**Complete Guide for iPhone Purchase Prediction Project**

**1. Project Overview**

**What the Project Is About**: This project develops a Decision Tree model to predict whether an individual will purchase an iPhone based on their Gender, Age, and Salary, using a dataset (iphone\_purchase\_records.csv) with 400 records. The analysis includes Exploratory Data Analysis (EDA) to uncover patterns, preprocessing to prepare data, model training, and evaluation to assess performance. The project is implemented in a Jupyter Notebook, exported as HTML for easy sharing.

**Purpose and Goals**:

* **Purpose**: To build an interpretable machine learning model that predicts iPhone purchase behavior and to demonstrate data science skills (EDA, modeling, visualization) for professional portfolios.
* **Goals**:
  + Identify key factors (e.g., Age, Salary) influencing iPhone purchases through EDA.
  + Train a Decision Tree model to achieve high accuracy (~85–90%).
  + Create visualizations (e.g., confusion matrix, feature importance) to communicate results.
  + Share a reproducible project on GitHub for recruiters, peers, or personal reference.

**Why It’s Useful**:

* **Problem Solved**: Helps understand consumer behavior for targeted marketing (e.g., identifying likely iPhone buyers).
* **Who Benefits**:
  + **Businesses**: Retailers or marketers can use the model to target high-potential customers.
  + **Data Scientists**: Showcases skills in Python, machine learning, and visualization for job applications.
  + **Students/Peers**: Provides a clear example of a machine learning workflow.

**Key Challenges and Solutions**:

* **Challenge**: Class imbalance in the target variable (Purchase iPhone: ~64% non-purchasers, ~36% purchasers).
  + **Solution**: Use a Decision Tree with tuned hyperparameters (max\_depth=5, min\_samples\_split=10) to mitigate bias and overfitting.
* **Challenge**: Categorical variable (Gender) requires encoding.
  + **Solution**: Apply One-Hot Encoding to handle Gender without implying ordinality.
* **Challenge**: Making the project reproducible and shareable.
  + **Solution**: Use Jupyter Notebook for interactive code, export to HTML with embedded plots, **2. Required Outputs**

**Deliverables**:

* **Code Files**:
  + iphone\_purchase\_analysis.ipynb: Jupyter Notebook with EDA, preprocessing, modeling, and evaluation.
  + create\_and\_execute\_notebook.py: Script to generate the notebook and HTML export.
* **Data File**:
  + iphone\_purchase\_records.csv: Dataset with 400 rows (Gender, Age, Salary, Purchase iPhone).
* **Visualizations**:
  + **Count Plot**: Distribution of Purchase iPhone (0 = No, 1 = Yes).
  + **Gender vs. Purchase Plot**: Count plot showing purchase behavior by Gender.
  + **Age/Salary Distributions**: Histograms and box plots for Age and Salary, segmented by Purchase iPhone.
  + **Scatter Plot**: Age vs. Salary, colored by Purchase iPhone.
  + **Correlation Heatmap**: Correlation between Age and Salary.
  + **Confusion Matrix Heatmap**: Model performance (true/false positives/negatives).
  + **Feature Importance Plot**: Bar plot showing importance of Gender, Age, Salary.
  + **Decision Tree Rules**: Text-based rules (graphical tree optional with Graphviz).
* **Report**:
  + iphone\_purchase\_analysis.html: HTML export of the notebook with embedded plots.
* **Supporting Files**:
  + requirements.txt: Lists Python dependencies.
  + visualizations/ folder: Contains saved plots (e.g., confusion\_matrix.png, feature\_importance.png).

**What Results Look Like**:

* **Accuracy Metrics**: Test accuracy ~85–90%, cross-validation mean score ~0.85–0.90.
* **Classification Report**: Precision, recall, and F1-score for classes 0 (No) and 1 (Yes), e.g.:

text

precision recall f1-score

0 0.92 0.94 0.93

1 0.85 0.80 0.82

* **Charts**:
  + Confusion matrix heatmap: 2x2 matrix (e.g., ~50 true negatives, ~20 true positives for 80 test samples).
  + Feature importance: Bar plot showing Age and Salary as dominant predictors.
  + EDA plots: Histograms showing purchasers are older (40–60) and have higher salaries (>100,000).
* **Predictions**: Model predicts 0 (No) or 1 (Yes) for new data, e.g., [Male, 45, 120000] → 1 (Purchase).

**Why Outputs Are Important**:

* **Validation**: Accuracy and classification report validate model performance; confusion matrix shows error types.
* **Sharing**: HTML report and visualizations make results accessible to non-technical audiences (e.g., recruiters).
* **Real-World Use**: Insights (e.g., Age/Salary importance) can guide marketing; reproducible code ensures others can use the model.

**3. Tools and Technologies Used**

**List of Tools and Technologies**:

* **Python 3**: General-purpose programming language.
  + **Why Chosen**: Beginner-friendly, widely used in data science, supports extensive libraries.
  + **Role**: Executes all project steps (data loading, EDA, modeling, visualization).
  + **Beginner-Friendly**: Easy syntax, large community, simple installation (python.org).
  + **Advanced**: Scalable for large datasets with optimized libraries.
* **pandas**: Data manipulation and analysis library.
  + **Why Chosen**: Efficient for handling tabular data (CSV), supports data cleaning and EDA.
  + **Role**: Loads dataset, performs summary statistics, and prepares data for modeling.
  + **Beginner-Friendly**: Intuitive DataFrame API, e.g., df.head().
  + **Advanced**: Optimized for large datasets with vectorized operations.
* **numpy**: Numerical computing library.
  + **Why Chosen**: Fast array operations, required by pandas and scikit-learn.
  + **Role**: Supports data preprocessing (e.g., array transformations).
  + **Beginner-Friendly**: Simple for basic math operations.
  + **Advanced**: Efficient for matrix computations in machine learning.
* **matplotlib**: Plotting library.
  + **Why Chosen**: Flexible for creating static visualizations (e.g., histograms).
  + **Role**: Generates plots (e.g., confusion matrix, feature importance).
  + **Beginner-Friendly**: Easy to create basic plots with plt.plot().
  + **Advanced**: Customizable for complex visualizations.
* **seaborn**: Statistical visualization library (built on matplotlib).
  + **Why Chosen**: Simplifies creating attractive, informative plots (e.g., heatmaps).
  + **Role**: Creates EDA plots (e.g., count plots, box plots) and confusion matrix heatmap.
  + **Beginner-Friendly**: High-level API, e.g., sns.countplot().
  + **Advanced**: Integrates with pandas for data-driven visualizations.
* **scikit-learn**: Machine learning library.
  + **Why Chosen**: Comprehensive, user-friendly for classification tasks like Decision Trees.
  + **Role**: Implements preprocessing (OneHotEncoder, StandardScaler), Decision Tree model, and evaluation metrics.
  + **Beginner-Friendly**: Simple API, e.g., DecisionTreeClassifier().fit().
  + **Advanced**: Supports hyperparameter tuning and cross-validation.
* **nbformat, nbconvert**: Jupyter Notebook creation and conversion libraries.
  + **Why Chosen**: Enable programmatic notebook creation and HTML export.
  + **Role**: Generate and execute iphone\_purchase\_analysis.ipynb, export to HTML.
  + **Beginner-Friendly**: Automate notebook tasks without manual intervention.
  + **Advanced**: Scalable for batch processing multiple notebooks.
* **Jupyter Notebook**: Interactive computing environment.
  + **Why Chosen**: Ideal for data science workflows, combining code, visualizations, and text.
  + **Role**: Hosts the analysis (iphone\_purchase\_analysis.ipynb).
  + **Beginner-Friendly**: Easy to install (pip install jupyter) and use interactively.
  + **Advanced**: Supports extensions for scalability.
* **Graphviz (Optional)**: Visualization tool for Decision Trees.
  + **Why Chosen**: Creates graphical Decision Tree diagrams.
  + **Role**: Visualizes tree structure (if enabled).
  + **Beginner-Friendly**: Optional; text-based rules are sufficient.
  + **Advanced**: Enhances interpretability for complex models.
* **GitHub**: Version control and project hosting platform.
  + **Why Chosen**: Industry-standard for sharing code, showcasing projects to recruiters.
  + **Role**: Hosts the repository for public access.
  + **Beginner-Friendly**: Web interface simplifies uploads.
  + **Advanced**: Supports branching, pull requests for collaboration.

**Installation**:

bash

pip install pandas numpy matplotlib seaborn scikit-learn nbformat nbconvert jupyter graphviz

* Install Graphviz software for graphical Decision Trees (e.g., sudo apt-get install graphviz on Linux).

**4. Step-by-Step Detailed Breakdown**

Below is a detailed breakdown of the project workflow, from data loading to sharing on GitHub. Each step includes what it does, why it’s performed, tools used, why those tools, the output, and its contribution to the project.

**Step 1: Set Up Environment**

* **What It Does**: Installs required libraries and sets up Jupyter Notebook.
* **Why Perform**: Ensures all tools are available to run the analysis.
* **Tools Used**: pip, Jupyter Notebook.
* **Why These Tools**: pip is the standard Python package manager; Jupyter provides an interactive environment for data science.
* **Output**: Installed libraries (pandas, scikit-learn, etc.) and running Jupyter server.
* **Reason**: Enables reproducible execution; critical for GitHub sharing as others can replicate the setup.
* **How**:

bash

pip install pandas numpy matplotlib seaborn scikit-learn nbformat nbconvert jupyter

jupyter notebook

* **Contribution**: Provides a consistent environment, documented in requirements.txt for GitHub.

**Step 2: Load and Inspect Data**

* **What It Does**: Loads iphone\_purchase\_records.csv and displays basic information (head, info, missing values, summary statistics).
* **Why Perform**: Understands dataset structure, checks for issues (e.g., missing values), and informs EDA.
* **Tools Used**: pandas.
* **Why This Tool**: pandas excels at handling tabular data, providing easy methods like df.info().
* **Output**: DataFrame with 400 rows, 4 columns (Gender, Age, Salary, Purchase iPhone); no missing values; summary stats (e.g., mean Age ~37, mean Salary ~69,000).
* **Reason**: Ensures data quality and informs preprocessing; shared in README.md for transparency.
* **Code**:

python

df = pd.read\_csv('iphone\_purchase\_records.csv')

print("First 5 rows of the dataset:")

print(df.head())

print("\nDataset Info:")

print(df.info())

print("\nMissing Values:")

print(df.isnull().sum())

print("\nSummary Statistics:")

print(df.describe())

**Step 3: Perform Exploratory Data Analysis (EDA)**

* **What It Does**: Visualizes data distributions and relationships (e.g., Gender vs. Purchase, Age/Salary distributions, correlations).
* **Why Perform**: Identifies patterns (e.g., older, high-salary individuals purchase more) and class imbalance (~64% non-purchasers).
* **Tools Used**: seaborn, matplotlib.
* **Why These Tools**: seaborn simplifies statistical plots; matplotlib supports customization.
* **Output**:
  + Count plot: ~256 non-purchasers, ~144 purchasers.
  + Gender vs. Purchase: Similar purchase rates for Male/Female.
  + Histograms/Box Plots: Purchasers have higher Age (40–60) and Salary (>100,000).
  + Scatter Plot: Visualizes Age vs. Salary by Purchase iPhone.
  + Correlation Heatmap: Low correlation between Age and Salary (~0.15).
* **Reason**: Guides feature selection and preprocessing; visualizations enhance README.md and HTML for GitHub sharing.
* **Code Example** (Count Plot):

python

plt.figure(figsize=(6, 4))

sns.countplot(x='Purchase Iphone', data=df)

plt.title('Distribution of iPhone Purchases')

plt.xlabel('Purchase iPhone (0 = No, 1 = Yes)')

plt.ylabel('Count')

plt.savefig('visualizations/purchase\_distribution.png')

plt.show()

**Step 4: Preprocess Data**

* **What It Does**: Encodes Gender (One-Hot Encoding), scales Age/Salary (StandardScaler), and splits data (80% train, 20% test).
* **Why Perform**: Prepares data for modeling; encoding handles categorical data, scaling ensures numerical features are comparable, splitting enables evaluation.
* **Tools Used**: scikit-learn (OneHotEncoder, StandardScaler, train\_test\_split).
* **Why This Tool**: scikit-learn provides robust preprocessing tools optimized for machine learning pipelines.
* **Output**: Encoded and scaled feature matrix (X\_train, X\_test: 320 and 80 rows), target arrays (y\_train, y\_test).
* **Reason**: Ensures model compatibility and fair evaluation; preprocessing steps are documented in README.md for reproducibility.
* **Code**:

python

X = df[['Gender', 'Age', 'Salary']]

y = df['Purchase Iphone']

column\_transformer = ColumnTransformer(transformers=[('encoder', OneHotEncoder(drop='first'), ['Gender'])], remainder='passthrough')

X\_encoded = column\_transformer.fit\_transform(X)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_encoded)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

print("Training set shape:", X\_train.shape)

print("Testing set shape:", X\_test.shape)

**Step 5: Train Decision Tree Model**

* **What It Does**: Trains a Decision Tree Classifier with tuned hyperparameters and performs cross-validation.
* **Why Perform**: Builds the predictive model; cross-validation ensures robust performance.
* **Tools Used**: scikit-learn (DecisionTreeClassifier, cross\_val\_score).
* **Why This Tool**: scikit-learn’s DecisionTreeClassifier is simple, interpretable, and supports hyperparameter tuning.
* **Output**: Trained model; cross-validation scores (~0.85–0.90 mean).
* **Reason**: Core to the project’s goal; model details shared in README.md and HTML for transparency.
* **Code**:

python

dt\_model = DecisionTreeClassifier(max\_depth=5, min\_samples\_split=10, random\_state=42)

dt\_model.fit(X\_train, y\_train)

cv\_scores = cross\_val\_score(dt\_model, X\_train, y\_train, cv=5)

print("Cross-Validation Scores:", cv\_scores)

print("Mean CV Score:", cv\_scores.mean())

**Step 6: Evaluate Model**

* **What It Does**: Predicts on test set, computes accuracy, confusion matrix, and classification report.
* **Why Perform**: Assesses model performance; visualizations communicate results.
* **Tools Used**: scikit-learn (accuracy\_score, confusion\_matrix, classification\_report), seaborn, matplotlib.
* **Why These Tools**: scikit-learn provides standard metrics; seaborn/matplotlib create clear visuals.
* **Output**:
  + Accuracy: ~85–90%.
  + Confusion Matrix: 2x2 heatmap (e.g., ~50 true negatives, ~20 true positives).
  + Classification Report: Precision/recall/F1-score for classes 0 and 1.
  + Saved plot: visualizations/confusion\_matrix.png.
* **Reason**: Validates model effectiveness; outputs enhance HTML and GitHub repository.
* **Code**:

python

y\_pred = dt\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Test Set Accuracy:", accuracy)

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.savefig('visualizations/confusion\_matrix.png')

plt.show()

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

**Step 7: Visualize Feature Importance**

* **What It Does**: Plots the importance of features (Gender\_Male, Age, Salary).
* **Why Perform**: Identifies key predictors (e.g., Age, Salary dominate).
* **Tools Used**: seaborn, matplotlib.
* **Why These Tools**: seaborn simplifies bar plots; matplotlib saves images.
* **Output**: Bar plot showing feature importance; saved as visualizations/feature\_importance.png.
* **Reason**: Enhances interpretability; shared in HTML and GitHub for insights.
* **Code**:

python

feature\_names = ['Gender\_Male', 'Age', 'Salary']

plt.figure(figsize=(8, 5))

sns.barplot(x=dt\_model.feature\_importances\_, y=feature\_names)

plt.title('Feature Importance in Decision Tree')

plt.xlabel('Importance')

plt.ylabel('Feature')

plt.savefig('visualizations/feature\_importance.png')

plt.show()

**Step 8: Visualize Decision Tree**

* **What It Does**: Generates text-based Decision Tree rules (graphical optional with Graphviz).
* **Why Perform**: Makes the model interpretable by showing decision rules.
* **Tools Used**: scikit-learn (export\_text).
* **Why This Tool**: export\_text is lightweight and doesn’t require Graphviz.
* **Output**: Text rules (e.g., “if Age <= 0.5, then…”).
* **Reason**: Enhances model transparency; included in HTML and GitHub.
* **Code**:

python

from sklearn.tree import export\_text

tree\_rules = export\_text(dt\_model, feature\_names=feature\_names)

print("Decision Tree Rules:\n", tree\_rules)

**Step 9: Export Notebook to HTML**

* **What It Does**: Converts the executed notebook to HTML with embedded plots.
* **Why Perform**: Creates a shareable, static report for non-technical audiences.
* **Tools Used**: nbformat, nbconvert.
* **Why These Tools**: Automate notebook execution and HTML export.
* **Output**: iphone\_purchase\_analysis.html with text and images.
* **Reason**: Facilitates sharing on GitHub and LinkedIn; accessible without Jupyter.
* **Code** (from create\_and\_execute\_notebook.py):

python

html\_exporter = HTMLExporter()

html\_data, \_ = html\_exporter.from\_notebook\_node(nb)

with open('iphone\_purchase\_analysis.html', 'w', encoding='utf-8') as f:

f.write(html\_data)