**Title: Battery Health Prediction Using Machine Learning and Flask Web App**

**🧩 Problem Statement**

🔋“Smartphone and laptop users often face battery performance issues but don’t have an easy way to estimate their battery health. The goal is to create a tool that predicts battery health (%) based on user usage data like charge cycles, temperature, charging habits, and device age.”

**✅ Objectives**

* Predict the battery health of a device using ML.
* Provide a user-friendly web interface.
* Help users understand when they should optimize or replace their battery.

**🔍 Step-by-Step Process**

**Step 1: Data Collection**

📄 File: battery\_health\_prediction\_20000.csv

* Collected 20,000 samples of devices with features like:
  + Device Type (Smartphone, Tablet, etc.)
  + Battery Capacity (in mAh)
  + Charge Cycles
  + Avg Temperature
  + Charge Time
  + Fast Charging
  + Screen On Time
  + Device Age (months)
  + Battery Health (%)

**Step 2: Data Preprocessing**

🛠️ Performed in Battery\_Health.ipynb

* Converted categories like "Yes"/"No" to 1/0
* Encoded device type using numbers (e.g., Smartphone = 0)
* Removed or handled any missing/null values
* Scaled or normalized data if needed

**Step 3: Model Training**

📊 Performed in the notebook (Battery\_Health.ipynb)

* Used scikit-learn library
* Trained models like:
  + Linear Regression
  + Random Forest (optional)
* Chose the model with **best accuracy**
* Saved model as battery\_health\_model.joblib using joblib

**Step 4: Web Application Development**

🖥️ File: app.py (Flask App)

* Flask framework used to build the web app
* Routes created:
  + / → Homepage
  + /predict → Prediction form and result
  + /contact → Contact page
* When user fills form, backend:
  + Takes inputs
  + Converts to ML-compatible format
  + Loads trained model
  + Predicts battery health
  + Displays feedback + colored progress bar

**Step 5: User Input Form (HTML Template)**

📥 Inputs collected:

* Device Type
* Battery Capacity (mAh)
* Charge Cycles
* Avg. Temperature (°C)
* Avg. Charge Time (hrs)
* Fast Charging: Yes/No
* Screen On Time (hrs)
* Device Age (months)

**Step 6: Prediction Output**

🔁 Backend gives:

* Prediction (e.g., 82%)
* Visual bar with color:
  + Green ✅ (Excellent health)
  + Orange ⚠️ (Moderate)
  + Red 🔻 (Poor)
* Suggestion message (optimize/replace/etc.)

**Step 7: Required Libraries**

📦 In requirements.txt

| **Library** | **Purpose** |
| --- | --- |
| numpy | Numerical operations |
| pandas | Data handling |
| matplotlib | Visualizations (if used) |
| seaborn | Statistical plots (optional) |
| scikit-learn | Machine learning model |
| joblib | Saving and loading ML models |

**🌟 Final Output**

* A working website where anyone can enter battery info and get instant battery health prediction.
* Suggests whether user should continue using, optimize, or replace their battery.

**📌 Tools & Technologies**

* Python
* Flask (for backend)
* HTML/CSS (for frontend)
* Pandas, Numpy, Scikit-learn (for ML)
* Joblib (for saving model)
* Jupyter Notebook (for model development)

**📈 Impact**

* Helps users avoid battery failure
* Improves awareness of device health
* Could be expanded into a full mobile app or service for electronics companies

**🔍 1. What is the problem your project is solving?**

**Answer:**  
Many people don’t know the actual condition of their device’s battery. My project helps them predict battery health (%) using their usage data like charge cycles, age, and temperature.

**🧠 2. Why did you choose this project?**

**Answer:**  
Because battery issues are very common, and there’s no easy tool for regular users to check their battery health. I wanted to solve this real-world problem using ML and web development.

**📁 3. What data did you use? Where did you get it from?**

**Answer:**  
I used a dataset of 20,000 battery records with features like device type, charge cycles, temperature, fast charging, etc. (You can mention it was self-created or synthetically generated if applicable.)

**🔄 4. How did you clean or preprocess the data?**

**Answer:**

* Converted Yes/No into 1/0
* Encoded device type (e.g., Smartphone = 0)
* Handled missing values
* Normalized or scaled the data if required

**🤖 5. Which machine learning model did you use and why?**

**Answer:**  
I used **Linear Regression** because it fits well for predicting percentage values. I also tried **Random Forest** for comparison and selected the one with better accuracy.

**📊 6. How did you evaluate the model?**

**Answer:**  
Using metrics like **R² Score** and **Mean Squared Error (MSE)**. I also split the data into training and testing sets to avoid overfitting.

**💾 7. How did you save and use the model in your app?**

**Answer:**  
I used joblib to save the trained model and loaded it in my Flask app to make predictions when the user submits the form.

**🌐 8. What is Flask and how did you use it?**

**Answer:**  
Flask is a Python framework for building web apps. I used it to create pages like home, contact, and predict. It connects the HTML form with my Python code and shows the output.

**🖥️ 9. What happens when a user enters data and clicks predict?**

**Answer:**

* The form sends data to Flask (app.py)
* Flask processes the input and calls the model
* The model gives battery health %
* The result is shown with color and message (like Good or Poor)

**🛠️ 10. What challenges did you face in this project?**

**Answer (Example):**

* Preparing the dataset properly
* Deciding which features affect battery health most
* Integrating ML model with Flask smoothly
* Creating a clear and simple user interface

**✅ 11. What are the final results of your project?**

**Answer:**  
A working web app where users can check battery health by entering their usage data. It gives a percentage and a color-coded suggestion.

**🚀 12. How can you improve this project in the future?**

**Answer:**

* Add real-time data input from a mobile app
* Add battery usage graph
* Train model on more real-world data
* Create separate models for different device types

BOX PLOT

**Final Output:**

You will get **one boxplot for each numeric column**, showing the spread of data and outliers, helping in data **visualization** and **cleaning** (like handling outliers later).

**🧠 Why is this useful?**

Boxplots are useful for:

* Checking **data quality**
* Finding **outliers** (which you may want to remove or fix)
* Understanding **how spread out** your data is

BAR CHART

* Each bar = average battery health for that device type
* Helps you **compare** performance across device categories

**✅ Why this plot is useful:**

* Helps you **see which device types** have better or worse average battery health.
* Useful in analysis, reporting, and model feature understanding.

LINE CHART

* Each point = average battery health at that age
* The **line connects points** to show the trend
* Helps you **visualize the decline** (or pattern) of battery health as time goes on

**🧠 Why this is useful:**

* Shows **how fast** battery health drops with age.
* Helps in **battery performance analysis**, **lifetime estimation**, or **model training** for prediction.

HISTOGRAM

* Tall bars = more devices in that range
* Smooth green curve = shape of the data distribution

**✅ Why this is useful:**

* Helps detect **normal ranges** and **extreme values**
* Great for understanding **how battery health is spread across all devices**
* Useful before modeling to know if data is **normally distributed**, **skewed**, or **has outliers**

PIE CHART

**Why this chart is useful:**

* Helps **visualize proportions** of devices with and without fast charging
* Easy to understand for reports and presentations

SCATTER PLOT

* **Each dot = one device**
* **Color = device type**
* You might see trends (like more screen-on time → lower battery health)

**✅ Why this is useful:**

* Helps **visualize the relationship** between screen usage and battery health
* You can also **compare device types** using colors
* Great for spotting **clusters**, **patterns**, or **outliers**

BAR CHART

* You’ll see **two main groups** (Yes/No or 1/0)
* Each group has **colored bars** for each device type (e.g., blue for Mobile, orange for Laptop)

**✅ Why this is useful:**

* Helps you compare how **different device types** are spread across **fast charging support**
* Useful in EDA (Exploratory Data Analysis) and visual presentations

| **Library** | **Purpose** | **Example Use** |
| --- | --- | --- |
| **pandas (pd)** | **Data loading & manipulation** | **Read CSV, clean data** |
| **numpy (np)** | **Math operations** | **Mean, std, random values** |
| **seaborn (sns)** | **Stylish visualizations** | **Boxplots, histograms, heatmaps** |
| **matplotlib (plt)** | **Basic plotting** | **Line, bar, scatter plots** |
| **train\_test\_split** | **Train-test data split** | **Split data before modeling** |
| **RandomForestRegressor** | **Machine learning algorithm** | **Predict battery health (regression)** |
| **LabelEncoder** | **Encode text labels to numbers** | **Convert "Yes"/"No" to 1/0** |
| **mean\_squared\_error** | **Model performance metric** | **Measure error in prediction** |
| **r2\_score** | **Model performance metric (R²)** | **Measure model accuracy** |
| **joblib** | **Save/load models** | **Save your trained model to use later** |

**What is the use of joblib or pickle files?**

**👉 Main Use:**

To **store trained machine learning models** (or any Python object) so that you can **reuse** them later **without retraining**.

**🔹 Example Use Case:**

Imagine you trained a Random Forest model. It took 2 hours to train. Instead of retraining every time, you can save it using joblib or pickle, and later load it instantly.

**✅ Difference between joblib and pickle**

| **Feature** | **pickle** | **joblib** |
| --- | --- | --- |
| Library Type | Built-in Python library | External library (pip install joblib) |
| Speed | Slower for large **NumPy arrays** | Faster with large NumPy data |
| File Size | Larger sometimes | Smaller with large models |
| Best For | Simple Python objects | Machine Learning models with NumPy |

**Why use train\_test\_split?**

To **evaluate model performance** on unseen data by splitting the dataset into:

* **Training set**: used to train the model
* **Testing set**: used to test accuracy and performance

There are **three main types of machine learning models**, based on how they learn from data:

**1. Supervised Learning**

The model learns from **labeled data** (i.e., input and correct output are both given).

**🔸 Examples:**

* Predicting house price
* Email spam detection
* Student pass/fail prediction

**🔸 Algorithms:**

| **Type** | **Algorithm Examples** |
| --- | --- |
| Classification | Logistic Regression, Decision Tree, Random Forest, KNN, SVM |
| Regression | Linear Regression, Ridge, Lasso, SVR |

**🔸 Code Example:**

python

Copy code

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X\_train, y\_train)

**✅ 2. Unsupervised Learning**

The model learns from **unlabeled data** (no correct output is given). It tries to find patterns or structure.

**🔸 Examples:**

* Customer segmentation
* Grouping similar products
* Detecting anomalies

**🔸 Algorithms:**

| **Type** | **Algorithm Examples** |
| --- | --- |
| Clustering | K-Means, Hierarchical Clustering |
| Association | Apriori, Eclat |
| Dimensionality Reduction | PCA, t-SNE |

**🔸 Code Example:**

python

Copy code

from sklearn.cluster import KMeans

model = KMeans(n\_clusters=3)

model.fit(X)

**✅ 3. Reinforcement Learning**

The model learns by **interacting with an environment** and **receiving rewards or penalties**.

**🔸 Examples:**

* Game playing AI (like Chess or Go)

Here’s the **difference between Classification and Regression** in **simple language**, with examples and a comparison table:

| **Feature** | **Classification** | **Regression** |
| --- | --- | --- |
| Goal | Predict class or category | Predict a continuous value |
| Output | Discrete (e.g., 0, 1, Yes/No) | Continuous (e.g., 12.5, 99.9) |
| Example | Spam or Not Spam | Predict house price |
| Algorithms | Logistic Regression, SVM (Classifier) | Linear Regression, SVR |
| Evaluation Metric | Accuracy, Precision, F1-Score | RMSE, MAE, R² Score |

| **what is confusion matrix-**  **Confusion Matrix is used for:** |
| --- |
| ✅ Evaluating classification performance |
| ✅ Understanding where the model is making mistakes |
| ✅ Calculating metrics like precision, recall, F1 |

**1. R² Score (R-squared or Coefficient of Determination)**

**🔹 Used In:**

🔸 **Regression problems** (when predicting continuous numbers)

**🔹 Definition:**

It tells **how well the model explains the variation in the actual data**.

**Meaning:**

* **1.0** → Perfect prediction
* **0.0** → Model predicts no better than the average
* **< 0.0** → Model performs worse than guessing

**🔹 Example:**

Predicting house prices.  
If R² = 0.85 → 85% of the variation in prices is explained by your model.

**2. Accuracy Score**

**🔹 Used In:**

🔸 **Classification problems** (when predicting categories like Yes/No, Spam/Not Spam)

**🔹 Definition:**

It is the **percentage of correctly predicted labels** out of all predictions.

**Meaning:**

* **Accuracy = 1.0 (100%)** → All predictions correct
* **Accuracy = 0.8 (80%)** → 80% predictions correct

**🔹 Example:**

Predicting if an email is spam.  
If 90 out of 100 emails are predicted correctly → Accuracy = 90%

Here’s a **simple explanation** of **Logistic Regression**, **Precision**, and **Recall** — with real-life examples and code if needed.

**✅ 1. Logistic Regression**

**🔹 What is it?**

A **classification algorithm** used to predict **binary** or **multi-class outcomes**.

Despite its name, it is **used for classification**, not regression.

**🔸 Example:**

* Predict if an email is **Spam (1)** or **Not Spam (0)**
* Predict if a customer will **buy (1)** or **not buy (0)**

**🔸 How it works:**

* Uses a **sigmoid function** to output a probability between 0 and 1.
* If the probability > 0.5 → Class 1  
  If the probability ≤ 0.5 → Class 0

**🔸 Sigmoid function:**

σ(z)=11+e−z\sigma(z) = \frac{1}{1 + e^{-z}}σ(z)=1+e−z1​

**🔸 Simple Python Code:**

python

Copy code

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train, y\_train)

predictions = model.predict(X\_test)

**✅ 2. Precision and Recall**

These are **evaluation metrics** used in **classification problems**.

**🔹 Precision**

Out of all **predicted positives**, how many were **actually positive**?

🔸 High precision = Few **false positives**

**🔸 Example:**

You predicted 100 people have a disease.  
Only 80 actually have it.

Precision = 80 / 100 = **0.8 or 80%**

**🔹 Recall (Sensitivity)**

Out of all **actual positives**, how many did we **correctly predict**?

🔸 High recall = Few **false negatives**

**🔸 Example:**

There are 100 people who actually have the disease.  
You only predicted 80 of them.

Recall = 80 / 100 = **0.8 or 80%**