Data Analysis with Python

Analyzing data with Python is an essential skill for Data Scientists and Data Analysts.
- collecting and importing data
- cleaning, preparing & formatting data
- data frame manipulation
- summarizing data
- building machine learning regression models
- model refinement
- creating data pipelines

Learning how to import data from multiple sources, clean and wrangle data, perform exploratory data analysis (EDA), and create meaningful data visualizations. Then predict future trends from data by developing linear, multiple, polynomial regression models & pipelines and learn how to evaluate them.

M1: Importing Data Sets

1.1 Understanding the Data

https://archive.ics.uci.edu/dataset/10/automobile

Before starting any data analysis, it is important to understand the dataset. We will work with a dataset on used car prices. This dataset, originally compiled by Jeffrey C. Schlimmer, is provided in CSV format (Comma-Separated Values).

Goal of Analysis

The main objective is to predict the price of a used car based on its features.

- The target variable is Price.
- The predictor variables are all other columns like engine size, body style, fuel type, etc.

1.2 Python Packages for Data Science

Python libraries for **data analysis** can be grouped into three main categories:

1. **Scientific Computing Libraries:** These help with **data manipulation and mathematical operations.**

- **Pandas**: Provides tools to work with **structured data** like tables, making it easier to manipulate and analyze data. It uses **DataFrames**, which are similar to tables in databases or spreadsheets.
- **NumPy**: Uses **arrays** to store data efficiently. It supports **fast numerical operations** and matrix manipulations.
- **SciPy**: Includes advanced mathematical functions such as **optimization, integration, and linear algebra**.

2. **Data Visualization Libraries:** These allow us to **create graphs, charts, and maps** to represent data visually.
    - **Matplotlib**: The most commonly used visualization library. It helps create **bar charts, line graphs, histograms, and scatter plots**. The graphs are also **customizable**, allowing us to change labels, colors, and styles.
    - **Seaborn**: A high-level visualization library built on **Matplotlib**. It makes it easy to create **heat maps, time series plots, violin plots, and statistical visualizations**. It also provides built-in **themes and color schemes** to improve the aesthetics of plots.
3. **Machine Learning Libraries:** These contain **algorithms** that help in building predictive models from data.
    - **Scikit-learn**: A widely used library that includes tools for **classification, regression, clustering, and dimensionality reduction**. It is built on top of **NumPy, SciPy, and Matplotlib**.
    - **Statsmodels**: Allows users to perform **statistical analysis, hypothesis testing, and regression modeling**. It provides advanced functions for **exploring data and estimating models**.

Using these Python libraries makes data analysis **efficient and accurate**. By leveraging these tools, we can perform **data wrangling, visualization, and machine learning** seamlessly.

1.3 Importing and Exporting Data in Python

1.4 Getting Started Analyzing Data in Python

1.5 Accessing Databases with Python(1.3,1.4,1.5 in github.ipynb)

https://github.com/Priyanka504595/Skills-certificates/blob/master/Data%20analysis%20using%20Python/Data%20Analysis%20with%20Python-Part1.ipynb

M2: Data Wrangling

Data wrangling is the process of cleaning and transforming raw data into a structured format that is ready for analysis. This step is crucial because real-world data is often messy, inconsistent, or incomplete.
Handling missing values in data, formatting data to standardize it and make it consistent, normalizing data, grouping data values into bins, and converting categorical variables into numerical quantitative variables.

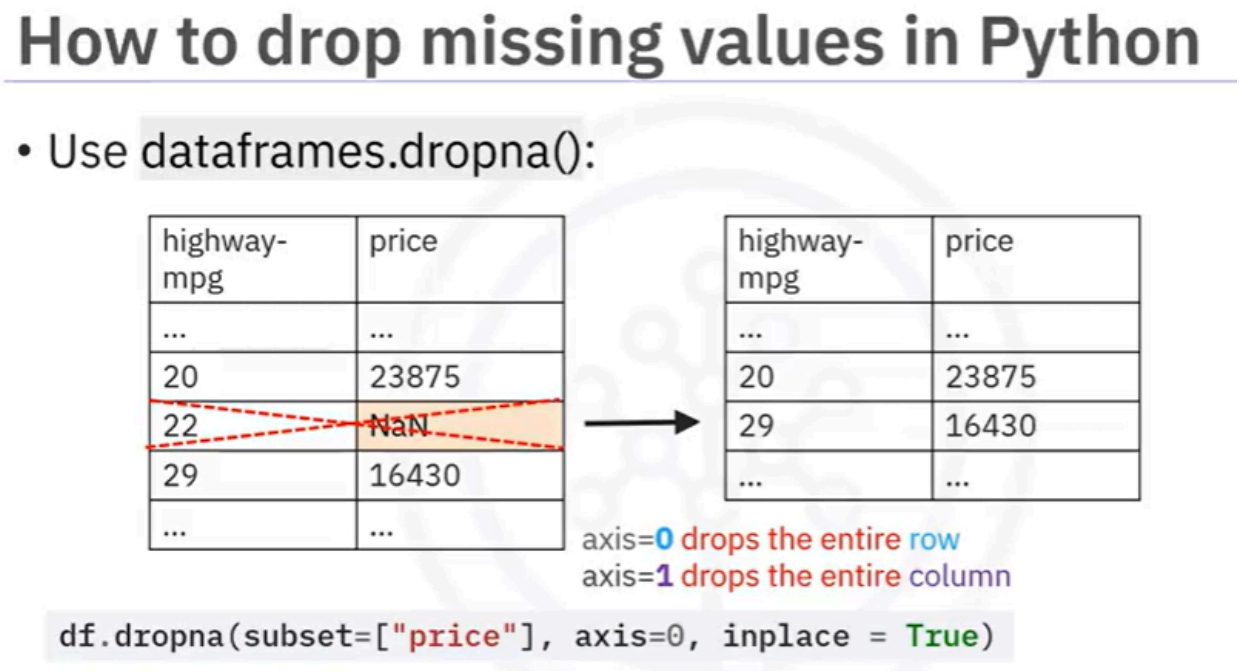
2.1. Pre-processing Data

Before conducting any analysis, raw data must be processed to ensure it is clean and structured. Pre-processing involves multiple steps, including handling missing values, correcting inconsistent formats, normalizing numerical values, binning data into categories, and converting categorical variables into numerical values.

2.2. Handling Missing Values

Missing values occur when some data points are not recorded in the dataset. These can appear as blank spaces, "NaN" (Not a Number), or placeholders like "N/A" or "?".
Ways to Handle Missing Values:

1. **Removing Missing Data:**
    - If only a few rows contain missing values, they can be dropped without affecting the overall analysis.
    - If an entire column has too many missing values, it may be removed.
    - In Python, the dropna() function removes missing values from the dataset.



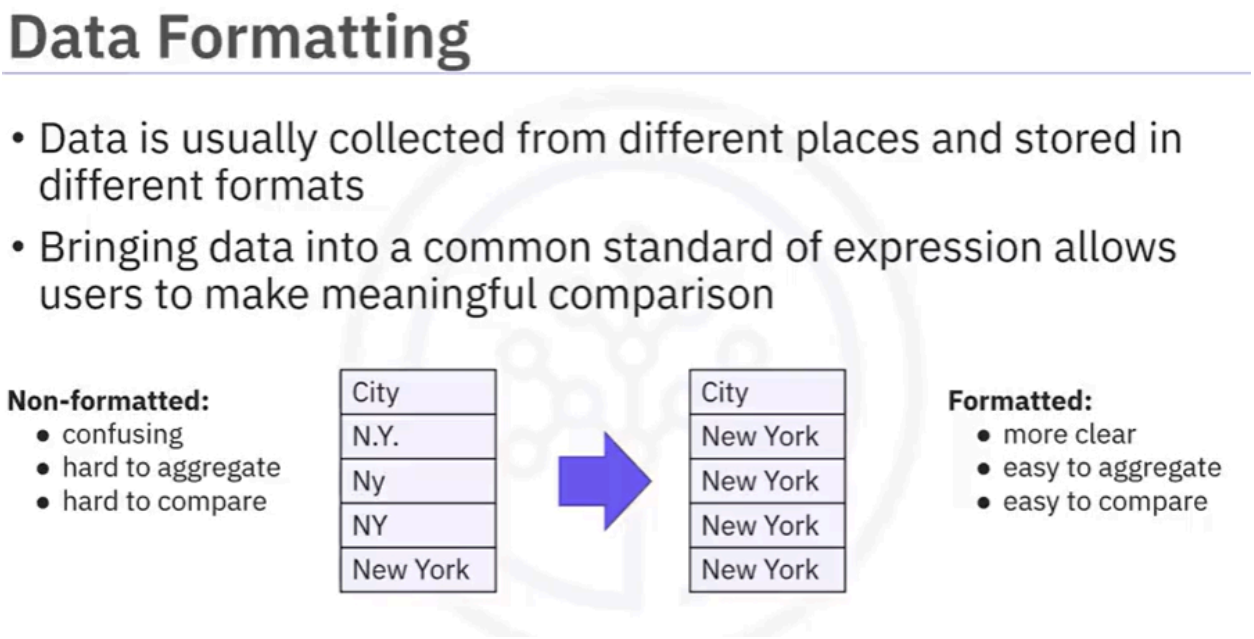
2. **Replacing Missing Data:**
   Instead of removing data, missing values can be replaced using appropriate techniques:
   - **For numerical data:** Replace missing values with the mean, median, or mode of the column.
   - **For categorical data:** Replace missing values with the most frequent category (mode).
   - The fillna() function in Pandas helps replace missing values.

2.3. Data Formatting

Data collected from different sources may have inconsistent formats, making it difficult to analyze. Formatting data ensures uniformity across all entries.

Common Formatting Issues:

- **Different representations of the same value:** Example: "New York", "NY", and "N.Y." should be standardized.



- **Inconsistent units:** Example: Fuel efficiency may be recorded in miles per gallon (mpg) in one dataset and liters per 100 kilometers (L/100km) in another.
- **Incorrect data types:** Example: A price column may be mistakenly stored as text instead of a numerical value.

Fixing Formatting Issues in Python:

- Convert inconsistent text to a single format using replace() or .str.lower().
- Convert numerical units using mathematical operations.
- Correct data types using .astype() function in Pandas.

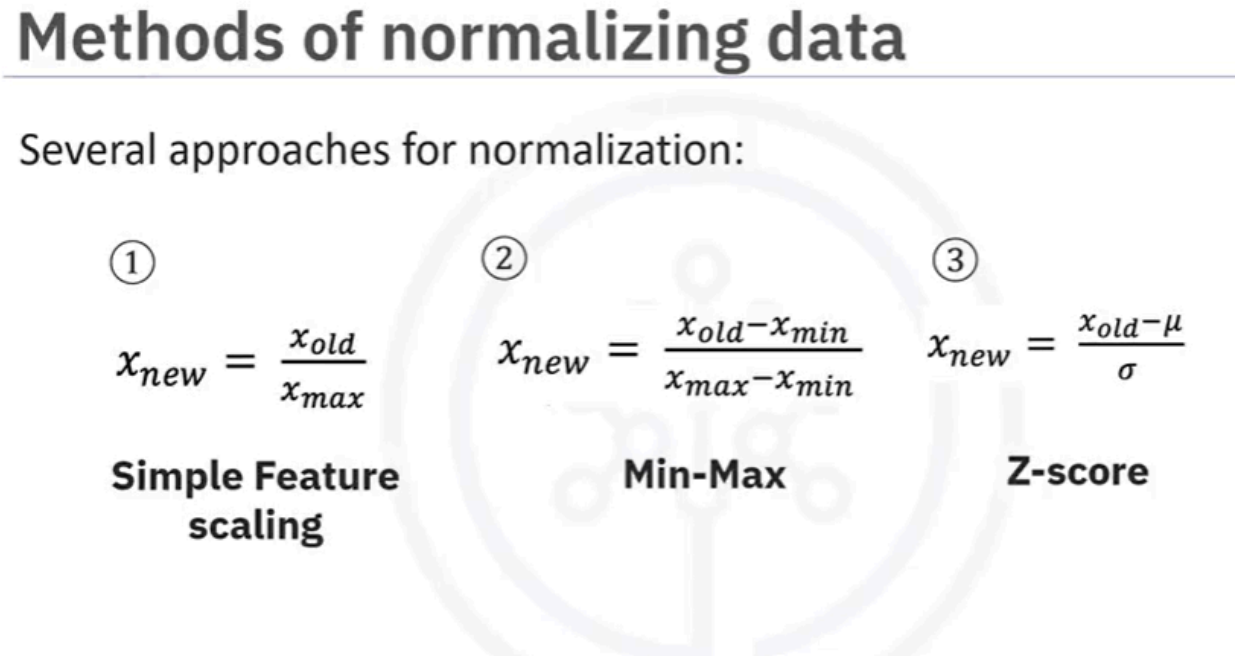
2.4. Data Normalization

Normalization adjusts numerical values to a consistent scale, making comparisons more meaningful. Without normalization, features with larger values may dominate the analysis.

Common Normalization Techniques:

1. **Simple Feature Scaling:**
   - Divides each value by the maximum value in the column.
   - Formula: X_new = X / X_max
   - Scales all values between 0 and 1.
2. **Min-Max Scaling:**
   - Adjusts values to a specific range (e.g., 0 to 1).
   - Formula: X_new = (X - X_min) / (X_max - X_min)
3. **Z-score Normalization:**
   - Standardizes data by centering it around the mean and scaling it based on standard deviation.
   - Formula: X_new = (X - mean) / standard deviation

- Most values will be between -3 and 3.



2.5. Binning (Grouping Numerical Data into Categories)

Binning is used to convert continuous numerical values into groups or categories. This helps simplify the analysis and detect trends more easily.

Example: Binning Car Prices

- If car prices range from **$5,000 to $45,000**, they can be divided into:
  - **Low Price:** $5,000 - $15,000
  - **Medium Price:** $15,001 - $30,000
  - **High Price:** $30,001 - $45,000

How to Perform Binning in Python:

- Use the **numpy.linspace()** function to create equal intervals.
- Assign categories using the **pandas.cut()** function.
- Visualize the bins using a histogram.

2.6. Converting Categorical Variables into Numerical Variables

Machine learning models cannot process categorical (text-based) data directly. They require numerical input, so categorical values must be converted into numbers.

Techniques for Converting Categorical Data:

1. **One-Hot Encoding:**
   - Creates a new binary column for each category.
   - Example: If "Fuel Type" has values "Gas" and "Diesel," two columns are created:

        - **Gas:** 1 if the car uses gas, 0 otherwise.
        - **Diesel:** 1 if the car uses diesel, 0 otherwise.
    - Done using pd.get_dummies() function in Pandas.
2. **Label Encoding:**
    - Assigns a unique number to each category.
    - Example: "Low", "Medium", and "High" values can be encoded as **0, 1, and 2** respectively.
    - Done using the LabelEncoder() function from Scikit-learn.

M3: Exploratory Data Analysis

In this module, you will learn what is meant by exploratory data analysis, and you will learn how to perform computations on the data to calculate basic descriptive statistical information, such as mean, median, mode, and quartile values, and use that information to better understand the distribution of the data. You will learn about putting your data into groups to help you visualize the data better, you will learn how to use the Pearson correlation method to compare two continuous numerical variables, and you will learn how to use the Chi-square test to find the association between two categorical variables and how to interpret them.
Exploratory Data Analysis (EDA) is an important step in understanding and preparing data for analysis. It involves summarizing data, identifying patterns, detecting relationships between variables, and visualizing trends. EDA helps data analysts and scientists make informed decisions before applying machine learning models or statistical techniques.

- Exploratory Data Analysis
- Descriptive Statistics
- GroupBy in Python
- Creating Different Types of Plots in Python
- Correlation and Correlation - Statistics
- Chi-Square Test for Categorical Variables

3.1: Understanding Exploratory Data Analysis (EDA)

EDA is the process of analyzing datasets to summarize their main characteristics. It helps in:

- Understanding the structure and distribution of data.
- Identifying missing values, outliers, and inconsistencies.
- Detecting relationships between variables.
- Preparing data for further processing and modeling.

EDA provides insights into how the data behaves and whether it requires transformation or cleaning before analysis.

3.2: Descriptive Statistics

Descriptive statistics summarize data using key measures that help in understanding the dataset. These measures include:

1. Measures of Central Tendency

- **Mean:** The average of all values.
- **Median:** The middle value when the data is sorted.
- **Mode:** The most frequently occurring value.

These measures provide insight into the typical value in a dataset.

2. Measures of Dispersion

- **Range:** The difference between the highest and lowest values.
- **Variance:** A measure of how much the values deviate from the mean.
- **Standard Deviation:** The square root of variance, showing how spread out the data is.

These measures help in understanding how much the values in a dataset vary.

3.3: Grouping Data (GroupBy)

Grouping data is a technique used to organize values based on a specific category. This is useful for summarizing statistics for different groups in a dataset. For example:

- Comparing average prices of different car types.
- Analyzing sales trends by region or customer category.

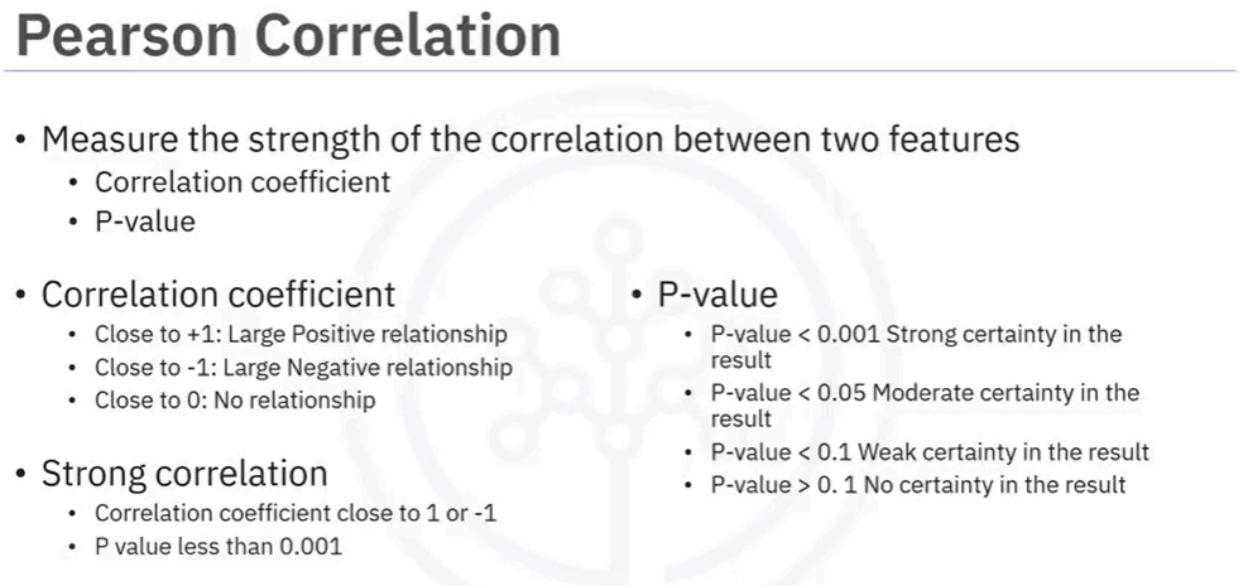
Grouping data allows analysts to compare trends across different segments and make data-driven decisions.

3.4: Correlation and Relationships Between Variables

Correlation measures the strength of the relationship between two numerical variables. Understanding correlation helps in identifying dependencies between different attributes.

1. Pearson Correlation

- A positive correlation means that as one variable increases, the other also increases.
- A negative correlation means that as one variable increases, the other decreases.
- A correlation close to zero indicates no significant relationship between the variables.



2. Correlation Heatmap

A heatmap visually represents the correlation between multiple variables. It provides an overview of how different attributes in a dataset relate to one another, making it easier to identify strong relationships.

3.5: Data Visualization for EDA

Data visualization is one of the most effective ways to understand patterns, distributions, and relationships within a dataset. Some commonly used visualization techniques include:

1. Histogram (Distribution of Data)

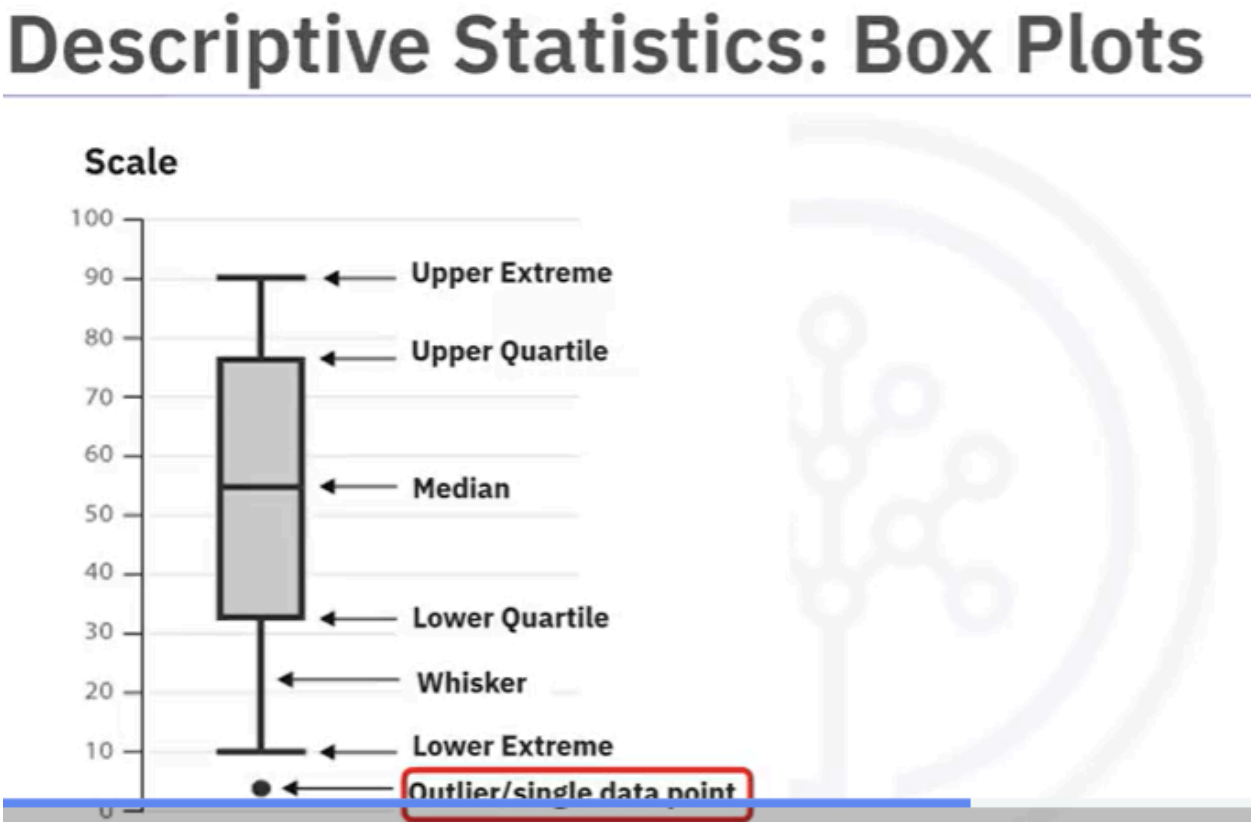
A histogram helps visualize how values are distributed in a dataset. It provides insights into the frequency of different values and highlights patterns such as skewness or unusual peaks.

2. Scatter Plot (Relationship Between Two Variables)

A scatter plot shows how two numerical variables are related. It helps identify trends and correlations, such as whether an increase in horsepower leads to an increase in car price.

3. Box Plot (Detecting Outliers)

A box plot is useful for identifying outliers in a dataset. It provides a summary of data distribution, including the median, quartiles, and extreme values.



M4: Model Development

In this module, you will learn how to define the explanatory variable and the response variable and understand the differences between the simple linear regression and multiple linear regression models. You will learn how to evaluate a model using visualization and learn about polynomial regression and pipelines. You will also learn how to interpret and use the R-squared and the mean square error measures to perform in-sample evaluations to numerically evaluate our model. And lastly, you will learn about prediction and decision making when determining if our model is correct.
Model development is the process of building mathematical equations that predict a target variable based on one or more input variables. In this module, we focus on different types of regression models used to predict car prices. The key topics covered include:

- **Simple and Multiple Linear Regression**
- **Model Evaluation using Visualization**
- **Polynomial Regression and Pipelines**
- **In-Sample Evaluation (R^2 and Mean Squared Error)**
- **Prediction and Decision Making**

4.1. Understanding Model Development

A **model** is a mathematical equation that defines the relationship between a dependent variable (target) and one or more independent variables (predictors). A good model should be able to:

- Capture meaningful patterns in the data.
- Make accurate predictions.
- Generalize well to new data.

For example, in our dataset, the goal is to predict **car price** based on features such as **horsepower, fuel type, engine size, and mileage**.

4.2. Simple and Multiple Linear Regression

Simple Linear Regression (SLR)

Simple Linear Regression models the relationship between **one independent variable (X)** and **one dependent variable (Y)** using a straight line equation:

Y=b0+b1XY = b_0 + b_1X

Where:

- $\mathbf{b_0}$ = Intercept (starting value of Y when X=0).
- $\mathbf{b_1}$ = Slope (rate at which Y changes with X).

For example, if **highway miles per gallon** is the predictor and **price** is the target, we can estimate price based on fuel efficiency.

Multiple Linear Regression (MLR)

When there are **multiple independent variables**, we use **Multiple Linear Regression**, which extends the equation:

Y=b0+b1X1+b2X2+...+bnXnY = b_0 + b_1X_1 + b_2X_2 + ... + b_nX_n

Here, $\mathbf{X_1}$, $\mathbf{X_2}$, ... Xn are the features (e.g., horsepower, weight, engine size), and $\mathbf{b_1}$, $\mathbf{b_2}$, ... bn are their respective coefficients.

Why Use Multiple Variables?

- A single variable may not provide enough information.
- Multiple features improve model accuracy.
- Helps understand how different factors influence the target variable.

4.3. Model Evaluation using Visualization

Visualizing a model helps us understand its accuracy and limitations. **Regression plots** and **residual plots** are common techniques.

Regression Plots

- Show the **relationship between predicted values and actual values**.
- Help determine **how well the model fits the data**.

Residual Plots

- Show **errors (residuals) between predicted and actual values**.
- If residuals are randomly distributed, the model is valid.
- If there is a pattern, it indicates a poor model fit.

A **good regression model** should:

- Have residuals evenly distributed around zero.
- Show **no clear patterns** in the residual plot.
- Predict values that are close to actual values.

4.4. Polynomial Regression and Pipelines

Linear regression assumes a straight-line relationship, but sometimes the data follows a **curved pattern**. **Polynomial Regression** helps in such cases.

Polynomial Regression

- Extends linear regression by adding **higher-degree terms**.
- Helps model **non-linear relationships**.

For example:

- **2nd-degree (Quadratic)** regression: Y=b0+b1X+b2X2Y = b_0 + b_1X + b_2X^2
- **3rd-degree (Cubic)** regression: Y=b0+b1X+b2X2+b3X3Y = b_0 + b_1X + b_2X^2 + b_3X^3

Using Pipelines

When building a complex model, we need to perform multiple steps, like:

1. **Transforming features** (e.g., creating polynomial terms).
2. **Scaling data** (normalization).
3. **Fitting a regression model**.

Pipelines automate these steps in sequence, making code more efficient and readable.

4.5. In-Sample Evaluation: R^2 and Mean Squared Error

Once a model is built, we need to evaluate how well it fits the training data.

Mean Squared Error (MSE)

- Measures the **average squared difference** between actual and predicted values.
- A **lower MSE** indicates a better model.

R^2 (Coefficient of Determination)

- Measures **how much of the variation in Y is explained by X**.
- R^2 values range from **0 to 1**:
  - **$R^2 = 1$** → Perfect model.
  - **$R^2 = 0.8$** → 80% of variations in Y are explained by X.
  - **R^2 close to 0** → Poor model.

A **good model** should have:

- **High R^2** (closer to 1).

- **Low MSE** (smaller errors).

4.6. Prediction and Decision Making

After evaluating the model, we use it to **make predictions**.

Key Considerations for Predictions

- **Check if the prediction makes sense** (e.g., car price should not be negative).
- **Ensure the model follows trends** (e.g., higher horsepower should lead to higher price).
- **Watch for overfitting** (if the model is too complex, it might not generalize well to new data).

Limitations of the Model

- If a model predicts **unrealistic values**, it may indicate a **poor fit or missing variables**.
- A **simple linear model** may not capture all relationships, requiring **polynomial or multiple regression**.

M5: Model Evaluation and Refinement

In this module, you will learn about the importance of model evaluation and discuss different data model refinement techniques. You will learn about model selection and how to identify overfitting and underfitting in a predictive model. You will also learn about using Ridge Regression to regularize and reduce standard errors to prevent overfitting a regression model and how to use the Grid Search method to tune the hyperparameters of an estimator.

Model evaluation and refinement are crucial steps in building a reliable predictive model. It ensures that the model generalizes well to new data and performs accurately. In this module, we cover:

- **Training and Testing Data Splitting**
- **Cross-Validation**
- **Overfitting and Underfitting**
- **Ridge Regression for Regularization**
- **Grid Search for Hyperparameter Tuning**

These techniques help improve model performance and prevent errors.

5.1. Model Evaluation

Why Evaluate a Model?

After developing a model, we need to check how well it performs on new data. If a model is evaluated only on the data used to train it, the performance might be **overestimated**.

Training vs. Testing Data

- **Training Data**: Used to train the model.

- **Testing Data**: Used to evaluate the model's performance on unseen data.

A common approach is to **split the dataset** into:

- **70% training data**
- **30% testing data**

This ensures the model can make accurate predictions on new data.

5.2. Cross-Validation

Why Cross-Validation?

Splitting data into training and testing sets gives only one estimate of model performance. **Cross-validation** improves this by splitting data multiple times.

How Cross-Validation Works

- The dataset is divided into **k equal parts (folds)**.
- The model is trained on **k-1 folds** and tested on the remaining fold.
- This process repeats **k times**, each time using a different fold as the test set.
- The final model performance is the **average of all iterations**.

This method helps reduce **bias and variance** and ensures **model reliability**.

5.3. Overfitting, Underfitting, and Model Selection

What is Overfitting?

- The model **fits the training data too well**, capturing **noise** instead of real patterns.
- **High accuracy on training data but poor performance on test data.**
- Happens when the model is **too complex** (e.g., high-degree polynomial regression).

What is Underfitting?

- The model **fails to capture patterns in the data**.
- **Low accuracy on both training and test data.**
- Happens when the model is **too simple** (e.g., using a straight line for complex data).

Finding the Right Balance

To choose the best model:

- Start with a **simple model** (linear regression).
- Increase complexity **gradually** (polynomial regression).
- Use **cross-validation** to measure model accuracy.
- Select the model with the **lowest test error**.

5.4. Ridge Regression

What is Ridge Regression?

Ridge Regression helps prevent **overfitting** by adding a **penalty (alpha)** to large coefficients. This helps control **complexity** while maintaining predictive power.

How Ridge Regression Works

- Regular linear regression may produce **very large coefficients**, leading to overfitting.
- Ridge Regression **shrinks coefficients** to avoid extreme values.
- The **alpha parameter** controls regularization strength:
    - **Low alpha** → Less penalty → Model behaves like regular regression.
    - **High alpha** → More penalty → Model generalizes better.

Choosing the Best Alpha

- **Cross-validation** helps find the **optimal alpha**.
- The goal is to **minimize error on test data**.

Ridge Regression is useful when **data has many correlated features**.

5.5. Grid Search for Hyperparameter Tuning

What is Grid Search?

Grid Search **automatically tests different hyperparameters** (like alpha in Ridge Regression) to find the best values.

How Grid Search Works

1. **Define a set of hyperparameter values**.
2. **Train multiple models**, each using different hyperparameter values.
3. **Evaluate each model** using cross-validation.
4. **Select the best-performing model**.

For example, in Ridge Regression, Grid Search tests **different alpha values** and selects the best one.

This method **saves time** and ensures we use the **best parameters** for our model.