**Summarization Report**

**First, we prepared an architecture for the project and learned key concepts along with it. Here is a summary of what we have learned so far:**

**Task-1: Learning About AI and ML Basics**

* **What We Did**: Learned what **Artificial Intelligence (AI)** and **Machine Learning (ML)** are. AI makes computers smart, like recognizing faces or suggesting movies on Netflix. ML is a part of AI where computers learn from data, like identifying cats from pictures. We explored:
  + Types of ML: **Supervised** (uses labeled data, e.g., spam emails), **Unsupervised** (finds patterns, e.g., grouping customers), and **Reinforcement** (trial and error, e.g., game AI).
  + **ML Pipeline**: Seven steps to build a model—collect data, clean it, prepare it, train a model, test it, deploy it (e.g., in apps), and maintain it.
  + **Classification**: Sorting data into groups (e.g., spam or not). Types include **binary** (two groups), **multi-class** (many groups), and **multi-label** (multiple tags).
  + **Classifiers**: Simple ones like Logistic Regression (straight lines) and complex ones like Decision Trees (many questions).
* **Mini Task 1: Iris with KNN**:
  + Used the **Iris dataset** to predict flower types (setosa, versicolor, virginica).
  + Explored data with pandas (checked features like petal length).
  + Plotted data with matplotlib (e.g., scatter plots of petal sizes).
  + Trained a **KNN classifier** (a non-linear classifier that groups flowers by nearby examples).
  + Pushed code to GitHub.

## Task-2: Preparing Data for ML

* **What We Did**: Learned how to **prepare data** for ML, like cleaning ingredients before cooking. Computers need clean, numerical data to learn. We covered:
  + **Preprocessing Steps**: Collect data (e.g., movie preferences), clean it (fix errors), organize it (turn text to numbers), split into training/testing sets, and scale numbers (make them similar ranges).
  + **Handling Missing Data**: Fill gaps with average (mean), most common (mode), random values, extreme values, or a fixed value (e.g., “Unknown”).
  + **Encoding**: Turn text into numbers—**label encoding** (e.g., “boy” = 0) or **one-hot encoding** (binary columns per category).
  + **Normalization**: Scale numbers to 0-1 (min-max) or adjust to mean 0 (standardization).
  + **Train/Test Split**: Use ~80% data to train, ~20% to test, to check if the model works on new data.
* **Mini Task 2: Titanic Data Cleaning**:
  + Loaded the **Titanic dataset** from Kaggle or Seaborn.
  + Cleaned data: Filled missing ages with mean, missing ports with mode (e.g., “Southampton”).
  + Encoded text: Converted “male/female” to 0/1 (label encoding) or used one-hot encoding for ports.
  + Split into train (80%) and test (20%) sets.
  + Pushed cleaned data and notebook to GitHub.

## Task-3: Training and Testing Models

* **What We Did**: Learned how to **train ML models**, use **Logistic Regression** for classification, and **evaluate** with accuracy. Training is like teaching a pet with examples and feedback. We covered:
  + **Training Models**: Models are math formulas adjusted to fit data (e.g., predict house prices). Algorithms like Decision Trees or Neural Networks find patterns. Needs lots of data and computing power.
  + **Logistic Regression**: A simple classifier for two groups (e.g., spam/not spam). It uses features (e.g., email words), gives probabilities, and picks the likely group.
  + **Evaluation with Accuracy**: Accuracy = percentage of correct predictions (e.g., 85/100 emails = 85%). Split data into training (learn) and testing (check). Accuracy can mislead if data is imbalanced (e.g., mostly not spam).
* **Mini Task 3: Titanic Logistic Regression**:
  + Used preprocessed **Titanic data** (from Mini Task 2).
  + Trained a **Logistic Regression model** to predict survival (survived or not).
  + Evaluated with accuracy\_score (e.g., 80% correct predictions).
  + Showed results (accuracy, maybe confusion matrix).
  + Pushed code to GitHub

## Task-4: Evaluating Models with Better Metrics

* **What We Did**: Learned how to check if a classification model works well using a **Confusion Matrix**, **Precision**, **Recall**, and **F1-Score**, instead of just accuracy. A confusion matrix is like a score sheet showing what a model got right or wrong (e.g., spam emails correctly or wrongly labeled). We also learned why accuracy can trick us when data is imbalanced (e.g., mostly healthy patients).
  + **Confusion Matrix**: Shows:
    - Correct positives (e.g., spam caught).
    - Correct negatives (e.g., non-spam right).
    - Wrong positives (e.g., non-spam called spam).
    - Wrong negatives (e.g., spam missed).
  + **Precision**: How many predicted positives are right (e.g., 83% of spam predictions correct).
  + **Recall**: How many actual positives are caught (e.g., 91% of spam emails found).
  + **F1-Score**: Balances precision and recall.
  + **Accuracy Issues**: High accuracy (e.g., 99%) can hide poor performance on rare classes (e.g., sick patients).
* **Mini Task 4: Titanic Confusion Matrix**:
  + Used the **Titanic dataset** to predict survival.
  + Generated a **confusion matrix** to show correct/incorrect predictions (e.g., survived or not).
  + Calculated **precision**, **recall**, and **F1-score** (e.g., how well the model catches survivors).
  + Wrote a short analysis: Accuracy isn’t enough when data is imbalanced (e.g., more non-survivors than survivors), as the model may ignore survivors.
  + Pushed code to GitHub.

## Task-5: Understanding Overfitting, Underfitting, and Improving Models

**What We Did**: Learned about **overfitting** (model too complex, memorizes noise) and **underfitting** (model too simple, misses patterns). We explored **cross-validation** to test models fairly and **ways to improve models**. It’s like a student cramming (overfitting) or not studying (underfitting).

* + **Overfitting**: Great on training data (e.g., 95%) but bad on test (e.g., 60%), like memorizing Titanic passenger details.
  + **Underfitting**: Bad on both (e.g., 50%), like using only Pclass.
  + **Cross-Validation**: Split data into parts (e.g., 10 folds), train on most, test on one, repeat, average results. Ensures model works on new data.
  + **Model Improvement**:
    - Clean/add data.
    - Create/remove features (e.g., derive age).
    - Try different models (e.g., Random Forest).
    - Tune settings (e.g., tree depth).
    - Reduce overfitting: Simplify, add data, use regularization.
    - Reduce underfitting: Add complexity, features.
* **Mini Task 5: Titanic Overfitting/Underfitting**:
  + Used the **Titanic dataset** to train two models:
    - **Simple model**: Few features (e.g., Pclass, Sex) to show underfitting (low accuracy, small gap).
    - **Overfitted model**: Many features (e.g., Age, Fare, Ticket) with high depth Random Forest to show overfitting (high training accuracy, lower test, large gap).
  + Compared train vs. test performance (e.g., accuracy, gap).
  + Wrote a brief write-up: Signs of overfitting (large train-test gap, high training accuracy).
  + Pushed code to GitHub.

## Task-6: Bias-Variance Tradeoff and Visualization

* **What We Did**: Learned about the **bias-variance tradeoff**, balancing **high bias (underfitting)** and **high variance (overfitting)**. We used examples like predicting rain or Titanic survival to understand it and visualized it with graphs.
  + **Bias**: Too simple model (e.g., only age for Titanic), high error, underfits.
  + **Variance**: Too complex model (e.g., Random Forest with ticket numbers), fits noise, overfits.
  + **Tradeoff**: Simple models have high bias, complex models have high variance. Find a middle ground or add data.
  + **High Bias vs. Variance**:
    - **High Bias**: Low train/test accuracy (e.g., 50%), small gap, too simple.
    - **High Variance**: High training accuracy (e.g., 95%), lower test (e.g., 60%), large gap, too complex.
  + **Visualization**: Graph with complexity (x-axis) and error (y-axis). High bias at low complexity, high variance at high, best model in the middle (U-shape).
* **Mini Task 6: Car Evaluation Learning Curves**:
  + Used the **Car Evaluation dataset** (predicting car acceptability, e.g., unacc, acc) from your prior work.
  + Trained a basic model (e.g., Decision Tree or Logistic Regression).
  + Plotted **learning curves** (train vs. validation error) to show how error changes with training size.
  + Wrote a short summary: “How to detect bias vs. variance from learning curves” (e.g., high bias: both errors high, small gap; high variance: low training error, high validation, large gap).
  + Pushed code to GitHub.