Project Report: Mobile Phone Pricing Classification

1. Introduction

This report details the development of a classification system designed to predict the price range of mobile phones based on a variety of features. The project aims to categorize mobile phones into four price ranges: *`[0: Low, 1: Medium, 2: High, and 3: Very High Cost]`*. This notebook utilizes a variety of techniques including exploratory data analysis and different classification models (`Logistic Regression`, `KNeighbors`, and `Random Forest Classifier`). This project showcases how different classification models can be used and compared to select the best model for the price prediction.

2. Project Objectives

- Develop a multi-class classification model capable of predicting mobile phone price ranges *`[0: Low, 1: Medium, 2: High, and 3: Very High Cost]`*.
- Provide insight into the features impacting price ranges of mobile phones.
- Use different machine learning classification models on the mobile phone dataset.
- Apply exploratory data analysis to understand the data distribution and feature correlations.
- Evaluate the performance of different classification models using a variety of metrics and visualizations.
- Use **MLflow** via **Dagshub** to enable experiment tracking and management.
- Finally, deploy the best model using `Streamlit`.

3. Methodology

The project follows these steps:

3.1. Data Acquisition and Preparation (Sections 1 & 3)

- **GitHub Repository Cloning:** The project starts by cloning a GitHub repository containing the required dataset. This allows for version control and easy access to all necessary files.

```python
!git clone https://github.com/PranayJagtap06/UFM_Mobile_Phone_Pricing.git
```

- **Dataset Extraction:** A zip archive (`mobile_phone_pricing.zip`) containing the mobile phone dataset is extracted to the local working directory. The extraction is done using the zipfile library.

```python
import zipfile
zip_ref = zipfile.ZipFile("/content/UFM_Mobile_Phone_Pricing/mobile_phone_pricing.zip",
'r')
zip_ref.extractall("/content")
zip_ref.close()
```

- **Dataset Inspection:** The notebook uses the `info()` and `isnull().sum()` functions to see if there are any null values, and what are the column types. It confirms there are no missing values.

```python
df.info()
df.isnull().sum()
```

3.2. Exploratory Data Analysis (Section 5)

- **Target Variable Analysis:** The value counts for the target variable (`price_range`) are checked to confirm that the classes are balanced.

```python
df.price_range.value_counts()
```

- **Data Distribution:** The code explores the distribution of key features:
    - `four_g` , `three_g` , `dual_sim` : Counts and visualizes 4G, 3G, and dual SIM presence.
- **Data Visualization:** The code generates several visualizations to understand the relationship between different columns and price ranges:
    - Scatter plot between `battery_power` and `ram` with the size of the datapoints representing the `int_memory` and colored by the `price_range` . Got insights that `battery_power` does not significantly affect price range, as high-capacity batteries are available across all ranges. `ram` and `int_memory` are critical features for premium phones, as seen from their strong positive correlation with price range. Feature Trade-offs in Budget Phones (Price Range 0) tend to compromise on `ram` and `int_memory` while offering competitive `battery_power` .
    - Box plots of `battery_power` across different price categories, providing insights into the distribution of `battery_power` across price ranges. Insights tells that `Higher` price ranges are generally associated with slightly higher battery power, but the overlap suggests that battery power is not a strong differentiator between price ranges. Manufacturers might prioritize other features besides battery power when justifying higher prices, or there might be diminishing returns in battery capacity for premium-priced devices.
    - Scatter plot between `ram` and price ranges, colored based on `price_range` and point sizes based on the `int_memory` . This plot indicates that `ram` plays a vital role in pricing mobile phones. Higher `ram` values lead to higher price ranges, and the size of the data points highlights the impact of internal memory on the cost.
    - Histogram of 3G and 4G availability by price range using the `barmode=group` . The plot suggests a strong correlation between price range and 4G availability. This indicates that phones with higher price tags are more likely to have 4G capabilities. This observation aligns with the expectation that newer and higher-end phones are more likely to incorporate advanced features like 4G.
- **Correlation Matrix:** The code also calculates and displays a correlation matrix which highlights how much different features are correlated with each other. This also gives an insight on how much the features correlate with the price range.

3.3. Data Preparation (Section 6)

- **Feature and Target Separation:** The features (independent variables) and target variable (`price_range`) are separated.

```python
 X = ds.drop("price_range", axis=1)
Y = ds.price_range
```

- **Train-Test Split:** The data is split into training and testing sets using `train_test_split` with a test size of 20% and stratified to maintain class proportions.

```python
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=ds.price_range, random_state=42)
```

3.4. Model Training and Evaluation (Section 8)

The notebook explores three classification models: `Logistic Regression` , `KNeighbors` , and `Random Forest Classifier` . Each model goes through similar steps:

- **Model Pipeline:** Pipelines are defined that include feature scaling and the respective classification models with grid search parameters.

- **Hyperparameter Tuning:** Grid search with cross-validation (`GridSearchCV`) is used to tune the hyperparameters for each model by optimizing the `accuracy`.

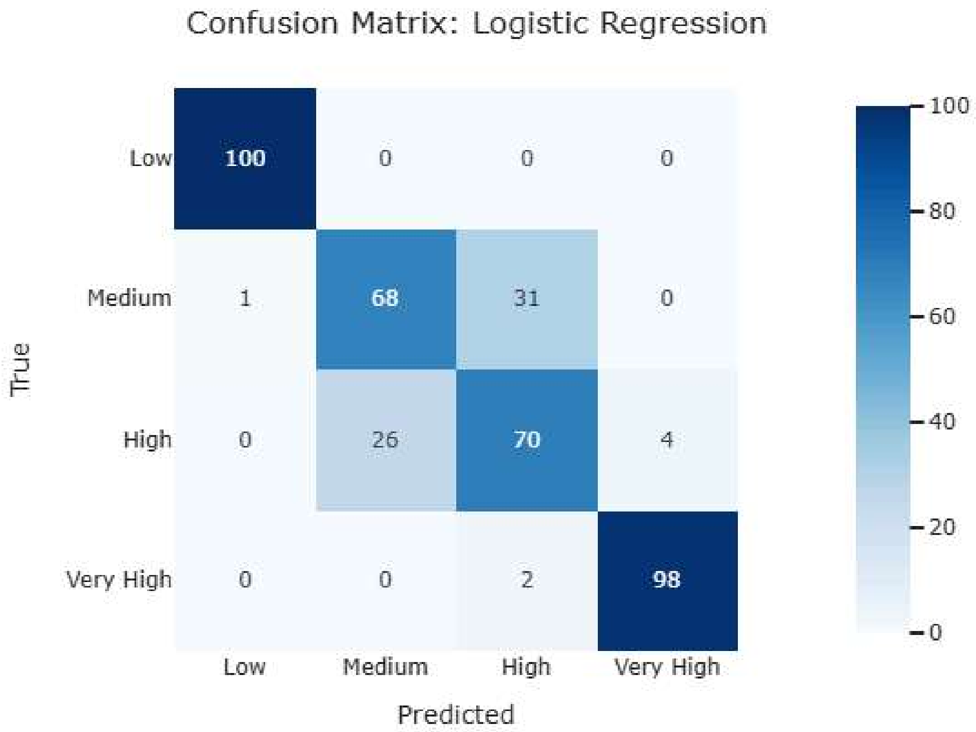
- **Model Fitting:** Each model is fitted using the train set.

- **Performance Evaluation:**

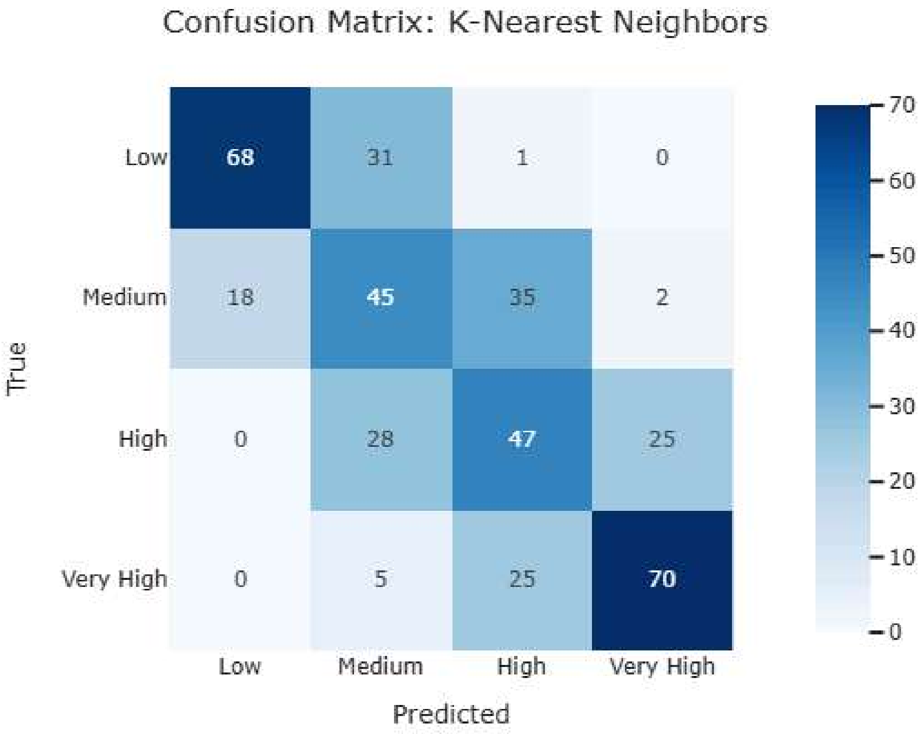
    - **Classification Report:** Generates classification reports that include key metrics such as precision, recall, and F1-score for each class.
        - **Logistic Regression Classifier:**
            - Achieved a training accuracy of 89.38% and a testing accuracy of 84.00%.
            - ```
              Logistic Regression Classification Report:
                            precision    recall  f1-score   support

                         0       0.99      1.00      1.00       100
                         1       0.72      0.68      0.70       100
                         2       0.68      0.70      0.69       100
                         3       0.96      0.98      0.97       100

                  accuracy                           0.84       400
                 macro avg       0.84      0.84      0.84       400
              weighted avg       0.84      0.84      0.84       400
              ```
            - Demonstrated high precision and recall for the `Low Cost` and `Very High Cost` classes but has lower scores on the `Medium Cost` and `High Cost` classes.
            - Indicates some overfitting but generalizes fairly well to unseen data.
        - **K-Nearest Neighbors Classifier:**
            - Shows a testing accuracy of 57.50% and a training accuracy of 100% which signifies a strong overfitting behavior of the model.
            - ```
              K-Nearest Neighbors Classification Report:
                            precision    recall  f1-score   support

                         0       0.79      0.68      0.73       100
                         1       0.41      0.45      0.43       100
                         2       0.44      0.47      0.45       100
                         3       0.72      0.70      0.71       100

                  accuracy                           0.57       400
                 macro avg       0.59      0.57      0.58       400
              weighted avg       0.59      0.57      0.58       400
              ```
            - Struggles to accurately predict all classes, especially `Medium Cost` and `High Cost` categories.
        - **Random Forest Classifier:**
            - Achieved high training and test accuracies (97.81% and 91.00% respectively) , indicating strong performance with some overfitting.

-
  ```
  Random Forest Classifier Classification Report:
                precision    recall  f1-score   support

             0       0.96      0.95      0.95       100
             1       0.86      0.87      0.87       100
             2       0.86      0.87      0.87       100
             3       0.96      0.95      0.95       100

      accuracy                           0.91       400
     macro avg       0.91      0.91      0.91       400
  weighted avg       0.91      0.91      0.91       400
  ```
- Shows excellent prediction capabilities and strong generalization to unseen data.

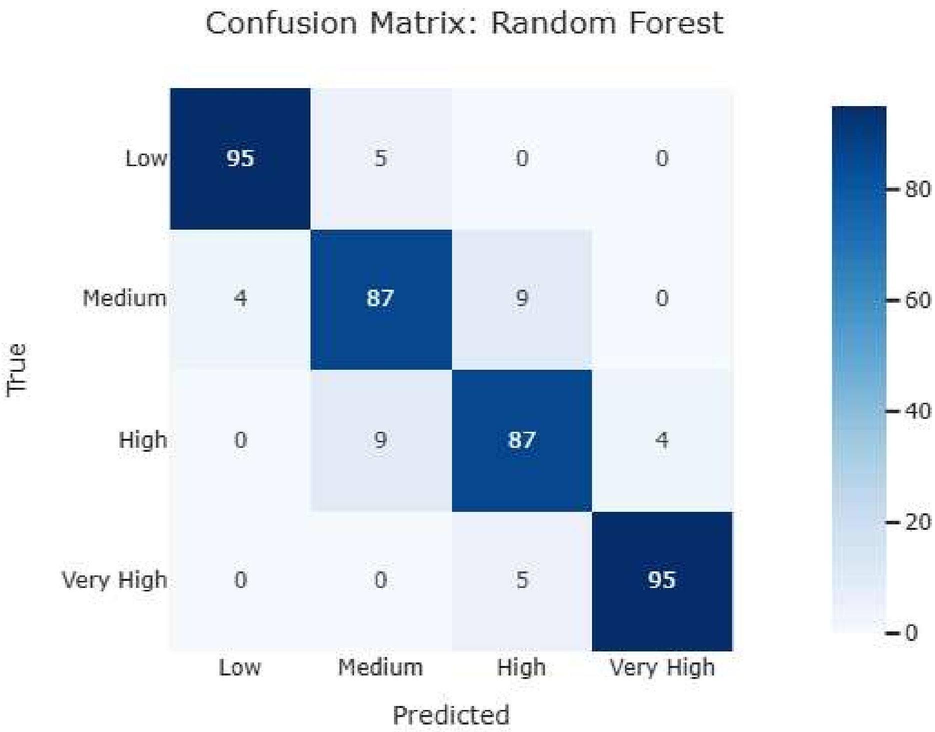
- **Confusion Matrix:** The confusion matrix visualizes the performance of the model to identify the predictions and misclassifications of the model.



Confusion Matrix: Logistic Regression

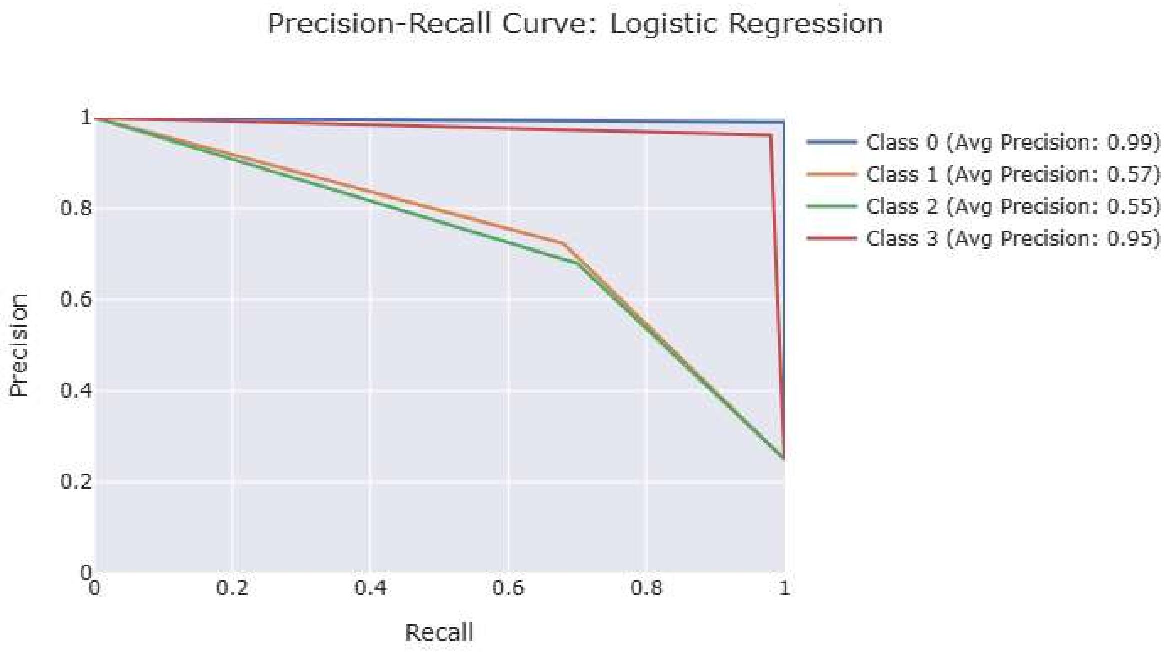


Confusion Matrix: KNeighbors

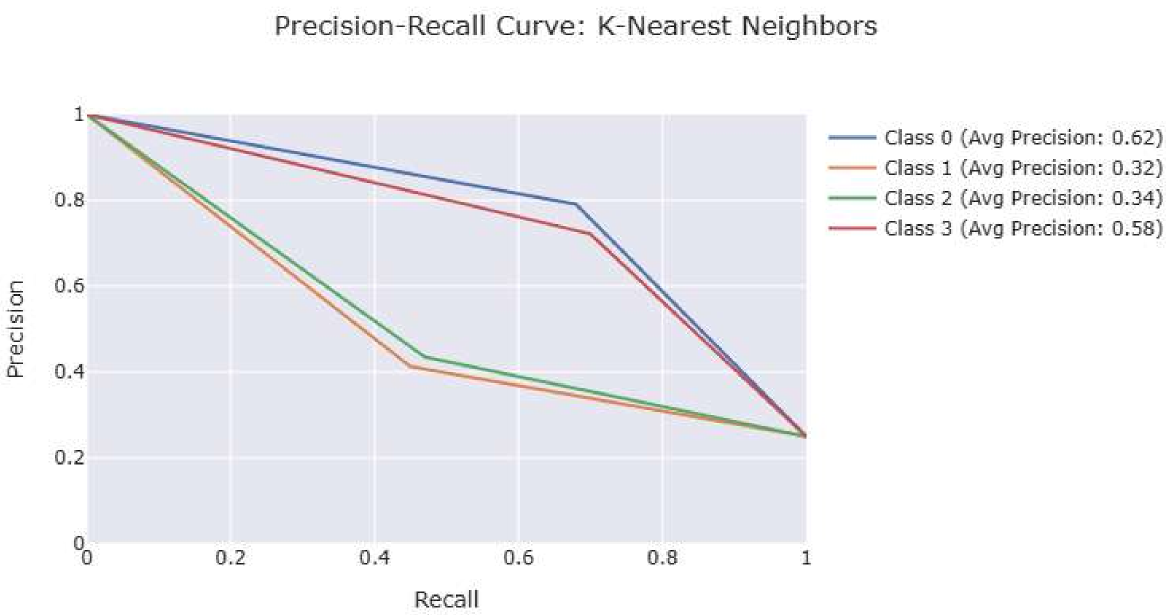


Confusion Matrix: Random Forest Classifier

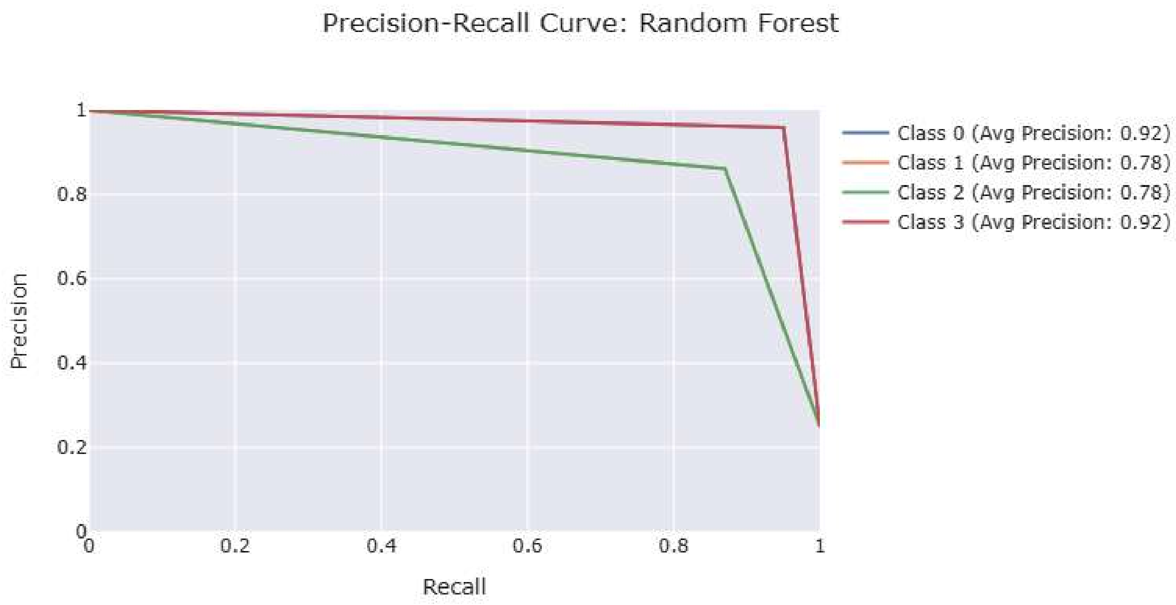
- **Precision-Recall Curve:** Precision-recall curves provide a further evaluation of the model's performance on identifying the different price ranges.



Precision Recall Curve: Logistic Regression

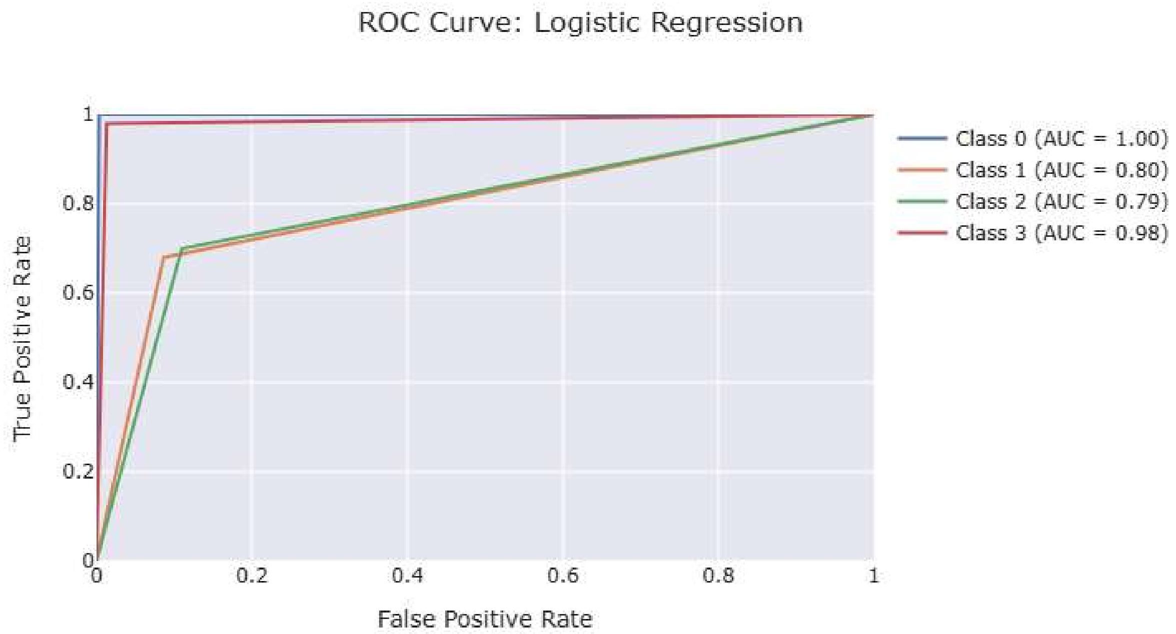


Precision Recall Curve: KNeighbors

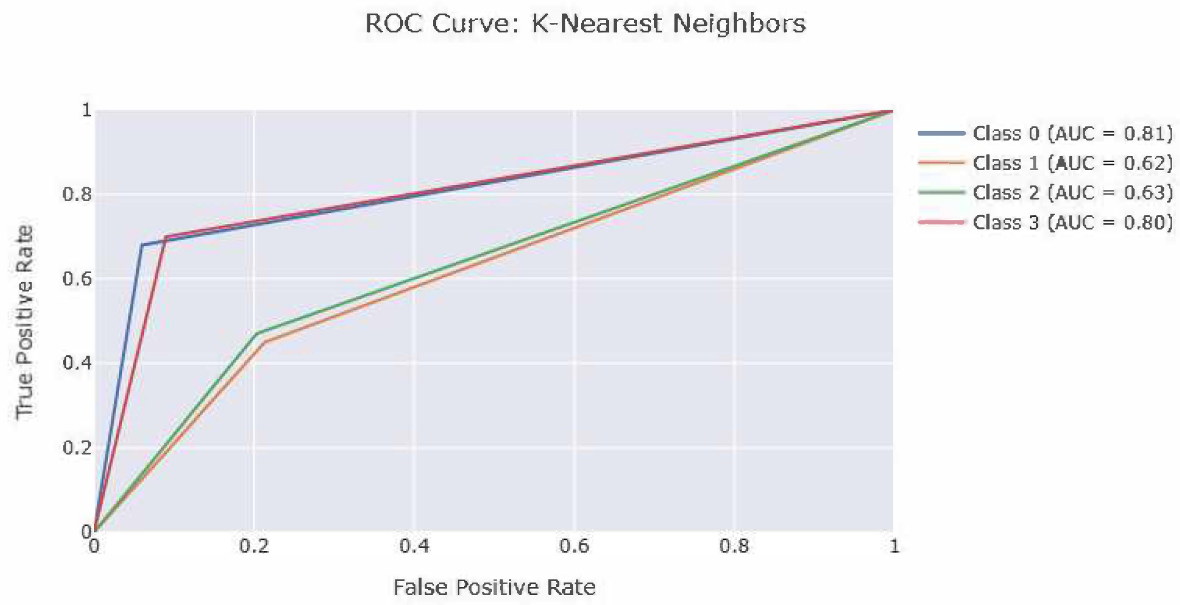


Precision Recall Curve: Random Forest Classifier

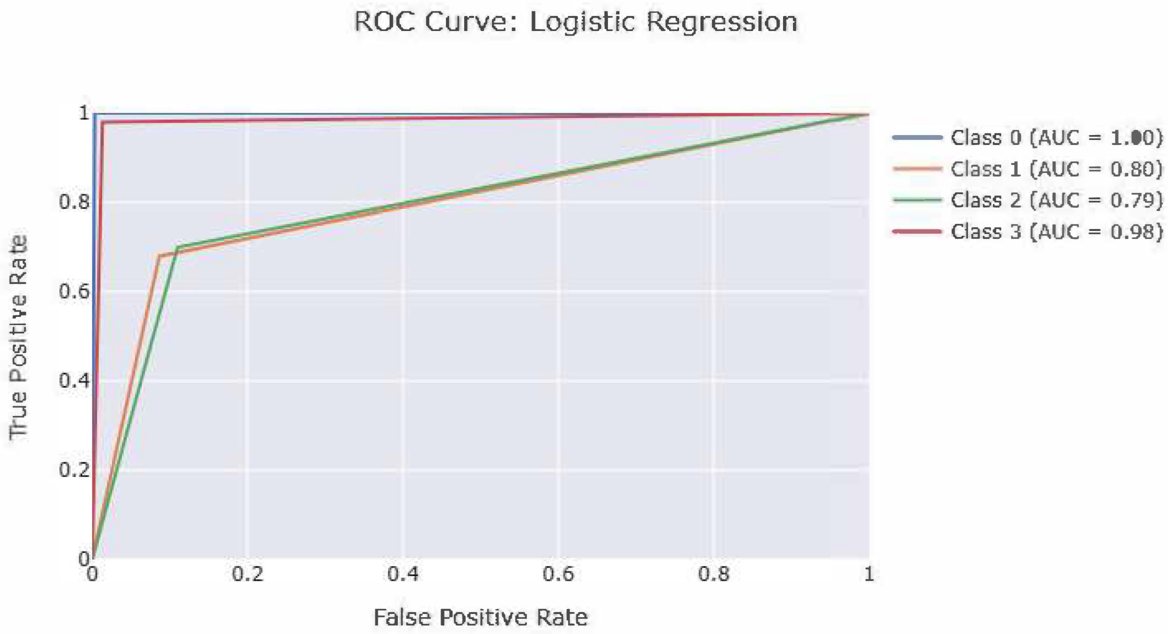
- **ROC Curve:** ROC curves provides a way to see how well the model is able to discriminate between the different price categories.



ROC Curve: Logistic Regression



ROC Curve: KNeighbors



ROC Curve: Random Forest Classifier

3.5. Model and Artifact Logging (Section 7 & 8)

- **Dagshub Integration:** The notebook uses Dagshub to track the experiments, providing experiment management.
- **MLflow Tracking:** MLflow is utilized for logging parameters, metrics, models, and artifacts of each experiment. The tracking uri is obtained from the dagshub UI. The models are logged using MLflow, along with their parameters and performance metrics.
- **Experiment Logging:** The logging is done by using the `create_experiment` function, to track the experiment results.

4. Results and Observations

After carefully analyzing the above trained models, we can clearly see that `Random Forest Classifier` is the best model followed by `Logistic Regression Classifier`. We will still deploy all the three models, for comparison, on a streamlit app.

5. Conclusion

This project successfully demonstrates the application of multiple machine learning classification models for predicting mobile phone price ranges. Through exploratory data analysis, the project identified the key features influencing price, and through the evaluation process of multiple models, the project highlighted which model will work best on the dataset. The results show that `Random Forest Classifier` performed the best achieving good performance on all price ranges. The project effectively used `MLflow` via `Dagshub` for experiment tracking and logging, along with a variety of visualizations to explain the results. A working `Streamlit` application was deployed with all the models to allow users to interact with the models and see the results in real-time. Find the project's streamlit app here