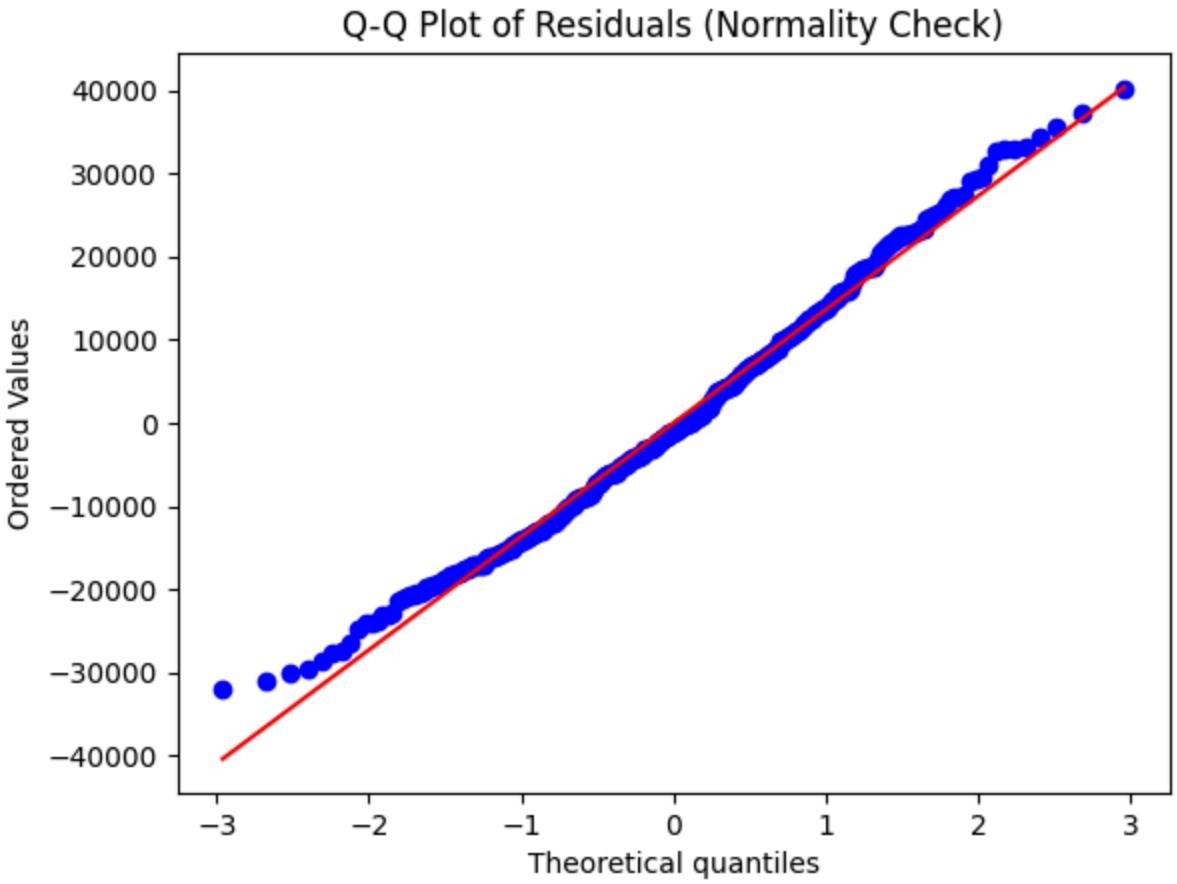
4. Normality of Residuals

To check if the residuals are normally distributed, we can use a histogram and a Q-Q plot. Alternatively, the Shapiro-Wilk test or Kolmogorov-Smirnov test can be used.

In [79]:
```
import scipy.stats as stats

# Histogram of residuals
plt.hist(residuals, bins=20, edgecolor='black')
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Histogram of Residuals (Normality Check)")
```

```
plt.show()

# Q-Q plot
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals (Normality Check)")
plt.show()
```





If the residuals are approximately normally distributed, the histogram should look bell-shaped, and the points in the Q-Q plot should roughly follow a straight line.

5. Multicollinearity Check (Only for Multiple Regression)

For multiple linear regression, multicollinearity among independent variables can be assessed using the Variance Inflation Factor (VIF). A VIF value greater than 10 indicates high multicollinearity.

Since we're using simple linear regression, this doesn't apply here. However, if we expand to multiple regression, you can calculate VIF as follows:

In [80]:
```
""" from statsmodels.stats.outliers_influence import variance_inflation_fact
import statsmodels.api as sm

# Assuming `X` contains all predictors for multiple regression
X = sm.add_constant(df.iloc[:, :-1])  # Add constant term for intercept
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.s
print(vif_data)
"""
```

Out[80]:
```
' from statsmodels.stats.outliers_influence import variance_inflation_facto
r\nimport statsmodels.api as sm\n\n# Assuming `X` contains all predictors f
or multiple regression\nX = sm.add_constant(df.iloc[:, :-1])  # Add constan
t term for intercept\nvif_data = pd.DataFrame()\nvif_data["feature"] = X.co
lumns\nvif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in r
ange(X.shape[1])]\nprint(vif_data)\n'
```

References

- https://www.geeksforgeeks.org/ml-linear-regression/
- https://www.javatpoint.com/simple-linear-regression-in-machine-learning