Documentation of AI Developer Task - Devices Price Classification System using Python and Spring Boot

Outline

1. EDA and Training the Model
2. Building RESTful API Endpoint
3. Building Spring Boot Endpoints
4. Testing the Functionalities Using Postman

EDA and Training the Model

1. Data Preparation:

- Data is downloaded in CSV format (`train-train.csv`, `test-test.csv`).
- CSV files are converted to SQLite database files (`train-train.db`, `test-test.db`).

Code Snippets:

```python
# Get the current working directory
current_directory = os.getcwd()

# Join the current directory with the file path
csv_file_path_train  = os.path.join(current_directory, '..', 'Maids Project', 'Dataset', 'train - train.csv')
csv_file_path_test  = os.path.join(current_directory, '..', 'Maids Project', 'Dataset', 'test - test.csv')

# Read the CSV file into a DataFrame
df_train = pd.read_csv(csv_file_path_train,header=0)
df_test = pd.read_csv(csv_file_path_test,header=0)

# Connect to the SQLite database (creates a new database if the file doesn't exist)
conn_tr = sqlite3.connect('Dataset/train-train.db')
conn_te = sqlite3.connect('Dataset/test-test.db')

# Use pandas to insert the DataFrame contents into the SQLite database
df_train.to_sql('devices', conn_tr, if_exists='replace', index=False)
df_test.to_sql('devices', conn_te, if_exists='replace', index=False)

# Commit the changes and close the connection
conn_tr.commit()
conn_tr.close()
conn_te.commit()
conn_te.close()
```

to show the head of the data to know the features.

```python
# Display the the head DataFrame
df=df_train
df.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram
0	842	0	2.2	0	1.0	0.0	7.0	0.6	188.0	2.0	...	20.0	756.0	2549.0
1	1021	1	0.5	1	0.0	1.0	53.0	0.7	136.0	3.0	...	905.0	1988.0	2631.0
2	563	1	0.5	1	2.0	1.0	41.0	0.9	145.0	5.0	...	1263.0	1716.0	2603.0
3	615	1	2.5	0	0.0	0.0	10.0	0.8	131.0	6.0	...	1216.0	1786.0	2769.0
4	1821	1	1.2	0	13.0	1.0	44.0	0.6	141.0	2.0	...	1208.0	1212.0	1411.0

5 rows × 21 columns

Now that we have the database in SQLite format, we can utilize pandas DataFrame to perform Exploratory Data Analysis (EDA) to discover and gain deeper insights into the data. Let's proceed with the analysis.

EDA

Dataset features:

-The training data contains 21 features are as follows

- id - ID
- battery_power - Total energy a battery can store in one time measured in mAh
- blue - Has Bluetooth or not
- clock_speed - The speed at which the microprocessor executes instructions
- dual_sim - Has dual sim support or not
- fc - Front Camera megapixels
- four_g - Has 4G or not
- int_memory - Internal Memory in Gigabytes
- m_dep - Mobile Depth in cm
- mobile_wt - Weight of mobile phone
- n_cores - Number of cores of the processor
- pc - Primary Camera megapixels
- px_height - Pixel Resolution Height
- px_width - Pixel Resolution Width
- ram - Random Access Memory in Megabytes
- sc_h - Screen Height of mobile in cm
- sc_w - Screen Width of mobile in cm
- talk_time - longest time that a single battery charge will last when you are
- three_g - Has 3G or not
- touch_screen - Has touch screen or not
- wifi - Has wifi or not
- price_range - This is the target variable with the value of:
- 0 (low cost)
- 1 (medium cost)
- 2 (high cost)
- 3 (very high cost)

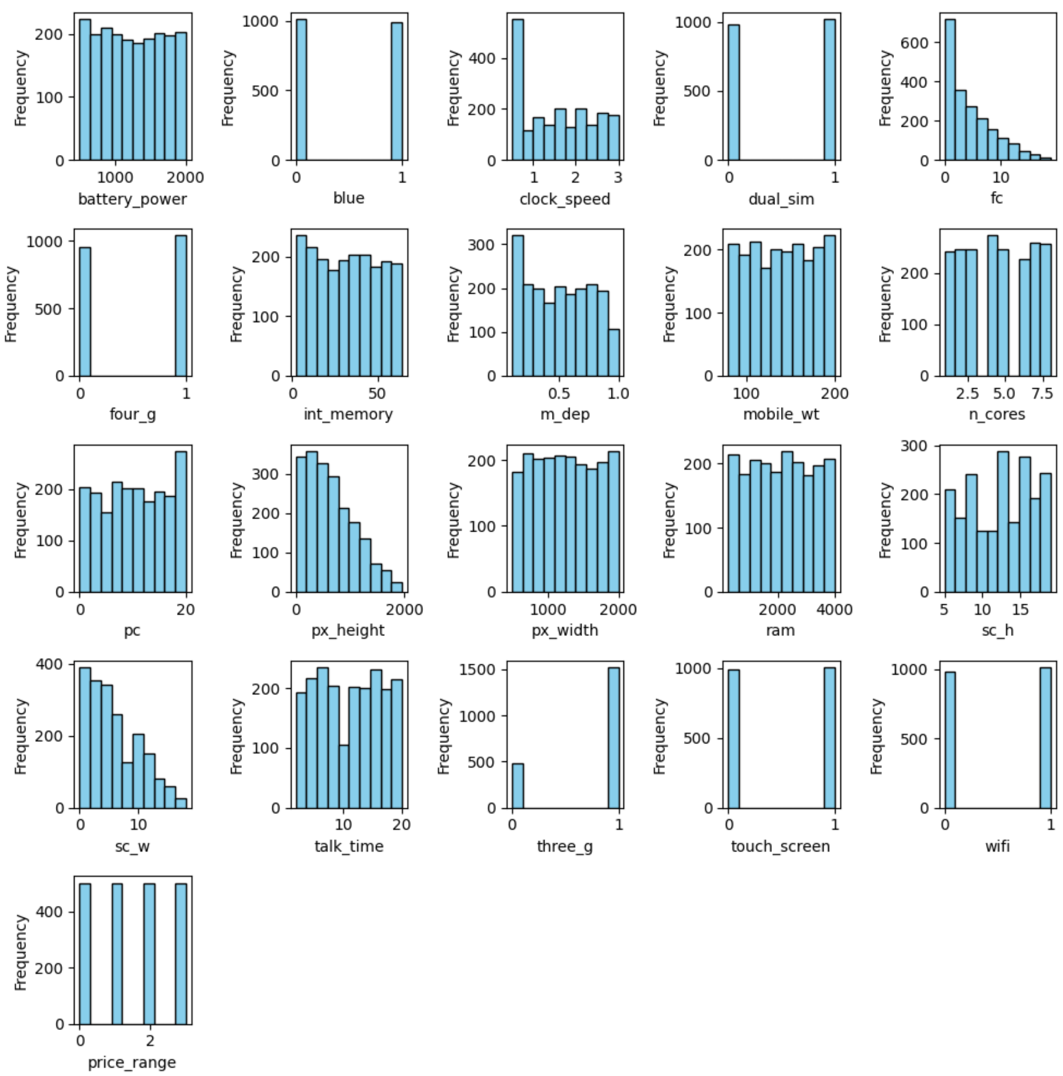
Now that we know the dataset has 21 features, let's determine its size:

Code Snippet:

```
print(f'The train size is {df.shape}')
print(f'Types of attributes are :\n{df.dtypes}')
```

```
The train size is (2000, 21)
Types of attributes are :
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc               float64
four_g           float64
int_memory       float64
m_dep            float64
mobile_wt        float64
n_cores          float64
pc               float64
px_height        float64
px_width         float64
ram              float64
sc_h             float64
sc_w             float64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object
```

- To visualize the distribution of the features in the dataset



Insights from previous figure

- ➔ Certain features such as 'three_g', 'touch_screen', 'wifi', 'dual_sim', 'blue', and 'four_g' are represented as binary values (0 or 1), where 1 indicates the presence of the feature and 0 indicates absence.
- ➔ The scale of features varies; for example, 'm_dep' has a scale from 0 to 1, while 'battery_power' has a scale from 0 to 2000. Normalizing or standardizing these features to a consistent scale will be necessary during the preprocessing step to ensure fair comparisons.
- ➔ The 'fc' feature exhibits skewness, which may require transformation to achieve a more symmetrical distribution and improve model performance.
- ➔ The target variable 'price_range' is categorized into four categories. The goal of our mission is to predict the price range of the device based on the given specifications, making this a multiclass classification problem.

Correlation

- ➔ The second step in Exploratory Data Analysis (EDA) involves identifying correlations to understand which features contribute most significantly to the 'price_range'. This analysis helps us uncover relationships between variables and determine which features are most influential in predicting the price range of the device.

```
# Calculate the correlation matrix
corr_matrix = df.corr()
# Sort the correlation matrix by values in descending order
corr_matrix['price_range'].sort_values(ascending=False)
```

```
price_range      1.000000
ram              0.917119
battery_power    0.200723
px_width         0.166094
px_height        0.148184
int_memory       0.042589
sc_w             0.038777
pc               0.034639
fc               0.024999
three_g          0.023611
sc_h             0.023300
talk_time        0.021859
blue             0.020573
wifi             0.018785
dual_sim         0.017444
four_g           0.015494
n_cores          0.004598
m_dep           -0.000083
clock_speed     -0.006606
touch_screen    -0.030411
mobile_wt       -0.032457
Name: price_range, dtype: float64
```

- ➔ The correlation analysis reveals that the 'ram' feature has the strongest positive correlation (0.917) with the 'price_range', indicating that it significantly impacts the mobile device's price. This finding emphasizes the importance of the 'ram' feature in predicting the price range and will be a crucial factor to consider during model training.

Now that we have a good insight into the dataset, it's time to clean and prepare the data for the training process.

Data Preprocessing

Handling Missing Values is the first step in the preprocessing phase. It involves identifying and addressing any missing or null values in the dataset. There are several methods to handle missing values, including imputation, removal of rows or columns with missing values, or using algorithms that can handle missing values directly.

Given our previous insight that missing values are very small compared to the dataset size, the best approach is to simply drop the instances corresponding to these missing values. This ensures that our analysis is not skewed by the presence of missing data and maintains the integrity of the dataset.

Code Snippet

```python
# display missing values
print(df.isnull().sum())
```

```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               5
four_g           5
int_memory       5
m_dep            5
mobile_wt        4
n_cores          4
pc               5
px_height        4
px_width         2
ram              2
sc_h             1
sc_w             1
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

- ➔ As shown, the highest number of missing values is 5, which represents only 0.25% of the total dataset size (5 * 100 / 2000). Therefore, the best option is to drop these rows, as it will have minimal impact on the overall dataset size and ensures that our analysis is not significantly affected by missing values.
- ➔ The shape after dropping missing value is (1991, 21)

Data Normalization

Normalizing the features ensures that they are on a similar scale, which can help the model converge faster during training. This step is crucial for improving the model's performance and stability, especially when features have different scales or units.

For the normalization process, we will utilize the MinMaxScaler module from the sklearn library, a powerful tool in machine learning. The formula for Min-Max scaling, also known as normalization, is given by:
Xscaled = X−Xmin / Xmax−Xmin
where:

- X is the original feature value,
- Xmin is the minimum value of the feature in the dataset,
- Xmax is the maximum value of the feature in the dataset, and
- Xscaled is the normalized feature value.

This formula scales the feature values to a range between 0 and 1, with 0 corresponding to the minimum value in the dataset and 1 corresponding to the maximum value. Using this approach ensures that all features are on a similar scale, which can help the model converge faster during training.

Note: We need to scale any data that will be used with the trained model to predict prices. Before feeding any data to the trained model, we must ensure that this data is scaled from 0 to 1.

Data after scaling :

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt
count	1991.000000	1991.000000	1991.00000	1991.000000	1991.000000	1991.000000	1991.000000	1991.000000	1991.000000
mean	0.492499	0.496233	0.40898	0.510799	0.226731	0.520844	0.485070	0.446677	0.502193
std	0.293681	0.500111	0.32620	0.500009	0.228407	0.499691	0.292551	0.320691	0.294990
min	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.234135	0.000000	0.08000	0.000000	0.052632	0.000000	0.225806	0.111111	0.241667
50%	0.483634	0.000000	0.40000	1.000000	0.157895	1.000000	0.483871	0.444444	0.508333
75%	0.744489	1.000000	0.68000	1.000000	0.368421	1.000000	0.741935	0.777778	0.750000
max	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

We can observe that the scales of the features are very close to each other, which is beneficial for the model's convergence speed. This similarity in scale reduces the risk of certain features dominating others during the training process, leading to a more stable and efficient model convergence.

Check skewness

```
skewness = normalized_df.skew()

# Display the skewness value
print("Skewness : \n", skewness)
```

```
Skewness :
 battery_power    0.031851
blue             0.015080
clock_speed      0.177621
dual_sim        -0.043237
fc               1.018337
four_g          -0.083511
int_memory       0.056349
m_dep            0.087387
mobile_wt        0.003900
n_cores          0.007925
pc               0.021024
px_height        0.662986
px_width         0.015752
ram              0.006836
sc_h            -0.098481
sc_w             0.633236
talk_time        0.013605
three_g         -1.224428
touch_screen    -0.007037
wifi            -0.021112
```

➔ We can observe that skewness has been effectively handled, as no values are greater than +1 or less than -1. This indicates that the data distribution is more symmetrical, which can lead to improved model performance and reliability.

Now that our data is ready to be fed into the model, missing data has been handled, and skewness as well as normalization has been addressed, we are moving on to the training model step.

2. Model Training:

- Machine learning model is trained using the data in the SQLite database.

Train the model

I chose the Support Vector Machine (SVM) model for several reasons.

➔ Firstly, the dataset contains 21 features, indicating a high-dimensional space where relationships between features and the target variable may be complex. SVMs are well-suited for high-dimensional datasets and can effectively capture intricate relationships.

➔ Secondly, SVMs are capable of capturing nonlinear relationships in the data. This is crucial as other models may struggle with nonlinearities, whereas SVMs excel at finding complex decision boundaries.

➔ Additionally, our previous insights revealed strong correlations between certain features and the target variable. SVMs support feature selection, allowing us to focus on the most relevant features and potentially improve model performance. This aligns perfectly with our goal of optimizing the model based on these insights.

Overall, the combination of SVM's ability to handle high-dimensional data, capture nonlinear relationships, and support feature selection makes it a suitable choice for this dataset.

Drawbacks

1. Computational Efficiency: SVMs can be computationally expensive, especially with large datasets. Training time may be a concern if your dataset is very large. (This may not be valid as we deal with very small dataset)

2. Scalability: While SVMs can handle high-dimensional data, they may not scale well to very large datasets with millions of samples. (This may not be valid as we deal with very small dataset)

➔ In order to evaluate the performance of our model, we need to have 'Y' values that are not present in the test CSV file. Therefore, we will set aside a portion of the training data to use as a test set. This will allow us to test our model's performance on unseen data and ensure that it generalizes well.

```
X = normalized_df
y = df_clean['price_range']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Training…

```
svm = SVC(kernel='linear', decision_function_shape='ovr',C =100)
svm.fit(X_train, y_train)
```

`kernel='linear'` specifies that a linear kernel should be used.

- If we observe that the model's performance is low and needs to capture nonlinearity, we will adjust the kernel to use nonlinear functions.

`decision_function_shape='ovr'` indicates the 'one-vs-rest' strategy.

- Since the problem is a multiclass classification task, the one-vs-rest strategy is suitable to train one classifier per class.

➔ Calculating the training and testing accuracy

```
# Calculate training accuracy
train_accuracy = svm.score(X_train, y_train)
print(f"Training Accuracy: {train_accuracy}")

# Calculate test accuracy
test_accuracy = svm.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")
```

➔ the obtained **accuracy** is

```
[105]
...  Training Accuracy: 0.9849246231155779
     Test Accuracy: 0.9799498746867168
```

➔ After optimizing the SVM model without using Grid Search, we achieved a peak accuracy of 98% on the training data and 97% on the test data. This optimization process involved manually tuning the hyperparameter C over a range of values from 0.1 to 1000. Additionally, we explored different kernels, including 'poly', 'rbf', and 'linear', to identify the best-performing configuration. The final selection of hyperparameters and kernel choices was made through iterative testing and evaluation, prioritizing a balance between model complexity and generalization performance.

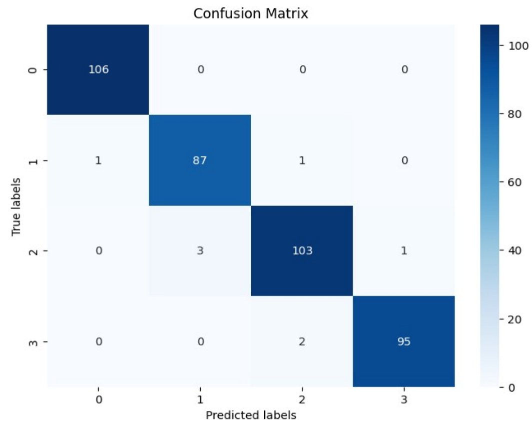
Another matrix to evaluate out model performance:

Confusion Matrix

Code Snippet :

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

y_pred = svm.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

➔



➔ Confusion Matrix insights

Class 0 (low cost):

True Negatives (TN): 101 instances were correctly predicted as not belonging to Class 0.
False Positives (FP): 5 instances were incorrectly predicted as belonging to Class 0 when they actually didn't.
False Negatives (FN): There are no false negatives for Class 0 in this confusion matrix.
True Positives (TP): There are no true positives for Class 0 in this confusion matrix.

Class 1 (medium cost):

True Negatives (TN): There are no true negatives for Class 1 in this confusion matrix.
False Positives (FP): 2 instances were incorrectly predicted as belonging to Class 1 when they actually didn't.
False Negatives (FN): 6 instances were incorrectly predicted as not belonging to Class 1 when they actually did.
True Positives (TP): 86 instances were correctly predicted as belonging to Class 1.

Class 2 (hight cost):

True Negatives (TN): 188 instances were correctly predicted as not belonging to Class 2.
False Positives (FP): 1 instance was incorrectly predicted as belonging to Class 2 when it actually didn't.
False Negatives (FN): 12 instances were incorrectly predicted as not belonging to Class 2 when they actually did.
True Positives (TP): 199 instances were correctly predicted as belonging to Class 2.

Class 3 (very high cost):

True Negatives (TN): 188 instances were correctly predicted as not belonging to Class 3.
False Positives (FP): 1 instance was incorrectly predicted as belonging to Class 3 when it actually didn't.
False Negatives (FN): 12 instances were incorrectly predicted as not belonging to Class 3 when they actually did.
True Positives (TP): 199 instances were correctly predicted as belonging to Class 3.

➔ Last Step is to save to be test it for the firth 10th examples

Code snippet:

Save and Load the model

```python
from joblib import dump, load

dump(svm, 'svm_model.joblib')
```

Then load the model and read records from database to make prediction on the first 10th records.

Code Snippets:

```python
# Connect to the SQLite database
conn = sqlite3.connect('Dataset/test-test.db')

# Load the data from the database
query = "SELECT * FROM devices;"
df = pd.read_sql_query(query, conn)

# Close the connection
conn.close()

# Load the saved SVM model
svm_loaded = load('svm_model.joblib')

# Fit the MinMaxScaler to all records in the DataFrame
scaler = MinMaxScaler()
scaler.fit(df.drop(df.columns[0], axis=1))  # Exclude the ID column

# Prepare the new data for prediction
X_new = df.drop(df.columns[0], axis=1)

# Make predictions for the first 10 devices
predictions = svm_loaded.predict(scaler.transform(X_new.iloc[:10]))

print(f'Price ranges of the first 10 devices are {predictions}')
```

```
Price ranges of the first 10 devices are [3 3 2 3 1 3 3 1 3 0]
```

➔ As the code shows, we first established a connection to the database and loaded all records into a DataFrame. We then closed the connection. Next, we loaded the previously saved model to make predictions. However, we must first normalize the data to be fed into the model. We created a MinMaxScaler to fit the records and applied this only to the first 10 records. Then, we passed the normalized data to the model to make predictions. Note that we dropped the ID column as we did in the training step.

Building RESTful API Endpoint

➔ Now, we will build a RESTful API endpoint using Flask so the model can receive requests from an endpoint, make predictions, and then send the results back to the established endpoint.
➔ First, build the app and load the trained SVM model.
code snippet:

```python
from flask import Flask, request, jsonify
from joblib import load
import sqlite3
import numpy as np
from sklearn.preprocessing import MinMaxScaler

app = Flask(__name__)

# Load the pre-trained SVM model
svm_loaded = load('svm_model.joblib')
```

➔ I implemented a predict function that will perform all necessary functionalities: open a connection with the database, make a query with the given ID, apply normalization to the retrieved record, pass it to the SVM model, and then send the result back to the endpoint URL.

1. establish connection to the database and make query with the passed id

code snippet:

```python
@app.route('/predict/<int:id>', methods=['POST'])
def predict(id):
    """
    Predict the price of a device based on its specifications.

    Args:
        id (int): The ID of the device to predict its price.

    Returns:
        dict: A JSON object containing the predicted price.
    """
    # Connect to the SQLite database
    conn = sqlite3.connect('Dataset/test-test.db')
    cursor = conn.cursor()

    # Query the database to get the record for the specified device I
    cursor.execute(f"SELECT * FROM devices WHERE id = {id}")
    record = cursor.fetchone()

    # Close the database connection
    conn.close()
```

➔ Then, we convert the retrieved record to a NumPy array so we can perform normalization techniques as used in the training process. But first, we must fit a MinMaxScaler to our dataset to properly apply normalization to the fetched record. Then we will make the prediction and returned it back to the endpoint.

```python
conn.close()
if record is None:
    # If no device with the specified ID is found, return a 404 error
    return jsonify({'error': 'Device not found'}), 404
else:
    # Prepare the record for prediction
    record = record[1:]  # Exclude the ID column
    record = np.array([record])  # Convert to a 2D NumPy array
    # Connect to the SQLite database again to retrieve all data
    conn = sqlite3.connect('Dataset/test-test.db')
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM devices")
    all_records = cursor.fetchall()

    # Close the database connection
    conn.close()
    all_records = np.array([record[1:] for record in all_records])

    # Define the MinMaxScaler
    scaler = MinMaxScaler()
    scaler.fit(all_records)
    normalized_record = scaler.transform(record)

    # Use the loaded SVM model to predict the price
    price = svm_loaded.predict(normalized_record)
    # Return the predicted price as a JSON response
    return jsonify({'price': price.tolist()})
```

Building Spring Boot Endpoints

➔ Now, we will build a Spring Boot application so that we can send requests to the Flask endpoint and perform CRUD operations on the database.

All endpoint rquired in the website are implmented

EndPoints: Implement RESTful endpoints to handle the following operations

● GET /api/devices/: Retrieve a list of all devices
● POST /api/devices/{id}: Retrieve details of a specific device by ID.
● POST /api/devices: Add a new device.
● POST /api/predict/{deviceId}
○ This will call the Python API to predict the price, and save the result in the device entity here.

● GET /api/devices/: Retrieve a list of all devices
is implnetd as follwos

```
@GetMapping("/api/devices")
public List<Map<String, Object>> getDevices() {
    /**
     * Retrieve all devices from the database.
     *
     * @return A list of devices.
     */
    String query = "SELECT * FROM devices";
    return jdbcTemplate.queryForList(query);
}
```

● POST /api/devices/{id}: Retrieve details of a specific device by ID.

Note that this endpoint construct json manually so the returned result could be readable.

```
@PostMapping("/api/devices")
public String getRecord(@RequestParam String id) {
    /**
     * Retrieve a specific device record from the database.
     *
     * @param id The ID of the device to retrieve.
     * @return A JSON string representing the device record.
     */
    String query = "SELECT * FROM devices WHERE id = ?";
    Map<String, Object> device = jdbcTemplate.queryForMap(query, id);
    if (!device.isEmpty()) {
        // Construct JSON manually
        return "{"
                + "\"id\":\"" + device.get("id") + "\","
                + "\"battery_power\":" + device.get("battery_power") + ","
                + "\"blue\":" + device.get("blue") + ","
                + "\"clock_speed\":" + device.get("clock_speed") + ","
                + "\"dual_sim\":" + device.get("dual_sim") + ","
                + "\"fc\":" + device.get("fc") + ","
                + "\"four_g\":" + device.get("four_g") + ","
                + "\"int_memory\":" + device.get("int_memory") + ","
                + "\"m_dep\":" + device.get("m_dep") + ","
                + "\"mobile_wt\":" + device.get("mobile_wt") + ","
                + "\"n_cores\":" + device.get("n_cores") + ","
                + "\"pc\":" + device.get("pc") + ","
                + "\"px_height\":" + device.get("px_height") + ","
                + "\"px_width\":" + device.get("px_width") + ","
                + "\"ram\":" + device.get("ram") + ","
                + "\"sc_h\":" + device.get("sc_h") + ","
```

- POST /api/devices: Add a new device.

```java
@PostMapping("/api/device")
public String addDevice(@RequestBody Device device) {
    /**
     * Add a new device record to the database.
     *
     * @param device The device object to add.
     * @return A string indicating the success or failure of adding the device.
     */
    String query = "INSERT INTO devices (id, battery_power, blue, clock_speed, dual_sim, fc, four_g,
            "VALUES (?, ?, ?, ?, ?  int com.maid.endpoint.Device.getBattery_power()  ?)";
    Object[] params = {
            device.getId(),device.getBattery_power(),device.getBlue(),device.getClock_speed(),device
            device.getPx_width(),
            device.getRam(),
            device.getSc_h(),
            device.getSc_w(),
            device.getTalk_time(),
            device.getThree_g(),
            device.getTouch_screen(),
            device.getWifi(),
    };
    try {
        jdbcTemplate.update(query, params);
        return "Device added successfully";
    } catch (Exception e) {
        e.printStackTrace();
        return "Failed to add device";
    }
}
```

- POST /api/predict/{deviceId}

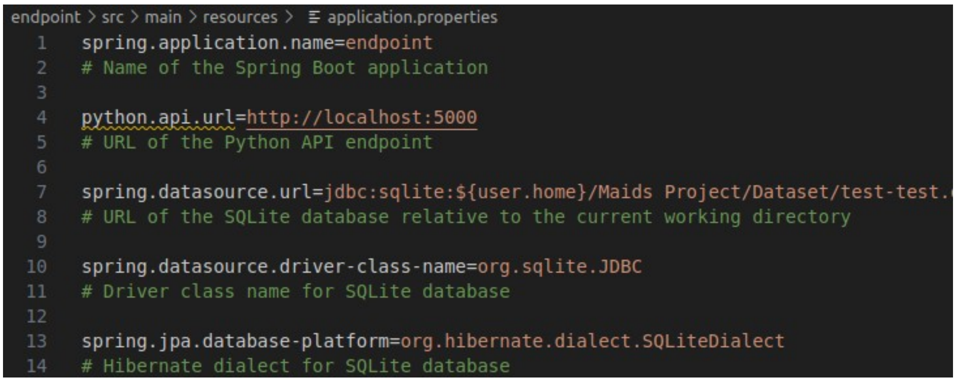
```java
@PostMapping("api/predict")
public String predict(@RequestParam Long id) {
    /**
     * Predict the price of a device based on its specifications.
     *
     * @param id The ID of the device to predict its price.
     * @return A string indicating the success or failure of setting the price range for the devic
     */
    String  String result - com.maid.endpoint.Controller.predict(Long)
    String result = restTemplate.postForObject(url, request:null, responseType:String.class);

    // Parse the JSON response to extract the price range
    try {
        JsonNode jsonNode = objectMapper.readTree(result);
        int price = jsonNode.get(fieldName:"price").get(index:0).asInt();
        // Set the price range in your entity or return it as needed
        // For example, if you have a Device entity
        Device device = new Device();
        device.setId(id.toString());
        device.setPrice_range(String.valueOf(price));

        // Save the device or return a success message
        // Example: deviceRepository.save(device);
        return "Price range of the device corresponding to the given id  " + id + " has been set s

    } catch (IOException e) {
        e.printStackTrace();
        return "Failed to set price range";
    }
```

Note the entity for the device hold its specification such as battery power bucktooth and etc. and constructed as following

```java
public class Device {

    // Unique identifier for the device
    private String id;

    // Device specifications
    private int battery_power;    // Battery power of the device
    private int blue;             // Blue color availability (1 for true, 0 for false)
    private double clock_speed;   // Clock speed of the device
    private int dual_sim;         // Dual SIM support (1 for true, 0 for false)
    private int fc;               // Front camera quality
    private int four_g;           // 4G availability (1 for true, 0 for false)
    private int int_memory;       // Internal memory capacity
    private double m_dep;         // Mobile depth
    private int mobile_wt;        // Mobile weight
    private int n_cores;          // Number of processor cores
    private int pc;               // Primary camera quality
    private int px_height;        // Pixel height of the device
    private int px_width;         // Pixel width of the device
    private int ram;              // RAM capacity of the device
    private int sc_h;             // Screen height of the device
    private int sc_w;             // Screen width of the device
    private int talk_time;        // Talk time of the device
    private int three_g;          // 3G availability (1 for true, 0 for false)
    private int touch_screen;     // Touch screen availability (1 for true, 0 for false)
    private int wifi;             // WiFi availability (1 for true, 0 for false)

    // Predicted price range for the device
    private String price_range;

    // Getters and setters for the device properties
```

➔ these privates specs are accessed and stetted only using the implemented methods for each variable inside the entity.

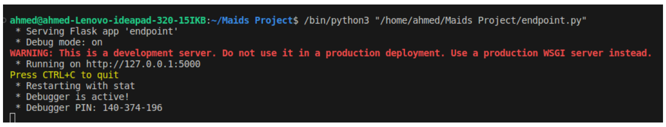
The endpoint configuration to the database and the flask API is defined as following

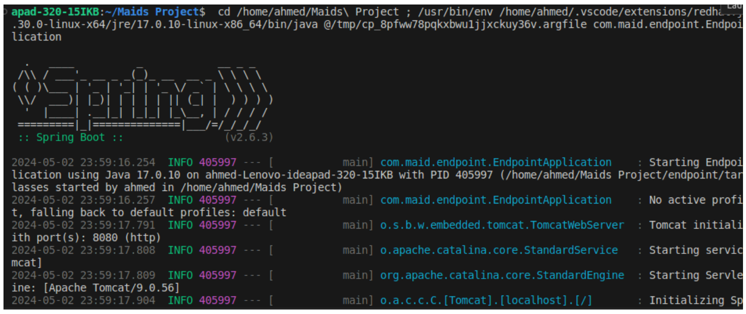


Testing the system Using Postman

1. initialize flask API endpoint

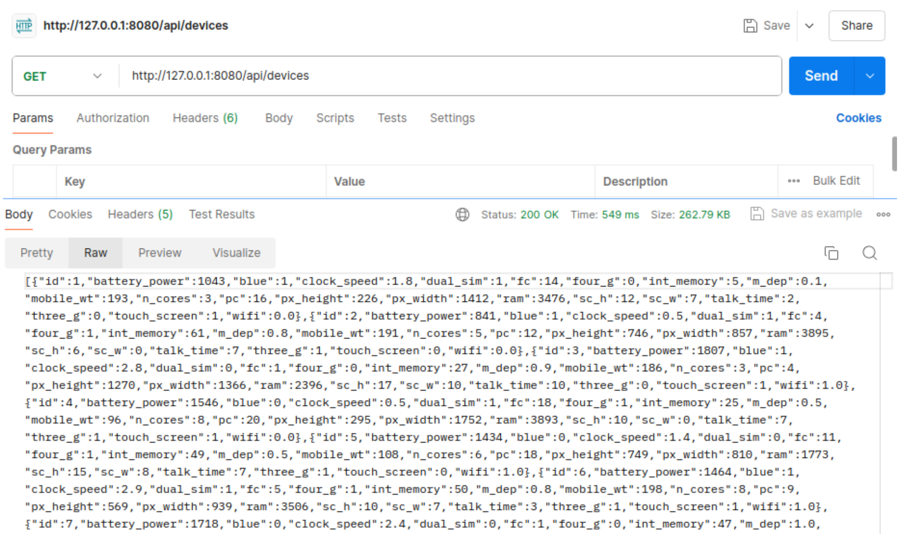
2. initialize spring boot App





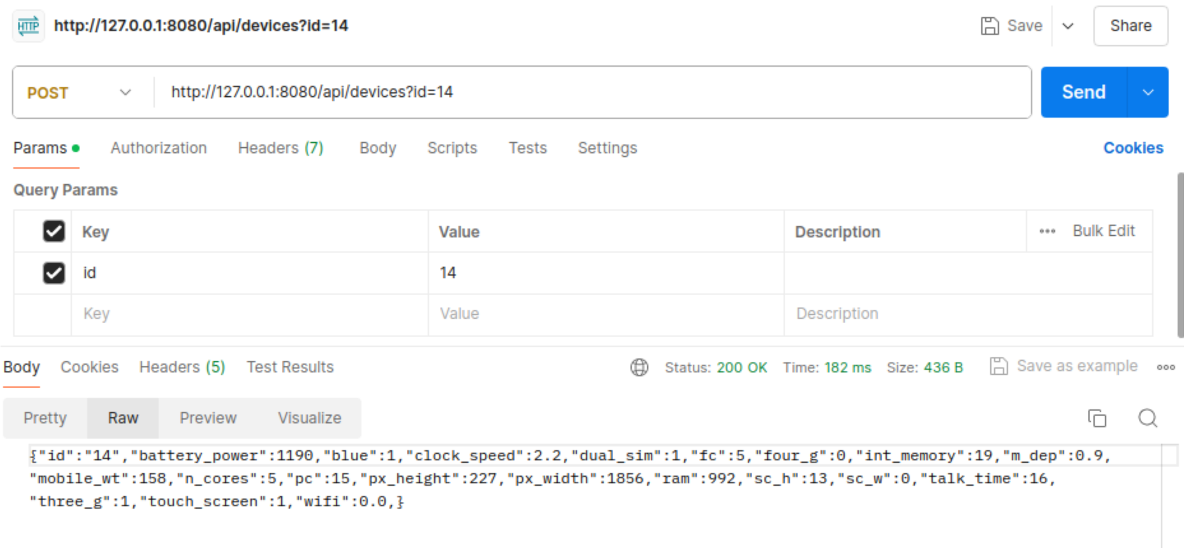
Now the endpoints are initialized and ready for requests. We will test each endpoint to verify if the results are as expected, using Postman.

- # ***GET /api/devices/: Retrieve a list of all devices***



✓ *Works as expected and retrieved all devices in the database*

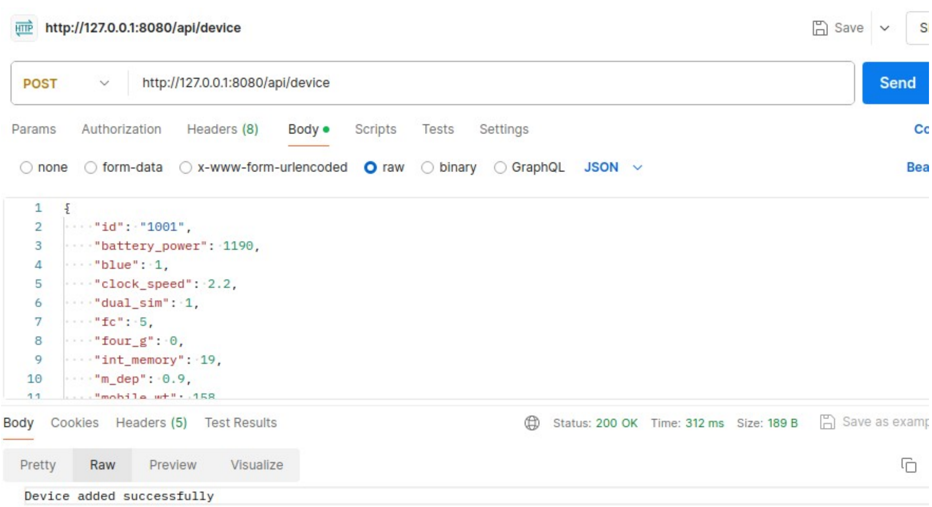
- ***POST /api/devices/{id}: Retrieve details of a specific device by ID.***

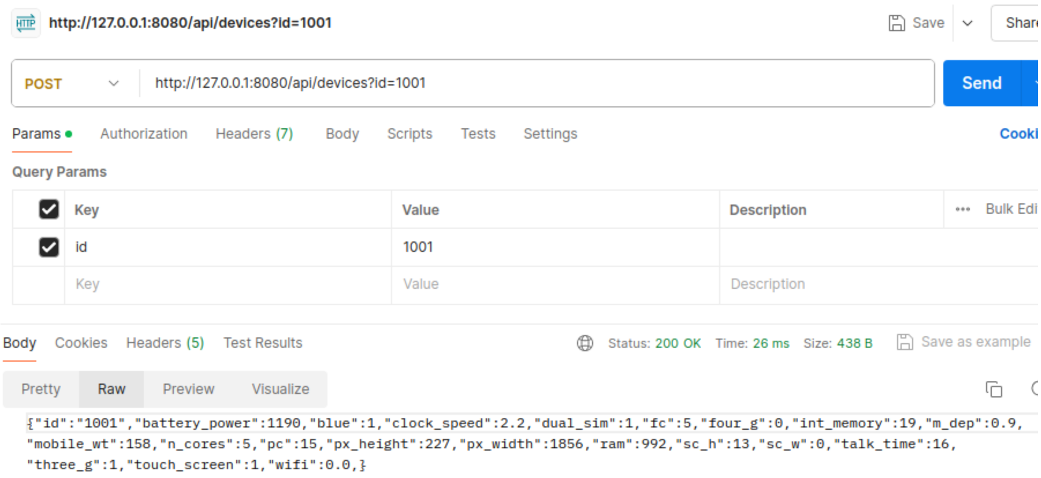


✓ *Works as expected and retrieved specs devices corresponding to the passed id 14 from the database.*

- ***POST /api/devices: Add a new device.***

This endpoint needs data to be passed in the request so it can store the data in the database. Then we should retive the record to make sure that is stored successfully



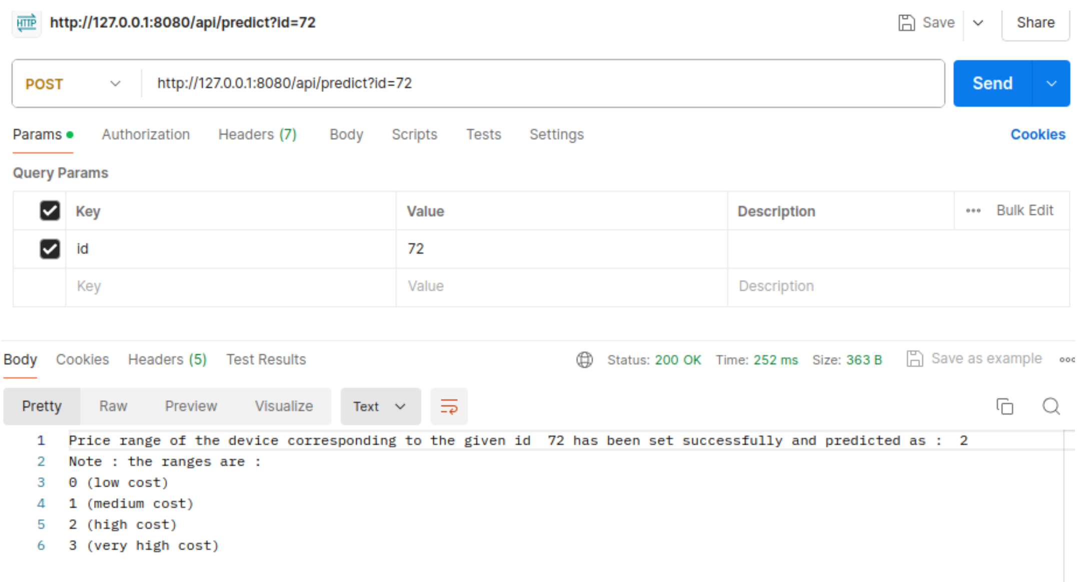


Returned same record which means it stored successfully

- ✓ ***Works as expected and stored specs of the passed devices.***

Last endpoint and the most important that will call Python API to predict the model price then save the result in the entity.

- ***POST /api/predict/{deviceId}***



- ***Works as expected and model predicted the price range of the passed device id as 2 which means high cost device.***

- ***Retrieved from the Python endpoint successfully***

```
TERMINAL                                                                                     Python
ahmed@ahmed-Lenovo-ideapad-320-15IKB:~/Maids Project$ /bin/python3 "/home/ahmed/Maids Project/endpoint.py"
 * Serving Flask app 'endpoint'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 140-374-196
/home/ahmed/.local/lib/python3.10/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but
 SVC was fitted with feature names
  warnings.warn(
127.0.0.1 - - [03/May/2024 00:20:49] "POST /predict/72 HTTP/1.1" 200 -
```

Note we set the price range in the entity as required

```
device.setId(id.toString());
device.setPrice_range(String.valueOf(price));
```

Now I will test 10th random record to show their price range.

ID	Predicted class
46	1
487	2
156	2
789	0
367	2
245	1
716	0
2	3
975	3
1001 (the one we added to the database)	0

Thank you, for this opportunity and for considering my application as AI Engineer. See u