Logo

Description automatically generated

**College of Professional Studies**

**Northeastern University San Jose**

**MPS Analytics**

**Course: ALY6020 – Predictive Analytics**

**Module 4 Project: Investing in Nashville**

**Submitted on:**

Oct 20th, 2023

**Submitted to:**  **Submitted by:**

Professor: BEHZAD AHMADI BHAGYASHRI KADAM

**INTRODUCTION**

**Given Problem Statement:**

You just started working for a real estate company and they are looking to make a huge investment into the growing Nashville area. They’ve acquired a dataset about recent sales and want you to build a model to help them accurately find the best value deals when they go to visit next week.  By looking at the variable, Sale Price Compared To Value, that will help us see which properties are being over/under valued. If we can build an accurate model, this could help the company identify what the key factors in finding the best deal may be.

**Understanding the Dataset**

The given dataset comprises a total of **22651 records and 26 columns/attributes** each signifying a distinct property transaction in the Nashville area. These columns detail attributes such as the "Parcel ID", "Land Use", "Property Address", and so on. Among these attributes, we have specifics like "Acreage", "Year Built", "Bedrooms", and "Finished Area", which provide insights into the physical characteristics of the properties. Additionally, the dataset captures transactional details like "Sale Date", "Legal Reference", and whether the property was "Sold As Vacant". The column "Sale Price Compared to Value" is our target variable. This column aims to indicate the valuation status of a property, helping us distinguish if it's over or under its perceived market value.

Below are the datatypes of the attributes:

A screenshot of a computer screen

Description automatically generated

**PART 1 : DATA CLEANING**

* **Checking the number of missing values for each Attribute in the dataset**

A screenshot of a computer screen

Description automatically generated

From the above output, we can observe that there are highest no of missing values for Suite/Condo # which is 22651, property city and property address have 2 missing values.

Further, Finished area, foundation type , bedrooms , full bath and half bath are also seen to have few missing values.

**Dropping non-imp attributes :**

Firstly, we need to drop the irrelevant/non-imp attribute ‘Suite/Condo #’ :

A group of words on a white background

Description automatically generated

**Checking the Distribution of Numerical Attributes:**

Further, before imputing the missing values, lets visualize the distribution of Income attribute :

A graph with blue bars

Description automatically generated

A graph of different sizes and colors

Description automatically generated with medium confidence

From the above distribution, we can observe the below :

1. For Acreage , the distribution appears to be right-skewed.
2. For Neighborhood – the distribution is somewhat uniform but with a slight right skewness , the data is almost evenly distributed across the bins , but there’s a noticeable concentration on the left-most bins.
3. For land value , building value and finished area - the distribution appears to be right-skewed.
4. For year built – the distribution appears to be uniform with data almost evenly distributed.
5. For Bedrooms– the distribution for ‘bedroom’ appears to be right-skewed which means that hat a majority of the properties have a lower number of bedrooms ( with peaks for 2 & 3) , with fewer properties having a higher number of bedrooms.
6. For Full Bath - the distribution for ‘Full Bath appears to be right-skewed which means that hat a majority of the properties have a lower number of full baths (with peaks for 1 & 2) , with fewer properties having a higher number of full baths.
7. Half Bath : the distribution for ‘Half Bath’ looks like a bimodal distribution as it has 2 distinct peaks for 0 and 1 and very very few for 2 and 3

**Imputing the missing values :**

A computer screen shot of a code

Description automatically generated

A screenshot of a computer screen

Description automatically generated

For the imputation of missing values , I have imputed the missing values for numerical attributes with their respective median and for categorical attributes with their respective mode such that the original distribution & characteristics of the data is preserved.

**Checking for Duplicate Records :**

There are no duplicate records in our dataset.

**DESCRIPTIVE CHARACTERISTICS OF THE DATASET**

**A screenshot of a computer

Description automatically generated**

The above displays the statistical information for several numerical columns and below are some key insights:

1. Acreage: Most properties are between 0.04 to 17.5 acres, with an average size of about 0.45 acres.
2. Neighborhood: The neighborhood codes range from 107 to 9530.
3. Land Value: The land values range widely from $900 to about $1.87 million, with an average value of around $70,137.
4. Building Value: Building values also vary significantly, from $1,400 to about $5.82 million.
5. Finished Area: The finished areas range from 450 to 19,728 square units, with an average of 1,915 square units.
6. Year Built: The properties in the dataset were built between 1832 and 2017.
7. Bedrooms: Most properties have between 0 and 11 bedrooms, with an average of about 3 bedrooms.
8. Full Bath: The number of full bathrooms ranges from 0 to 10, with most properties having around 2.
9. Half Bath: Most properties have between 0 and 3 half bathrooms, with an average of about 0.27.

**EDA**

**Barplots for Categorical Features**

A screenshot of a computer

Description automatically generated

**A comparison of a graph

Description automatically generated**

**A comparison of a bar graph

Description automatically generated**

From the above barplots of categorical variables , below are some key insights observed :

1. Nashville city ofcourse has the highest number of properties. The other cities have significantly fewer properties.
2. Crawl Foundation Type is the most highest observed foundation type in the properties suggesting that it’s the most common construction choice in the dataset’s region.
3. For Land Use – Single Family homes dominates the data suggesting they are the most common types of properties followed by Duplex which is less frequent.
4. For exterior wall – Brick & Frame is the most common exterior wall and popular choice for property construction for our dataset.
5. C & B Grades are the most common grades for properties and highest observed which might
6. For Sale price compared to value – Over sold properties are observed to be more than the under. Almost 60% of the properties have value ‘over’ for their sale price compared to value.

**Timeseries plot for ‘Year Built’:**

A screenshot of a computer code

Description automatically generated

A graph showing a number of years

Description automatically generated

The above provides visualizations depict the distribution of properties based on their construction year (`Year Built`). Below are some key insights:

1. This plot shows the number of properties built each year.
2. There's a noticeable increase in the number of properties built around the mid-20th century, with peaks observed in certain years.
3. The number of properties built seems to have decreased after the peaks, with some fluctuations in the recent years.
4. The earlier years (before the mid-20th century) have a relatively steady but lower number of properties built.

**Pairplot for Numerical features:**

**A close-up of a message

Description automatically generated**

**A screenshot of a graph

Description automatically generated**

**A graph of a number of dots

Description automatically generated with medium confidence**

The above pairplots provide scatter plots for the selected numerical columns and below are some key insights :

1. **Finished Area vs. Bedrooms:**

As the finished area increases, the number of bedrooms tends to increase. This is expected as larger properties usually have more rooms.

1. **Finished Area vs. Full Bath:**

There's a positive correlation between the finished area and the number of full baths as Larger properties tend to have more bathrooms.

1. **Finished Area vs. Half Bath:**

The relationship between finished area and half baths might be less pronounced than full baths. However, larger properties might still have more half baths and they show a positive correlation

1. **Bedrooms vs. Full Bath:**

Properties with more bedrooms usually have more full baths. This is logical as larger homes designed to accommodate more people would need more bathrooms and hence its showing a positive correlation.

**CORRELATION PLOT**

A black and green text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

From the above correlation plot, we can understand below key insights :

1. Finished Area:

It has a strong positive correlation with Land value, building value Bedrooms and full bath. This means as the finished area of a property increases, the number of bedrooms and bathrooms also tends to increase

1. Land Value:

Has a strong positive correlation with building value, finished area and also Full bath/

1. Full Bath:

Full baths has a strong positive correlation with Building value, finished, area and bedrooms.

1. Year Built:

It doesn't show a strong correlation with other variables in this subset. This suggests that the year a property was built might not have a strong linear relationship with its size or number of rooms.

**PART 2 : LOGISTIC REGRESSION MODEL**

Before building the model, we need to prepare the data by splitting the target variable (Sale Price compared to value) and our features and then split the data into **training & test sets (20% - test and 80% train)**

A screenshot of a computer code

Description automatically generated

Next, I have initialized the logistic regression model with a maximum iteration of 1000 and a fixed random state for reproducibility. The model is then trained on the training data using the ‘fit’ method.

Further, using this trained model I have predicted the target variable on the test test.

Lastly , the performance of this model is evaluated using metrics like accuracy , precision , recall etc. from the classification report generated and also confusion matrix.

**A screenshot of a computer screen

Description automatically generated**

From the above results, we understand the below:

* **Accuracy**: Our model's accuracy is approximately **75%.** This means that the model correctly predicted the target variable (Sale Price compared to value) for about 75% of the test data.
* **Precision**: Also, the precision of our model for correctly predicting the sale price compared to value is **75%** which means that out of all the properties the model predicted as "Over" in terms of sale price compared to value, 75% of them were actually "Over."

On the other hand, the precision for “under” is 0.64 or 64% which means that out of all the properties the model predicted as "Under" in terms of sale price compared to value, 64% of them were actually "Under."

* **Recall (Sensitivity)**: The recall for predicting the sale price compared to its value is 1.00 or 100 %. This means that the model correctly identified 100% of the properties that were categorized as "Over" in the test set and is an excellent recall rate. On the other hand, the recall for “under” is 0.01 or 1% which model correctly identified only 1% of the properties that were actually categorized as "Under" in the test set. In other words, out of all the properties that had a sale price compared to value as "Under," the model could only correctly predict 1% of them.
* **F1 Score:** An F1-Score of **0.86** for the "Over" class indicates that the model has a good balance between Precision and Recall for “Over” class whereas an F1-Score of **0.02** for the "Under" class is very low, indicating that the model struggles to balance Precision and Recall for “Under” class.
* **Confusion Matrix**: The model correctly classified **3381** properties as "Over" and made only **8** mistakes by classifying "Under" properties as "Over."

On the other hand, the model correctly classified only **14** properties as "Under" but misclassified a significant number (**1128**) of "Over" properties as "Under."

* **Summary**: The logistic regression model exhibits a high accuracy of approximately 75%. However, its performance is notably imbalanced between the two classes. For properties priced "Over" their value, the model is highly proficient, correctly identifying almost all instances. however, it struggles to detect properties priced "Under" their value, often misclassifying them. While the model's precision for both classes is reasonable, the recall for the "Under" class is significantly low, indicating many missed opportunities to correctly identify such properties.

**Further, lets understand what attributes are strongly determining the property prices with the help of coefficients values of our model which give insights into its importance :**

A computer code with text

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**After plotting the coefficients values , we can understand the below :**

**Year Built:** The coefficient is negative, which means that as the year a property was built increases (i.e., newer properties), the likelihood of the property being classified as "Over" in the "Sale Price Compared To Value" decreases. This could suggest that newer properties might be priced more accurately or closer to their actual value.

**Finished Area:** The coefficient is positive, indicating that as the finished area of a property increases, the likelihood of the property being classified as "Over" in the "Sale Price Compared To Value" also increases. This suggests that larger properties might be priced higher than their actual value.

**Neighborhood:** The positive coefficient suggests that properties in certain neighborhoods might have a higher likelihood of being classified as "Over" in the "Sale Price Compared To Value". This could be due to various factors like amenities, demand, or historical trends in those neighborhoods.

**Building Value:** The positive coefficient indicates that as the building value increases, the likelihood of the property being classified as "Over" in the "Sale Price Compared To Value" also increases. This might suggest that properties with higher building values are priced higher than their actual value.

**Land Value:** The negative coefficient suggests that as the land value increases, the likelihood of the property being classified as "Over" decreases. This could mean that properties with higher land values might be priced closer to their actual value.

In summary, the model suggests that factors like the year a property was built, its finished area, its neighborhood, and its building and land values are among the primary drivers influencing whether a property's sale price is higher or lower than its actual value.

**PART 3 : DECISION TREE MODEL**

The below code for Decision Tree model initializes, trains and evaluates the model using decision tree classifier. After training, it predicts on the test data and finally the model’s performance is understood using accuracy & other classification metrics.

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

From the above results, we understand that the below:

* **Accuracy**: Our model's accuracy is approximately **71%.** This means that the model correctly predicted the target variable (Sale Price compared to value) for about 71% of the test data.
* **Precision**: The precision of our model for correctly predicting the sale price compared to value is **0.77** or **77%** which means that out of all the properties the model predicted as "Over" in terms of sale price compared to value, 77% of them were actually "Over."

On the other hand, the precision for “under” is **0.38 or 38%** which means that out of all the properties the model predicted as "Under" in terms of sale price compared to value, 38% of them were actually "Under."

* **Recall (Sensitivity)**: The recall for predicting the sale price compared to its value is 0.86 or **86 %.** This means that the model correctly identified **86%** of the properties that were categorized as "Over" in the test set and is an good recall rate. On the other hand, the recall for “under” is 0.25 **or 25%** which model correctly identified only 25% of the properties that were categorized as "Under" in the test set. In other words, out of all the properties that had a sale price compared to value as "Under," the model could only

correctly predict 25% of them.

* **F1 Score:** An F1-Score of **0.82** for the "Over" class indicates that the model has a good balance between Precision and Recall for “Over” class whereas an F1-Score of **0.31** for the "Under" class is very low, indicating that the model struggles to balance Precision and Recall for “Under” class.
* **Confusion Matrix**: The model correctly classified **2290** properties as "Over" and made **469** mistakes by classifying "Under" properties as "Over."

On the other hand, the model correctly classified only **290** properties as "Under" but misclassified a significant number (**852**) of "Over" properties as "Under."

* **Summary:** The Decision Tree model demonstrates a reasonable accuracy of 71%. However, its performance is notably imbalanced between the two classes. While it performs well for the "Over" class, it struggles with the "Under" class, especially in terms of recall. The model tends to misclassify a significant number of "Under" properties, as indicated by the high number of false negatives. This suggests that while the model can identify properties priced "Over" their value reasonably well, it has challenges in accurately identifying properties priced "Under" their value.

**PART 4 : RANDOM FOREST MODEL**

Before building the Random Forest model, we will initialize the random forest classifier with specific hyperparameters such as n\_estimators (no of trees in the forest) , max\_depth(max depth of the trees) , n\_jobs and class\_weight (having it as balanced value as it automatically adjusts weights since our data looks a bit imbalanced)

Next, train it and then predict it on the test set. Finally, we will evaluate the model’s performance using accuracy & other classification metrics.

A computer code with text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

From the above model results, we understand the below:

* **Accuracy**: Our model's accuracy is approximately **68%.** This means that out of all predictions, around **68%** were correct.
* **Precision**: Also, the precision of our model for correctly predicting the sale price compared to value is **0.80** or **80%** which means that out of all the properties the model predicted as "Over" in terms of sale price compared to value, 80% of them were actually "Over."

On the other hand, the precision for “under” is 0.38 or 38% which means that out of all the properties the model predicted as "Under" in terms of sale price compared to value, 38% of them were actually "Under."

* **Recall (Sensitivity)**: The recall for predicting the sale price compared to its value is **0.76 or 76 %.** This means that the model correctly identified 76% of the properties that were categorized as "Over" in the test set and is an excellent recall rate. On the other hand, the recall for “under” is **0.44 or 44%** which means model correctly identified 44% of the properties that were actually categorized as "Under" in the test set. In other words, out of all the properties that had a sale price compared to value as "Under," the model could only correctly predict 5% of them.
* **F1 Score:** An F1-Score of **0.78** for the "Over" class indicates that the model has a good balance between Precision and Recall for “Over” class whereas an F1-Score of **0.41** for the "Under" class is low, indicating that the model struggles to balance Precision and Recall for “Under” class.
* **Confusion Matrix**: The model correctly classified **2586** properties as "Over" and made only **803** mistakes by classifying "Under" properties as "Over."

On the other hand, the model correctly classified only **502**  properties as "Under" but misclassified a good number (**640**) of "Over" properties as "Under."

* **Summary**: The Gradient boosting model shows a reasonable accuracy of approximately 76%. It performs exceptionally well for the "Over" class in terms of both precision and recall. However, it faces significant challenges with the "Under" class, especially in terms of recall. The model tends to misclassify a large number of "Under" properties, as indicated by the high number of false negatives. While its precision for the "Under" class is commendable, the low recall suggests that the model misses a significant portion of the "Under" properties. This indicates that while the model can identify properties priced "Over" their value very well, it has challenges in accurately identifying properties priced "Under" their value.

**PART 5 : GRADIENT BOOST MODEL**

Before building the gradient boost model, we will initialize the gradient boosting classifier, train it and then predict it on the test set. Finally, we will evaluate the model’s performance using accuracy & other classification metrics.

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

From the above model results, we understand the below:

* **Accuracy**: Our model's accuracy is approximately **76%.** This means that out of all predictions, around **76%** were correct.
* **Precision**: Also, the precision of our model for correctly predicting the sale price compared to value is **0.76** or **76%** which means that out of all the properties the model predicted as "Over" in terms of sale price compared to value, 76% of them were actually "Over."

On the other hand, the precision for “under” is 0.71or 71% which means that out of all the properties the model predicted as "Under" in terms of sale price compared to value, 64% of them were actually "Under."

* **Recall (Sensitivity)**: The recall for predicting the sale price compared to its value is **0.99 or 99 %.** This means that the model correctly identified 99% of the properties that were categorized as "Over" in the test set and is an excellent recall rate. On the other hand, the recall for “under” is **0.05 or 5%** which model correctly identified only 5% of the properties that were actually categorized as "Under" in the test set. In other words, out of all the properties that had a sale price compared to value as "Under," the model could only correctly predict 5% of them.
* **F1 Score:** An F1-Score of **0.86** for the "Over" class indicates that the model has a good balance between Precision and Recall for “Over” class whereas an F1-Score of **0.09** for the "Under" class is very low, indicating that the model struggles to balance Precision and Recall for “Under” class.
* **Confusion Matrix**: The model correctly classified **3366** properties as "Over" and made only **23** mistakes by classifying "Under" properties as "Over."

On the other hand, the model correctly classified only **56** properties as "Under" but misclassified a significant number (**1086**) of "Over" properties as "Under."

* **Summary**: The Gradient boosting model shows a reasonable accuracy of approximately 76%. It performs exceptionally well for the "Over" class in terms of both precision and recall. However, it faces significant challenges with the "Under" class, especially in terms of recall. The model tends to misclassify a large number of "Under" properties, as indicated by the high number of false negatives. While its precision for the "Under" class is commendable, the low recall suggests that the model misses a significant portion of the "Under" properties. This indicates that while the model can identify properties priced "Over" their value very well, it has challenges in accurately identifying properties priced "Under" their value.

**PART 6 : COMPARISON & RECOMMENDATION**

Lets compare the results , observations & metrics of all the Models and summarize the conclusion :

* 1. **Logistic Regression:**

Accuracy: 75.52%

Precision: Over (76%), Under (71%)

Recall: Over (99%), Under (5%)

F1-Score: Over (86%), Under (9%)

* 1. **Decision Tree:**

Accuracy: 70.85%

Precision: Over (77%), Under (38%)

Recall: Over (86%), Under (25%)

F1-Score: Over (82%), Under (31%)

* 1. **Random Forest:**

Accuracy: 68.15%

Precision: Over (80%), Under (38%)

Recall: Over (76%), Under (44%)

F1-Score: Over (78%), Under (41%)

* 1. **Gradient Boosting:**

Accuracy: 75.52%

Precision: Over (76%), Under (71%)

Recall: Over (99%), Under (5%)

F1-Score: Over (86%), Under (9%)

A graph with a line going up

Description automatically generated

**Recommendation:**

Based on the benchmarking metrics:

1. Logistic Regression and Gradient Boosting have the highest accuracy and similar performance metrics. They excel in identifying the "Over" class but struggle with the "Under" class, especially in terms of recall.
2. Decision Tree has a balanced performance between the two classes but has a lower overall accuracy compared to Logistic Regression and Gradient Boosting.
3. Random Forest, despite being an ensemble method, has the lowest accuracy among the models. However, it has a more balanced recall between the two classes compared to Logistic Regression and Gradient Boosting.
4. Considering the metrics and the business context, if the real estate company prioritizes overall accuracy and precision, Logistic Regression or Gradient Boosting would be the best choice. However, if they want a more balanced performance between the two classes, especially in terms of recall for the "Under" class, they might consider the Random Forest model.

**References**

1. https://www.analyticsvidhya.com/blog/2021/10/building-an-end-to-end-logistic-regression-model/
2. <https://www.datacamp.com/tutorial/random-forests-classifier-python>
3. <https://towardsdatascience.com/master-machine-learning-random-forest-from-scratch-with-python-3efdd51b6d7a>
4. <https://northeastern.instructure.com/courses/160443/pages/lesson-4-3-gradient-boosting-introduction?module_item_id=9500073>

**Appendix**

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  #Loading data#  df = pd.read\_csv('/Users/bhagyashrikadam/Documents/NEU\_ASSIGNMENTS/ALY6020/Module4/week 4 - Nashville\_housing\_data.csv')  df.shape  print(df.dtypes)  df.head(10)  **## PART1 ##**  **## Data Cleaning ##**  **# Checking the number of missing values ##**  **missing\_values = df.isnull().sum()**  **print(missing\_values)**  # Drop the non-imp variables #  df = df.drop(['Suite/ Condo #','Unnamed: 0'],axis = 1)  missing\_values = df.isnull().sum()  print(missing\_values)  # Checking the distribution before imputation #  df.hist(figsize=(16,14),bins=25);  # Impute missing values #  # Numerical columns  numerical\_cols = ['Finished Area', 'Bedrooms', 'Full Bath', 'Half Bath']  for col in numerical\_cols:  median\_value = df[col].median()  df[col].fillna(median\_value, inplace=True)  # Categorical columns  categorical\_cols = ['Property Address', 'Property City', 'Foundation Type']  for col in categorical\_cols:  mode\_value = df[col].mode()[0]  df[col].fillna(mode\_value, inplace=True)  # Check if there are any missing values left  df.isnull().sum()  # Checking for outliers #  import matplotlib.pyplot as plt  import seaborn as sns  # Plot boxplots for numerical columns to visualize outliers  plt.figure(figsize=(16, 10))  for i, col in enumerate(numerical\_cols, 1):  plt.subplot(2, 2, i)  sns.boxplot(y=df[col])  plt.title(f'Box plot of {col}')  plt.ylabel(col)  plt.tight\_layout()  plt.show()  # Descriptive statistics for the Dataset #  numerical\_data = df.select\_dtypes(include=['int64', 'float64'])  numerical\_data.describe()  # Bar plots for categorical columns  categorical\_cols = ['Property City', 'Foundation Type', 'Land Use', 'Exterior Wall', 'Grade' ,'Sale Price Compared To Value']  plt.figure(figsize=(20, 20))  for i, col in enumerate(categorical\_cols, 1):  plt.subplot(3, 2, i)  df[col].value\_counts().plot(kind='bar')  plt.title(f'Bar plot of {col}')  plt.ylabel('Count')  plt.xticks(rotation=45)  plt.tight\_layout()  plt.show()  # Time Series Plot and Density Plot for 'Year Built'  plt.figure(figsize=(18, 8))  # Time Series Plot  plt.subplot(1, 2, 1)  df['Year Built'].value\_counts().sort\_index().plot()  plt.title('Time Series Plot of Year Built')  plt.xlabel('Year Built')  plt.ylabel('Number of Properties')  # Density Plot  plt.subplot(1, 2, 2)  sns.kdeplot(df['Year Built'], shade=True)  plt.title('Density Plot of Year Built')  plt.xlabel('Year Built')  plt.ylabel('Density')  plt.tight\_layout()  plt.show()  # Correlation plot  plt.figure(figsize=(15, 10))  correlation\_matrix = df.corr()  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', linewidths=0.5)  plt.title('Correlation Matrix')  plt.show()  # Pair plots for a subset of numerical columns  # Pair plots for a subset of numerical columns with categorization based on 'Sale Price Compared To Value'  selected\_cols = ['Finished Area', 'Bedrooms', 'Full Bath', 'Half Bath']  sns.pairplot(df[selected\_cols + ['Sale Price Compared To Value']], hue='Sale Price Compared To Value', kind='scatter', diag\_kind='kde', palette='viridis')  plt.suptitle('Pair Plots for Numerical Columns Categorized by Sale Price Compared To Value', y=1.02)  plt.show()  ## Logistic Regression Model using sklearn ##  from sklearn.linear\_model import LogisticRegression  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  # Splitting the data  X = pd.get\_dummies(df.drop(['Sale Price Compared To Value'], axis=1), drop\_first=True)  y = df['Sale Price Compared To Value']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Initializing and training the logistic regression model  logreg = LogisticRegression(max\_iter=1000 , random\_state=42)  logreg.fit(X\_train, y\_train)  # Predicting on the test set  y\_pred = logreg.predict(X\_test)  # Evaluating the model  accuracy = accuracy\_score(y\_test, y\_pred)  classification\_rep = classification\_report(y\_test, y\_pred)  conf\_matrix = confusion\_matrix(y\_test, y\_pred)  print(accuracy)  print(classification\_rep)  print(conf\_matrix)  ## Computing Coefficients for our Model ##  import matplotlib.pyplot as plt  import numpy as np  # Extracting the coefficients  coefficients = logreg.coef\_[0]  features = X.columns  # Creating a DataFrame for coefficients  coef\_df = pd.DataFrame({'Feature': features, 'Coefficient': coefficients})  # Sorting the DataFrame by the absolute value of the coefficients  coef\_df = coef\_df.sort\_values(by='Coefficient', key=abs, ascending=False)  coef\_df  # Plotting the top features  plt.figure(figsize=(10, 10))  plt.barh(coef\_df['Feature'][:20], coef\_df['Coefficient'][:20]) # Top 20 features  plt.xlabel('Coefficient Value')  plt.ylabel('Feature')  plt.title('Top 20 Features by Coefficient Value')  plt.gca().invert\_yaxis() # To display the top feature at the top  plt.show()  # PART 3 : Decision Tree Model #  from sklearn.tree import DecisionTreeClassifier  # Initializing and training the decision tree model  dt\_model = DecisionTreeClassifier(random\_state=42)  dt\_model.fit(X\_train, y\_train)  # Predicting on the test set  y\_pred\_dt = dt\_model.predict(X\_test)  # Evaluating the decision tree model  accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt)  classification\_rep\_dt = classification\_report(y\_test, y\_pred\_dt)  conf\_matrix\_dt = confusion\_matrix(y\_test, y\_pred\_dt)  print(accuracy\_dt)  print(classification\_rep\_dt)  print(conf\_matrix\_dt)  # PART 4 : Random Forest Model #  from sklearn.ensemble import RandomForestClassifier  # Initializing and training the random forest model  ##rf\_model = RandomForestClassifier(random\_state=42)  rf\_model = RandomForestClassifier(n\_estimators=50, max\_depth=10, n\_jobs=-1, random\_state=42 , class\_weight='balanced')  rf\_model.fit(X\_train, y\_train)  # Predicting on the test set  y\_pred\_rf = rf\_model.predict(X\_test)  # Evaluating the random forest model  accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)  classification\_rep\_rf = classification\_report(y\_test, y\_pred\_rf)  conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)  print(accuracy\_rf)  print(classification\_rep\_rf)  print(conf\_matrix\_rf)  # PART 5 : Gradient Boost Model #  from sklearn.ensemble import GradientBoostingClassifier  # Initializing and training the gradient boost model  gb\_model = GradientBoostingClassifier(random\_state=42)  gb\_model.fit(X\_train, y\_train)  # Predicting on the test set  y\_pred\_gb = gb\_model.predict(X\_test)  # Evaluating the gradient boost model  accuracy\_gb = accuracy\_score(y\_test, y\_pred\_gb)  classification\_rep\_gb = classification\_report(y\_test, y\_pred\_gb)  conf\_matrix\_gb = confusion\_matrix(y\_test, y\_pred\_gb)  print(accuracy\_gb)  print(classification\_rep\_gb)  print(conf\_matrix\_gb)  ## Comparison ##  import matplotlib.pyplot as plt  # Model names  models = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'Gradient Boosting']  # Accuracy values for each model (based on the provided results)  accuracies = [0.7552, 0.7085, 0.6815, 0.7552]  plt.figure(figsize=(10, 6))  plt.plot(models, accuracies, marker='o', linestyle='-', color='b')  plt.title('Model Performance Comparison')  plt.xlabel('Models')  plt.ylabel('Accuracy')  plt.grid(True, which='both', linestyle='--', linewidth=0.5)  plt.tight\_layout()  plt.show() |
|  |