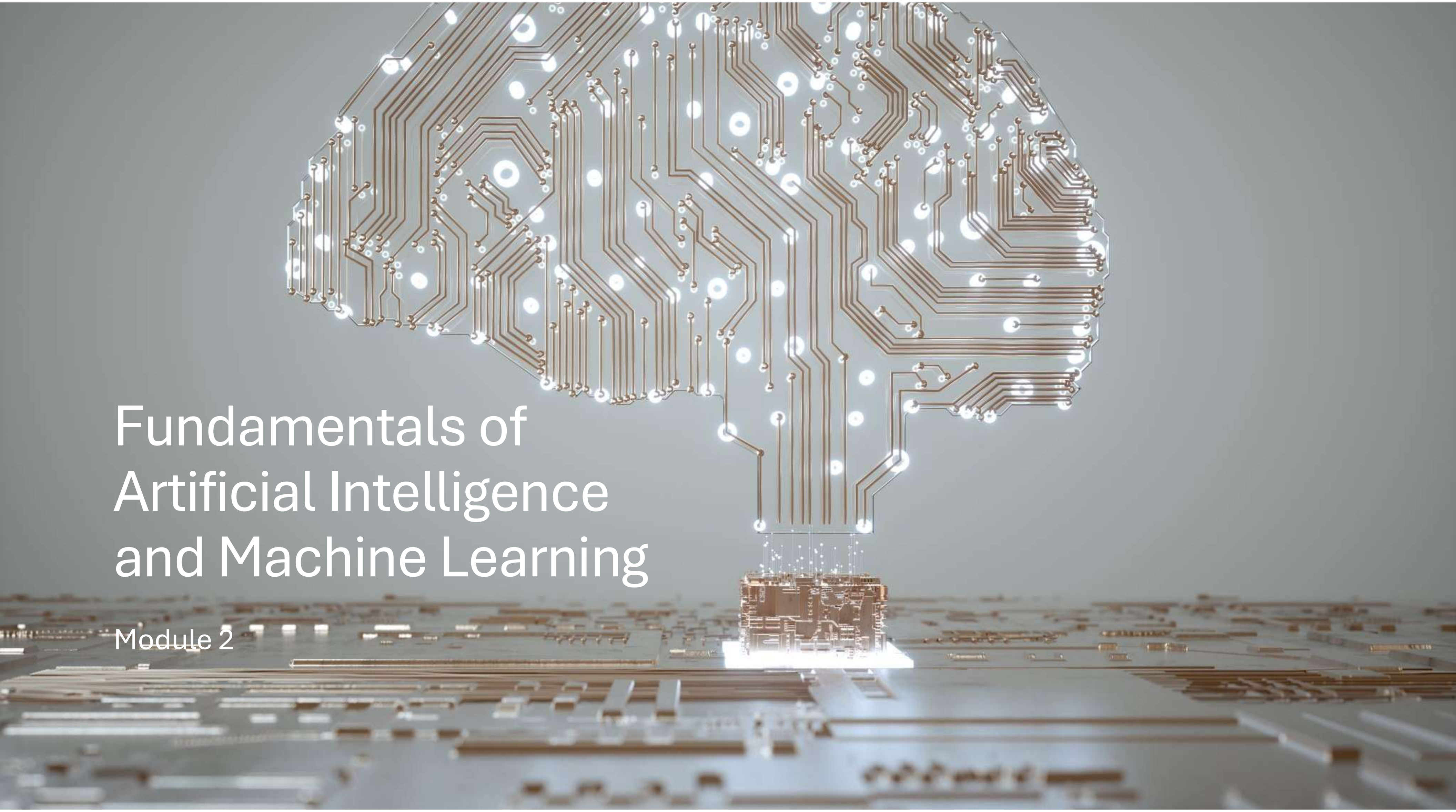


Fundamentals of Artificial Intelligence and Machine Learning

Module 2

Review of Last Class

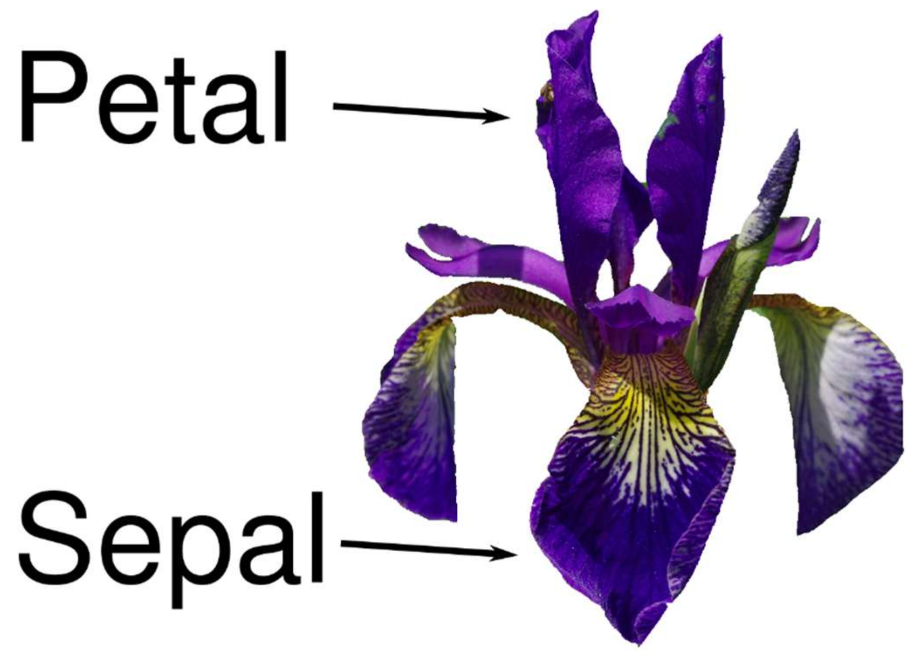
- Types of Analytics
    - Descriptive
    - Diagnostic
    - Predictive
    - Prescriptive
- Data Analytics vs Data Science
- Artificial Intelligence
    - Machine Learning
        - Supervised Learning
        - Unsupervised Learning
    - Other AI Subfields

Review of Last Class (continued)

- Classification
  - Binary Classification
  - Multi-Class Classification
- Machine Learning Classification Techniques
  - Decision Trees and Random Forests
    - Branch Splitting
    - Gini Index/Entropy
    - Bootstrapping
  - K-NN Algorithm
- Sci-kit Learn
- Iris Classification Lab

Highlights from Iris Classification

- Use of a k-NN Algorithm to classify an iris into 3 categories
  - Setosa
  - Versicolor
  - Virginica



Dataset Exploration

- Data is downloaded from the sklearn datasets library
- Shows the features (columns) and the targets

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

```
Feature names:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
[11]: # the data itself is contianed in the target and data fields. data contains the num
      print("Type of data:", type(iris_dataset['data']))
```

```
Type of data: <class 'numpy.ndarray'>
```

```
[12]: #the rows in the data array correspond to each indiidual flower, while the columns
      print("Shape of data:", iris_dataset['data'].shape)
```

```
Shape of data: (150, 4)
```

```
[13]: print("First five rows of data:\n", iris_dataset['data'][:5])
```

```
First five rows of data:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

```
In [16]: # species are encoded from 0 to 2.  0 - setosa, 1 - versicolor, 2 - virginica
         print("Target:\n", iris_dataset['target'])
```

```
Target:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Creating a Test Set and a Training Set

- 75% of the data was used in the training set
  - Training set will be used to train the algorithm
- 25% of the data was used in the test set
  - Testing set will be used to validate the model's accuracy
  - Predictions will be made and validated

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

```python
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

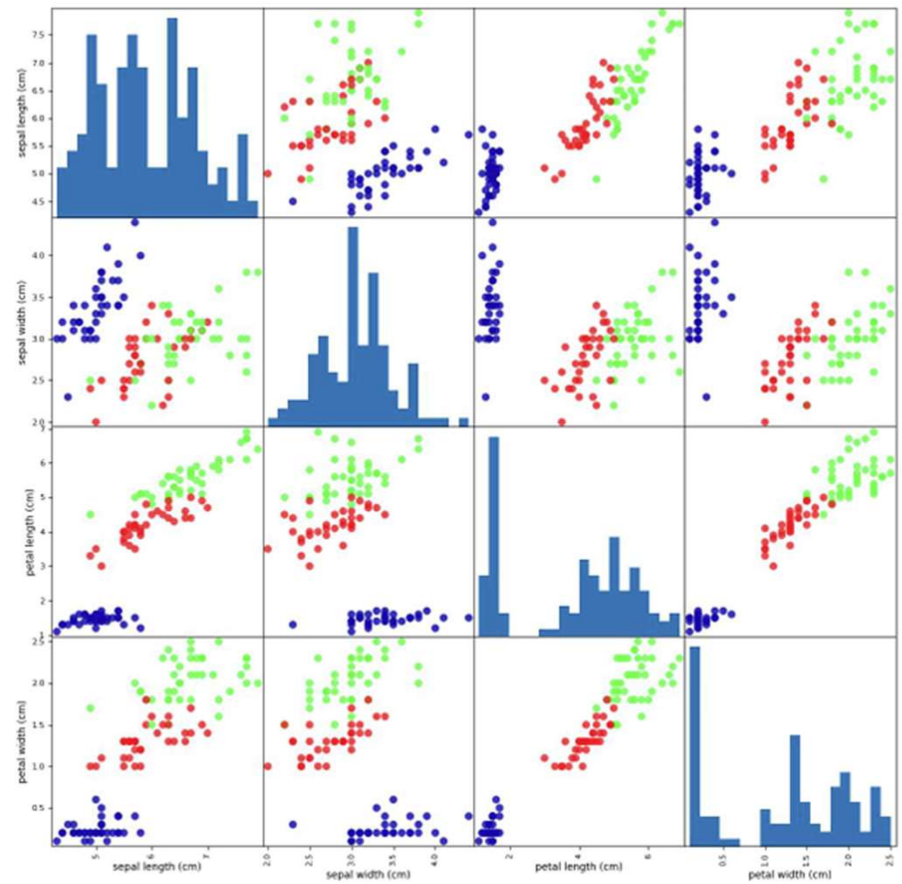
```
X_train shape: (112, 4)
y_train shape: (112,)
```

```python
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_test shape: (38, 4)
y_test shape: (38,)
```

Creating a Scatter Matrix to view 2D Dimensions of the Data

- We can only see 2 Dimensions of data
- Can be used to visually scan for insights and inconsistencies



Creating the k-NN Model

- Classification Process: Finds k-nearest neighbors based on a distance metric and assigns the majority class (or single class for k=1k=1).
- Parameter n_neighbors=1:
  - Considers only the closest neighbor for classification.
- Distance Metric:
  - Default: Euclidean distance. Customizable via the metric parameter.

[22]:
```
# the algorithm finds the point in the training set that is closest to the new point, then it assigns the label of this training point
# to the new data point. The k in k-nearest signifis that instead of only using the closest neighbor to the new data point, we can consider any fixed
# number k of neighbors in the training. (for example, the closest three or five neighbors). Then, we can make a prediction using the majority class
# among these neighbors.

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

[23]:
```
# fit method returns the knn object itself and modifies it in-place.
knn.fit(X_train, y_train)
```

[23]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

Making a Prediction

- Put in new data – and predict the classification of the iris

```
# fit method returns the knn object itself and modifies it in-place.
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

```
import numpy as np
# Feed a new iris with a 5cm sepal length, 2.9cm sepal width, 1cm petal length, and 0.2cm petal width
# Place this data into a num-py array
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape:", X_new.shape)
```

```
X_new.shape: (1, 4)
```

```
prediction = knn.predict(X_new)
print("Prediction:", prediction)
print("Predicted target name:",
       iris_dataset['target_names'][prediction])
```

```
Prediction: [0]
Predicted target name: ['setosa']
```

Evaluate the Model

- Evaluate the test predictions for each iris in the test set. See if we were accurate based on what they really were according to the data.
- Model had a 97% Accuracy Score!

27]:

```
# Evaluate the prediction based on the test set. Test each iris in the data set using the model we created with the fitted data.
# Measure how well the model works by computing the accuracy.
y_pred = knn.predict(X_test)
print("Test set predictions:\n", y_pred)
```

```
Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```

28]:

```
# Compute the accuracy - fraction of flowers for which the right species was predicted.
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.97
```

Business Applications of Iris Classification

Customer Segmentation

Group customers based on behaviors or preferences for targeted marketing

Product Categorization

Classify products into categories like “luxury” or “budget” based on features. (Ebay tags)

Quality Control

Identify defective products in manufacturing based on test results

Employee Evaluation

Group employees into performance tiers using key metrics

Class Overview

- Target's AI-Driven Transformation
- IBM Watson
- Supervised vs Unsupervised Learning
  - GAN Algorithms
- Regression and Logistic Regression
  - Regression as a Statistical Technique
  - Regression as a ML Technique
- Lab - Predicting Customer Churn
- Overfitting vs Underfitting
- Cross-Validation Techniques

Target's AI-Driven transformation

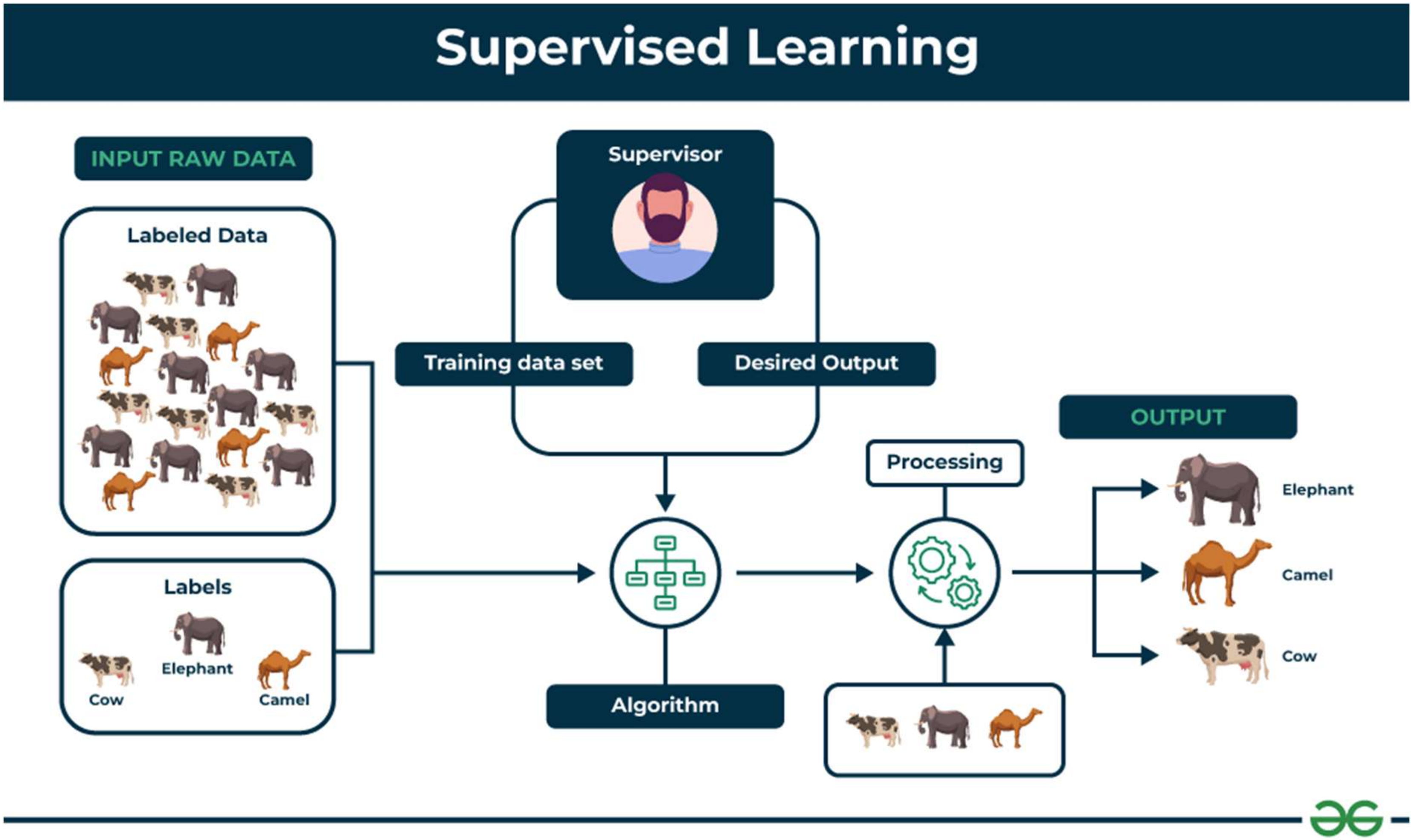
- Article
- Early Adoption
  - AI data collection began over a decade ago.
  - Used AI to forecast customer pregnancies for personalized marketing (famously there was a case where target new before the customer new).
- Customer Experience
  - Personalized Shopping
  - Improved product Search (AI-Powered Summaries)
  - Generative AI for Product Descriptions
- Supply Chain Innovation
  - Sorting, and Order Fulfillment Automation
  - Auto Re-Bin and Robotic Shipment Sorter
- Impact
  - 2025 Revenue is $85.90 Billion
  - Continued growth in digital businesses

IBM Watson

- How Watson Works
- Natural Language Processing (NLP)
  - AI that can read, understand, and generate human Languages
  - Ingests Large Datasets, Understands and Learns the data, Provides Analysis and insights
- Machine Learning Cognitive Computing
  - Simulates human cognitive process by interpreting and making decisions, based on context, knowledge, and data.
- As a product
  - Watson for Health
  - Watson for Business
  - Watson Assistant
  - Watson Studio
- Even won Jeopardy!
  - https://www.youtube.com/watch?v=P18EdAKuC1U&t=166s
  - https://www.ibm.com/history/watson-jeopardy

Types of Machine Learning

- Supervised Learning
  - Uses labeled data
  - Inputs features and labeled output
  - Machine learns the relationships between the inputs and the outputs, and makes trained predictions
  - Example: predicting customer churn, iris prediction
- Unsupervised Learning
  - Finds hidden patterns in unlabeled data
  - Inputs features with no labels or outcomes
  - Example: Clustering customers by purchasing behavior

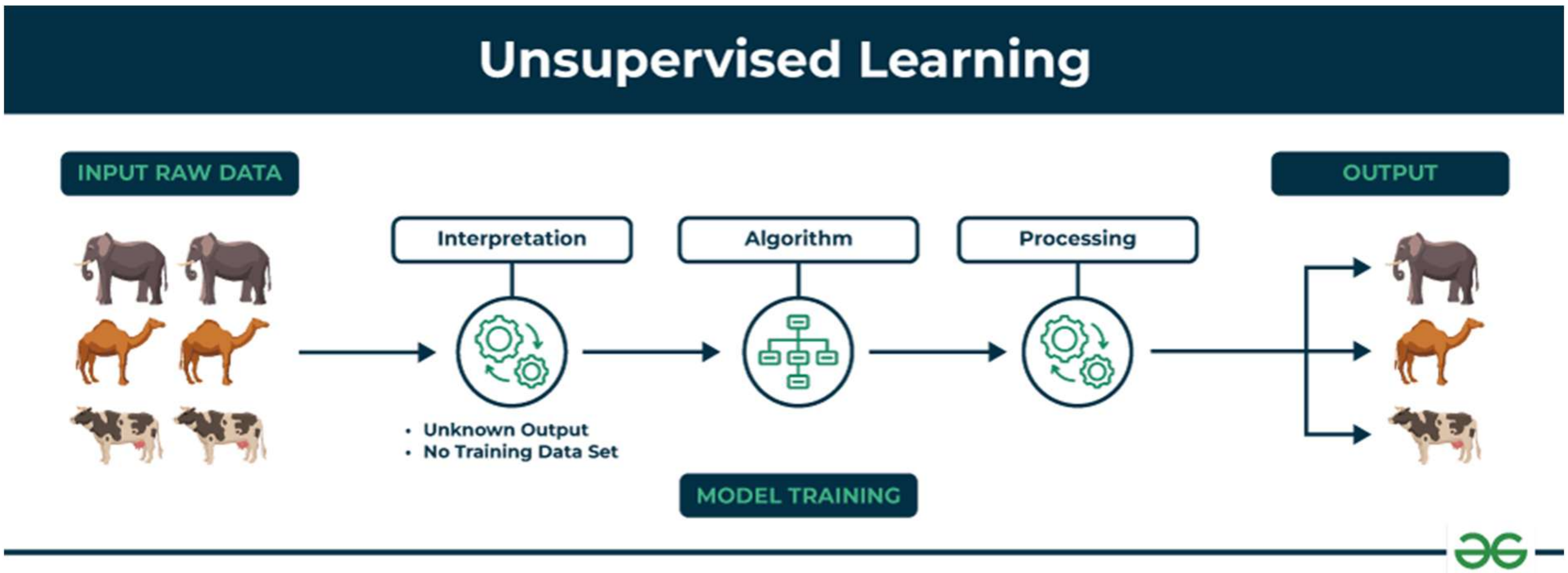

Supervised Learning
INPUT RAW DATA
Labeled Data
Labels
Cow
Elephant
Camel
Supervisor
Training data set
Desired Output
Algorithm
Processing
OUTPUT
Elephant
Camel
Cow

Application of Supervised Learning

- Spam filtering
  - Algorithms can be trained to identify and classify spam emails based on content
- Image Classification
  - Classify image into different categories for image search, content moderation, and image-based product recommendations.
- Fraud Detection
  - Analyze financial transactions and identify patterns that indicate fraudulent activity

When to choose Supervised Learning

- Advantages
  - Collecting data and producing predictions from previous experience.
  - Control of the number of classes we want in training data.
  - Estimating or mapping results to a new sample.
- Disadvantages
  - Classifying big data is difficult.
  - Requires a cleaned and labeled data set.
  - Requires a training process.
  - Needs a lot of computation and time.


Unsupervised Learning
INPUT RAW DATA
OUTPUT
Interpretation
Algorithm
Processing
• Unknown Output
• No Training Data Set
MODEL TRAINING

Application of Unsupervised Learning

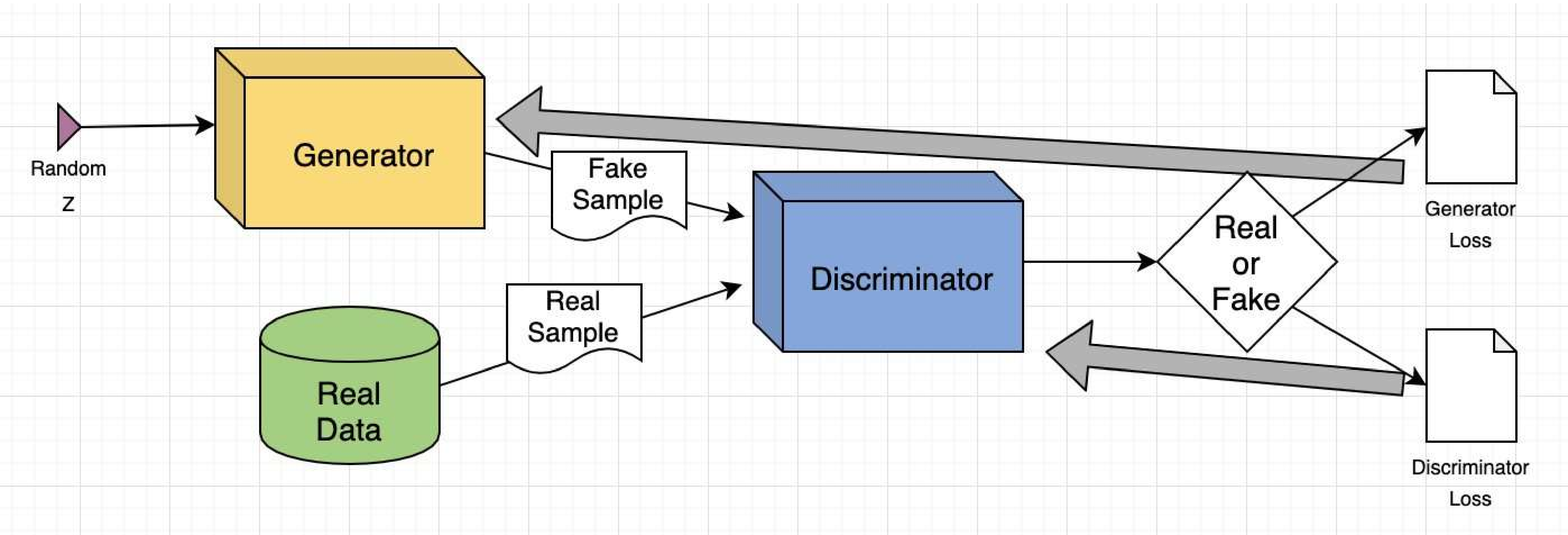
- Anomaly detection
  - Detecting deviations from normal behavior of data such as fraud detection, home intrusion, and system failures.
- New Discovery
  - Can uncover hidden relationships in data such as scientific data, leading to new insights and hypothesis an scientific fields.
- Customer Segmentation
  - Identify groups of customers with similar characteristics, allowing businesses to target marketing campaigns and tailor customer service.

When to choose Unsupervised Learning

- Advantages
  - Training data doesn't need to be labeled
  - Can find patterns and relationships without being told what to look for
  - Can help gain unknown insights you might not have been able to get otherwise
- Disadvantages
  - Difficult to measure accuracy or effectiveness due to lack of answers during training
  - Unsupervised learning can be sensitive to data quality such as missing values, outliers, and noisy data
  - Without labeled data, we can't asses the performance of the algorithm

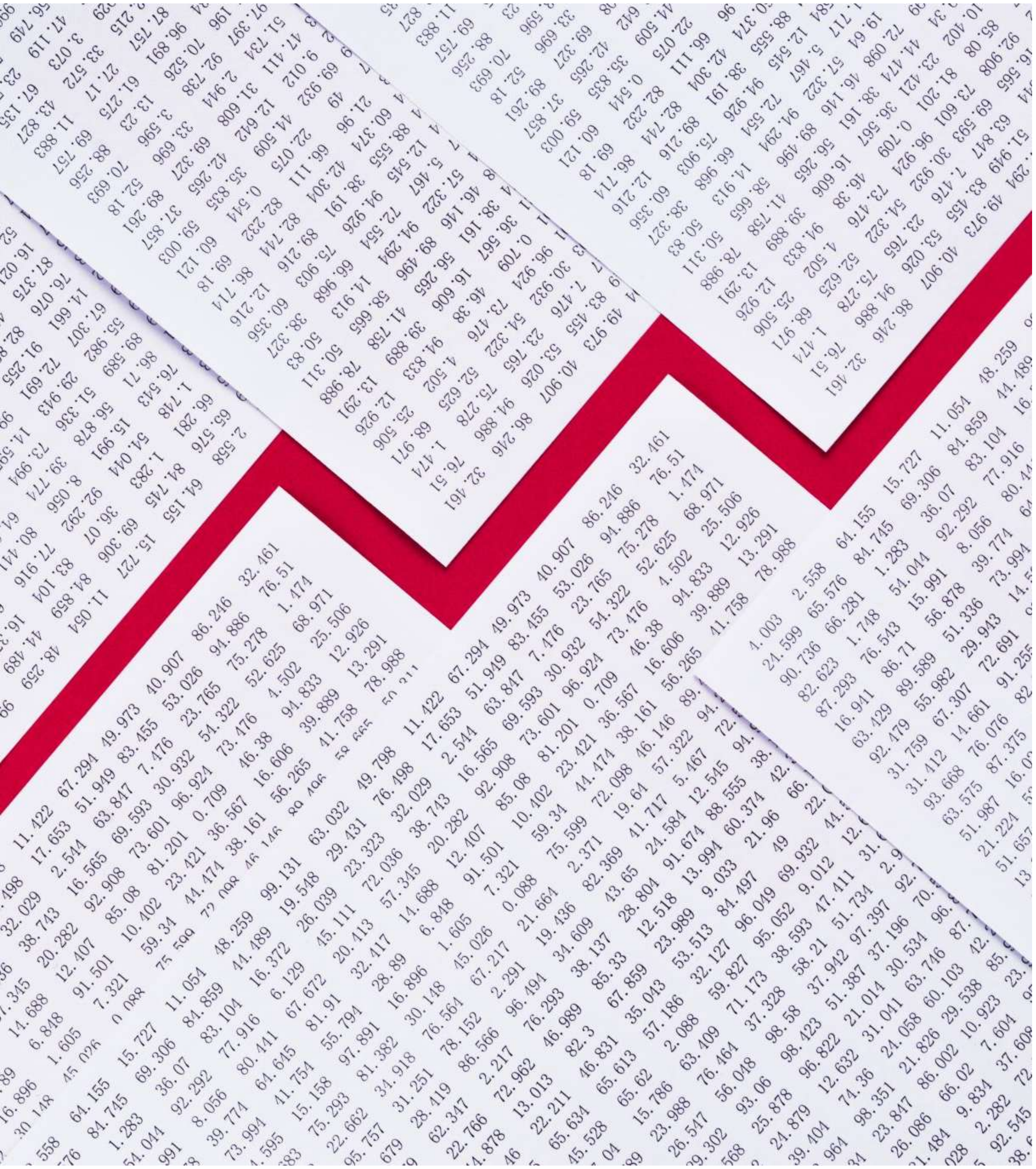
Generative Adversarial Network

- Considered an **unsupervised learning** algorithm.
- Two learning models:
    - **Generator:** Creates fake data
    - **Discriminator:** Distinguishes fake from real
- Applications:
    - Image/Voice Generation
    - Image to Text Translation and Synthesis
    - Text to Image Synthesis
    - Data Generation for training
        - Example: Healthcare (realistic images of organs for surgical planning and simulation)
- Example: https://thispersondoesnotexist.com/


Random
Z
Generator
Fake Sample
Real Sample
Real Data
Discriminator
Real or Fake
Generator Loss
Discriminator Loss

Quick Quiz!

- What is one disadvantage to Unsupervised Learning?
- Is predicting the species of Iris an example of Supervised Learning or Unsupervised Learning?
- What does GAN Stand for?
- Is a GAN Artificial Intelligence an example of Supervised or Unsupervised Learning?



Regression Techniques

- **Regression**
  - Predicts continuous values (example: house prices)
- **Logistic Regression**
  - Predicts probabilities or binary outcomes (e.g. churn or no churn)

Regression as a Machine Learning Technique

- Used to make predictions based on data
- Treated as a learning algorithm rather than a purely statistical tool
- Algorithms
    - Linear Regression
    - Polynomial Regression
    - Ridge and Lasso Regression
    - Support Vector Regression
- Main difference is the focus on minimizing prediction error and optimizes performance on unseen data. However, unlike other machine learning models like neural networks, it doesn’t involve iterative learning.

Quick Quiz!

- What is the difference between Regression and Logistic Regression?

Real-World Example – Churn Prediction

- Telephone Company Data
- Customer Attributes
  - Income, overage, leftover, house, over_15_min_calls_per_month, average_call_duration, college2
- Outcome: Leave
  - Whether the customer decides to leave for another telephone company.
  - This is the value we will be attempting to predict!
  - If they are predicted to churn, it might be worth the company's resources to reach out to them, or provide them a special offer to stay.

```
COLLEGE                          object
INCOME                            int64
OVERAGE                           int64
LEFTOVER                          int64
HOUSE                             int64
HANDSET_PRICE                     int64
OVER_15MINS_CALLS_PER_MONTH       int64
AVERAGE_CALL_DURATION             int64
REPORTED_SATISFACTION            object
REPORTED_USAGE_LEVEL             object
CONSIDERING_CHANGE_OF_PLAN       object
LEAVE                            object
```

Cleaning the Data – College Column

- Convert “College” column to integer, so it can be categorized as a boolean

[4]:
```
# Transform COLLEGE column to a numeric variable
df["COLLEGE2"] = (df.COLLEGE == "one").astype(int)
df.head(5)
```

Cleaning the Data – Outcome Column

- Convert the Outcome to an integer so it can be evaluated as a boolean

```
[6]: df["LEAVE2"] = (df.LEAVE == "STAY").astype(int)
     df.head(5)
```

Select Predictor Columns

- Assign the columns we want to use to make the prediction, and the target column we want to predict

```
# Names of different columns
predictor_cols = ["INCOME", "OVERAGE","LEFTOVER","HOUSE","OVER_15MINS_CALLS_PER_MONTH","AVERAGE_CALL_DURATION","COLLEGE2"]
target_col = "LEAVE2"
```

Split the Data Set

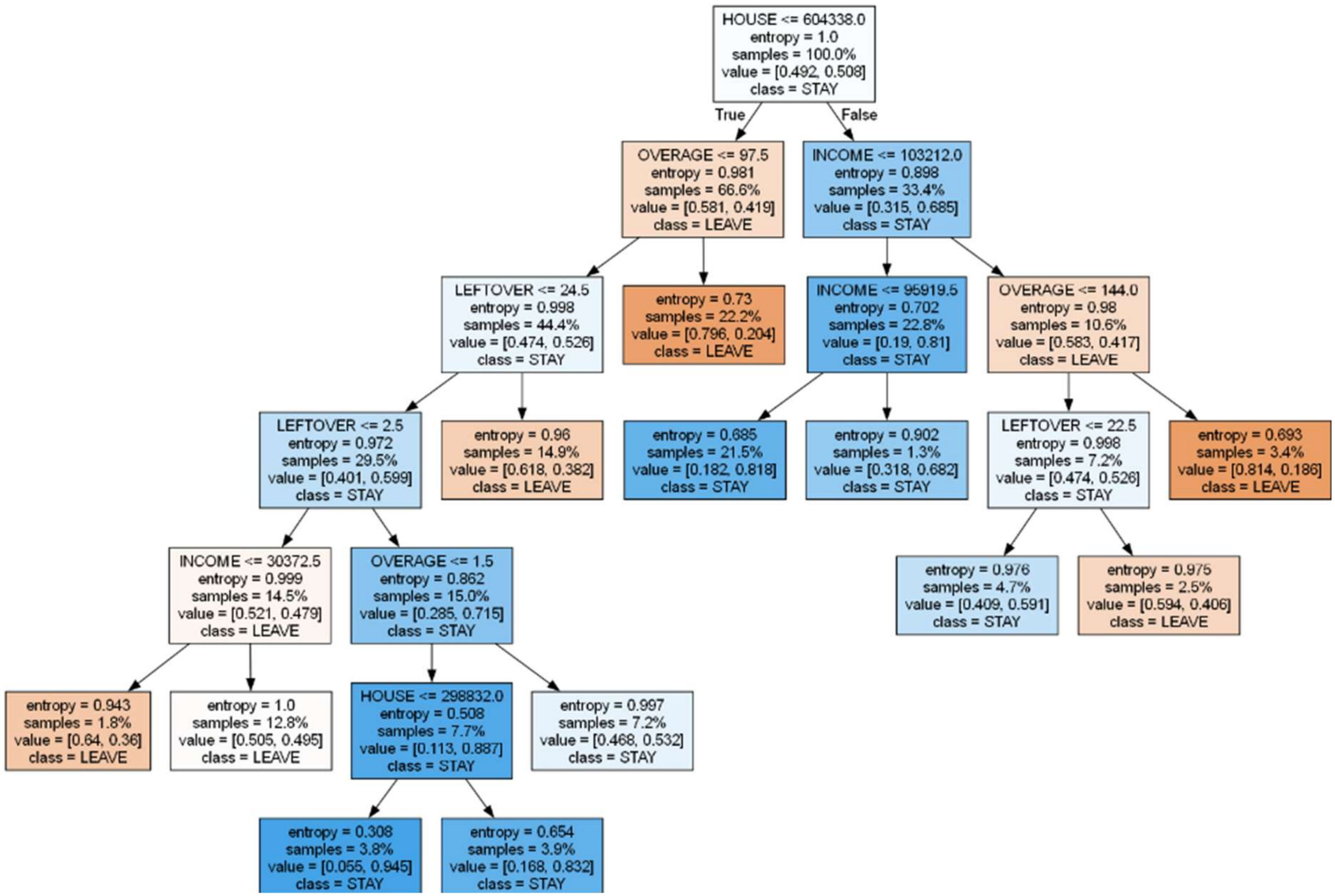
- Function **train_test_split** splits the data set into training and datasets to evaluate the performance of the machine learning model
    - Training set – used to train the machine learning model. The model learns from these data
    - Testing set – used to evaluate the performance of the trained model on unseen data.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[predictor_cols],df[target_col],test_size = 0.5,random_state = 0)
```

Fit the Model

- Define a DecisionTree
  - Define parameters about depth, leafs and criterion

```
[10]: from sklearn.tree import DecisionTreeClassifier
      # Let's define the model (tree)
      decision_tree = DecisionTreeClassifier(max_depth=6, criterion="entropy",max_leaf_nodes = 12, min_samples_leaf = 1)
      # Let's tell the model what is the data
      decision_tree.fit(X_train, y_train)
```


HOUSE <= 604338.0
entropy = 1.0
samples = 100.0%
value = [0.492, 0.508]
class = STAY
True
False
OVERAGE <= 97.5
entropy = 0.981
samples = 66.6%
value = [0.581, 0.419]
class = LEAVE
INCOME <= 103212.0
entropy = 0.898
samples = 33.4%
value = [0.315, 0.685]
class = STAY
LEFTOVER <= 24.5
entropy = 0.998
samples = 44.4%
value = [0.474, 0.526]
class = STAY
entropy = 0.73
samples = 22.2%
value = [0.796, 0.204]
class = LEAVE
INCOME <= 95919.5
entropy = 0.702
samples = 22.8%
value = [0.19, 0.81]
class = STAY
OVERAGE <= 144.0
entropy = 0.98
samples = 10.6%
value = [0.583, 0.417]
class = LEAVE
LEFTOVER <= 2.5
entropy = 0.972
samples = 29.5%
value = [0.401, 0.599]
class = STAY
entropy = 0.96
samples = 14.9%
value = [0.618, 0.382]
class = LEAVE
entropy = 0.685
samples = 21.5%
value = [0.182, 0.818]
class = STAY
entropy = 0.902
samples = 1.3%
value = [0.318, 0.682]
class = STAY
LEFTOVER <= 22.5
entropy = 0.998
samples = 7.2%
value = [0.474, 0.526]
class = STAY
entropy = 0.693
samples = 3.4%
value = [0.814, 0.186]
class = LEAVE
INCOME <= 30372.5
entropy = 0.999
samples = 14.5%
value = [0.521, 0.479]
class = LEAVE
OVERAGE <= 1.5
entropy = 0.862
samples = 15.0%
value = [0.285, 0.715]
class = STAY
entropy = 0.976
samples = 4.7%
value = [0.409, 0.591]
class = STAY
entropy = 0.975
samples = 2.5%
value = [0.594, 0.406]
class = LEAVE
entropy = 0.943
samples = 1.8%
value = [0.64, 0.36]
class = LEAVE
entropy = 1.0
samples = 12.8%
value = [0.505, 0.495]
class = LEAVE
HOUSE <= 298832.0
entropy = 0.508
samples = 7.7%
value = [0.113, 0.887]
class = STAY
entropy = 0.997
samples = 7.2%
value = [0.468, 0.532]
class = STAY
entropy = 0.308
samples = 3.8%
value = [0.055, 0.945]
class = STAY
entropy = 0.654
samples = 3.9%
value = [0.168, 0.832]
class = STAY

Evaluate the Model

- Get the Test Set Score and accuracy score
  - Make a prediction on all of the values from the test set, and determine the accuracy of the predictions

```
[20]: y_pred = decision_tree.predict(X_test)
      print("Test set score: {: 2f}".format(np.mean(y_pred == y_test)))

      Test set score:  0.696800

[21]: from sklearn import metrics
      print ( "Accuracy = %.3f" % (metrics.accuracy_score(decision_tree.predict(X_test), y_test) ))

      Accuracy = 0.697
```

Make a prediction for a new customer

```
]: predictor_cols = ["INCOME", "OVERAGE","LEFTOVER","HOUSE","OVER_15MINS_CALLS_PER_MONTH","AVERAGE_CALL_DURATION","COLLEGE2"]
X_new = np.array([[90000, 100,30,500000,3, 7,1]])
def Predict_for_New_Value(X_new):
    print("X_new.shape: {}".format(X_new.shape))
    prediction = decision_tree.predict(X_new)
    print("Prediction: {}".format(prediction))
    if(prediction == 0):
        return("LEAVE")
    elif(prediction == 1):
        return("STAY")
    else:
        return("UNKNOWN STATUS..")

predicted_status = Predict_for_New_Value(X_new)
print("Predicted value for new record is %s", predicted_status)
```

```
X_new.shape: (1, 7)
Prediction: [0]
Predicted value for new record is %s LEAVE
```

Logistic Regression

- Logistic Regression
  - A type of regression used for binary classification (two possible outcomes)
  - Will the customer leave? **Yes** or **No**
- Core of the Logistic Regression is the Logistic Function (sigmoid function) which transforms any input value into a value between – and 1.

$$P(y = 1 \,|X) = \frac{1}{1 + \, e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n)}}$$

- Decision Boundary
  - Output of the function is a probability. A common threshold would be 0.5
    - If $P(y = 1 \,|X) \geq 0.5$ the model is classified as 1 or "LEAVE"
    - If $P(y = 1 \,|X) < 0.5$ the model is classified as 1 or "STAY"

Logistic Regression – Prepare Data

- Declare X variable which contains all columns except the prediction ("LEAVE")
- Declare Y variable which contains the target column, which the model will predict

[53]:
```
# Define predictor (X) and target (y) columns
X = df[["COLLEGE2", "INCOME", "OVERAGE", "LEFTOVER", "HOUSE", "HANDSET_PRICE",
       "OVER_15MINS_CALLS_PER_MONTH", "AVERAGE_CALL_DURATION"]]
y = df["LEAVE"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Logistic Regression – Data Splitting

- Variable **test_train_split** will split the data into training (70%) and testing (30%) subsets.
- The training set is used to fit the model and the testing set will evaluate it's performance

```
[53]: # Define predictor (X) and target (y) columns
X = df[["COLLEGE2", "INCOME", "OVERAGE", "LEFTOVER", "HOUSE", "HANDSET_PRICE",
        "OVER_15MINS_CALLS_PER_MONTH", "AVERAGE_CALL_DURATION"]]
y = df["LEAVE"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Logistic Regression – Model Training

- Initialize the model with a maximum number of iterations to ensure it's convergence.
- Train the model using log_reg.fit() on the training data (x_train, y_train)

```
# Initialize logistic regression model
log_reg = LogisticRegression(max_iter=1000, random_state=0)

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

Logistic Regression- Make a Prediction

- Make predictions on the test data given

```
# Predict on the testing data
y_pred = log_reg.predict(X_test)
```

Logistic Regression – Model Evaluation

- Calculate Accuracy – percentage of correctly predicted instances
- Confusion Matrix – Shows the number of true positives, true negatives, false positives, and false negatives
- Classification Report – Precision, Recall, F1-score, and support

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)


# Print evaluation metrics
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.64
Confusion Matrix:
[[1811 1119]
 [1013 2057]]
Classification Report:
              precision    recall  f1-score   support

       LEAVE       0.64      0.62      0.63      2930
        STAY       0.65      0.67      0.66      3070

    accuracy                           0.64      6000
   macro avg       0.64      0.64      0.64      6000
weighted avg       0.64      0.64      0.64      6000
```

Resources

- https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning
- https://www.geeksforgeeks.org/supervised-unsupervised-learning/#
- https://aws.amazon.com/what-is/gan/#:~:text=A%20generative%20adversarial%20network%20(GAN,from%20a%20database%20of%20songs.
- https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/