# Critical Analysis Report

Name: Pradyot Jain

Student ID: 48479985

Number of Words : 1065

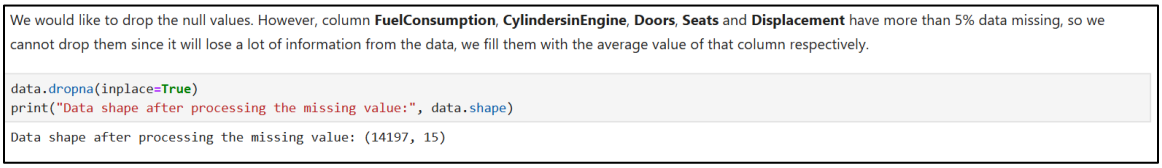
## 1. Introduction

This report presents a critical analysis of a data analysis project aimed at predicting vehicle prices using three machine learning models: Linear Regression (LR), Decision Tree Regression (DTR), and Multi-Layer Perceptron (MLP). The project aimed to identify the best-performing model while ensuring that data preprocessing, feature selection, and model tuning steps align with the project objectives. This analysis addresses critical issues in the code implementation and statements, focusing on areas that impact the models' predictive accuracy, interpretability, and alignment with the stated aims. The objective of this report is to identify, justify, and correct these issues to improve the overall quality of the project.

## 2. Critical Issues: Identification, Justification, and Correction

### 2.1. Issue with Null Value Handling

**Identification:** The initial code for handling null values was inconsistent with the approach described in the statement. The statement indicated that columns with more than 5% missing data would have missing entries filled with the mean, while columns with less than 5% missing data would have rows with missing values dropped. However, the code did not fully implement this approach.


We would like to drop the null values. However, column **FuelConsumption**, **CylindersinEngine**, **Doors**, **Seats** and **Displacement** have more than 5% data missing, so we cannot drop them since it will lose a lot of information from the data, we fill them with the average value of that column respectively.

```
data.dropna(inplace=True)
print("Data shape after processing the missing value:", data.shape)
```

```
Data shape after processing the missing value: (14197, 15)
```

**Justification:** Properly handling missing values is essential for maintaining data quality and ensuring unbiased predictions. A mismatch between the code and the statement could lead to an incorrect data preprocessing process, potentially impacting the model's reliability.

**Correction:** The code was updated to align with the stated approach:

- Columns with more than 5% missing values were filled with the mean of the column.
- Rows with missing values in columns with less than 5% missing data were dropped.

This correction ensures that the preprocessing step is consistent with the project's objectives, preserving valuable data while maintaining data integrity.

We would like to drop the null values. However, column **FuelConsumption**, **CylindersinEngine**, **Doors**, **Seats** and **Displacement** have more than 5% data missing, so we cannot drop them since it will lose a lot of information from the data, we fill them with the average value of that column respectively.

```
# Setting the threshold for missing data percentage
threshold = 0.05 * len(data)  # 5% of the total rows

# Identifying columns with more than 5% missing data
columns_to_fill = [col for col in data.columns if data[col].isna().sum() > threshold]

# Filling missing values in these columns with the column's mean
for col in columns_to_fill:
    data[col].fillna(data[col].mean(), inplace=True)

# Dropping rows with missing values in columns with less than 5% missing data
columns_to_dropna = [col for col in data.columns if data[col].isna().sum() <= threshold]
data.dropna(subset=columns_to_dropna, inplace=True)

print("Data shape after processing missing values:", data.shape)
```

```
Data shape after processing missing values: (15306, 15)
```

## 2.2. Use of One-Hot Encoding Instead of Ordinal Encoding

**Identification:** In the initial implementation, OrdinalEncoder was used to encode categorical variables, which assigns an integer value to each category based on its order. This approach implies a ranking among categories, which was inappropriate for categorical features like Brand and FuelType,etc, where no such ranking exists.

### Encoding

We encode the categorical features to integers with OrdinalEncoder and drop the original column for simplicity.

```
ord_enc = OrdinalEncoder(dtype=int)

data["BrandCode"] = ord_enc.fit_transform(data[["Brand"]])
data["UsedCode"] = ord_enc.fit_transform(data[["UsedOrNew"]])
data["TransmissionCode"] = ord_enc.fit_transform(data[["Transmission"]])
data["DriveTypeCode"] = ord_enc.fit_transform(data[["DriveType"]])
data["FuelTypeCode"] = ord_enc.fit_transform(data[["FuelType"]])
data["LocationCode"] = ord_enc.fit_transform(data[["Location"]])
data["BodyTypeCode"] = ord_enc.fit_transform(data[["BodyType"]])
```

```
# Drop the categorical columns
clean = data.drop(
    columns=['Brand', 'UsedOrNew', 'Transmission', 'DriveType',
             'FuelType', 'Location', 'BodyType'])
```

Now this `clean` will be used for following tasks. We check the shape and statistic info.

**Justification:** Using ordinal encoding for non-ordinal categorical features can mislead the model by introducing an artificial hierarchy. For example, encoding Brand categories in a specific order might imply that one brand is inherently "higher" or "lower" than another, which does not reflect the actual relationships in the data.

**Correction:** The encoding method was changed from ordinal encoding to one-hot encoding. One-hot encoding creates binary columns for each category, ensuring that all categories are treated independently without implying any ordinal relationship. This change aligns the encoding method with the nature of the categorical features, enhancing the interpretability and performance of the model.

### Corrected Code with One-Hot Encoding

The following code replaces ordinal encoding with one-hot encoding for the specified categorical columns:

```
# Applying one-hot encoding to the categorical columns and drop the original columns
clean = pd.get_dummies(data, columns=['Brand', 'UsedOrNew', 'Transmission', 'DriveType',
                                      'FuelType', 'Location', 'BodyType'], drop_first=True)

# Now 'data' is ready for further tasks without any implied ranking in categorical features
print("Data shape after one-hot encoding:", clean.shape)
```

```
Data shape after one-hot encoding: (15306, 103)
```

## 2.3. Feature Selection Based on Correlation with Price

**Identification:** The initial selection of features for model training included Kilometre, Year, Displacement, CylindersinEngine, and FuelConsumption. This choice did not fully utilize the correlation analysis to select the features most strongly associated with Price. Given the high number of features, a correlation analysis was conducted to identify features with the highest correlations to Price.

Then we keep the 5-top most correlated features and split the dataset. We want the training set the size of 80% of full dataset.

```
X = clean[['Kilometre','Year','Displacement','CylindersinEngine','FuelConsumption']]
y = clean['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=student_id)
```

**Justification:** Selecting features based on their correlation with the target variable improves model interpretability and accuracy. Features with weak correlations dilute the model's predictive power and can obscure significant relationships in the data.

**Correction:** Based on the correlation analysis, we updated the feature selection to include the top 5 features most correlated with Price: Kilometres, Year, Brand_Ferrari, Displacement, and FuelType_Unleaded. This update ensures that the feature selection aligns with the strongest indicators of vehicle price, improving the foundation for model performance.

Then we keep the 5-top most correlated features and split the dataset. We want the training set the size of 80% of full dataset.

```
X = clean[['Kilometres','Year','Displacement','Brand_Ferrari','FuelType_Unleaded']]
y = clean['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=student_id)
```

## 2.4. Decision Tree Regression (DTR) Model Not Tuned.

**Identification:** The statement specified that the Decision Tree Regressor (DTR) should be tuned to avoid overfitting or underfitting. However, the initial implementation of DTR was not tuned, and the model was overfitting, as indicated by significantly better performance on the training set compared to the test set.

**Decision Tree Regression (DTR)**

We perform decision tree regression to check the performance. We want to tune the model so that it's not overfitting nor underfitting.

```
regressor = DecisionTreeRegressor(max_depth=2)
regressor.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=2)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
# evaluate testing set
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("The results for testing set with DTR:")
print("MSE:", mse)
print("Root MSE:", rmse)
print("R^2:", r2)
```

```
The results for testing set with DTR:
MSE: 639786891.5874575
Root MSE: 25294.009005838863
R^2: 0.3371519478754226
```

```
# evaluate training set
y_pred = regressor.predict(X_train)
mse = mean_squared_error(y_train, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_train, y_pred)
print("The results for training set with DTR:")
print("MSE:", mse)
print("Root MSE:", rmse)
print("R^2:", r2)
```

```
The results for training set with DTR:
MSE: 477459804.80115074
Root MSE: 21850.853640101814
R^2: 0.36514363934091043
```

As the performance on training is higher then testing set , we can say it is overfitting.

**Justification:** Tuning the DTR model is essential for finding the optimal model complexity. Without tuning, the model may learn specific details from the training data that do not generalize well, leading to poor performance on the test data.

**Correction:** To improve model complexity and avoid overfitting, we used GridSearchCV to tune the max_depth, min_samples_split, and min_samples_leaf parameters, following the statement's direction to tune the DTR to avoid overfitting or underfitting. This tuning allowed the DTR model to generalize better on the test set without underfitting.

```
from sklearn.model_selection import GridSearchCV

# Defining the parameter grid
param_grid = {
    'max_depth': [2, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initializing the Decision Tree Regressor
dtr = DecisionTreeRegressor(random_state=student_id)

# Setting up the grid search with cross-validation
grid_search = GridSearchCV(estimator=dtr, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

# Fitting the model to find the best parameters
grid_search.fit(X_train, y_train)

# Output the best parameters and best score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validated score (negative MSE):", grid_search.best_score_)

# Training the model with the best parameters
best_dtr = grid_search.best_estimator_
best_dtr.fit(X_train, y_train)

# Evaluating on the test set
y_pred = best_dtr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("The results for testing set with tuned DTR:")
print("MSE:", mse)
print("Root MSE:", rmse)
print("R^2:", r2)
```

```
Best parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 5}
Best cross-validated score (negative MSE): -567322353.1127985
The results for testing set with tuned DTR:
MSE: 431561898.68258387
Root MSE: 20774.067937757976
R^2: 0.45861436168025294
```

## 2.5. Use of wrong ('Identity') Activation Function in MLP to Capture Non-Linear Relationships

**Identification:** The initial report stated that the MLP model was configured to capture non-linear relationships in the data. However, the activation function used was identity, which is linear and does not capture non-linearity.

**Multi-Layer Perceptron (MLP)**

We then deploy a MLP regressior and try to capture the inner non-linear relationship.

We would like a double hidden-layer structure with 100 and 100 for each layer respectively.

To save the calculation time, we would like to set the max_iteration as 1000.

```
regressor = MLPRegressor(
    solver='adam',
    activation='identity',
    alpha=0.0001,
    random_state=student_id,
    hidden_layer_sizes=(100,100),
    max_iter=1000
)

regressor.fit(X_train, y_train)
```

**Justification:** The identity activation function limits the MLP's capacity to capture complex patterns. Since the project required the MLP to capture non-linear relationships, using ReLU instead aligns better with this aim and improves the model's ability to learn from the data.

**Correction:** The activation function was changed from identity to ReLU, introducing non-linearity to the MLP model. This adjustment allows the model to capture complex relationships in the data, aligning with the project's aim of capturing non-linear patterns.

These adjustments allowed the MLP to align better with the project's aim of capturing non-linear relationships, potentially enhancing model performance, though further tuning may be necessary.

```
# Initialize MLP with ReLU activation function
regressor = MLPRegressor(
    solver='adam',
    activation='relu',  # Using ReLU instead of identity
    alpha=0.0001,
    random_state=student_id,
    hidden_layer_sizes=(100, 100),
    max_iter=1000
)

# Fit the model on the training data
regressor.fit(X_train, y_train)
```

```
MLPRegressor
MLPRegressor(hidden_layer_sizes=(100, 100), max_iter=1000,
             random_state=48479985)
```

## 3. Conclusion

The critical issues identified in this project highlight several areas where discrepancies between the code implementation and the project statements affected the model's accuracy and interpretability. Initially, issues such as inconsistent handling of null values, lack of tuning in the DTR model, inappropriate feature selection, the use of ordinal encoding for non-ordinal features, and the use of a linear activation function (identity) in MLP limited the models' ability to perform accurately and predictively.

By addressing these issues—ensuring null value handling aligns with project objectives, tuning the DTR model to avoid overfitting, updating the MLP activation function to ReLU to capture non-linearity, selecting features based on correlation with Price, and replacing ordinal encoding with one-hot encoding—the project's predictive performance was enhanced. These corrections ensure that the models are now more aligned with the project's objectives, providing a more accurate and interpretable approach to predicting vehicle prices.

Overall, these critical analyses and corrections strengthen the project's foundation for vehicle price prediction. Future steps may include experimenting with more complex neural networks, advanced feature engineering techniques, and further hyperparameter tuning to optimize model performance and generalization.