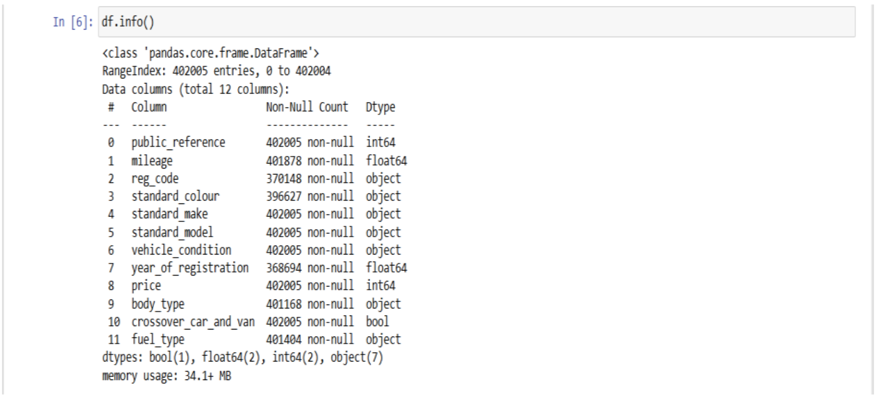
Student name: Pratik Bhutada
Student_ID: 24840332

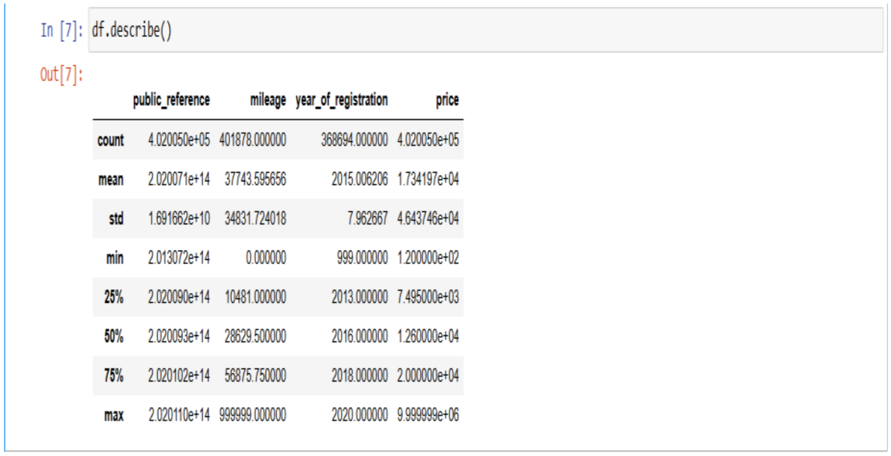
1. Data/Domain Understanding and Exploration

Our dataset Adverts.csv file has details about cars sold by AutoTrader till year 2021. Auto Trader is one of the UK's largest automotive marketplaces, it enables buying and selling of new and used vehicles by private sellers and marketplaces. The dataset has 402,005 rows and 12 columns which are 'public_reference', 'mileage', 'reg_code', 'standard_colour', 'standard_make', 'standard_model', 'vehicle_condition', 'year_of_registration', 'price', 'body_type', 'crossover_car_and_van', 'fuel_type' and they can be categorized into qualitative and quantitative types.



```
In [6]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 402005 entries, 0 to 402004
Data columns (total 12 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   public_reference       402005 non-null  int64
 1   mileage                401878 non-null  float64
 2   reg_code               370148 non-null  object
 3   standard_colour        396627 non-null  object
 4   standard_make          402005 non-null  object
 5   standard_model         402005 non-null  object
 6   vehicle_condition      402005 non-null  object
 7   year_of_registration   368694 non-null  float64
 8   price                  402005 non-null  int64
 9   body_type              401168 non-null  object
 10  crossover_car_and_van  402005 non-null  bool
 11  fuel_type              401404 non-null  object
dtypes: bool(1), float64(2), int64(2), object(7)
memory usage: 34.1+ MB
```

Fig 1.1



```
In [7]: df.describe()

Out[7]:
```

	public_reference	mileage	year_of_registration	price
count	4.020050e+05	401878.000000	368694.000000	4.020050e+05
mean	2.020071e+14	37743.595656	2015.006206	1.734197e+04
std	1.691662e+10	34831.724018	7.962667	4.643746e+04
min	2.013072e+14	0.000000	999.000000	1.200000e+02
25%	2.020090e+14	10481.000000	2013.000000	7.495000e+03
50%	2.020093e+14	28629.500000	2016.000000	1.260000e+04
75%	2.020102e+14	56875.750000	2018.000000	2.000000e+04
max	2.020110e+14	999999.000000	2020.000000	9.999999e+06

Fig 1.2

1.1. Meaning and Type of Features:
'vehicle_condition': The vehicle_condition feature is a categorical variable, stored as an object data type, used to classify vehicles based on their condition. It is binary, with two possible values: 'NEW' (indicating the vehicle is new) and 'USED'(indicating the vehicle is pre-owned). This feature is essential for categorizing vehicles, particularly when analyzing factor such as price. However, there is a significant class imbalance between the two categories, with a majority of vehicles classified as 'USED'. This imbalance could potentially impact predictive models, as the model may become biased toward the majority class. An analysis of its correlation with price reveals a weak negative correlation,

suggesting that vehicles classified as 'USED' tend to have lower prices compared to 'NEW' vehicles. (Refer to Fig 1.5)

'mileage': The mileage feature is a numerical variable, stored as a float, representing the distance a vehicle has traveled, typically measured in miles or kilometers. It is a continuous variable with a wide range of values, as some vehicles may have minimal mileage, while others may have significantly higher mileage due to extensive usage. This feature is crucial for evaluating vehicle wear and tear and plays a significant role in pricing analysis, as higher mileage often corresponds to lower prices. An analysis of its correlation with price reveals a negative correlation, indicating that as mileage increases, vehicle prices tend to decrease. Additionally, some missing values in this feature may need to be addressed. (Refer to Fig 1.4)

'year_of_registration': The year_of_registration feature is a numerical variable, stored as a float, representing the year in which the vehicle was registered. This feature is critical for understanding the depreciation of vehicle value over time and is strongly associated with pricing trends. Vehicles registered in more recent years tend to have higher prices compared to older ones. An analysis of its correlation with price reveals a moderate positive correlation, indicating that newer vehicles generally command higher prices. Attention must be given to potential outliers, such as registration years that are far in the past or future and any missing values.

1.2. Analysis of Predictive Power of Features:
This section aims to evaluate the relationships between the features in the dataset and the target variable (price) using correlation analysis. A heatmap was plotted to visualize the correlation coefficients between the features and the target variable (price). Correlation values range from -1 to +1, where +1 indicates a strong positive relationship, -1 indicates a strong negative relationship and 0 indicates no correlation.

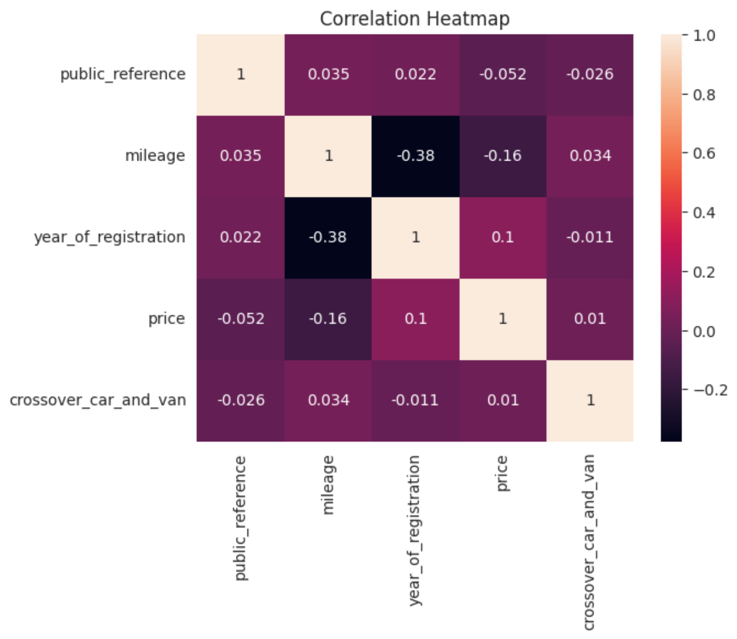


Fig 1.3

'year_of_registration' shows a mild positive correlation (e.g., +0.1), indicating newer vehicles tend to have higher prices. Mileage exhibits a negative correlation, indicating vehicles with higher mileage generally have lower prices. 'crossover_car_and_van' has a very mild correlation with our target variable 'price', which indicates that this feature might contribute slightly (close to nothing) to the variation in car prices.

1.3. Data Exploration and Visualizations:

Distribution of numerical columns:

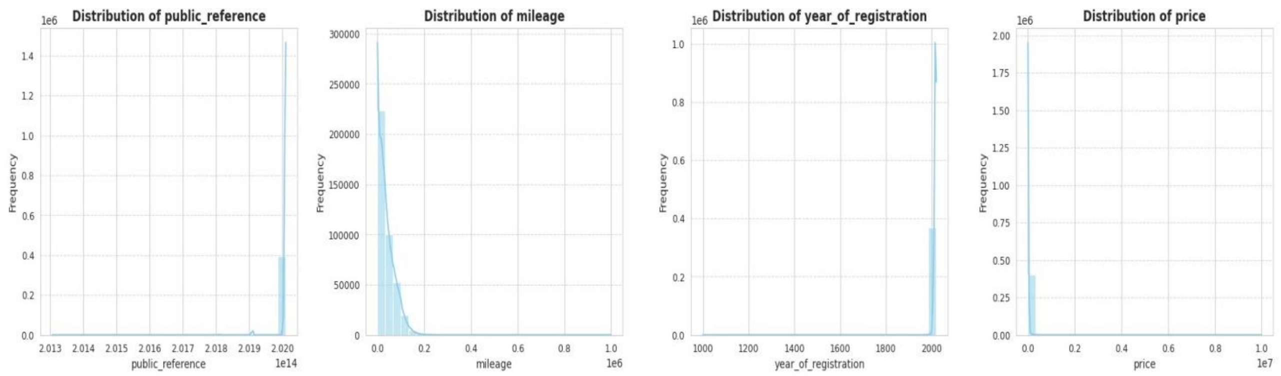


Fig 1.4

Distribution of categorical columns:

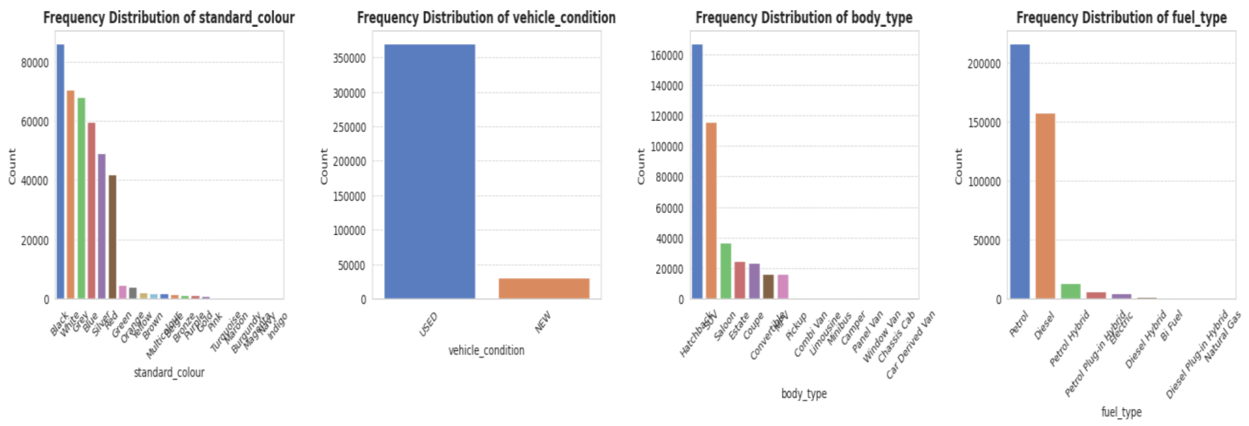


Fig 1.5

2. Data Processing for Machine Learning

2.1. Dealing with Missing Values, Outliers

The dataset has missing values in six columns, with reg_code and year_of_registration having the most. A bar chart shows how many values are missing in each column, making it easy to see the gaps in the data.

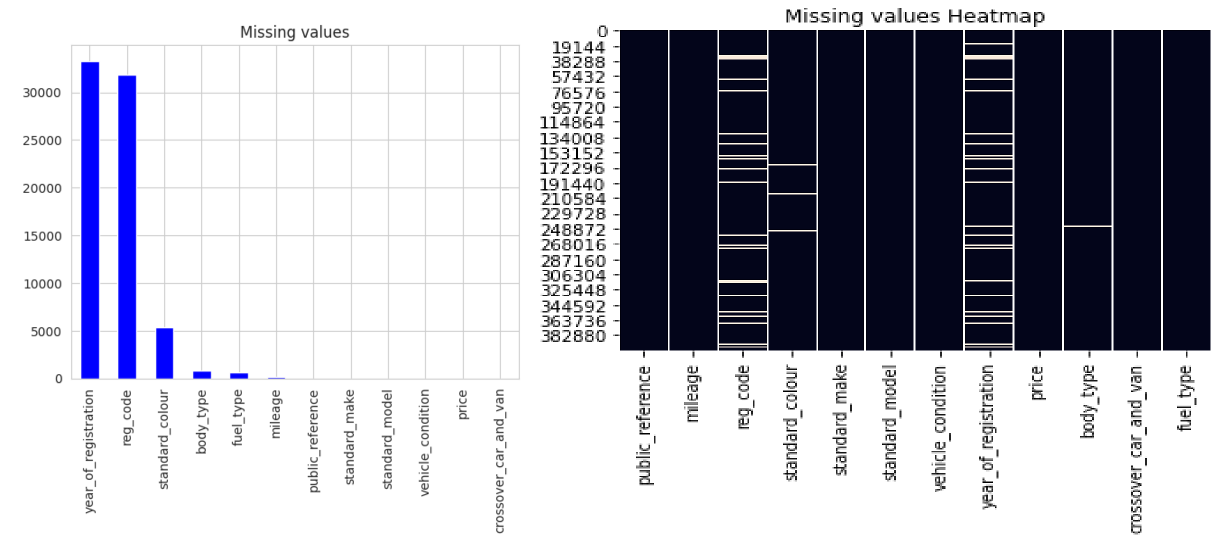


Fig 2.1

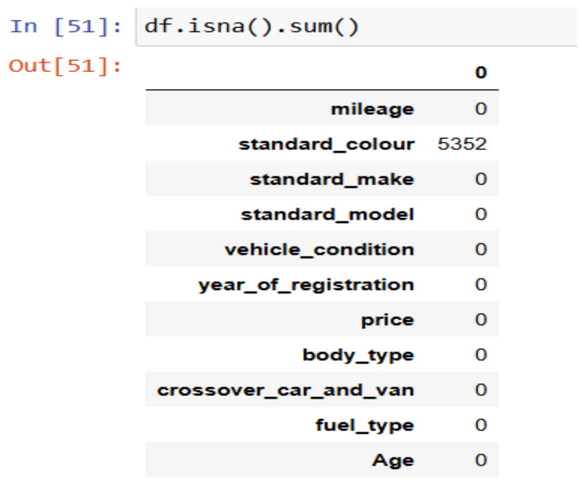
Handling missing values for year_of_registration and reg_code: To address missing values in the year_of_registration and reg_code columns, it was observed that these two features are highly correlated. For NEW cars, missing year_of_registration values were filled with 2021, and missing reg_code values were filled with 21. For other cases, the missing year_of_registration was imputed using the reg_code, based on a mapping function that followed UK vehicle registration rules. The mapping helped assign years to registration codes, with alphabetical codes corresponding to specific years (e.g., A = 1963, B = 1964).

Imputing Categorical Features: The missing values in categorical features like fuel_type and body_type were filled using the most frequent value (mode) in the respective columns. This approach ensures that the imputation is consistent with the dataset's overall distribution while minimizing bias from missing data.

Imputation of mileage: For vehicles with missing mileage values, different strategies were applied for NEW and USED cars. For NEW cars, missing mileage values were flagged for further handling. For USED cars, the vehicle's age was calculated and used to compute the average mileage per year. Missing mileage values were then imputed based on the average mileage per year and the vehicle's age, ensuring a more accurate estimate.

Outlier Removal: Outliers in the year_of_registration column were handled by removing vehicles with registration years earlier than 1892 or later than the current year (2021). This ensured that the dataset contained only logically valid data, removing entries that would otherwise disturb the model’s analysis.

After handling all the missing values & outliers:



```
In [51]: df.isna().sum()
Out[51]:
                              0
                 mileage      0
         standard_colour   5352
           standard_make      0
          standard_model      0
       vehicle_condition      0
    year_of_registration      0
                   price      0
               body_type      0
   crossover_car_and_van      0
               fuel_type      0
                     Age      0
```

Fig 2.2

2.2. Feature Engineering, Data Transformations, Feature Selection

To simplify the representation of the vehicle's make and model, a new column model_name was created by combining the existing standard_make and standard_model columns.

standard_colour: With the groupby of model_name, I have found the most frequent colour of that car and filled the missing values with the most frequent one, then just kept the colours count which are greater than 2000 (the rest are replaced by 'Other'). Logic behind this: Most cars are manufactured of that colour or that colour is the most sold.

To calculate the average mileage per year for each vehicle, a new column average_mileage_per_year was created. This column is calculated by dividing the total mileage (mileage) of each vehicle by its age (Age). The age is determined by subtracting the vehicle's year_of_registration from the current year.

Target Encoding for Categorical Features: The categorical features in the dataset are transformed using target encoding. For each categorical feature, the mean of the target variable (price) is calculated for each category. This mean is then used to replace the original categorical values, allowing the model to learn the relationship between the categorical variables and the target. This encoding technique is useful when there is a significant relationship between the feature and the target variable, as it captures the average effect of each category on the target.

```python
categorical_features = ['standard_colour','standard_make', 'body_type', 'model_name', 'fuel_type']
```

```python
# Target encoding for categorical_features
for feature in categorical_features:
    target_mean = df.groupby(feature)['price'].mean()

    df[feature] = df[feature].map(target_mean)
```

One-Hot Encoding for Vehicle Condition: The vehicle_condition feature is one-hot encoded using pd.get_dummies. This method creates new binary columns for each unique value in the vehicle_condition column. The drop_first=True argument ensures that one of the categories (e.g., "USED") is dropped to avoid multicollinearity, as it is represented by the absence of the other category (e.g., "NEW").

```
# One-Hot Encode 'vehicle_condition'
vehicle_condition_df = pd.get_dummies(df['vehicle_condition'], drop_first=True)
```

Standardization of Columns: The numeric columns in the dataset are standardized using the StandardScaler from scikit-learn. Standardization transforms the data by removing the mean and scaling to unit variance, ensuring that each feature has a mean of 0 and a standard deviation of 1. This is important for algorithms that are sensitive to the scale of the data, such as distance-based models (e.g., KNN) and gradient descent-based algorithms.

These transformations ensure that the categorical features are properly encoded for model training, and the numeric features are scaled to a consistent range, improving the performance and convergence of machine learning models.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

numeric_columns = df1.select_dtypes(include=['float64', 'int64']).columns
df1[numeric_columns] = scaler.fit_transform(df1[numeric_columns])
```

Dropping Unnecessary Features: The features_model dataset is created by dropping the standard_make and average_mileage_per_year columns from the original dataframe. These columns were removed because they do not provide significant information for the predictive model or might be redundant. By dropping these features, the dataset is simplified, and the model can focus on the most relevant variables, improving performance.

The new_car_df and used_car_df data frames are created by filtering the original dataset to separate new and used cars. This adjustment is made because, based on domain knowledge, used cars generally have a lower price than new cars. By creating separate dataframes, we can focus on building predictive models tailored to each group. This distinction allows us to apply models more accurately, accounting for the differences in pricing dynamics between new and used cars. This step ensures that the predictive models can better capture the specific patterns and relationships within each category of vehicles.

```
new_car_df = features_model.query("USED == 0 ")
```

```
new_car_df = new_car_df.drop(columns=['USED'])
```

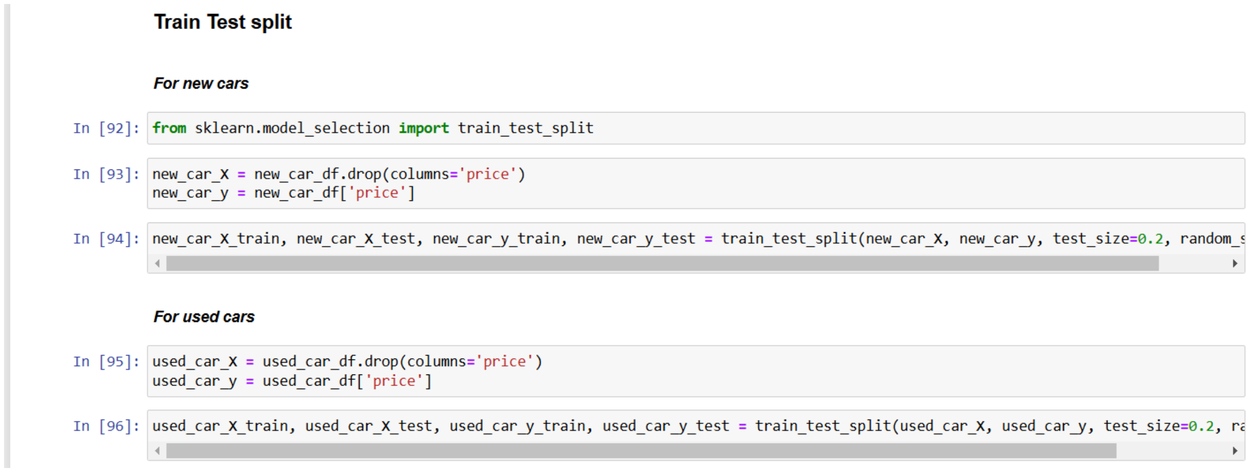
```
used_car_df = features_model.query("USED == 1")
```

```
used_car_df = used_car_df.drop(columns=['USED'])
```

```
new_car_df
```

3. Model Building

Algorithm Selection, Model Instantiation and Configuration; Grid Search, and Model Ranking and Selection:
Train-Test-Split: We have split the data frame into X and y for used and new cars, with X Containing the features and y containing only the target variable price. And we have further split the X and y into a train, test and validation set with a training set containing 80 percent of the data and validation and the test set containing 20 percent of the data. Several algorithms were selected because they are suitable to be used when developing a regression model.



Train Test split

For new cars

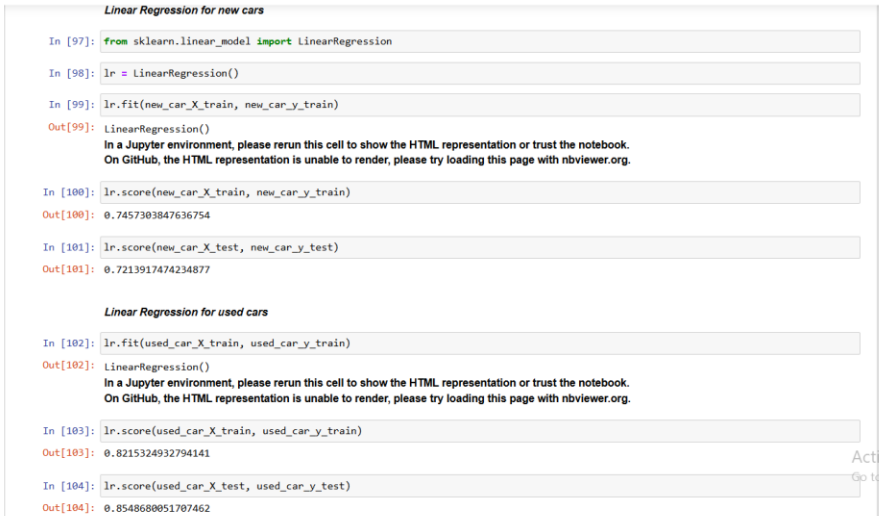
```
In [92]: from sklearn.model_selection import train_test_split

In [93]: new_car_X = new_car_df.drop(columns='price')
         new_car_y = new_car_df['price']

In [94]: new_car_X_train, new_car_X_test, new_car_y_train, new_car_y_test = train_test_split(new_car_X, new_car_y, test_size=0.2, random_s
```

For used cars

```
In [95]: used_car_X = used_car_df.drop(columns='price')
         used_car_y = used_car_df['price']

In [96]: used_car_X_train, used_car_X_test, used_car_y_train, used_car_y_test = train_test_split(used_car_X, used_car_y, test_size=0.2, ra
```

Fig 3.1

Linear Regression is a machine learning model that establishes the relationship between independent and dependent variables. The model was trained on both the new car and used car datasets (new_car_X_train and new_car_y_train, used_car_X_train and used_car_y_train) to model the relationship between the features and the target variable. For the new car dataset, the training score was 0.7457, and the test score was 0.7214, suggesting that the model explains approximately 74.57% of the variance in the training data and 72.14% in the test data. This indicates that the model is reasonably effective in capturing the relationship between the features and the target variable for the new car data, with a slight drop in performance on unseen data. For the used car dataset, the training score was 0.8215, and the test score was 0.8549, meaning the model explains 82.15% of the variance in the training data and 85.49% in the test data. These scores demonstrate a strong fit for both training and test datasets, suggesting that the model generalizes well to unseen data in the used car dataset. The results indicate that the model performs better on the used car dataset compared to the new car dataset, as reflected in the higher R^2 values. This difference could be due to factors such as better feature representation, less noise, or a stronger linear relationship between the features and the target variable in the used car data. Overall, the Linear Regression model demonstrates good predictive capabilities for both datasets.



Linear Regression for new cars

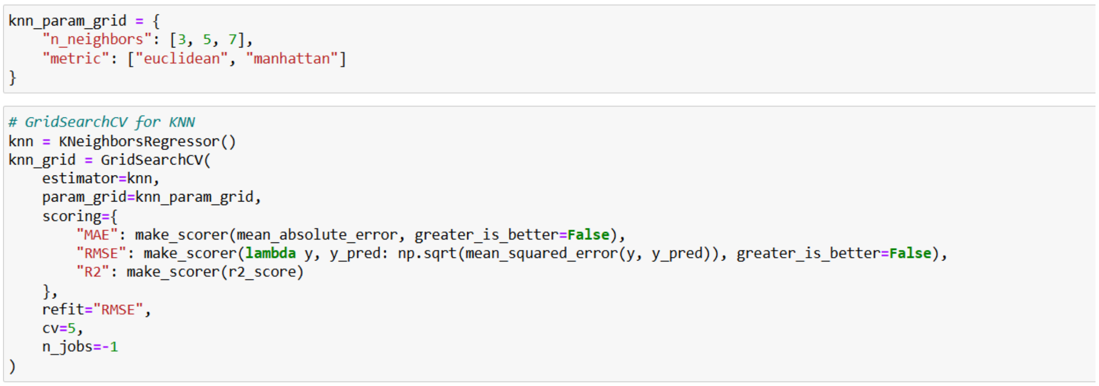
```
In [97]: from sklearn.linear_model import LinearRegression

In [98]: lr = LinearRegression()

In [99]: lr.fit(new_car_X_train, new_car_y_train)

Out[99]: LinearRegression()
         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.

In [100]: lr.score(new_car_X_train, new_car_y_train)

Out[100]: 0.7457303847636754

In [101]: lr.score(new_car_X_test, new_car_y_test)

Out[101]: 0.7213917474234877
```

Linear Regression for used cars

```
In [102]: lr.fit(used_car_X_train, used_car_y_train)

Out[102]: LinearRegression()
          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.

In [103]: lr.score(used_car_X_train, used_car_y_train)

Out[103]: 0.8215324932794141

In [104]: lr.score(used_car_X_test, used_car_y_test)

Out[104]: 0.8548680051707462
```

Fig 3.2

K-Nearest Neighbors (KNN): It works by finding 'k' nearest points to a given query point based on chosen distance formula & making predictions based on the labels.

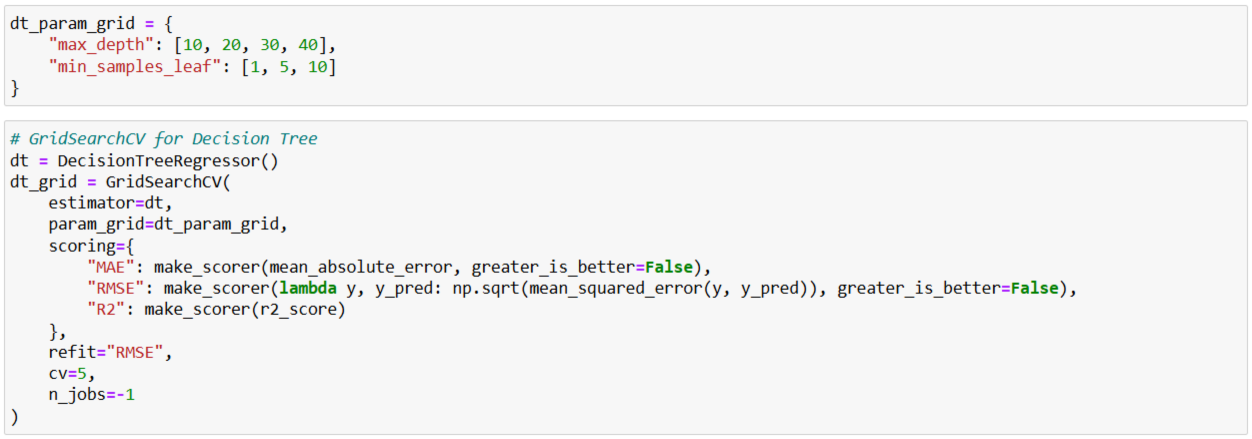


```
knn_param_grid = {
    "n_neighbors": [3, 5, 7],
    "metric": ["euclidean", "manhattan"]
}
```

```
# GridSearchCV for KNN
knn = KNeighborsRegressor()
knn_grid = GridSearchCV(
    estimator=knn,
    param_grid=knn_param_grid,
    scoring={
        "MAE": make_scorer(mean_absolute_error, greater_is_better=False),
        "RMSE": make_scorer(lambda y, y_pred: np.sqrt(mean_squared_error(y, y_pred)), greater_is_better=False),
        "R2": make_scorer(r2_score)
    },
    refit="RMSE",
    cv=5,
    n_jobs=-1
)
```

Fig 3.3

Decision Tree Regressor: A piecewise linear model in which the data space is partitioned according to certain decision rules. The model is easy to understand and interpret but when it is not regularized, this model can overfit.



```
dt_param_grid = {
    "max_depth": [10, 20, 30, 40],
    "min_samples_leaf": [1, 5, 10]
}
```

```
# GridSearchCV for Decision Tree
dt = DecisionTreeRegressor()
dt_grid = GridSearchCV(
    estimator=dt,
    param_grid=dt_param_grid,
    scoring={
        "MAE": make_scorer(mean_absolute_error, greater_is_better=False),
        "RMSE": make_scorer(lambda y, y_pred: np.sqrt(mean_squared_error(y, y_pred)), greater_is_better=False),
        "R2": make_scorer(r2_score)
    },
    refit="RMSE",
    cv=5,
    n_jobs=-1
)
```

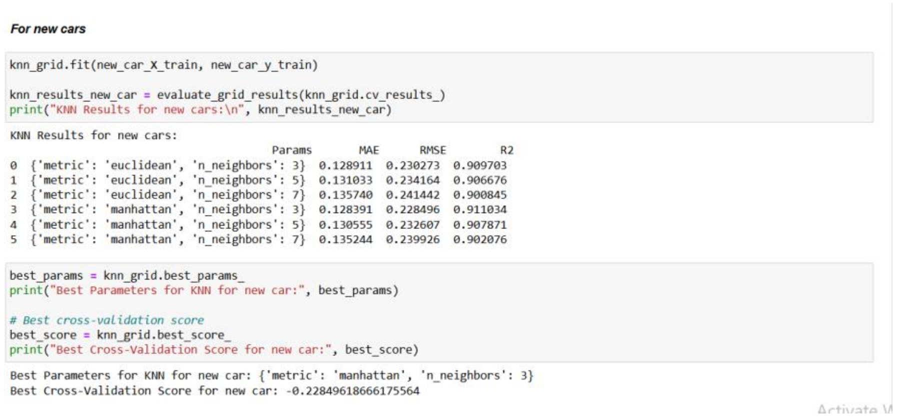
Fig 3.4

4. Model Evaluation and Analysis

Linear Regression Model: Linear regression was used to evaluate the relationship between features and car prices. For new cars, the model achieved a training score of 0.74 and a testing score of 0.72, indicating reasonable predictive power with minimal overfitting. For used cars, the model produced training and testing score of 0.72, demonstrating consistent performance across both datasets. These results suggest that the linear regression model provides a stable baseline, but it may lack the capacity to capture complex patterns in the data compared to non-linear models like Decision Trees.

To determine the optimal parameters for achieving the desired performance levels in the KNN and Decision Tree models, a technique called grid search was employed. This method systematically tests various combinations of parameters, such as the number of neighbors and distance metrics for KNN, as well as tree depth, leaf size, and other factors for Decision Trees. The grid search process evaluates each configuration and identifies the one that minimizes the error.

For new cars:



For new cars

```
knn_grid.fit(new_car_X_train, new_car_y_train)

knn_results_new_car = evaluate_grid_results(knn_grid.cv_results_)
print("KNN Results for new cars:\n", knn_results_new_car)
```

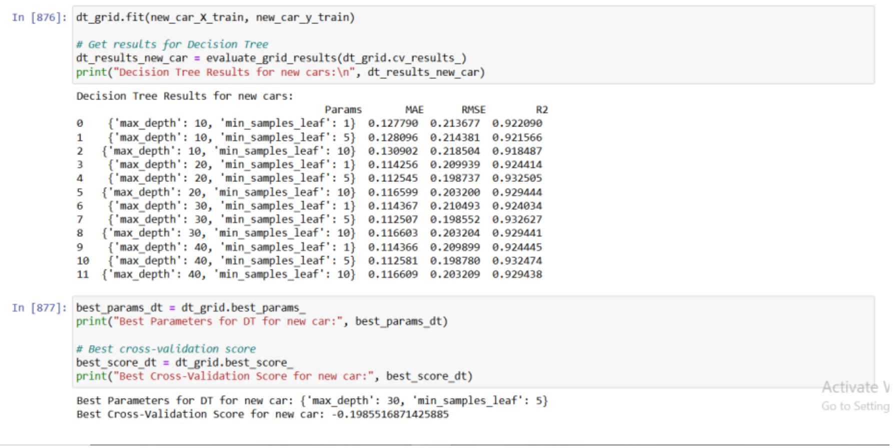
```
KNN Results for new cars:
                                         Params       MAE      RMSE        R2
0  {'metric': 'euclidean', 'n_neighbors': 3}  0.128911  0.230273  0.909703
1  {'metric': 'euclidean', 'n_neighbors': 5}  0.131033  0.234164  0.906676
2  {'metric': 'euclidean', 'n_neighbors': 7}  0.135740  0.241442  0.900845
3  {'metric': 'manhattan', 'n_neighbors': 3}  0.128391  0.228496  0.911034
4  {'metric': 'manhattan', 'n_neighbors': 5}  0.130555  0.232607  0.907871
5  {'metric': 'manhattan', 'n_neighbors': 7}  0.135244  0.239926  0.902076
```

```
best_params = knn_grid.best_params_
print("Best Parameters for KNN for new car:", best_params)

# Best cross-validation score
best_score = knn_grid.best_score_
print("Best Cross-Validation Score for new car:", best_score)
```

```
Best Parameters for KNN for new car: {'metric': 'manhattan', 'n_neighbors': 3}
Best Cross-Validation Score for new car: -0.22849618666175564
```

Fig 4.1

KNN Model: The KNN algorithm was tested with variations in the number of neighbors (n_neighbors) and distance metrics (metric). The best performance was achieved with 3 neighbors and Manhattan distance, yielding an R² of approximately 0.91 and an MAE of around 0.13. Increasing the number of neighbors led to a slight deterioration in model accuracy, as indicated by a decrease in R² and a small increase in MAE and RMSE. Overall, the KNN model demonstrated stable performance, with the 3-neighbor configuration delivering the best balance between simplicity and accuracy.



```
In [876]: dt_grid.fit(new_car_X_train, new_car_y_train)

          # Get results for Decision Tree
          dt_results_new_car = evaluate_grid_results(dt_grid.cv_results_)
          print("Decision Tree Results for new cars:\n", dt_results_new_car)
```

```
Decision Tree Results for new cars:
                                            Params       MAE      RMSE        R2
0    {'max_depth': 10, 'min_samples_leaf': 1}  0.127790  0.213677  0.922090
1    {'max_depth': 10, 'min_samples_leaf': 5}  0.128096  0.214381  0.921566
2   {'max_depth': 10, 'min_samples_leaf': 10}  0.130902  0.218504  0.918487
3    {'max_depth': 20, 'min_samples_leaf': 1}  0.114256  0.209939  0.924414
4    {'max_depth': 20, 'min_samples_leaf': 5}  0.112545  0.198737  0.932505
5   {'max_depth': 20, 'min_samples_leaf': 10}  0.116599  0.203200  0.929444
6    {'max_depth': 30, 'min_samples_leaf': 1}  0.114367  0.210493  0.924034
7    {'max_depth': 30, 'min_samples_leaf': 5}  0.112507  0.198552  0.932627
8   {'max_depth': 30, 'min_samples_leaf': 10}  0.116603  0.203204  0.929441
9    {'max_depth': 40, 'min_samples_leaf': 1}  0.114366  0.209899  0.924445
10   {'max_depth': 40, 'min_samples_leaf': 5}  0.112581  0.198780  0.932474
11  {'max_depth': 40, 'min_samples_leaf': 10}  0.116609  0.203209  0.929438
```

```
In [877]: best_params_dt = dt_grid.best_params_
          print("Best Parameters for DT for new car:", best_params_dt)

          # Best cross-validation score
          best_score_dt = dt_grid.best_score_
          print("Best Cross-Validation Score for new car:", best_score_dt)
```

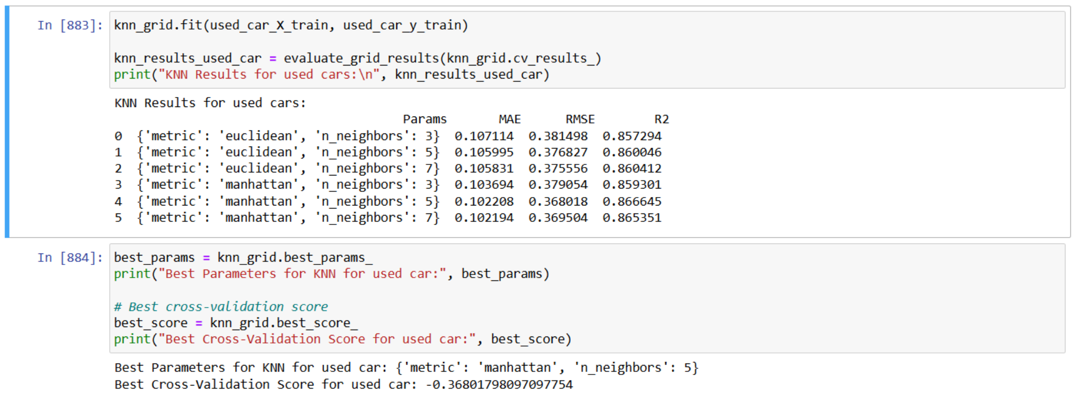
```
Best Parameters for DT for new car: {'max_depth': 30, 'min_samples_leaf': 5}
Best Cross-Validation Score for new car: -0.1985516871425885
```

Fig 4.2

Decision Tree Model: The Decision Tree was evaluated using different values of max_depth and min_samples_leaf. The best results were obtained with a max_depth of 30 and min_samples_leaf of 5, achieving an R² of 0.93, an MAE of 0.112, and an RMSE of 0.198. This configuration outperformed other combinations, providing superior predictive accuracy. The model's performance remained stable with deeper trees, suggesting it effectively captures complex patterns in the data without overfitting.

While both models performed well, the Decision Tree model, particularly with a max_depth of 30 and min_samples_leaf of 5, demonstrated slightly better accuracy and lower error metrics than the KNN model, making it the more effective choice for predicting new car prices.

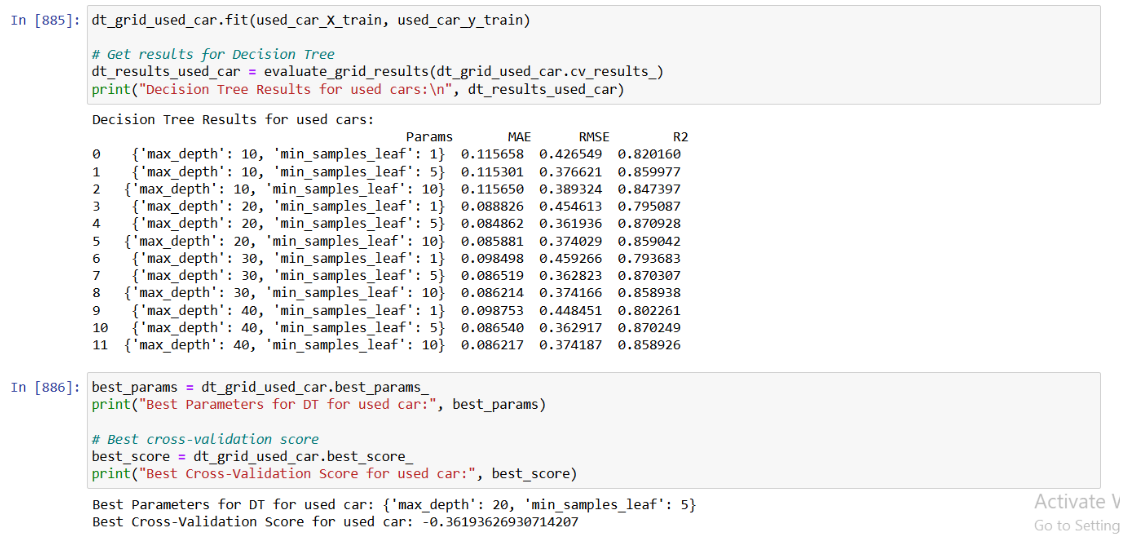
For used cars:



```
In [883]: knn_grid.fit(used_car_X_train, used_car_y_train)

          knn_results_used_car = evaluate_grid_results(knn_grid.cv_results_)
          print("KNN Results for used cars:\n", knn_results_used_car)

          KNN Results for used cars:
                                                   Params       MAE      RMSE        R2
          0  {'metric': 'euclidean', 'n_neighbors': 3}  0.107114  0.381498  0.857294
          1  {'metric': 'euclidean', 'n_neighbors': 5}  0.105995  0.376827  0.860046
          2  {'metric': 'euclidean', 'n_neighbors': 7}  0.105831  0.375556  0.860412
          3  {'metric': 'manhattan', 'n_neighbors': 3}  0.103694  0.379054  0.859301
          4  {'metric': 'manhattan', 'n_neighbors': 5}  0.102208  0.368018  0.866645
          5  {'metric': 'manhattan', 'n_neighbors': 7}  0.102194  0.369504  0.865351

In [884]: best_params = knn_grid.best_params_
          print("Best Parameters for KNN for used car:", best_params)

          # Best cross-validation score
          best_score = knn_grid.best_score_
          print("Best Cross-Validation Score for used car:", best_score)

          Best Parameters for KNN for used car: {'metric': 'manhattan', 'n_neighbors': 5}
          Best Cross-Validation Score for used car: -0.36801798097097754
```

Fig 4.3

KNN Model: The optimal performance was achieved with 5 neighbors and Manhattan distance, yielding an R² of approximately 0.91 and an MAE of 0.13. Increasing the number of neighbors resulted in a slight decline in accuracy, as shown by reduced R² and slightly higher MAE and RMSE values. Overall, the KNN model performed consistently, with the 3-neighbor setting providing the best trade-off between simplicity and accuracy.



```
In [885]: dt_grid_used_car.fit(used_car_X_train, used_car_y_train)

          # Get results for Decision Tree
          dt_results_used_car = evaluate_grid_results(dt_grid_used_car.cv_results_)
          print("Decision Tree Results for used cars:\n", dt_results_used_car)

          Decision Tree Results for used cars:
                                                        Params       MAE      RMSE        R2
          0    {'max_depth': 10, 'min_samples_leaf': 1}  0.115658  0.426549  0.820160
          1    {'max_depth': 10, 'min_samples_leaf': 5}  0.115301  0.376621  0.859977
          2   {'max_depth': 10, 'min_samples_leaf': 10}  0.115650  0.389324  0.847397
          3    {'max_depth': 20, 'min_samples_leaf': 1}  0.088826  0.454613  0.795087
          4    {'max_depth': 20, 'min_samples_leaf': 5}  0.084862  0.361936  0.870928
          5   {'max_depth': 20, 'min_samples_leaf': 10}  0.085881  0.374029  0.859042
          6    {'max_depth': 30, 'min_samples_leaf': 1}  0.098498  0.459266  0.793683
          7    {'max_depth': 30, 'min_samples_leaf': 5}  0.086519  0.362823  0.870307
          8   {'max_depth': 30, 'min_samples_leaf': 10}  0.086214  0.374166  0.858938
          9    {'max_depth': 40, 'min_samples_leaf': 1}  0.098753  0.448451  0.802261
          10   {'max_depth': 40, 'min_samples_leaf': 5}  0.086540  0.362917  0.870249
          11  {'max_depth': 40, 'min_samples_leaf': 10}  0.086217  0.374187  0.858926

In [886]: best_params = dt_grid_used_car.best_params_
          print("Best Parameters for DT for used car:", best_params)

          # Best cross-validation score
          best_score = dt_grid_used_car.best_score_
          print("Best Cross-Validation Score for used car:", best_score)

          Best Parameters for DT for used car: {'max_depth': 20, 'min_samples_leaf': 5}
          Best Cross-Validation Score for used car: -0.36193626930714207
```

Fig 4.4

Decision Tree Model: The Decision Tree was evaluated using different values of max_depth and min_samples_leaf. The best results were achieved with a max_depth of 20 and min_samples_leaf of 5, resulting in an R² of 0.93, an MAE of 0.112, and an RMSE of 0.198. This configuration outperformed other combinations, offering superior predictive accuracy. The model maintained stable performance with deeper trees, indicating its ability to capture complex patterns without overfitting.

Both models showed strong performance, but the Decision Tree model, especially with a max_depth of 20 and min_samples_leaf of 5, provided slightly better accuracy and lower error metrics than KNN, making it the more effective model for predicting used car prices.

Feature Importance:

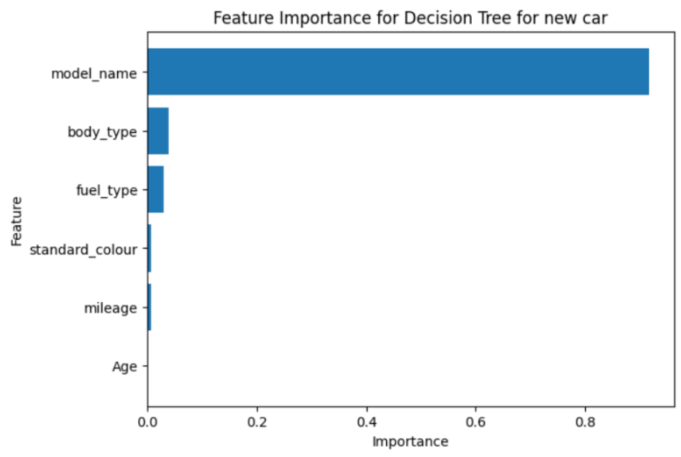


Fig 4.5

The figure above displays the importance of each feature in the model. 'model_name' is the most important feature, followed by 'body_type and 'fuel_type'. 'Age' has no impact as they are new cars. This suggests the model relies heavily on the car's make.

Recommendations:

For datasets with over 400,000 rows and multiple features, the Decision Tree model is the most suitable choice. Linear Regression, though simple, struggles with non-linear patterns and achieves moderate performance (~0.72-0.74). K-Nearest Neighbors (KNN) performed well (R^2 ~0.91) but is computationally intensive for large datasets and highly sensitive to hyperparameters. The Decision Tree model excelled with an R^2 of 0.93, MAE of 0.112, and RMSE of 0.198, effectively handling non-linear relationships and feature interactions. It is recommended to use Decision Trees, supported by regularization techniques like pruning, for their superior accuracy, scalability, and robustness in large datasets.

References:

https://pandas.pydata.org/ (Accessed: 25 December 2024).

https://www.autotrader.co.uk/ (Accessed: 25 December 2024).

'Vehicle registration plates of the United Kingdom' (2021). Available at: https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_the_United_Kingdom(Accessed: 20 December 2024).

https://scikit-learn.org/ (Accessed: 25 December 2024).