CSE 587 Data Intensive Computing
Project Phase 2

Title: Property Sales: Melbourne City

Team Details:

Harshavardhan Sivadanam -hsivadan (50478880)
Nikhil Kumar vadigala -nvadigal (50485135)
Sai Sree Depa- saisreed(50478013)

Problem Statement

The main objective of this project is to develop predictive models for forecasting real estate prices in Melbourne City accurately. The objective is to create models that can predict property sale prices by considering a wide range of factors, including the number of rooms, property type, location (suburb, postcode, latitude, longitude), building characteristics (bedroom count, bathroom count, car parking, land size, building area), year of construction, and other relevant attributes available in the dataset. This project should deliver practical insights for people and parties interested in the Melbourne housing market by forecasting property values. Making decisions on the purchase, sale, or investment of real estate in the city can be aided by these forecasts.

Data Source:

We obtained the dataset "Property Sales of Melbourne City" from Kaggle, and which can be accessed using the following link: Property sales: Melbourne City on Kaggle.
The idea is to develop predictive models for forecasting real estate prices in Melbourne City accurately. The objective is to create models that can predict property sale prices by considering a wide range of factors, including the number of rooms, property type, etc.

Selecting Features and Splitting the Dataset:

```
In [323]: #Selecting required columns in the dataset
          selected_columns = ['Rooms', 'Distance', 'Postcode', 'Bathroom', 'Car', 'Landsize', 'BuildingArea',
                              'YearBuilt', 'Lattitude', 'Longtitude', 'Propertycount',
                              'BuildingAge', 'CouncilArea_encoded', 'Suburb_encoded', 'Year',
                              'Month', 'Day', 'Type_t', 'Type_u', 'Method_S',
                              'Method_SA', 'Method_SP', 'Method_VB', 'Regionname_Eastern Victoria',
                              'Regionname_Northern Metropolitan', 'Regionname_Northern Victoria',
                              'Regionname_South-Eastern Metropolitan',
                              'Regionname_Southern Metropolitan', 'Regionname_Western Metropolitan',
                              'Regionname_Western Victoria',
                              'SellerG_encoded']

In [324]: # Features used for prediction
          input_data = housing_data[selected_columns]
          # Target variable to be predicted
          output_data= housing_data[['Price']]

In [327]: # Splitting the dataset into training and testing sets
          X_train,X_test,Y_train,Y_test = train_test_split(input_data,output_data,test_size = 0.25)
```

In the dataset, required columns are selected and then the dataset is divided into training and testing datasets, with a proportion of 25%.

The Models we have used for the prediction of Melbourne City house prices are:

1. Linear Regression
2. Decision Tree
3. Gradient Boosting Regressor
4. a. AdaBoost Regressor with Default base estimator
   b. AdaBoost Regressor with Decision Tree as a base estimator
5. Random Forest
6. XGB Regressor

MODEL I: LINEAR REGRESSION

Linear Regression is a basic statistical method used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. Finding the linear regression line that minimizes the sum of squared differences between the observed and predicted values is the objective of the best-fitting straight-line method.

```
In [343]: linear_regression = LinearRegression()
          linear_regression.fit(X_train,Y_train)
Out[343]: LinearRegression()

In [344]: lr_pred= linear_regression.predict(X_test)
```

Reasons for Choosing the Linear Regression Model:

Linearity in the Dataset: In the dataset, there are a lot of features in the Melbourne housing dataset that may be related linearly to the target variable (price). Features like "Landsize," "Distance," "Bathroom,"

and "Rooms" can have a linear impact on house prices. Since linear regression is predicated on the idea that input features and the target variable have a linear relationship, it works best with datasets that are predicted to exhibit these kinds of linear trends.

Interpretability: The influence of each feature on the intended home prices can be easily understood through the use of linear regression. The flexibility of use and interpretability of linear regression makes it a favorable option in the real estate industry, where stakeholders frequently want unambiguous insights into the elements influencing property values.

Ease and Simplicity: Linear regression is an ideal place to start for regression tasks since it is easy to use and has low computational cost. A simple linear model is a useful baseline for the Melbourne housing dataset because of its many variables and possible linear correlations. On top of this foundation, complex models can be constructed for comparison.

Analysis Using Model Evaluation Metrics:

```
In [345]: r2_score_lr= r2_score(Y_test,lr_pred)
          print("R-squared score using Linear regression",r2_score_lr)

          R-squared score using Linear regression 0.636081270087905

In [346]: mae_lr= mean_absolute_error(Y_test,lr_pred)
          print("Mean Absolute Error using Linear regression",mae_lr)

          Mean Absolute Error using Linear regression 205623.0546559237

In [347]: mse_lr= mean_squared_error(Y_test,lr_pred)
          rmse_lr= np.sqrt(mse_lr)
          print("Root Mean Square Error using Linear regression",rmse_lr)

          Root Mean Square Error using Linear regression 265825.09969081404

In [348]: pred_score_lr= linear_regression.score(X_test, Y_test)*100
          print("Accuracy using Linear regression",pred_score_lr)

          Accuracy using Linear regression 63.608127008790504
```

R-squared Score: 0.636

The R-squared value shows that the linear regression model accounts for roughly 63.6% of the variance in the target variable (price). This metric measures how well the linear model fits the data.

Mean Absolute Error (MAE): 205623.05

The average absolute difference (MAE) between the actual and predicted prices is shown. This metric shows that predictions of the linear regression model are, on average, around 205623.05 away from the actual prices.

Root Mean Square Error (RMSE): 265825.09

The mean proportion of the model's errors is indicated by the RMSE. This indicates that, on average, the prices that are predicted by the linear regression model and the actual prices differ by around 265825.09.

Accuracy: 63.60%

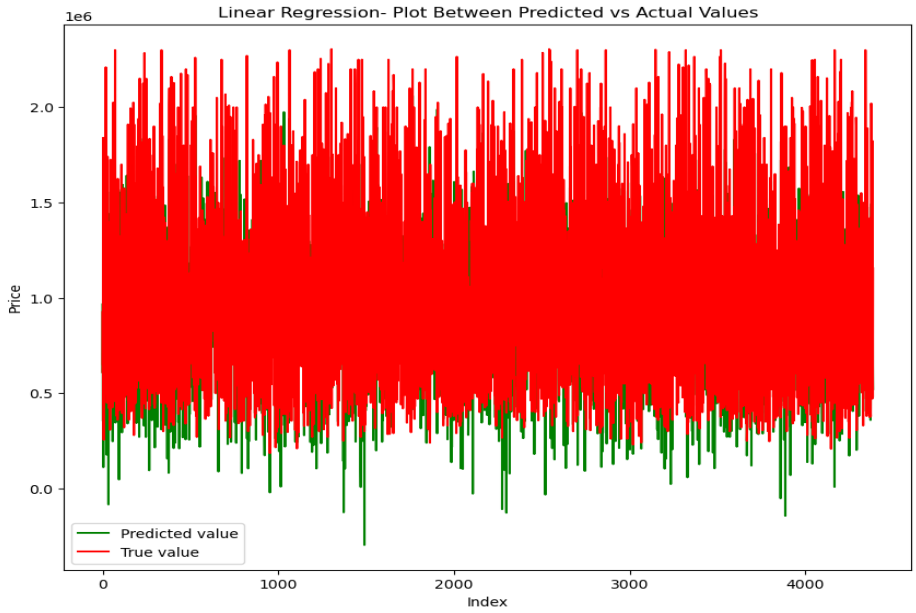
Although accuracy is not a commonly used metric for regression problems, in this context it signifies the proportion of accurate predictions that fall inside a given range. The linear regression model's predictions are within an acceptable range of the actual prices for roughly 63.60% of the cases, according to the model's accuracy.

Visualization

1. Line Plot Between Predicted vs Actual Values

```
n [130]: # Line plot between Predicted vs Actual values

         # Generates the range of data points
         data_points = range(X_test.shape[0])
         plt.figure(figsize=(10, 8))
         plt.plot(data_points, lr_pred, 'g', label='Predicted value')
         plt.plot(data_points, Y_test, 'r', label='True value')
         plt.title('Linear Regression- Plot Between Predicted vs Actual Values')
         plt.ylabel('Price')
         plt.xlabel('Index')
         plt.legend()
         plt.show()
```

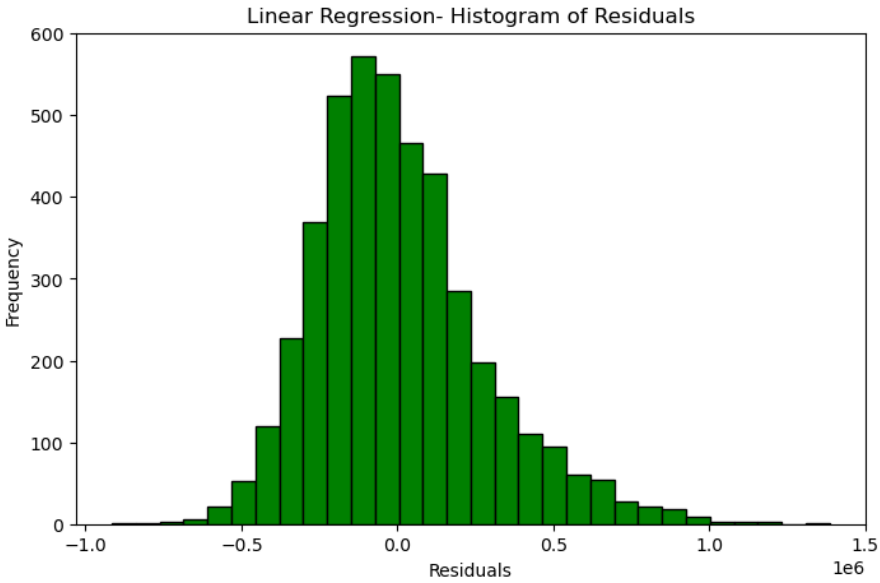
The provided line plot compares the predicted values (lr_pred) and the actual values (Y_test) for each data point in a linear regression model. The green line represents the predicted values generated by the Linear Regression model. The red line corresponds to the true values (actual housing prices) from the dataset. Along the range of data points, the green line nearly matches the red line. However, for various values, there are differences between the two lines, which show places where the model's predictions differ from the actual values. Outliers, or instances in which the green line deviates noticeably from the red line, may highlight to certain data points where the model may have problems or where actual values behave differently.



2. Histogram of Residual Graph:

```
In [131]: # Histogram of residuals graph

          #Ensuring Y_test is 1-D array
          Y_test = Y_test.squeeze()
          lr_pred = lr_pred.ravel()

          residuals = Y_test - lr_pred
          plt.figure(figsize=(8, 5))
          plt.hist(residuals, bins=30, edgecolor='black', color='green')
          plt.xlabel("Residuals")
          plt.ylabel("Frequency")
          plt.title("Linear Regression- Histogram of Residuals")
          plt.show()
```

The histogram shows the residuals for a linear regression model that visualizes the distribution of prediction errors. A centered and symmetrical histogram indicates unbiased predictions, while patterns or asymmetry suggest areas for model improvement. Positive residuals signify overpredictions, and negative residuals indicate underpredictions. As histogram is almost centered at zero, it indicates there is no major bias in the model.



Conclusion:

The Linear Regression model uses the expected linear relationships observed in the data to provide moderate predictive accuracy for the Melbourne housing dataset. Additional feature engineering and model optimization may improve the overall accuracy and performance.

MODEL II: DECISION TREE REGRESSION

A Decision Tree Regressor is a variant of the decision tree algorithm specifically designed for regression tasks. The algorithm creates a tree structure with each leaf node representing a predicted numerical value after recursively dividing the dataset into subsets according to feature conditions. Decision Tree Regressors are capable of capturing complex non-linear relationships in the data and are interpretable.

```
In [333]: decision_tree = DecisionTreeRegressor(random_state=42)

In [334]: decision_tree.fit(X_train, Y_train)
Out[334]: DecisionTreeRegressor(random_state=42)

In [335]: DT_pred = decision_tree.predict(X_test)
```

Reasons for Choosing the Decision Tree Regression Model:

Non-linearity in the Dataset: From the Melbourne housing dataset, it can be observed that there are some features that could be non-linear with the target variable since they do not strictly follow a linear pattern. Decision Tree Regression, which is known for capturing complex, non-linear relationships, is particularly well-suited for our dataset where the underlying patterns are complex and resist simple linear associations.

Interpretability: The decision-making process is made easier to understand by the clear and interpretable algorithm of Decision Tree Regression. Decision trees are helpful because they are interpretable, especially in the real estate setting as in the dataset property prices can be determined by a variety of factors.

Feature Importance: Decision trees provide a clear picture of the significance of each factor, making it easier to determine which features have a significant impact on the target variable, Price.

Handling Non-Normality: Decision trees are sensitive to variations and outliers and do not rely on a normal distribution of the data. As in the dataset, the distribution of housing prices is diverse and non-uniform. Decision tree regression is a useful tool for managing this non-normality.

Analysis Using Model Evaluation Metrics:

```
In [336]: r2_score_dt= r2_score(Y_test,DT_pred)
          print("R-squared score using Decision Tree regression",r2_score_dt)

          R-squared score using Decision Tree regression 0.6086946940793614

In [337]: mae_dt= mean_absolute_error(Y_test,DT_pred)
          print("Mean Absolute Error using Decision Tree regression",mae_dt)

          Mean Absolute Error using Decision Tree regression 193390.613999088

In [338]: mse_dt= mean_squared_error(Y_test,DT_pred)
          rmse_dt= np.sqrt(mse_dt)
          print("Root Mean Square Error using Decision Tree regression",rmse_dt)

          Root Mean Square Error using Decision Tree regression 275645.9714963617

In [339]: pred_score_dt= decision_tree.score(X_test, Y_test)*100
          print("Accuracy using Decision Tree regression",pred_score_dt)

          Accuracy using Decision Tree regression 60.86946940793614
```

R-squared Score: 0.608

The decision tree regression model accounts for roughly 60.8% of the variance in the target variable (Price), according to the R-squared score. This highlights how well the model captures and explains the non-linear variability in house prices.

Mean Absolute Error (MAE): 193390.61

The average absolute difference between the actual and expected prices is represented by the MAE. In this instance, it indicates that the average divergence of the decision tree regression model's predictions from the true prices is about 193390.61, indicating that these variances are not linear.

Root Mean Square Error (RMSE): 275645.97

The average magnitude of the errors in the model is measured by the RMSE. This illustrates even further the influence of non-linearity: on average, the decision tree regression model's forecasts deviate from the actual prices by roughly 275645.97.

Accuracy: 60.86%

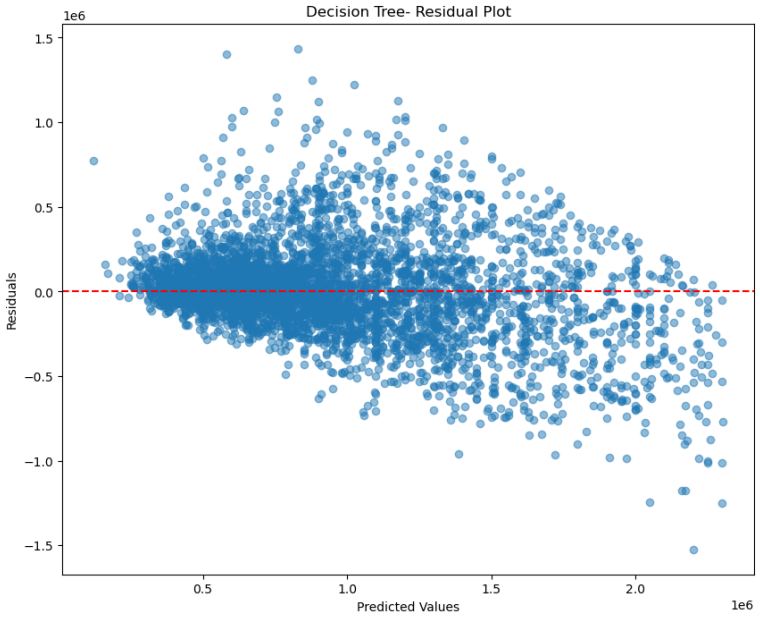
In this context it is understood to mean the proportion of accurate predictions that fall within a certain range. The model's accuracy of 60.86% highlights its capacity to generate accurate predictions in the non-linear context of variations in house prices.

Visualization:

1. Scatter Plot Between Predicted Values vs Residuals

In [121]:
```
# Scatter plot of decision tree predicted values vs residuals

#Ensuring Y_test is 1-D array
Y_test = Y_test.squeeze()
DT_pred = DT_pred.ravel()

# calculating the residuals
residuals = Y_test - DT_pred

plt.figure(figsize=(10, 8))
plt.scatter(DT_pred, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Decision Tree- Residual Plot")
plt.show()
```

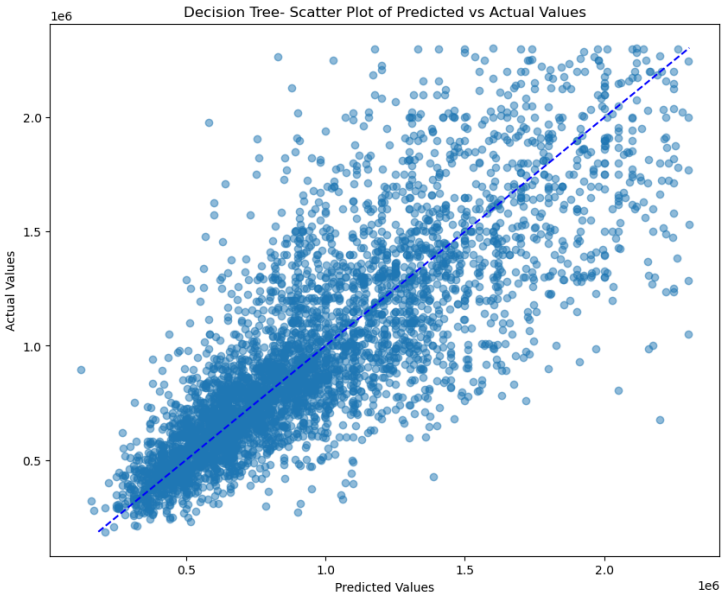
The Scatter plot shows the link between the residuals of the model on the given test dataset (Y_test) and the predicted values produced by a decision tree model (DT_pred). The differences between the true target values (Y_test) and the forecasted values (DT_pred) are used to compute the residuals. The red dashed line at y=0 represents the ideal scenario where the residuals are exactly zero for all predicted values. The residuals show a random scatter around the horizontal axis, which suggests that the model is capturing the underlying patterns well. However, the spread with the widening of residuals suggests that the model's variance does not remain constant, indicating unreliable predictions, and the degree of error increases as we move along the predicted values for some values.



2. Scatter Plot Between Predicted Values vs Actual Values

In [122]:
```
# scatter plot between actual and predicted values
plt.figure(figsize=(10, 8))
plt.scatter(DT_pred, Y_test, alpha=0.5)
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], 'b--')
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.title("Decision Tree- Scatter Plot of Predicted vs Actual Values")
plt.show()
```

The Scatter plot of predicted values (DT_pred) against actual values (Y_test) provides a visual representation of how well a decision tree regression model performs on the dataset. From the graph, it can be seen that for some values of the data, the predicted values are not same as the actual values.



Conclusion:

In conclusion, the non-linear complications found in the Melbourne housing dataset are moderately handled by the Decision Tree Regression model. The model shows potential by utilizing its ability to give understanding and capture complicated relationships. Additional improvements via feature engineering and model tuning may improve its accuracy and overall performance in this non-linear area.

MODEL III: GRADIENT BOOSTING REGRESSOR

The Gradient Boosting Regressor is an ensemble learning method that generates a strong predictive model by combining the predictions of several weak learners. It is a member of the boosting algorithm family, and its main concept is to increase overall prediction performance by combining and building weak models one after the other. The algorithm creates trees one after the other, trying to fix the errors of the earlier trees. Every tree in the series is trained to forecast the ensemble's current residuals, or the variations between actual and expected values. Because of its resistance to outliers, it can be used to recognize unusual patterns.

```
In [352]: GBR = GradientBoostingRegressor(n_estimators=100, random_state=42)
          GBR.fit(X_train, Y_train)

          /Users/saisreereddy/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_gb.py:494: DataConversionWarning
          column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for exa
          using ravel().
            y = column_or_1d(y, warn=True)

Out[352]: GradientBoostingRegressor(random_state=42)

In [353]: GBR_pred = GBR.predict(X_test)
```

Reasons for Choosing the Gradient Boosting Regressor:

Ensemble Learning for Improved Performance: The Gradient Boosting Regressor is selected for its ensemble learning properties in the Melbourne housing dataset. Many factors affect the housing market, and a set of weak learners can better understand the complex and diverse links between these features and home prices. The model is predicted to achieve improved prediction performance by adding trees one after the other to correct errors caused by the previous ones. This is especially expected in situations where we have seen that the individual models would find it difficult to capture the complexity of the housing market.
Non-linearity and Complex Relationships: There are many factors in the Melbourne housing dataset, and each one has the potential to have a non-linear impact on the overall housing prices. Because the gradient boosting regression is so good at capturing complicated patterns and non-linear correlations, it is a good fit for this dataset. The model's capacity to manage complicated relationships between variables such as age, size, and location improves its reliability in making accurate predictions in the housing market.
Robustness to Overfitting: When working with datasets that have complicated relationships, overfitting is an issue. By using regularisation approaches during the ensemble construction process, the Gradient Boosting Regressor ensures resistance against overfitting in the Melbourne housing dataset. This is especially important for real estate datasets because overfitting to specific property instances is a common issue.

Analysis Using Model Evaluation Metrics:

```
354]: r2_score_GBR= r2_score(Y_test,GBR_pred)
      print("R-squared score using Gradient Boosting Regressor",r2_score_GBR)

      R-squared score using Gradient Boosting Regressor 0.7648486528342653

355]: mae_GBR= mean_absolute_error(Y_test,GBR_pred)
      print("Mean Absolute Error using Gradient Boosting Regressor",mae_GBR)

      Mean Absolute Error using Gradient Boosting Regressor 154690.03585334562

356]: mse_GBR= mean_squared_error(Y_test,GBR_pred)
      rmse_GBR= np.sqrt(mse_GBR)
      print("Root Mean Square Error using Gradient Boosting Regressor",rmse_GBR)

      Root Mean Square Error using Gradient Boosting Regressor 213681.7918875081

357]: pred_score_GBR= GBR.score(X_test, Y_test)*100
      print("Accuracy using Gradient Boosting Regressor",pred_score_GBR)

      Accuracy using Gradient Boosting Regressor 76.48486528342653
```

R-squared Score: 0.764

The Gradient Boosting Regressor accounts for roughly 76.4% of the variance in the target variable (Price), according to the R-squared score. The model appears to be able to adequately represent the variation in house prices, based on the strong R-squared score.

Mean Absolute Error (MAE): 154690.03

The average absolute difference between the actual and expected prices is represented by the MAE. In this instance, it shows that the average deviation between the true prices and the forecasts made by the Gradient Boosting Regressor is about 154690.03.

Root Mean Square Error (RMSE): 213681.79

The average magnitude of the errors in the model is measured by the RMSE. In this case, it demonstrates that the average difference between the prices predicted by the Gradient Boosting Regressor, and the actual prices is about 213681.79.

Accuracy: 76.4%

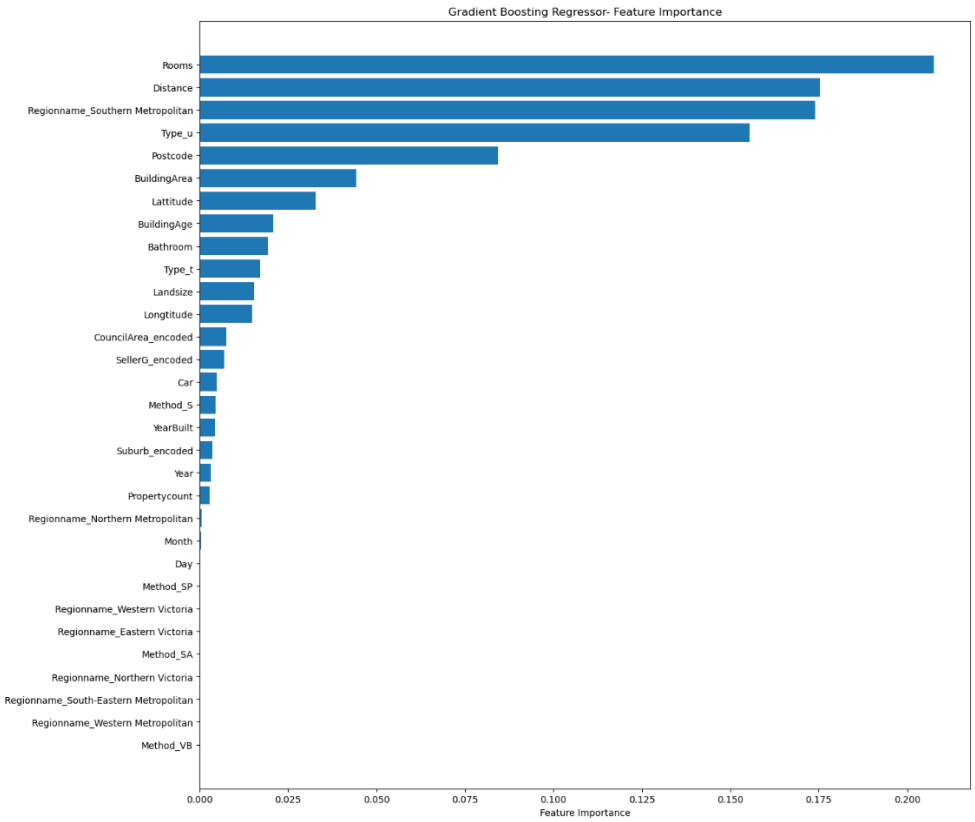
In this context, it is understood that the proportion of accurate predictions that fall inside a certain range. With an accuracy of 76.4%, it can be concluded that for around 76.4% of the cases, the predictions made by the Gradient Boosting Regressor are within an acceptable range of the actual prices.

Visualization:

1. Feature Importance Graph

```
In [139]: # Feature importance Graph from the trained Gradient Boosting Regressor model
feature_importance = GBR.feature_importances_
data_columns = input_data.columns

# Sort the indices based on feature importance
sorted_idx = feature_importance.argsort()
plt.figure(figsize=(15, 15))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx])
plt.yticks(range(len(sorted_idx)), data_columns[sorted_idx].tolist())
plt.xlabel("Feature Importance")
plt.title("Gradient Boosting Regressor- Feature Importance")
plt.show()
```

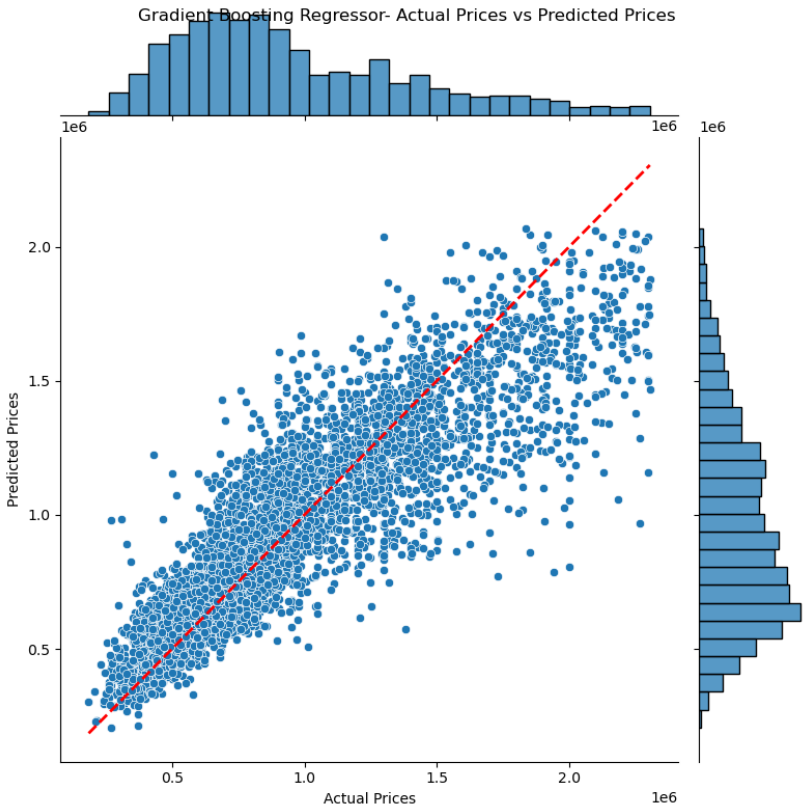
The horizontal bar chart represents the feature importance of input variables in a trained Gradient Boosting Regressor model. Longer bars indicate higher importance, i.e., Rooms have more importance, followed by Distance, Regionname_Southern Metropolitan, Type_u, etc. Showcasing which features have the most significant impact on the model's predictions. This visualization aids in identifying key contributors to the model's decision-making process. These key predictors can offer valuable insights into the factors that strongly influence the real estate market in Melbourne.



2. Joint Plot Between Actual and Predicted Values:

In [140]:
```
# DataFrame with Actual and Predicted prices
comparison_data = pd.DataFrame({'Actual Prices': Y_test.squeeze(), 'Predicted Prices': GBR_pred})
plt.figure(figsize=(15, 15))
sns.jointplot(x='Actual Prices', y='Predicted Prices', data=comparison_data, kind='scatter', height=8)
plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], linestyle='--', color='red', linewidth=2)
plt.suptitle('Gradient Boosting Regressor- Actual Prices vs Predicted Prices')
plt.show()
```

The below joint scatter plot illustrates the relationship between actual and predicted prices from a Gradient Boosting Regressor model. Points clustered along the diagonal indicate accurate predictions, while deviations from the red dashed line highlight discrepancies. The plot provides a visual assessment of the model's performance in predicting prices.



Conclusion:

In the Melbourne dataset, the Gradient Boosting Regressor shows itself to be an effective model for predicting property prices. Its noteworthy prediction success is a result of its ability to identify non-linear correlations, manage complex patterns, and reduce overfitting. Additional fine-tuning of the model might improve its accuracy even further.

MODEL IV: ADABOOST REGRESSOR

AdaBoost Regressor is an ensemble learning algorithm used for regression tasks. It is part of the boosting algorithm family, which involves the sequential training of several weak learners (usually decision trees) to fix the errors made before them. Instances that were incorrectly classified by previous models are given greater weights, highlighting their significance for further model training. All of the weak learners' predictions together, weighted, form the final prediction. Models that perform better contribute more to the final prediction.

```
]: adaboost_regressor = AdaBoostRegressor(random_state=42)
   adaboost_regressor.fit(X_train, Y_train)

   /Users/saisreereddy/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/validation.py:993: DataConversionWarnin
   g: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for ex
   ample using ravel().
     y = column_or_1d(y, warn=True)

]: AdaBoostRegressor(random_state=42)

]: ABR_pred = adaboost_regressor.predict(X_test)

]: # AdaBoost Regressor with a Decision Tree as the base estimator
   abr_dt = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(random_state=42), random_state=42)
   abr_dt.fit(X_train, Y_train)

   /Users/saisreereddy/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/validation.py:993: DataConversionWarnin
   g: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for ex
   ample using ravel().
     y = column_or_1d(y, warn=True)

]: AdaBoostRegressor(base_estimator=DecisionTreeRegressor(random_state=42),
                     random_state=42)

]: ABR_DT_pred = abr_dt.predict(X_test)
```

Reasons for Choosing the Gradient Boosting Regressor:

Sequential Error Correction: AdaBoost Regressor's sequential training technique is a useful choice while analyzing the Melbourne housing dataset. Numerous factors affect the real estate market, and the model can improve its predictions iteratively through sequential error correction. AdaBoost attempts to improve in order to deal with the complex and constantly changing nature of the housing dataset.

Weighted Focus on Incorrectly classified Instances: When an instance is previously mispredicted by the ensemble, AdaBoost gives it a higher weight. Some properties in the Melbourne housing dataset can have unique features or outliers that affect how much they are priced. AdaBoost makes sure that following weak learners pay extra attention to difficult cases by emphasizing examples with prediction errors.

Ensemble Combination: The information on the Melbourne housing market includes a range of features, each of which influences home prices in a distinct way. In this situation, the ensemble synergy of AdaBoost becomes essential. The model takes advantage of each learner's unique set of skills by aggregating the forecasts of several weak learners. This ensemble technique improves the model's capacity to generalise well to new, unseen data and helps prevent overfitting.

Reason for Default Base Estimator:

AdaBoost Regressor's default setup was selected due to its ease of use and simplicity. Usually, the default configurations offer a decent place to start, making the algorithm's implementation simple.

Reason for Decision Tree Base Estimator:

Because decision trees are flexible in capturing complex relationships in the housing dataset, they make good base estimators for AdaBoost. Decision trees concentrate on cases in the previous trees incorrectly classified in each boosting iteration, which helps the model discover complex patterns.

Analysis Using Model Evaluation Metrics:

```
r2_score_ABR= r2_score(Y_test,ABR_pred)
print("R-squared score using AdaBoost Regressor",r2_score_ABR)
```

```
R-squared score using AdaBoost Regressor 0.35829306567828045
```

```
mae_ABR= mean_absolute_error(Y_test,ABR_pred)
print("Mean Absolute Error using AdaBoost Regressor",mae_ABR)
```

```
Mean Absolute Error using AdaBoost Regressor 304152.2366688532
```

```
mse_ABR= mean_squared_error(Y_test, ABR_pred)
rmse_ABR= np.sqrt(mse_ABR)
print("Root Mean Square Error using AdaBoost Regressor",rmse_ABR)
```

```
Root Mean Square Error using AdaBoost Regressor 352989.7929569652
```

```
pred_score_ABR= adaboost_regressor.score(X_test, Y_test)*100
print("Accuracy using AdaBoost Regressor",pred_score_ABR)
```

```
Accuracy using AdaBoost Regressor 35.82930656782805
```

```
r2_score_ABR_DT= r2_score(Y_test,ABR_DT_pred)
print("R-squared score using ADABoost Regressor with Decision tree",r2_score_ABR_DT)
```

```
R-squared score using ADABoost Regressor with Decision tree 0.7834050161646516
```

```
mae_ABR_DT= mean_absolute_error(Y_test,ABR_DT_pred)
print("Mean Absolute Error using ADABoost Regressor with Decision tree",mae_ABR_DT)
```

```
Mean Absolute Error using ADABoost Regressor with Decision tree 142282.921218512
```

```
mse_ABR_DT= mean_squared_error(Y_test,ABR_DT_pred)
rmse_ABR_DT= np.sqrt(mse_ABR_DT)
print("Root Mean Square Error using ADABoost Regressor with Decision tree",rmse_ABR_DT)
```

```
Root Mean Square Error using ADABoost Regressor with Decision tree 205077.48303777105
```

```
pred_score_ABR_DT= abr_dt.score(X_test, Y_test)*100
print("Accuracy using ADABoost Regressor with Decision tree",pred_score_ABR_DT)
```

```
Accuracy using ADABoost Regressor with Decision tree 78.34050161646516
```

R-squared Score (Default): 0.358

The AdaBoost Regressor accounts for about 35.8% of the variance in the target variable (Price). This implies a moderate ability to represent variations in housing prices.

R-squared Score (Decision Tree): 0.783

The AdaBoost Regressor with a Decision Tree base estimator accounts for roughly 78.3% of the variation in the target variable (Price). Compared to the default AdaBoost Regressor, this significant improvement indicates the model's improved capacity to capture the variability in housing prices.

Mean Absolute Error (MAE) (Default): 304152.23

The average absolute difference (MAE) between the actual and predicted prices is shown. In this instance, it shows that the average variance between the true prices and predictions made by the AdaBoost Regressor is about 304152.23.

Mean Absolute Error (MAE) (Decision Tree): 142282.921

The average absolute difference (MAE) between the actual and predicted prices is shown. This example shows that the average deviation between the true prices and the predictions made by the AdaBoost Regressor with Decision Tree is about 142282.921.

Root Mean Square Error (RMSE) (Default): 352989.792

The average magnitude of the errors in the model is measured by the RMSE. In this case, it shows that the average difference between the prices predicted by the AdaBoost Regressor, and the actual prices is about 352989.792.

Root Mean Square Error (RMSE) (Decision Tree): 205077.483

The average magnitude of the errors in the model is measured by the RMSE. In this case, it shows that the average difference between the prices predicted by the AdaBoost Regressor with Decision Tree and the actual prices is about 205077.483.

Accuracy (Default): 35.82%

In this context, it is understood that the proportion of accurate predictions fall within a certain range. With an accuracy of 35.82%, the AdaBoost Regressor's forecasts are roughly 35.82% of the time within a reasonable range from the actual prices.

Accuracy (Decision Tree): 78.34%

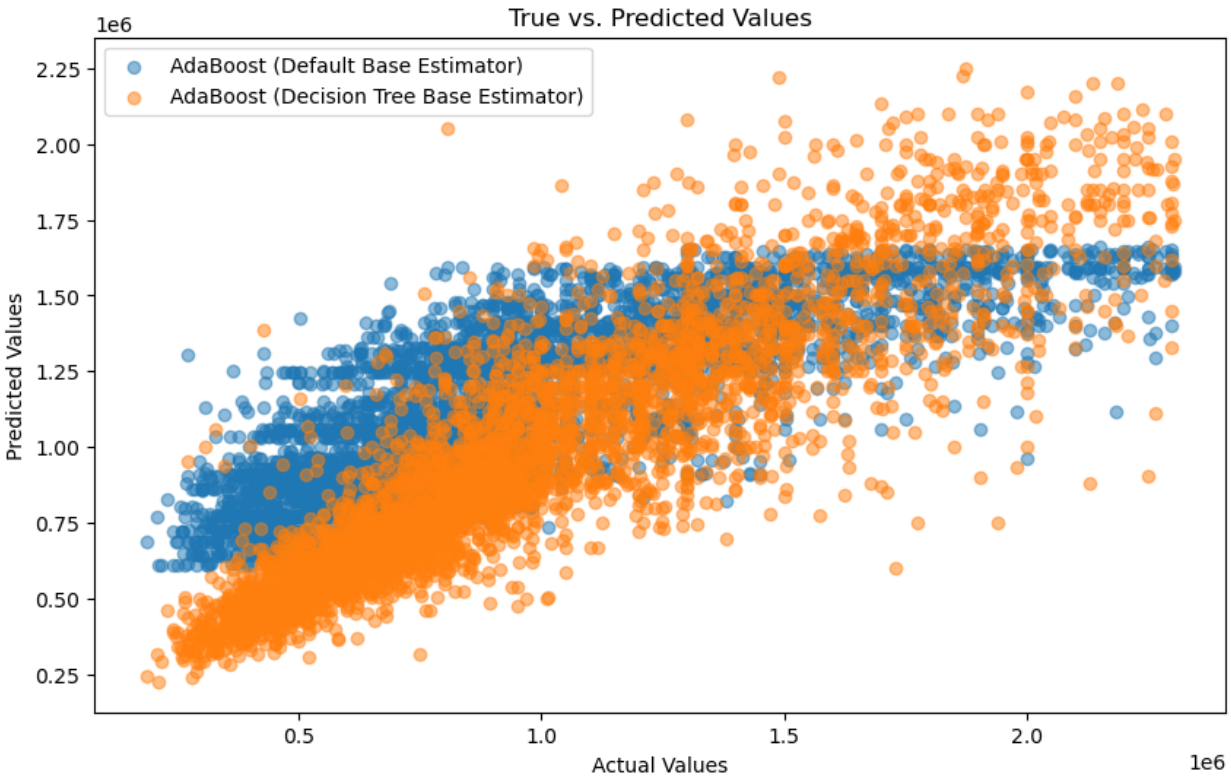
The accuracy of 78.34% indicates that, for around 78.34% of the cases, the predictions made by the AdaBoost Regressor with Decision Tree fall within an acceptable range of the actual prices.

Visualization:

1. Scatter plot Between Actual And Predicted Values

In [154]:
```
# Scatter plot between Actual and Predicted values of Adaboost Regressor
plt.figure(figsize=(10, 6))
# Scatter plot for AdaBoost with the default base estimator
plt.scatter(Y_test, ABR_pred, label="AdaBoost (Default Base Estimator)", alpha=0.5)
# Scatter plot for AdaBoost with a Decision Tree base estimator
plt.scatter(Y_test, ABR_DT_pred, label="AdaBoost (Decision Tree Base Estimator)", alpha=0.5)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("True vs. Predicted Values")
plt.legend()
plt.show()
```

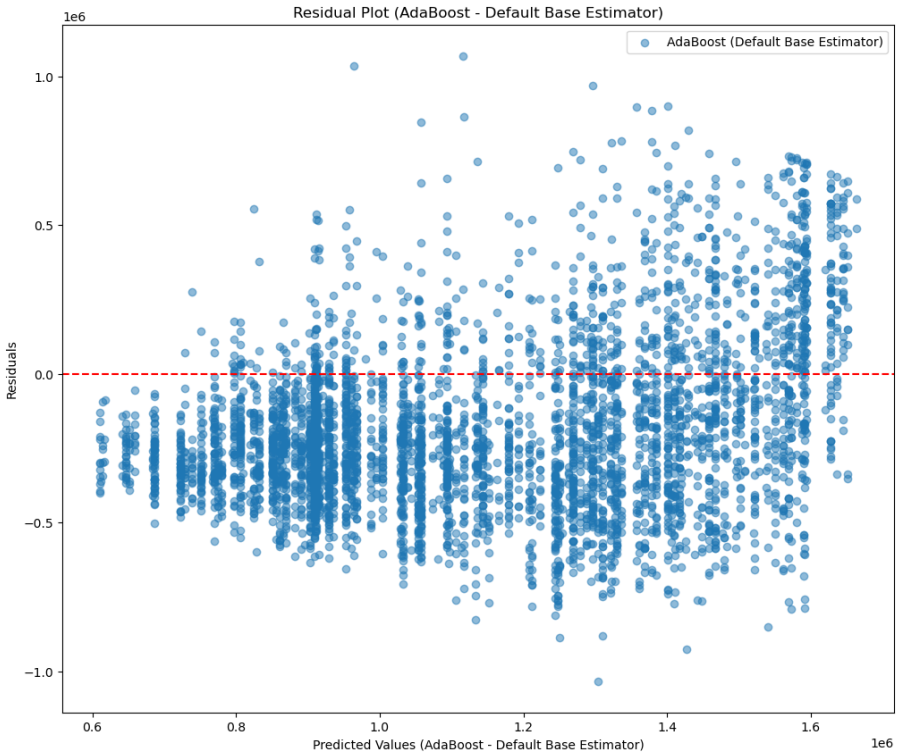
In the below scatter plot, predictions from two AdaBoost Regressor models—one with the default base estimator and the other with a Decision Tree base estimator—are contrasted with actual values (Y_test). Every dot denotes a pair of data, demonstrating the predictive power of the models. Whereas deviations show differences between actual and projected values, points along the diagonal line show correct forecasts.



2. Residual Plot for Default Base Estimator in AdaBoost Regressor

In [155]:
```
# Residual plot for Adaboost Regressor with Default base Estimator
# Residuals for the first AdaBoost model
Y_test = Y_test.squeeze()
ABR_pred = ABR_pred.ravel()
residuals = Y_test - ABR_pred
plt.figure(figsize=(12, 10))
# Scatter plot for residuals of the first AdaBoost model
plt.scatter(ABR_pred, residuals, alpha=0.5, label="AdaBoost (Default Base Estimator)")
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted Values (AdaBoost - Default Base Estimator)")
plt.ylabel("Residuals")
plt.title("Residual Plot (AdaBoost - Default Base Estimator)")
plt.legend()
plt.show()
```

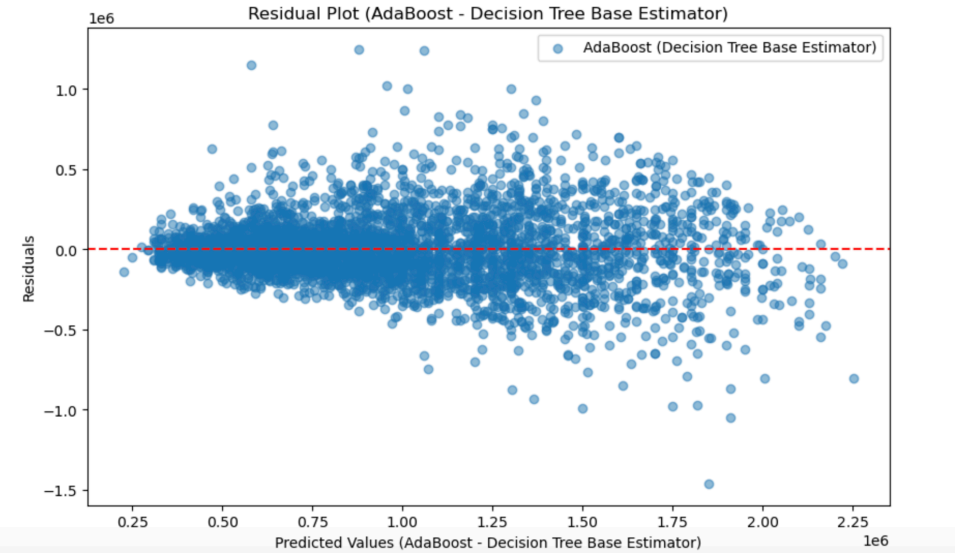
The below residual plot for the AdaBoost Regressor with the default base estimator visualizes the differences between actual and predicted values. Each point represents a data instance, with the x-axis indicating predicted values and the y-axis showing residuals. The red dashed line at y=0 serves as a reference, highlighting where the model predictions align with actual values (on the line) or deviate (above or below the line). Since the scatter of data points is very wide, the model fails to accurately predict the target outcome for various data points.



3. Residual Plot for Decision Tree Base Estimator in AdaBoost Regressor

```
In [375]: # Residual plot for Adaboost Regressor with Decision Tree Estimator
          # Residuals for the second AdaBoost model
          residuals_ = Y_test - ABR_DT_pred
          plt.figure(figsize=(10, 6))
          plt.scatter(ABR_DT_pred, residuals_, alpha=0.5, label="AdaBoost (Decision Tree Base Estimator
          plt.axhline(y=0, color='r', linestyle='--')
          plt.xlabel("Predicted Values (AdaBoost - Decision Tree Base Estimator)")
          plt.ylabel("Residuals")
          plt.title("Residual Plot (AdaBoost - Decision Tree Base Estimator)")
          plt.legend()
          plt.show()
```

The below residual plot for the AdaBoost Regressor with the Decision Tree base estimator visualizes the differences between actual and predicted values. Each point represents a data instance, with the x-axis indicating predicted values and the y-axis showing residuals. The red dashed line at y=0 serves as a reference, highlighting where the model predictions align with actual values (on the line) or deviate (above or below the line). Since the scatter of data points is uniform near the horizontal axis, the model successfully accurately predicts the target outcome for various data points.



Conclusion:

The AdaBoost Regressor is a notable model for housing price prediction, especially when using a Decision Tree as the base estimator. Improved accuracy is a result of its ensemble learning technique and flexibility in dealing with complex relationships. This model may have more potential that can be realized with extra optimization and fine-tuning.

MODEL V: RANDOM FOREST

Random Forest Regressor is an ensemble learning algorithm where a random subset of the data and features is used to train each decision tree. In order to achieve the best possible prediction, the forecasts from various trees are combined. By doing so, overfitting is less likely to occur, and an additional layer of variability is added. When compared to individual decision trees, Random Forest reduces variation more effectively. It is highly adaptable to new data because of its ensemble structure.

```
: random_forest = RandomForestRegressor(n_estimators = 100, random_state=1)
  random_forest.fit(X_train, Y_train)
```

```
  /var/folders/kp/xfqjcgz93rx88km955rkxtjm0000gn/T/ipykernel_37497/1452581609.py:2: DataConversionWarning: A co
  ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example us
  vel().
    random_forest.fit(X_train, Y_train)
```

```
: RandomForestRegressor(random_state=1)
```

```
: rf_pred = random_forest.predict(X_test)
```

Reasons for Choosing Random Forest:

Ensemble Learning for Understanding Patterns: The Melbourne housing dataset has a variety of features that may have non-linear correlations with selling prices. Using an ensemble of decision trees, the

Random Forest Regressor performs in identifying the complex varied patterns found in the dataset. This is essential for precisely predicting house prices, which might be impacted by numerous variables interacting in complex ways.

Robustness Against Overfitting: Real estate datasets frequently include noise and outliers, which can cause conventional models to overfit. The Random Forest Regressor offers resistance against overfitting by combining numerous decision trees and utilizing an ensemble technique. This makes it a good fit for the Melbourne housing dataset, which frequently contains outliers.

Features of Importance for Real Estate Insights: In the context of real estate, Random Forest's interpretability is useful. The model enables stakeholders to determine the relative influence of different features on house prices by providing a measure of feature relevance. Understanding the variables influencing Melbourne real estate values involves doing this.

Analysis Using Model Evaluation Metrics:

```
r2_score_rf= r2_score(Y_test,rf_pred)
print("R-squared score using Random Forest Regressor",r2_score_rf)
```

```
R-squared score using Random Forest Regressor 0.7916070511619395
```

```
mae_rf= mean_absolute_error(Y_test,rf_pred)
print("Mean Absolute Error using Random Forest Regressor",mae_rf)
```

```
Mean Absolute Error using Random Forest Regressor 141453.23110883113
```

```
mse_rf= mean_squared_error(Y_test,rf_pred)
rmse_rf= np.sqrt(mse_rf)
print("Root Mean Square Error using Random Forest Regressor",rmse_rf)
```

```
Root Mean Square Error using Random Forest Regressor 201157.06548278473
```

```
pred_score_rf= random_forest.score(X_test, Y_test)*100
print("Accuracy using Random Forest Regressor",pred_score_rf)
```

```
Accuracy using Random Forest Regressor 79.16070511619395
```

R-squared Score: 0.791

The Random Forest Regressor successfully captures variations in housing prices within the Melbourne dataset, as shown by the high R-squared score of 79.1%.

Mean Absolute Error (MAE): 141453.23

The average absolute difference between the expected and actual prices is represented by the MAE, which is 141453.23. With the complex structure of the dataset, this metric, which is specific to the housing setting of Melbourne, demonstrates the model's capacity to generate good predictions.

Root Mean Square Error (RMSE): 201157.06

The average magnitude of errors in the model's predictions is shown by the RMSE of 201157.06. This measure, which is particular to the Melbourne housing dataset, highlights the moderate amount of precision with which the Random Forest Regressor can produce predictions.

Accuracy: 79.16%

With an accuracy of 79.16%, it can be concluded that for approximately 78.16% of the cases, the Random Forest predictions are within an appropriate range of the true price.

Visualization:

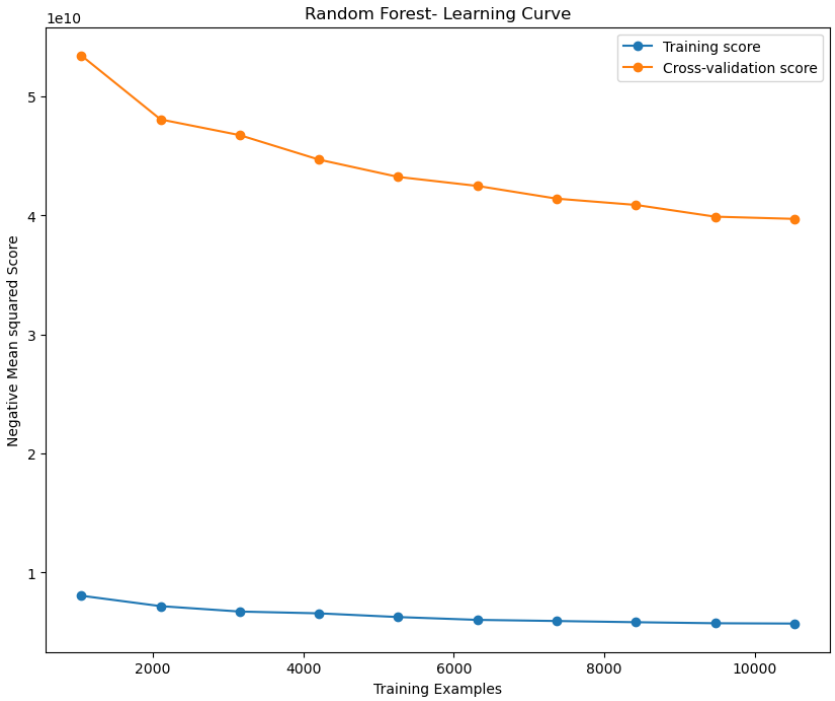
1. Learning Curve of Random Forest

In [399]:
```
# Learning Curve for Random Forest
# Using negative mean squared error as the scoring metric
train_sizes, train_scores, test_scores = learning_curve(
    random_forest, X_train, Y_train, cv=5, scoring="neg_mean_squared_error", train_sizes=np.linspace(0.1, 1.0, 10))
train_scores_mean = -np.mean(train_scores, axis=1)
test_scores_mean = -np.mean(test_scores, axis=1)
plt.figure(figsize=(10, 8))
plt.plot(train_sizes, train_scores_mean, marker='o', label='Training score')
plt.plot(train_sizes, test_scores_mean, marker='o', label='Cross-validation score')
plt.xlabel("Training Examples")
plt.ylabel("Negative Mean squared Score")
plt.title("Random Forest- Learning Curve")
plt.legend()
plt.show()
```

The below Random Forest model's learning curve shows how different training set sizes affect the model's performance as indicated by Negative mean squared scores. The model's performance on the training set is represented by the Training score curve, and its ability to generalize on a validation set is indicated by the Cross-validation score curve. Plotting aids in determining whether further training instances might help the model or if its performance has peaked. The model's performance on the training set and the cross-validated set is visualized as the number of training samples increases.

Key factors to be noticed:

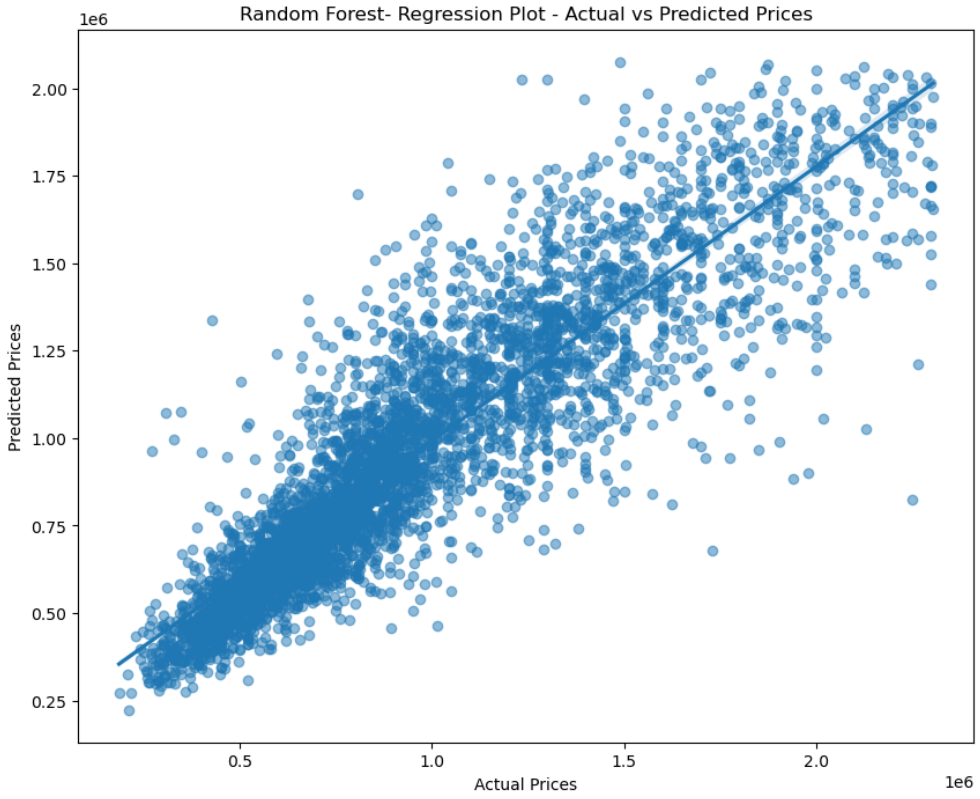
The decreasing trend in the cross-validation score suggests that as more data is added, the model's performance on unseen data is not improving consistently. This indicates that the model is struggling to generalize well to new examples. The fact that the training score is either constant or slightly decreasing implies that the model may already be fitting the training data well. However, this could indicate that the model is reaching a point of saturation where additional data doesn't much contribute to better fitting.



2. Regression Plot between Actual vs Predicted Prices

In [166]:
```
# Regression plot using seaborn's regplot
plt.figure(figsize=(10, 8))
sns.regplot(x=Y_test.squeeze(), y=rf_pred, scatter_kws={'alpha': 0.5})
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Random Forest- Regression Plot - Actual vs Predicted Prices")
plt.show()
```

The below regression plot illustrates the relationship between actual and predicted prices from a Random Forest regression model. The scatter points represent individual data pairs, and the regression line captures the overall trend in the predictions. Deviations from the line reveal instances where the model's predictions differ from the actual values.



Conclusion:

In the Melbourne dataset, the Random Forest Regressor is shown to be an effective choice for housing price prediction. Its notable predictive performance can be due to its robustness against overfitting, feature importance providing, and ensemble learning approach that is specifically designed to manage non-linear connections. Its accuracy might be improved for practical uses in the Melbourne real estate market with more research, optimization, and feature engineering.

MODEL VI: XG BOOST REGRESSOR

One of the most powerful ensembles learning algorithms in the gradient boosting family is the XGBoost (Extreme Gradient Boosting) Regressor. It performs a variety of machine learning tasks, particularly problems with regression, and is known for its effectiveness, speed, and high performance. Regularization terms are included in the function's objective by XGBoost, which helps to improve model generalization and prevent overfitting. Regression problems benefit from their effectiveness since it is well-suited for predicting continuous numerical values.

```
XGBR = XGBRegressor(random_state=42)
XGBR.fit(X_train, Y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

```
XGBR_pred = XGBR.predict(X_test)
```

Reasons for Choosing XGBoost Regressor:

Robust Handling of Heterogeneous Features: A range of factors with varying sizes and degrees of impact on housing prices are present in the Melbourne housing dataset. The XGBoost Regressor is a perfect fit for this dataset because of its ability to handle heterogeneous features with robustness. With its ability to automatically handle numerical features without requiring considerable preprocessing, it successfully controls the diversity of feature types. This capacity simplifies the modeling procedure and makes it easier to integrate a variety of features into the prediction task.

Regularization to Avoid Overfitting: XGBoost's regularization algorithms provide protection against overfitting in a dataset free of outliers or missing data, where the model may attempt to fit noise in the absence of observable patterns, leading to a more robust and dependable predictive model.

Managing Non-linearity in Feature Relationships: A model that can capture the complex non-linear relationships between features and housing prices is necessary because the Melbourne housing dataset does not contain any outliers or missing data (as data preprocessing is done in phase 1). Because of its natural ability to manage non-linear relationships efficiently, it is an excellent fit for datasets in which the influence of features on housing prices might not show a straightforward linear trend.

Analysis Using Model Evaluation Metrics:

```
r2_score_XGBR= r2_score(Y_test,XGBR_pred)
print("R-squared score using XGBoost Regressor",r2_score_XGBR)
```

```
R-squared score using XGBoost Regressor 0.811171682589503
```

```
mae_XGBR= mean_absolute_error(Y_test,XGBR_pred)
print("Mean Absolute Error using XGBoost Regressor",mae_XGBR)
```

```
Mean Absolute Error using XGBoost Regressor 135981.66746466028
```

```
mse_XGBR= mean_squared_error(Y_test,XGBR_pred)
rmse_XGBR= np.sqrt(mse_XGBR)
print("Root Mean Square Error using XGBoost Regressor",rmse_XGBR)
```

```
Root Mean Square Error using XGBoost Regressor 191481.7301815415
```

```
pred_score_XGBR= XGBR.score(X_test, Y_test)*100
print("Accuracy using XGBoost Regressor",pred_score_XGBR)
```

```
Accuracy using XGBoost Regressor 81.1171682589503
```

R-squared Score: 0.811

The XGBoost Regressor does exceptionally well in explaining the variance in housing prices within the Melbourne dataset, as shown by its high R-squared score of 81.1%. This is critical for a practical application where accurate predictions and an understanding of the changing nature of housing prices are important.

Mean Absolute Error (MAE): 135981.66

The average absolute difference between the expected and actual prices is shown by the MAE of 135981.66. This statistic shows that the XGBoost Regressor is capable of accurately predicting the outcome in a clean dataset without being affected by anomalous data.

Root Mean Square Error (RMSE): 191481.73

The average magnitude of errors in the model's predictions is shown by the RMSE of 191481.73. This statistic highlights the XGBoost Regressor's capacity to produce precise predictions with a high degree of accuracy in a clean dataset.

Accuracy: 81.11%

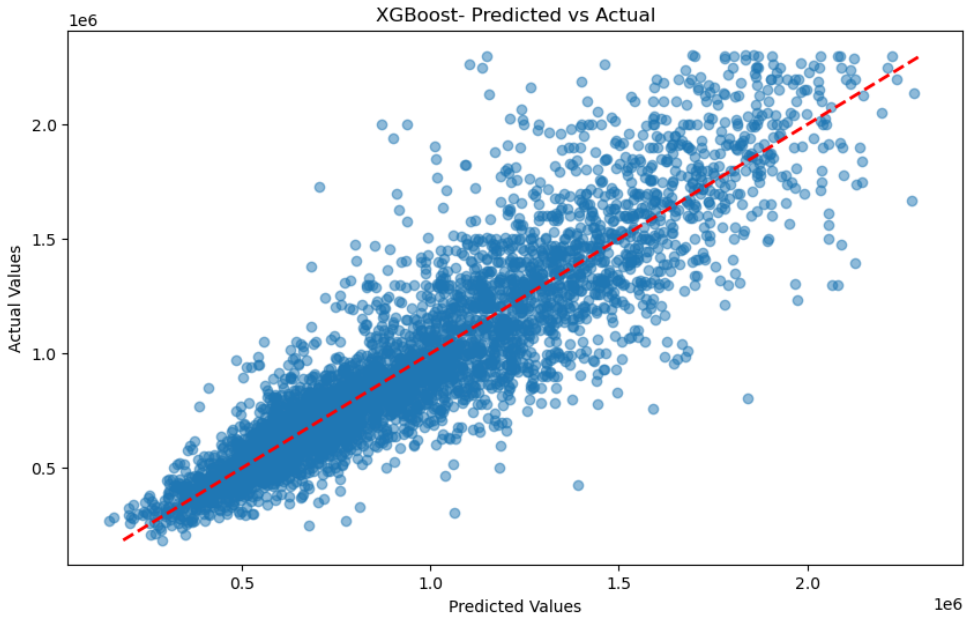
In this context it is understood that the proportion of accurate predictions that fall within a certain range. The XGBoost Regressor's predictions are within an appropriate range of the true prices for approximately 81.11% of the cases.

Visualization:

1. Scatter Plot Between predicted and Actual values

```
In [174]: #Scatter Plot between Predicted and Actual Values
          plt.figure(figsize=(10, 6))
          plt.scatter(XGBR_pred, Y_test, alpha=0.5)
          plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], linestyle='--', color='red', linewidth=2)
          plt.xlabel("Predicted Values")
          plt.ylabel("Actual Values")
          plt.title("XGBoost- Predicted vs Actual")
          plt.show()
```

The below scatter plot visualizes the relationship between predicted and actual values from an XGBoost Regressor model. Points clustered around the diagonal red dashed line indicate accurate predictions, while deviations reveal discrepancies. This plot provides a quick assessment of the model's performance in capturing the underlying patterns in the data. It can be noticed that the spread is uniform across the red line, indicating correct predictions of the target variable (Price).



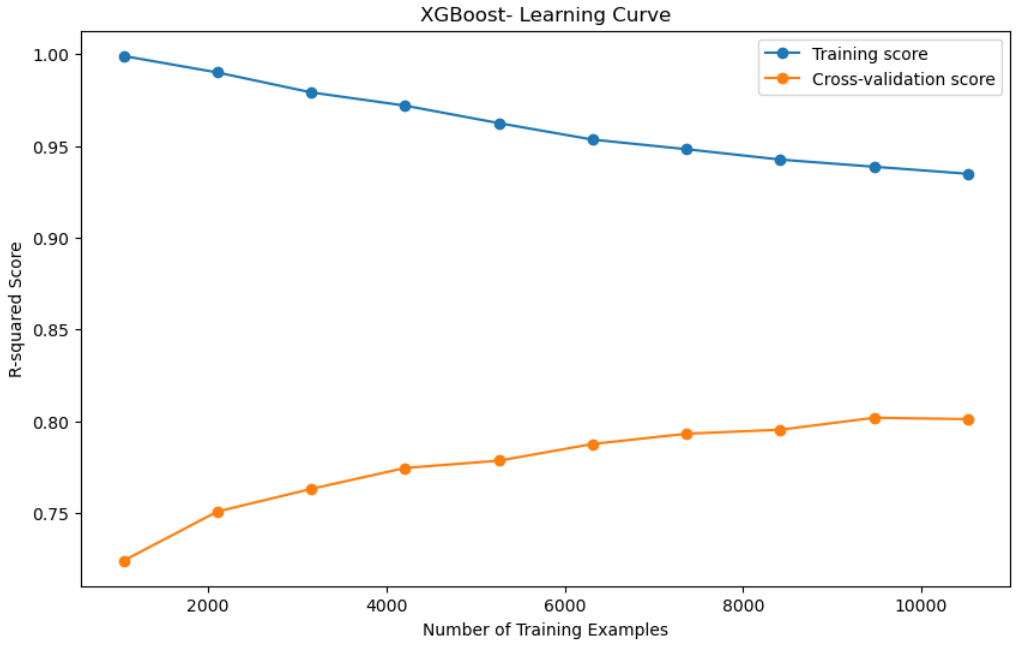
2. Learning Curve of XGBoost Regressor

In [175]:
```
# Learning Curve for XGBoost Regressor
train_sizes, train_scores, test_scores = learning_curve(
    XGBR, X_train, Y_train, cv=5, scoring="r2", train_sizes=np.linspace(0.1, 1.0, 10)
)

train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_scores_mean, marker='o', label='Training score')
plt.plot(train_sizes, test_scores_mean, marker='o', label='Cross-validation score')
plt.xlabel("Number of Training Examples")
plt.ylabel("R-squared Score")
plt.title("XGBoost- Learning Curve")
plt.legend()
plt.show()
```

The below XGBoost Regressor's learning curve shows how the amount of training instances affects the regressor's performance as determined by R-squared scores. The Cross-validation score curve shows how well the model generalizes to a validation set, whereas the Training score curve shows how well the model performs on the training set. The plot offers information about the model's ability to generalize and aids in determining if the model would benefit from more training instances.

Key factors to be noticed:

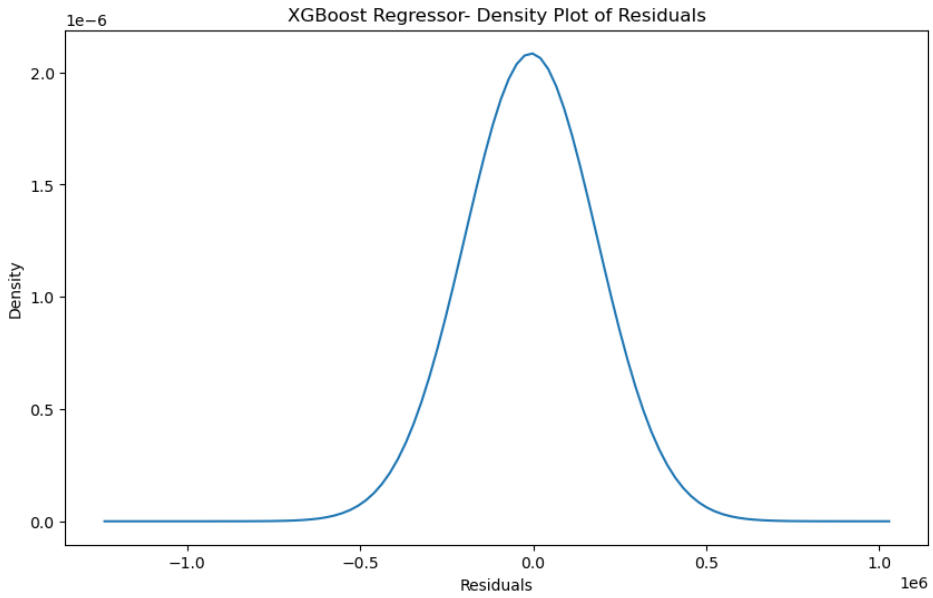
The cross-validation score shows a rise, indicating that the XGBoost Regressor's performance on unseen data steadily increases with additional data. This is a sign that the model can effectively generalize to new instances. The training score declining trend suggests that the XGBoost Regressor may be having increasing difficulty fitting the training data accurately as the training set size grows. This is good because it shows the model is generalizing more efficiently rather than just learning the training instances by memory.



3. Density Plot of Residuals:

In [395]:
```
# Density Plot of Residuals
residuals = Y_test - XGBR_pred

# mean and standard deviation of the residuals
residual_mean = np.mean(residuals)
residual_std = np.std(residuals)

# array of values within the range of residuals
x_range = np.linspace(residuals.min(), residuals.max(), 100)

# Probability Density Function of the residuals using a normal distribution
density_values = norm.pdf(x_range, residual_mean, residual_std)

plt.figure(figsize=(10, 6))
plt.plot(x_range, density_values)
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.title('XGBoost Regressor- Density Plot of Residuals')
plt.show()
```

A representation of the distribution of the errors (residuals) made by the XGBoost Regressor on the test data is shown by the density plot of residuals. For every data point, the residual shows the error in the model's prediction. As seen in the graph, the symmetry in the density plot indicates that the residuals are uniformly distributed around the mean. The peak at 0 suggests that the majority of residuals cluster around this point. It suggests that the model is accurately making predictions without any bias.



Conclusion:

The XGBoost Regressor is an accurate choice for estimating house prices in the Melbourne dataset. Its noteworthy prediction success can be due to its advanced gradient boosting methods, regularisation strategies, and capacity to manage non-linear interactions. Additional investigation, adjustment of parameters, and development of features may be necessary to enhance its precision and resistance for practical implementations in Melbourne real estate.

Comparison of Models:

The following table shows the various models with their Prediction Score and Mean Absolute Error.

	Algorithm	Prediction_score	Mean_Absolute_Error
0	DecisionTreeRegressor	60.869469	193390.613999
1	LinearRegression	63.608127	205623.054656
2	GradientBoostingRegressor	76.484865	154690.035853
3	AdaBoostRegressor	35.829307	304152.236669
4	AdaBoostRegressor With DT	78.340502	142282.921219
5	RandomForestRegressor	79.160705	141453.231109
6	XGBRegressor	81.117168	135981.667465

Top Performer: XGBoost Regressor is the top-performing model, showing the highest prediction score (81.12%) and the lowest Mean Absolute Error (135,981.67).

Ensemble Methods: Ensemble methods, including GradientBoostingRegressor, AdaBoostRegressor with Decision Tree, and RandomForestRegressor, consistently outperform standalone models. The collaborative nature of ensemble learning proves effective in considering diverse patterns in the data.

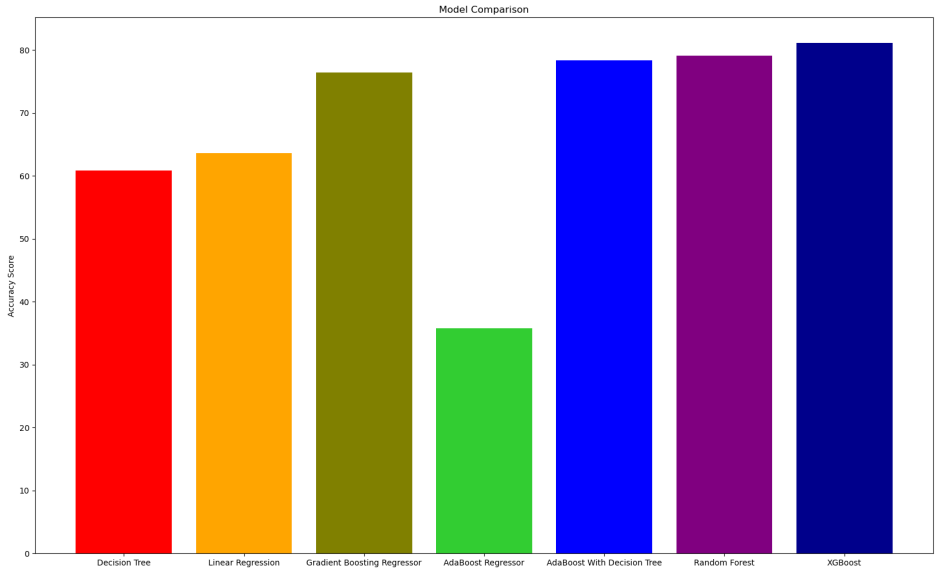
AdaBoost Improvement: AdaBoostRegressor with Decision Tree significantly improves upon the standalone AdaBoostRegressor, highlighting the importance of choosing an appropriate base estimator within the AdaBoost framework.

Moderate Performance: Decision tree and Linear Regression shows moderate performance as they do not completely able to identify complex features and diversity in the dataset.

Visualization of the Models:

```
In [397]: # different Models Comparison
models = ['Decision Tree', 'Linear Regression', 'Gradient Boosting Regressor', 'AdaBoost Regressor', 'AdaBoost With
accuracy_scores = [pred_score_dt, pred_score_lr, pred_score_GBR, pred_score_ABR, pred_score_ABR_DT, pred_score_rf ,
plt.figure(figsize=(20, 12))
colors = ['red', 'orange', 'olive', 'limegreen', 'blue', 'purple', 'darkblue']
plt.bar(models, accuracy_scores, color=colors)
plt.ylabel('Accuracy Score')
plt.title('Model Comparison')
plt.show()
```

The bar chart above compares the accuracy scores of different regression models applied to the Melbourne housing dataset. Each model is represented by a bar, and the height of the bar corresponds to the accuracy score achieved by the respective algorithm. It can be seen that highest Performing model is XGBoost Regressor.



REFERENCES:

1. https://www.geeksforgeeks.org/decision-tree/
2. https://www.learndatasci.com/tutorials/predicting-housing-prices-linear-regression-using-python-pandas-statsmodels/
3. https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html
5. https://www.geeksforgeeks.org/house-price-prediction-using-machine-learning-in-python/
6. https://www.geeksforgeeks.org/xgboost-for-regression/?ref=header_search