Table of Contents

Problem Definition

Pat McGee is gearing up to venture into the mobile phone industry, but he's facing a crucial problem of determining how to gauge the pricing for his products. In an industry dominated by tech giants such as Apple or Samsung, the pricing model is paramount to his business's success, so relying on intuition alone isn't sufficient and has decided to seek a predictive model data-driven solution instead. To address this challenge, Pat has shared a comprehensive dataset encompassing various mobile phone attributes and specifications and has tasked us to find the relationships within the dataset's features to develop a predictive model that should accurately identify whether a price range falls within the high or low spectrum based on the current industry standards. Throughout this report, we aim to cover this predictive approach and provide a robust model that optimizes his pricing strategies in a way that his business can compete in the dynamic mobile market.

Exploratory Data Analysis

This section highlights the key features that include RAM, Battery Power, Pixel Height and Pixel Width, and explores its relationship with the target variable Price Range by providing visualizations and insightful findings. We will focus on summarizing the data presented in the original dataset, further examine its correlation with Price Range and present a feature segmentation that categorizes the feature sets into groups of categories.

Summary of the Data

We will first examine the provided dataset and attributes. Figure 1 below showcases the first 5 rows of the provided dataset, comprising a total of 2000 rows of data and 21 columns of features. Shifting our attention to the key features, Battery Power represents the total energy a battery can store in one time that is measured in mAh, both Pixel Height and Width are the screen resolution dimensions, RAM (Random Access Memory) is essentially the memory that stores the operating system ranging from 256 to 4000 Mega Bytes, and finally, the Price Range column is the target variable with a value ranging between 0 to 3, representing low to very high cost respectively.

(2000, 21)

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	2	20	756	2549	9	7	19	0	0	1	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	6	905	1988	2631	17	3	7	1	1	0	2
2	563	1	0.5	1	2	1	41	0.9	145	5	6	1263	1716	2603	11	2	9	1	1	0	2
3	615	1	2.5	0	0	0	10	0.8	131	6	9	1216	1786	2769	16	8	11	1	0	0	2
4	1821	1	1.2	0	13	1	44	0.6	141	2	14	1208	1212	1411	8	2	15	1	1	0	1

Figure 1 — First 5 rows of the original dataset

A more intuitive representation of the features is presented in Figure 2 below, depicting a scatterplot matrix for Pixel Height and RAM against Price Range. The Pixel Height data can be seen to fade out as the value gets higher and we can make an observation that the screen height ranging between 1500 to 2000 pixels tends to be a bit more costly as we can see very few samples are shown in the pricing range value of 0 with that screen height range. As for RAM, the relationship seems more apparent since we can see that the larger the RAM size is, the higher the price range becomes. For example, RAM values ranging between approximately 2200 to 4000 MB are predominantly in the price range of 2 to 3, and RAM values ranging between approximately 300 to 2000 MB tend to be in the low price range of 0 or 1.

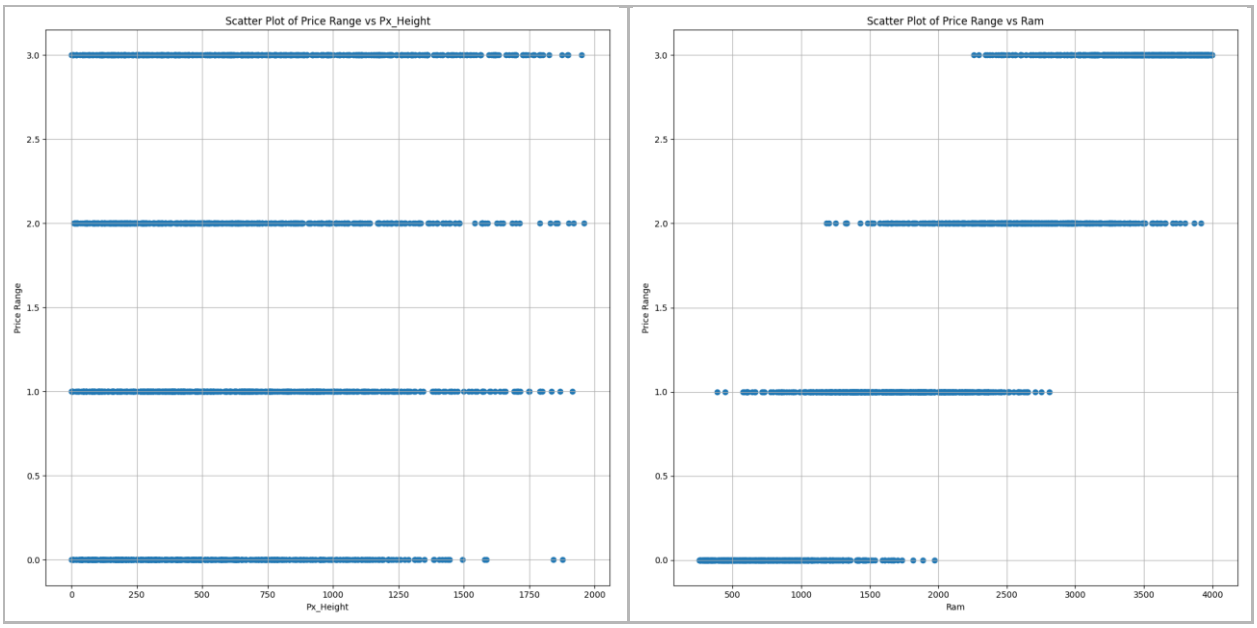

Figure 2 — Scatterplot Matrix of each key feature against Price Range

Another way to visualize the data is through plotting a barplot which is illustrated in Figure 3 below. Very similar results can be seen in comparison to the scatterplot matrix especially the feature RAM against the Price Range in terms of how the price range increases correspondingly when the RAM storage space increases showing a strong correlation. The other features don't seem to uphold that same level of correlation in the plots but we can noticeably see a positive correlation in terms of the higher the feature values are, the higher the price range.

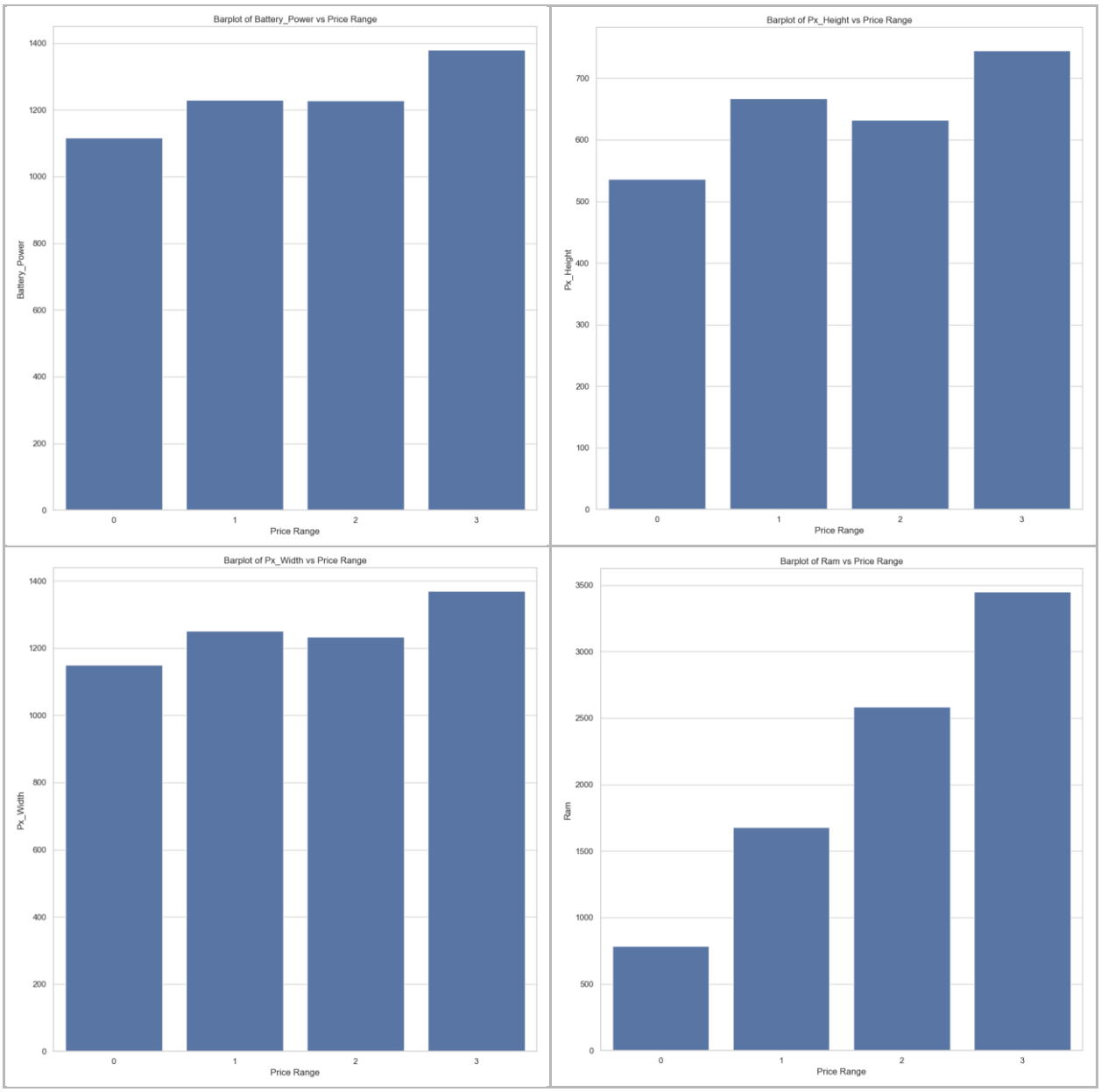

Figure 3 — Barplot of each key feature against Price Range

Correlation

Figure 4 below presents a correlation heatmap that displays each key feature's correlation coefficient with the target variable Price Range. We can see that the RAM is highly correlated positively with the target as it has a coefficient value of 0.92, while the other features seem to present a very low correlation with a coefficient value of 0.2 or lower. Although these features present a low correlation, it simply implies there may be no linear relationship involved but, having to tune the hyperparameters of the models that we build can often negate this issue.

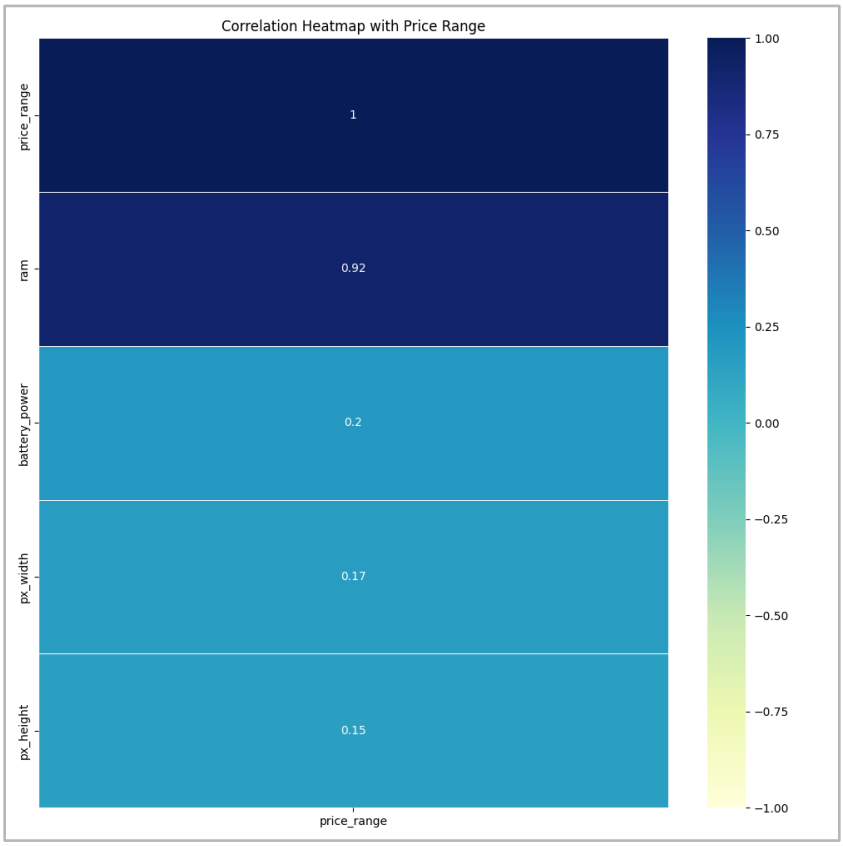

Figure 4 —Correlation heatmap of key features with Price Range

Boxplots can also often compare the distribution of each key feature across different levels of the target variable which can help identify any significant differences in those distributions. Figure 5 below depicts a boxplot for each feature with the Price Range. Starting with Battery Power, we can see that the values are evenly spread across the different price ranges. We can observe that the second quantile, the 50% mark of the values for Battery Power gradually increases as it goes up the price range, indicating a small positive correlation. The range of Battery Power values increases correspondingly as well as we can see at price range 0, the values range from approximately 800 to 1400 mHa and at price range 3, the values range from approximately 1000 to 1600 mHa. The next feature Pixel Height can be seen to have outliers in the dataset as depicted in price ranges 0 and 1. We can also make an observation of how 50% of the values for all price ranges tend to be within 250 to 1000 pixels and its second quantile tends to change throughout those price ranges. As for the feature Pixel Width, there doesn't seem to be a noticeable difference between the values and its ranges for price ranges 1 and 2 but we can see that price range 3 tends to have a larger pixel width with a range between 1100 to 1700 pixels. Finally, with the feature RAM, the positive correlation that was revealed from the previous plots becomes even more evident in this boxplot as we can see its progression in

the range of values as the price range increases. We can additionally see some outliers presented throughout the dataset.

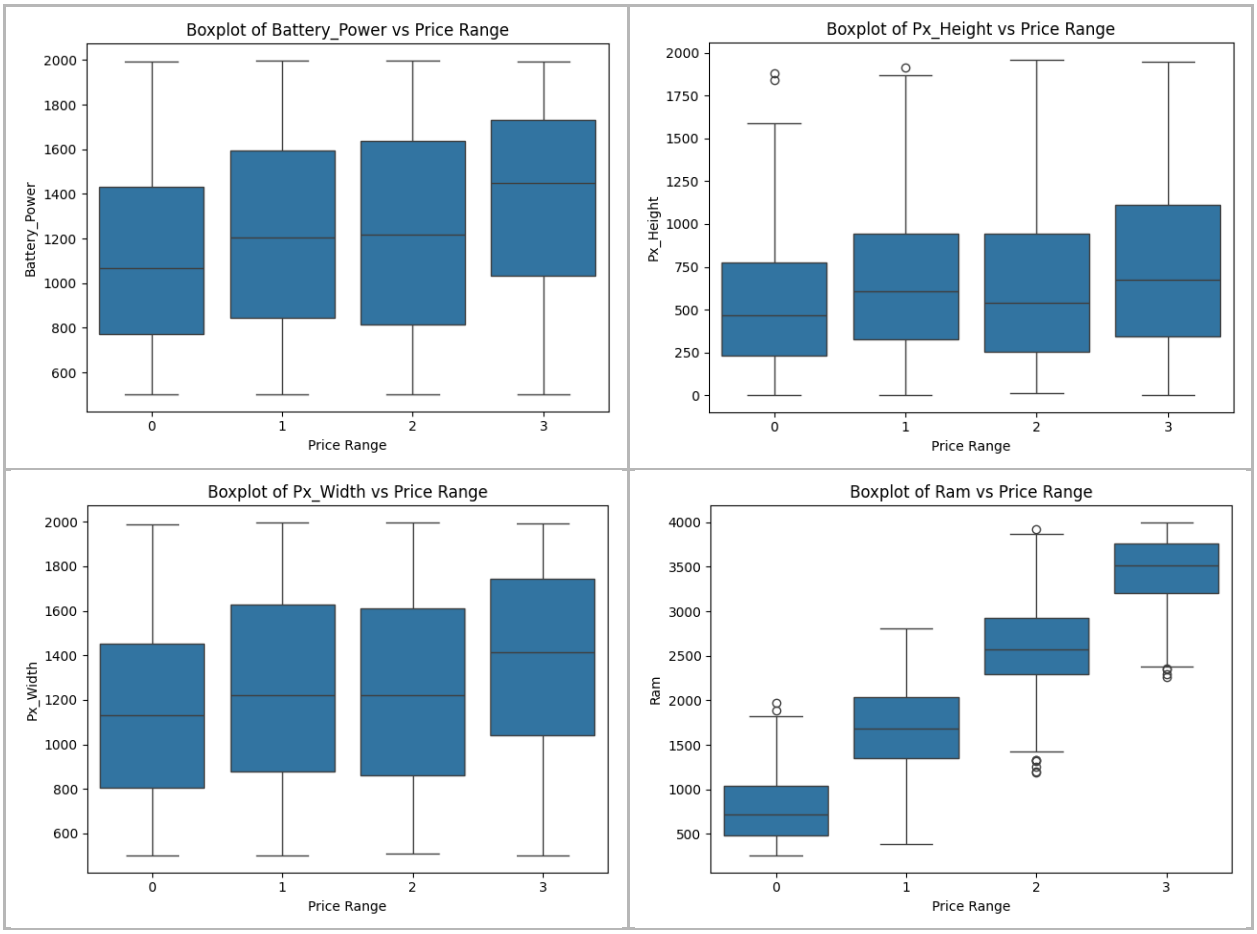

Figure 5 — Boxplot of each key feature against Price Range

Feature Segmentation

We have performed a K-means clustering analysis to segment the dataset based on the key features into 2 to 4 groups. Within the different combinations of the key features, the following 2 plots below separate the clusters nicely and allow us to understand how the different features contribute to the segmentation of mobile phones into different categories and performances.

Starting with Figure 6 below, we have performed the K-means with 3 clusters of the dataset based on features RAM and Pixel Width. The cluster coloured in green represents mobile phones that have any range of RAM but have low pixel width denoting users who prioritize performance over high resolution. The cluster coloured in yellow represents mobile phones that have low RAM but high pixel width denoting users who prioritize visual quality and display over heavy computational tasks and performance. The cluster coloured in purple represents the

opposite denoting users with high RAM and high pixel width who prioritize performance and quality over the costs of the mobile phones.

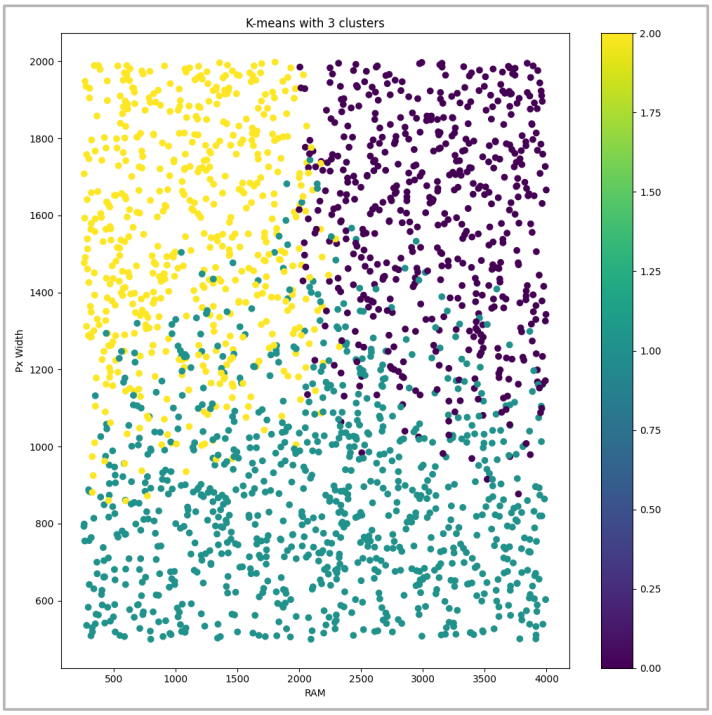

Figure 6 — K-Means with 3 clusters of the dataset based on RAM and Pixel Width

As for Figure 7 below, we have segmented in 2 clusters based on pixel width and pixel height. Although there are fewer clusters presented, the K-means clustering does a decent job of separating the segments and categorizing the types of users, The clusters coloured in yellow represent low pixel height but low to moderate range of pixel width denoting users who prioritize affordability and basic functionality over high-resolution displays. The clusters coloured in purple represent quite a large range of pixel height but prefer larger pixel width denoting users who prioritize high-quality displays for multimedia consumption such as gaming or other productivity apps and may target users in a higher range or premium level mobile phone market.

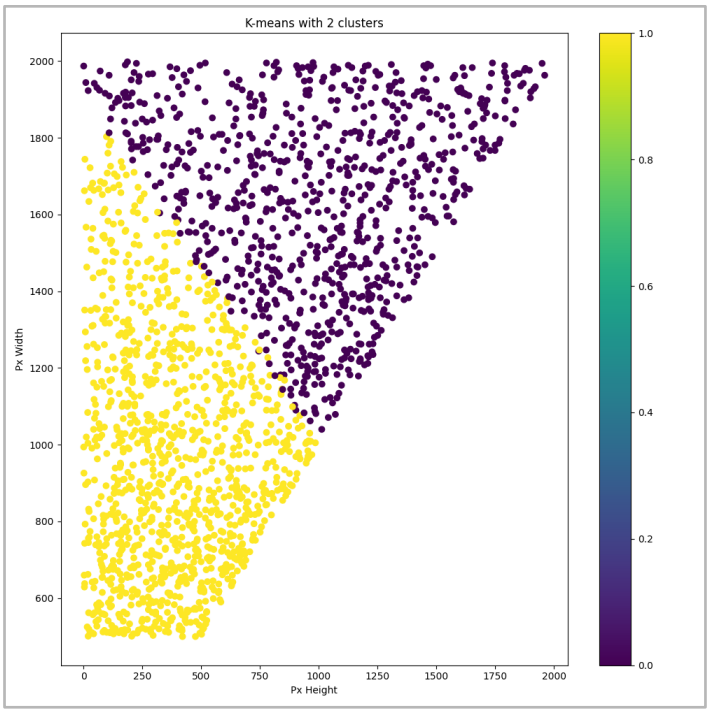

Figure 7 — K-Means with 2 clusters of the dataset based on the Pixel Height and Pixel Width

Model and Feature Selection Analysis Breakdown

This section explores various automated feature selection techniques that were used to find the most optimal features and evaluates the performance of 4 different types of predictive models that were constructed to predict our target variable Price Range. We will focus on comparing these models' metrics and further examine which of the 4 models is the best model that can accurately make the predictions with the most optimal number of features.

Model and Feature Selection Comparisons

Table 1 below presents 3 different types of automated feature selection techniques we have used to find an optimal number of best practical features that can be used for our models. Upon observation, it's apparent that the feature RAM seems to be a significant feature since it appears in 2 of the 3 feature selection techniques in addition to the exploratory data analysis revealing a high potential for significance as a predictor variable which we have presented in the previous section.

Feature Selection	Recursive Feature Elimination	Forward Feature Selection	Feature Importance (Random Forest)
Features	blue, three_g, wifi	ram, battery_power, px_width, px_height	ram
# of Features	3	4	1

Table 1 — Comparison table of automated feature selection routines and selected features.

Table 2 below presents 4 predictive models we have constructed that include Bagging Regression, Ensemble Regression, Stacked Regression and the traditional OLS linear regression. Throughout the data analysis section, we identified the relevance in the relationship between the features RAM, Pixel Width, Pixel Height and Battery Power with the target variable Price Range. The models below also exhibit one if not all of those features and show very satisfying metrics with a Root Mean Square Error (RMSE) all scoring below 1 with a high accuracy rate. Stacked Regression seems to dominate its performance out of the 4 models presenting the lowest RMSE score of 0.218 and the highest accuracy rate of 96.1%. The standard deviations for both the average RMSE and average accuracy are also very favourable as they show a score of 0.019 and 0.0068 respectively indicating that these metrics are very consistent. Although the model requires 4 features in comparison to Bagging Regression which requires only 1, the performance is extremely favourable and may be considered to be chosen as our best predictive model. Bagging Regression seems to perform the worst out of the 4 models with an average RMSE of 0.669 that was produced over 10 splits of cross-fold validations and an average accuracy of 63.9%. OLS and Ensemble regression models show relatively good metrics that can compete with Stacked Regression as they present a high accuracy rate and low RMSE.

Model	Bagging Regression	Ensemble Regression	Stacked Regression	OLS Regression
Features	ram	battery_power, ram, px_height, px_width	battery_power, ram, px_height, px_width	battery_power, px_width, ram
# of Features	1	4	4	3
Average RMSE	0.6690334129354889	0.2675888070673829	0.2183174091671136	0.3389750407910169
Std of RMSE	0.0453688763411217	0.0144053569512029	0.0193370621890597	0.0087683072099979
Average Accuracy	0.6391157493209161	0.9418399171027717	0.9614960502259707	0.9076069210811598
Std of Accuracy	0.0454440511992963	0.0094551125703628	0.0068312570950877	0.0051580573459826

Table 2 — Comparison table for the models and their respective features, RMSE, accuracy and standard deviations.

Model Evaluation

After examining the performance between the 4 models, Stacked Regression dominated the metrics having the highest accuracy rate and lowest RMSE score with very favourable standard deviations for the respective metrics. The features that were used for the models seem to exhibit the features that were presented from the automated feature selection techniques as well and we have also identified a highly correlated relationship from these features with the target variable Price Range from the data analysis sections.

As far as the Stacked Model goes, we have stacked Elastic Net linear regression, Support Vector Regression (SVR) with a gamma parameter of “scale”, AdaBoost Regression, Decision Tree Regression, Random Forest Regression with 200 estimators, Bagging Regression of 200 estimators, Extra Trees Regression of 200 estimators as well. No scaling was performed, but we split the dataset into training, testing and validation sets for 10 splits of cross-fold validations to validate its consistency in its model stability and predictions. We have provided the code in the Code Appendix section below that will run this model over cross-fold validations without errors.

Code Appendix

```
import pandas as pd
import statsmodels.api as sm
import numpy as np
from sklearn.linear_model import LinearRegression, ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, KFold
from sklearn.ensemble import (
    BaggingRegressor,
    RandomForestRegressor,
    AdaBoostRegressor,
    ExtraTreesRegressor,
)
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
pd.set_option("display.width", 1000)


def get_data():
```

```
    # @TODO: Locate path of the dataset here
    path = "/Users/elber/Documents/COMP 4948 - Datasets/assignment1/"
    return pd.read_csv(path + “train.csv”)


def model_stacked_regressor(df):
    # Split the data into features and target
    x = df[
        [
            "battery_power",
            "px_height",
            "px_width",
            "ram",
        ]
    ]
    x = sm.add_constant(x)
    y = df["price_range"]

    def get_unfit_models():
        models = list()
        models.append(ElasticNet())
        models.append(SVR(gamma="scale"))
        models.append(AdaBoostRegressor())
        models.append(DecisionTreeRegressor())
        models.append(RandomForestRegressor(n_estimators=200))
        models.append(BaggingRegressor(n_estimators=200))
        models.append(ExtraTreesRegressor(n_estimators=200))
        return models

    def evaluate_model(x_test, y_test, predictions, model):
        rmse = np.sqrt(mean_squared_error(y_test, predictions))
        print("Model:", model.__class__.__name__)
        print("Root Mean Squared Error:", rmse)
        print("Accuracy:", model.score(x_test, y_test))

    def fit_base_models(x_train, y_train, x_test, models):
        # Fit the base models and store its predictions in a dataframe
        df_predictions = pd.DataFrame()
        for i in range(len(models)):
            models[i].fit(x_train, y_train)
            predictions = models[i].predict(x_test)
            df_predictions[i] = predictions
        return df_predictions, models

    def fit_stacked_model(x, y):
```

```
    model = LinearRegression()
    model.fit(x, y)
    return model

# Cross fold validation
k_fold = KFold(n_splits=10, shuffle=True)
rmse_list = []
accuracy_list = []
print("\nCross Fold Validation:")
for train_index, val_index in k_fold.split(x):
    x_train, x_val = x.iloc[train_index], x.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    # Get the base models
    unfit_models = get_unfit_models()

    # Fit the base models and stacked model
    df_predictions, models = fit_base_models(x_train, y_train, x_val, unfit_models)
    stacked_model = fit_stacked_model(df_predictions, y_val)

    # Evaluate the base models and validation data
    print("\n** Evaluate Base Models ***")
    df_validation_predictions = pd.DataFrame()
    for i in range(len(models)):
        predictions = models[i].predict(x_val)
        df_validation_predictions[i] = predictions
        evaluate_model(x_val, y_val, predictions, models[i])

    # Evaluate the stacked model and validation data
    print("\n** Evaluate Stacked Model ***")
    stacked_predictions = stacked_model.predict(df_validation_predictions)
    evaluate_model(
        df_validation_predictions, y_val, stacked_predictions, stacked_model
    )

    rmse_list.append(np.sqrt(mean_squared_error(y_val, stacked_predictions)))
    accuracy_list.append(stacked_model.score(df_validation_predictions, y_val))

print("\nAverage RMSE:", np.mean(rmse_list))
print("Standard Deviation RMSE:", np.std(rmse_list))
print("Average Accuracy:", np.mean(accuracy_list))
print("Standard Deviation Accuracy:", np.std(accuracy_list))

# Standalone Model Evaluation
print("\nStandalone Model Evaluation:")
```

```
    # Split the data into training, testing and validation sets
    x_train, x_temp, y_train, y_temp = train_test_split(x, y, test_size=0.7)
    x_test, x_val, y_test, y_val = train_test_split(x_temp, y_temp, test_size=0.5)

    # Get the base models
    unfit_models = get_unfit_models()

    # Fit the base models and stacked model
    df_predictions, models = fit_base_models(x_train, y_train, x_val, unfit_models)
    stacked_model = fit_stacked_model(df_predictions, y_val)

    # Evaluate the base models and validation data
    print("\n** Evaluate Base Models ***")
    df_validation_predictions = pd.DataFrame()
    for i in range(len(models)):
        predictions = models[i].predict(x_test)
        df_validation_predictions[i] = predictions
        evaluate_model(x_test, y_test, predictions, models[i])

    # Evaluate the stacked model and validation data
    print("\n** Evaluate Stacked Model ***")
    stacked_predictions = stacked_model.predict(df_validation_predictions)
    evaluate_model(
        df_validation_predictions, y_test, stacked_predictions, stacked_model
    )


def main():
    df = get_data()

    # Build Stacked Regressor Model
    model_stacked_regressor(df)


if __name__ == "__main__":
    main()
```