**Machine Learning (ML)** is teaching computers to learn from data and make predictions.
It uses **statistics** and **Python** to analyze data sets and create prediction models.

**Data Examples:**

- Array: `[99,86,87,88,...]`
- Table (e.g., car info: brand, color, speed, etc.)

**Goal:** Analyze data to predict outcomes (e.g., if a car has AutoPass).

**Data Types:**

- **Numerical:** Numbers
  - *Discrete:* Countable (e.g., number of cars)
  - *Continuous:* Measurable (e.g., price)
- **Categorical:** Non-numeric (e.g., color, yes/no)
- **Ordinal:** Ranked categories (e.g., grades A > B)

Understanding data types helps choose the right analysis method.

**Mean, Median, Mode in ML**

- **Mean**: Average value
  → `numpy.mean()`
- **Median**: Middle value (after sorting)
  → `numpy.median()`
- **Mode**: Most frequent value
  → `scipy.stats.mode()`

**Example List**:
`[99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]`

- Mean = 89.77
- Median = 87
- Mode = 86

These stats help analyze data in Machine Learning.

## Standard Deviation & Variance (ML Basics)

- **Standard Deviation (σ)**:
  Tells how spread out values are.
  → `numpy.std()`
- **Variance ($\sigma^2$)**:
  Square of standard deviation.
  → `numpy.var()`

---

**Examples:**

- `speed = [86,87,88,86,87,85,86]`
  → Mean = 86.4, **Std Dev = 0.9** (values close to mean)
- `speed = [32,111,138,28,59,77,97]`
  → Mean = 77.4, **Std Dev = 37.85** (values widely spread)

---

**Formulas:**

- **Variance** = Average of squared differences from mean
- **Standard Deviation** = √Variance

## Percentiles (ML Basics)

- **Percentile**: Value below which a given % of data falls.
  → `numpy.percentile(data, percent)`

---

**Example:**

```
python
CopyEdit
import numpy as np

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

np.percentile(ages, 75)  # Output: 43
np.percentile(ages, 90)  # Output: 61.0
```

---

**Meaning**:

- 75% are aged **43 or younger**
- 90% are aged **61 or younger**

## Data Distribution (ML Basics)

- Use **NumPy** to create **random datasets**:

  ```
  python
  CopyEdit
  np.random.uniform(0.0, 5.0, 250)
  ```

- Use **Matplotlib** to **visualize with histograms**:

  ```
  python
  CopyEdit
  import matplotlib.pyplot as plt
  plt.hist(data, bins)
  plt.show()
  ```

---

## Examples

### □ 250 Random Values + 5 Bars:

```
python
CopyEdit
x = np.random.uniform(0, 5, 250)
plt.hist(x, 5)
```

### □ 100,000 Random Values + 100 Bars:

```
python
CopyEdit
x = np.random.uniform(0, 5, 100000)
plt.hist(x, 100)
```

---

✅Each bar = value range count
✅Larger datasets help simulate real-world scenarios

## Normal Data Distribution (Gaussian/Bell Curve)

- Values cluster around a **mean**
- Uses `numpy.random.normal(mean, std_dev, size)`

---

## Example:

```
python
CopyEdit
import numpy as np
import matplotlib.pyplot as plt

x = np.random.normal(5.0, 1.0, 100000)  # Mean = 5, SD = 1
plt.hist(x, 100)
plt.show()
```

---

✅**Mean = 5.0**
✅**Standard Deviation = 1.0**
✅**Most values lie between 4.0 and 6.0**

---

□ **Shape**: Bell Curve
□ **Use case**: Simulates real-world natural data distributions

## Scatter Plot Overview

- Plots **dots** using two arrays: one for **x-axis**, one for **y-axis**
- Shows relationships between two variables

---

## Example 1: Car Age vs Speed

```
python
CopyEdit
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```

□ Newer cars appear faster (possible correlation)

---

## Example 2: Random Data

```
python
CopyEdit
import numpy as np
import matplotlib.pyplot as plt

x = np.random.normal(5.0, 1.0, 1000)
y = np.random.normal(10.0, 2.0, 1000)

plt.scatter(x, y)
plt.show()
```

✅x-values centered at 5.0
✅y-values centered at 10.0
□ Y-axis shows a wider spread

## What is Regression?

- **Regression**: Finding relationships between variables
- **Linear Regression**: Draws a straight line to predict future values

---

## Example: Car Age vs Speed

```
python
CopyEdit
import matplotlib.pyplot as plt
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```

---

## Add Linear Regression Line

```
python
CopyEdit
from scipy import stats

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x): return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

---

✓Line shows trend between car age and speed
□ Useful for predictions based on existing data

## Polynomial Regression (ML)

- **Definition**: Used when data doesn't fit a straight line (non-linear patterns).
- **Goal**: Predict outcomes using a curved line that fits the data.

## How It Works

- Use `numpy.polyfit()` to create a polynomial model.
- Visualize it using `matplotlib.pyplot`.

```
python
CopyEdit
```

```
import numpy as np
import matplotlib.pyplot as plt

x = [...]
y = [...]

model = np.poly1d(np.polyfit(x, y, 3))
line = np.linspace(min(x), max(x), 100)

plt.scatter(x, y)
plt.plot(line, model(line))
plt.show()
```

## $R^2$ Score

- Measures fit accuracy:
    - `1.0` = perfect
    - `0.0` = no fit

```
python
CopyEdit
from sklearn.metrics import r2_score
print(r2_score(y, model(x)))
```

## Prediction

- Use the model to predict:

```
python
CopyEdit
print(model(17))  # Predict at x = 17
```

## Bad Fit Example

- Poorly related data → low $R^2$ (e.g., `0.00995`) → polynomial regression not suitable.
- **Multiple Regression in Machine Learning**
- Multiple regression predicts a value using two or more independent variables.
  **Example**: Predicting a car's $CO_2$ emissions using weight and volume.
- **Steps in Python**:
- `python`
- `CopyEdit`
- `import pandas`
- `from sklearn import linear_model`
- 
- `df = pandas.read_csv("data.csv")`
- `X = df[['Weight', 'Volume']]`
- `y = df['CO2']`
- 
- `regr = linear_model.LinearRegression()`
- `regr.fit(X, y)`

- 
- `# Predict CO2 for a car (Weight=2300kg, Volume=1300cm3)`
- `predictedCO2 = regr.predict([[2300, 1300]])`
- `print(predictedCO2)`
- **Output**:
  ```
  [107.2087328]
  ```

**Scaling Features in Machine Learning**

Scaling helps make data comparable, especially when values have different units or magnitudes (e.g., weight vs. volume). We use **standardization** to scale data.

**Formula for Standardization**:
z=x−usz = \frac{x - u}{s}z=sx−u
Where:

- zzz = scaled value
- xxx = original value
- uuu = mean
- sss = standard deviation

**Example**:

- Weight: (790−1292.23)/238.74=−2.1(790 - 1292.23) / 238.74 = -2.1(790−1292.23)/238.74=−2.1
- Volume: (1.0−1.61)/0.38=−1.59(1.0 - 1.61) / 0.38 = -1.59(1.0−1.61)/0.38=−1.59

**Python Code**:

```
python
CopyEdit
import pandas
from sklearn.preprocessing import StandardScaler

scale = StandardScaler()
df = pandas.read_csv("data.csv")
X = df[['Weight', 'Volume']]
scaledX = scale.fit_transform(X)
print(scaledX)
```

**Result**:
Scaled values like [-2.1, -1.59] are easy to compare.

**Prediction Example** (for scaled data):

```
python
CopyEdit
scaled = scale.transform([[2300, 1.3]])
predictedCO2 = regr.predict([scaled[0]])
```

```
print(predictedCO2)
```

**Output**:
```
[107.2087328]
```

**Machine Learning - Train/Test**
Train/Test is a method to evaluate the accuracy of a model by splitting the dataset into training (80%) and testing (20%) sets. The model is trained on the training set and tested on the testing set.

**Steps:**

1. **Split Data:** Randomly divide data into training and testing sets.
2. **Fit Model:** Use a regression model to fit the training data (e.g., polynomial regression).
3. **Evaluate with R2 Score:** Calculate the R2 score to measure how well the model fits the data.
4. **Test Model:** Use the testing set to verify the model's performance on unseen data.
5. **Predict:** Once the model is validated, use it to predict new values.

Example code demonstrates splitting data, fitting a polynomial regression model, calculating R2, and predicting future values.

**Machine Learning - Data Distribution**

To create large data sets for testing, use Python's NumPy module to generate random data. For visualization, Matplotlib can be used to draw histograms.

**Steps:**

1. **Generate Random Data:** Use `numpy.random.uniform()` to create random values (e.g., 250 random floats between 0 and 5).
2. **Draw Histogram:** Use Matplotlib to plot a histogram, showing how data is distributed across different ranges.
3. **Big Data:** Increase data size (e.g., 100,000 values) for larger datasets and adjust the histogram bars accordingly.

Example code demonstrates generating random data and visualizing its distribution with a histogram.

**Machine Learning - Normal Data Distribution**

A normal (Gaussian) distribution has values concentrated around a mean value, forming a bell curve. To create it in Python, use `numpy.random.normal(mean, std_dev, size).`

**Example:**

- `mean = 5.0`, `std_dev = 1.0`, 100,000 values.

- Use Matplotlib to plot a histogram.

Most values cluster around the mean (5.0), with fewer values further away, creating the bell curve shape.

**Machine Learning - Scatter Plot**

A scatter plot shows data points as dots, where each dot represents a pair of values. Use `plt.scatter(x, y)` to create one.

**Example:**

- x = car ages, y = car speeds.
- Scatter plot shows relationships (e.g., newer cars tend to be faster).

For random data, generate two arrays with `numpy.random.normal(mean, std_dev, size)` and plot with `plt.scatter(x, y).`

**Machine Learning - Linear Regression**

**Regression** finds relationships between variables to predict future outcomes. **Linear Regression** uses data points to draw a straight line for predictions.

**Example:**

- Use `stats.linregress(x, y)` to calculate the slope, intercept, and correlation coefficient (r).
- `r` measures the relationship, where 1 or -1 is a perfect fit, and 0 means no relation.

**Prediction:**

- To predict values (e.g., speed of a 10-year-old car), use the formula `y = slope * x + intercept.`

**Bad Fit:**

- If `r` is close to 0, the data is not suitable for linear regression.

Polynomial regression is used when data points do not fit a straight line, offering a more flexible way to model relationships between variables. It uses the polynomial relationship to draw a curve through data points.

## Steps:

1. **Scatter Plot**: Plot your data to visualize the relationship.
2. **Model**: Use `numpy.polyfit()` to create a polynomial model.

3. **Plot Curve**: Use `matplotlib` to plot the regression curve.

## Example:

```
python
CopyEdit
import numpy as np
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

model = np.poly1d(np.polyfit(x, y, 3))
line = np.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(line, model(line))
plt.show()
```

## Evaluate Fit:

Use the **R-squared** value to measure how well the model fits your data. A value close to 1 means a good fit.

```
python
CopyEdit
from sklearn.metrics import r2_score
print(r2_score(y, model(x)))
```

## Predict:

You can predict future values using the model, for example, predicting the speed at hour 17:

```
python
CopyEdit
speed = model(17)
print(speed)
```

## Bad Fit Example:

If data doesn't fit well, the R-squared value will be low, indicating a poor fit.

**Multiple Regression Overview**
Multiple regression predicts a value based on two or more independent variables. For example, predicting CO2 emissions based on car weight and engine volume.

**Steps in Python:**

1. **Import libraries**:

```
python
CopyEdit
import pandas as pd
from sklearn.linear_model import LinearRegression
```

2. **Load data**:

```
python
CopyEdit
df = pd.read_csv("data.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
```

3. **Train model**:

```
python
CopyEdit
regr = LinearRegression()
regr.fit(X, y)
```

4. **Make prediction**:

```
python
CopyEdit
predictedCO2 = regr.predict([[2300, 1300]])
print(predictedCO2)
```

5. **Interpret results**:
   Coefficients show how weight and volume affect CO2. For instance, increasing weight by 1kg increases CO2 by 0.0075g.
6. **Example**:
   If the car weight increases from 2300kg to 3300kg, the predicted CO2 emission rises from 107g to 115g.
   Formula:
   `107.2087328 + (1000 * 0.00755095) = 114.75968`

Scaling features is essential when data has different values or measurement units. One common method is **standardization**, which transforms data to a comparable scale using the formula:

z=(x−u)sz = \frac{(x - u)}{s}z=s(x−u)

Where:

- zzz is the scaled value
- xxx is the original value
- uuu is the mean
- sss is the standard deviation

In Python, you can use **StandardScaler** from `sklearn` to standardize data easily.

## Example:

```
python
CopyEdit
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load data
df = pd.read_csv("data.csv")
X = df[['Weight', 'Volume']]

# Standardize the data
scale = StandardScaler()
scaledX = scale.fit_transform(X)

print(scaledX)
```

## Predicting CO2 Emissions:

To predict CO2 emissions after scaling:

```
python
CopyEdit
scaled = scale.transform([[2300, 1.3]])
predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

This scales the data and helps make predictions more reliable.

**Machine Learning - Train/Test**

**Train/Test Method**: Split data into training (80%) and testing (20%) sets. Train the model on training data and test its accuracy on testing data.

**Steps**:

1. **Data Set**: Create a data set with input and output variables (e.g., shopping habits).
2. **Split Data**: Use 80% for training and 20% for testing.
3. **Fit the Model**: Apply a polynomial regression to the training data.
4. **R-Squared Score ($R^2$)**: Measure the model's fit. $R^2$ ranges from 0 (no fit) to 1 (perfect fit).
5. **Test the Model**: Validate with the testing data.
6. **Prediction**: Predict outcomes for new data.

**Example**: Predict customer spending based on time spent in the shop. Model showed an $R^2$ of 0.809, confirming a good fit.

A Decision Tree is a flowchart-like model used for decision-making based on data. It splits data into branches based on feature values to predict outcomes. In the example, the decision to attend a comedy show is made based on factors like Age, Experience, Rank, and Nationality.

Steps to create a Decision Tree:

1. **Convert data to numerical values** (e.g., Nationality and 'Go' columns).
2. **Separate features and target columns** for prediction.
3. **Build the tree** using `DecisionTreeClassifier` in Python.
4. **Plot the tree** using `plot_tree` from `matplotlib`.

The decision tree splits the data based on certain features (like Rank or Age), showing the probability of an outcome (e.g., attending the show). The `predict()` method can be used for new predictions.

A confusion matrix is a table used to evaluate classification models by comparing actual vs. predicted values. It helps identify errors by showing true positives, false positives, true negatives, and false negatives. Key metrics derived from it include:

- **Accuracy**: (TP + TN) / Total predictions
- **Precision**: TP / (TP + FP)
- **Sensitivity (Recall)**: TP / (TP + FN)
- **Specificity**: TN / (TN + FP)
- **F1-Score**: 2 * (Precision * Recall) / (Precision + Recall)

These metrics help assess model performance, especially in imbalanced datasets.

Hierarchical clustering is an unsupervised learning method used to group data points based on their dissimilarities. Agglomerative Clustering, a bottom-up approach, starts with each data point as its own cluster and merges the closest ones iteratively.

Steps:

1. Visualize data points using a scatter plot.
2. Compute the linkage (distance between clusters) using Euclidean distance and Ward's linkage.
3. Use a dendrogram to visualize the hierarchy of clusters.
4. Alternatively, use scikit-learn's AgglomerativeClustering for clustering and visualization.

Key libraries:

- **NumPy**: For data manipulation.
- **Matplotlib**: For plotting.
- **SciPy**: For computing linkage.
- **scikit-learn**: For clustering and prediction.

Result: Visual representation of data points grouped into clusters.

**Logistic Regression**: A classification method predicting binary (e.g., cancerous or not) or multinomial outcomes.

- **Binomial Example**: Predicting tumor malignancy (0 = No, 1 = Yes).

**Steps**:

1. Import modules: `import numpy`, `from sklearn import linear_model`.
2. Define data:
    - X: Tumor sizes
    - Y: Tumor outcomes (0 or 1)
3. Fit model:

    ```
    python
    CopyEdit
    logr = linear_model.LogisticRegression()
    logr.fit(X, y)
    ```

4. Predict tumor outcome for new size:

    ```
    python
    CopyEdit
    predicted = logr.predict([3.46])
    ```

**Coefficient**: Measures log-odds change per unit increase in X.

- **Odds**: Exponentiate coefficient to interpret odds.

    ```
    python
    CopyEdit
    odds = numpy.exp(logr.coef_)
    ```

**Probability**: Convert log-odds to probability:

```
python
CopyEdit
probability = numpy.exp(log_odds) / (1 + numpy.exp(log_odds))
```

Example Output: Probability of a tumor being cancerous based on size.

**Grid Search for Logistic Regression:**

1. **Model Parameters**: Logistic regression has parameters like `C`, which controls regularization and model complexity.
2. **Goal**: Find the best value for `C` by evaluating different values (e.g., 0.25, 0.5, 1.75).
3. **Implementation**:

    - Use `LogisticRegression(max_iter=10000)` and fit to data.
    - Evaluate with different `C` values and store results in `scores[]`.
4. **Results**: Higher values of `C` (like 1.75) improve accuracy. Going beyond doesn't help.
5. **Overfitting Caution**: Use train/test split to avoid overfitting.

## Here's a step-by-step breakdown:

1. **Dataset**:
    - You have a dataset with categorical columns like `Car` and `Model` which contain string values.
    - In machine learning models, string data cannot be directly used, so you need to transform these into numeric values.
2. **One-Hot Encoding**:
    - One hot encoding involves creating a new binary column for each category (unique value) in the original categorical column.
    - If a row corresponds to a category, the new column for that category gets a `1`, and all others get `0`.
    - For example, if a car is a "Ford", the `Car_Ford` column will be `1`, and all other `Car_<brand>` columns will be `0`.
3. **Using Pandas for One-Hot Encoding**:
    - `pd.get_dummies()` is a pandas function that automatically performs one-hot encoding on the selected categorical column(s).

### Code Example:

```
python
CopyEdit
import pandas as pd

# Read the data
cars = pd.read_csv('data.csv')

# Perform one-hot encoding on the 'Car' column
ohe_cars = pd.get_dummies(cars[['Car']])

# Print the resulting DataFrame
print(ohe_cars.to_string())
```

4. **Output**: The result of one-hot encoding will look like this:
    - Each unique car brand (like Audi, BMW, Ford, etc.) will have its own column.
    - If the car is from that brand, the value in the corresponding column will be `1`; otherwise, it will be `0`.

## Example Output:

```
plaintext
CopyEdit
      Car_Audi  Car_BMW  Car_Fiat  Car_Ford  Car_Honda  ...
  0          0        0         0         0          0    ...
```

```
1          0       0        0        0         0    ...
2          0       0        0        0         0    ...
3          0       0        1        0         0    ...
4          0       0        0        0         0    ...
...
```

## Benefits:

- This transformation allows the categorical variables (`Car`) to be included in machine learning models as numeric input.
- It avoids any assumptions about the relationship between the categories (since the machine learning model will treat them as independent features).

Let me know if you'd like further clarification or additional examples!

K-means is an unsupervised clustering algorithm that divides data into K clusters by minimizing variance. The process involves assigning data points to clusters, calculating centroids, and iterating until assignments stabilize. The "elbow method" helps choose the best K by plotting inertia for different K values. Once K is determined, the model is trained, and the data is clustered. Example code shows how to visualize data, find the optimal K using inertia, and visualize the clustered results. Bagging is an ensemble method that improves accuracy by using random subsets of data with replacement. It reduces overfitting by training classifiers or regressors on these subsets and combining their predictions through majority vote (classification) or averaging (regression). For example, using a Decision Tree classifier on the wine dataset, bagging improves accuracy by aggregating predictions from multiple models.

Cross-validation (CV) improves model performance by preventing overfitting. It involves splitting the data into multiple subsets for training and validation. Common CV methods include:

- **K-Fold**: Divides the data into k parts; trains on k-1 and validates on the remaining fold.
- **Stratified K-Fold**: Ensures class balance in both training and validation sets.
- **Leave-One-Out (LOO)**: Uses one data point for validation, training on the rest.
- **Leave-P-Out (LPO)**: Similar to LOO but uses p data points for validation.
- **Shuffle Split**: Randomly splits data into training and testing sets, with customizable split ratios.

Each method is useful for different data scenarios, and you can average scores across folds for model evaluation.

The **AUC-ROC Curve** is a performance evaluation metric for classification models. It plots the **True Positive Rate (TPR)** vs **False Positive Rate (FPR)** at different thresholds.

- **Accuracy** can be misleading, especially with imbalanced data, as a model may predict the majority class well but fail on the minority class.

- **AUC** (Area Under the Curve) gives a better measure by evaluating how well the model distinguishes between classes. A value close to 0.5 means no distinction, while closer to 1 means good class separation.

**AUC Example**:

- Model 1 (low confidence): AUC = 0.5
- Model 2 (high confidence): AUC = 0.83
  Even if models have similar accuracy, a higher AUC score indicates a better model for distinguishing classes.

Use **AUC** when dealing with imbalanced data to get a more reliable model performance.

K-nearest neighbors (KNN) is a simple machine learning algorithm used for classification or regression. It predicts a data point's class based on the majority vote of its K nearest neighbors. The larger the K, the more stable the decision boundary. For classification, KNN assigns the most common class among the K closest points.

## Example:

1. **Data**: Points with features `(x, y)` and class labels.
2. **KNN Implementation**: Using `KNeighborsClassifier` from `scikit-learn`, we train the model with K=1 and K=5 to classify a new point.
3. **Prediction**: The class of the new point changes based on the number of neighbors.

**Key Points**:

- K=1: Sensitive to noise.
- K=5: More stable predictions.
- Larger K values help mitigate outliers.

KNN is intuitive but can be slow for large datasets.