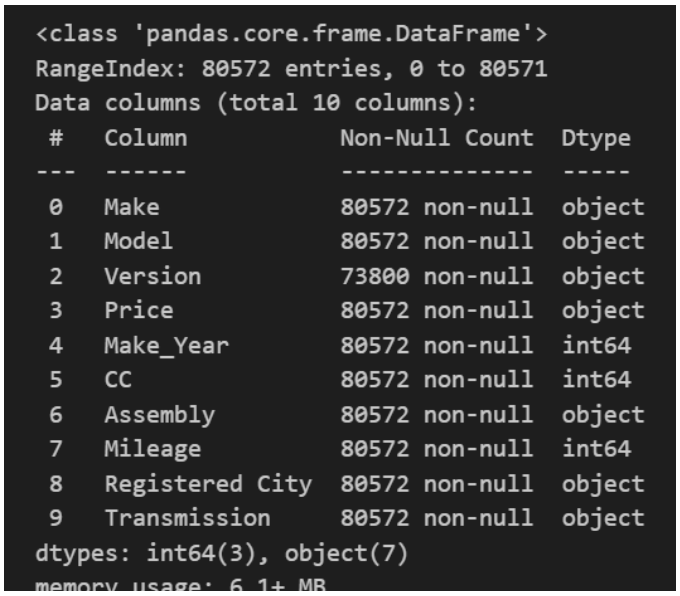
Cowlar Machine Learning Task

Task: 1

We are given a dataset about second hard cards. We have various features related to the car and the price it is being listed for sale. We want to build a machine learning model that inputs the features of an old car and predicts the amount it should be sold for. The features of a car include:

- Make: The company of the car (Toyota)
- Model: Name of a car (Corolla)
- Version: The variant of a car. Such as a corolla may be a GLI, XLI etc
- Price: The price a car is listed for
- Make_year: The year the car was made in
- CC: Capacity of a car's engine
- Assembly: If a car is locally assembled or imported
- Mileage: The number of miles a car has been driven
- Registered City; The city a car is registered in
- Transmission: Manual or Automatic



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80572 entries, 0 to 80571
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Make             80572 non-null  object
 1   Model            80572 non-null  object
 2   Version          73800 non-null  object
 3   Price            80572 non-null  object
 4   Make_Year        80572 non-null  int64
 5   CC               80572 non-null  int64
 6   Assembly         80572 non-null  object
 7   Mileage          80572 non-null  int64
 8   Registered City  80572 non-null  object
 9   Transmission     80572 non-null  object
dtypes: int64(3), object(7)
memory usage: 6.1+ MB
```

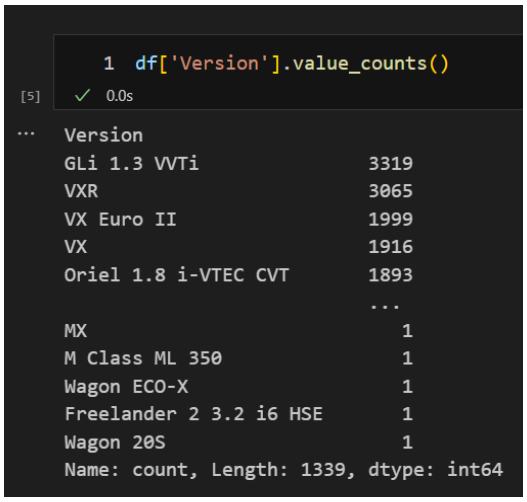
Data Cleaning

We can see from our initial look at the data that there are a total of ***80.572*** rows of data or we have that many cars that have been listed for sale. We can observe that the only column that has missing values is the ***Version*** column. Let us carry out further analysis to find ways to fill in the missing values.

Looking at the above Versions we can tell a car's version is specific to its model and make. Since versions is a categorical variable we have two options to fill the missing values

- Fill the missing values with the mode of the column
- Fill the missing values with a new category

But since there are a lot of variations in the data we can not just fill the missing values with the mode of the column. Moreover, the version of a car is specific to its model and make. So let us fill the missing values with a new category called "missing", Then we can carry out data analysis and see if we can fill the missing values with a more specific category.



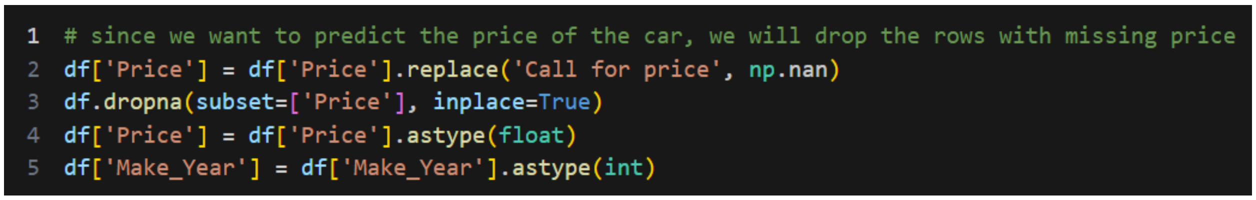
```
df['Version'].value_counts()
```

```
[5]  ✓ 0.0s
```

```
Version
GLi 1.3 VVTi              3319
VXR                       3065
VX Euro II                1999
VX                        1916
Oriel 1.8 i-VTEC CVT      1893
                          ...
MX                           1
M Class ML 350               1
Wagon ECO-X                  1
Freelander 2 3.2 i6 HSE      1
Wagon 20S                    1
Name: count, Length: 1339, dtype: int64
```

Correcting Data Types

There are some data points where price is set as “Call for price”. These are not important for us as we are trying to predict a car’s price so we will drop these columns



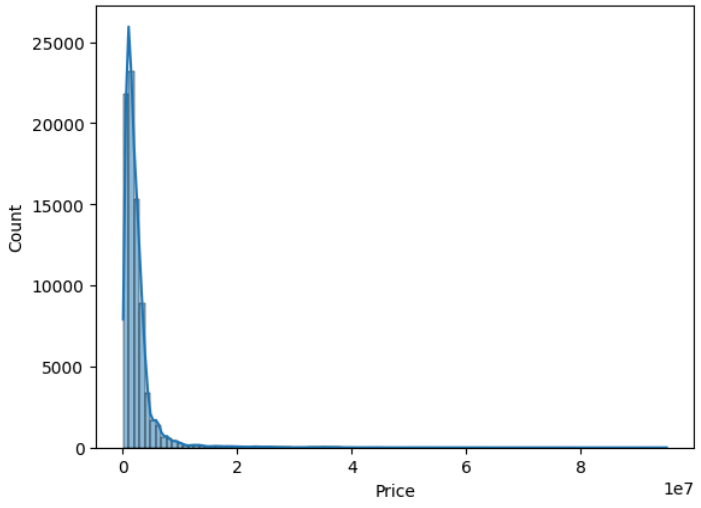
```python
# since we want to predict the price of the car, we will drop the rows with missing price
df['Price'] = df['Price'].replace('Call for price', np.nan)
df.dropna(subset=['Price'], inplace=True)
df['Price'] = df['Price'].astype(float)
df['Make_Year'] = df['Make_Year'].astype(int)
```

Data Analysis

After carrying out a detailed analysis of the dataset. Below are some of the insights gained from the data. This does contain all the data analysis which is included in the Python notebook for the sake of brevity.

1. Skewness of Price

We observed that the distribution of car prices is heavily skewed. Most of the cars have low prices but there exist all out cars whose price is well outside the middle values. This is because most normal cars exist in a normal price range. There are few luxury cars that are priced way higher than normal cars.

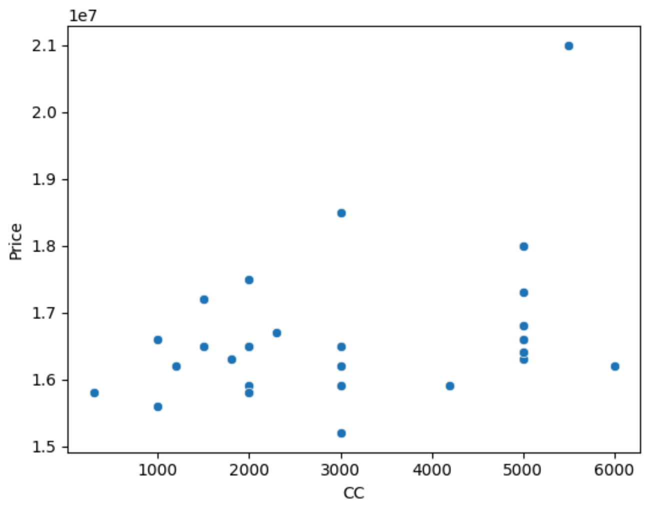


2. CC of Cars

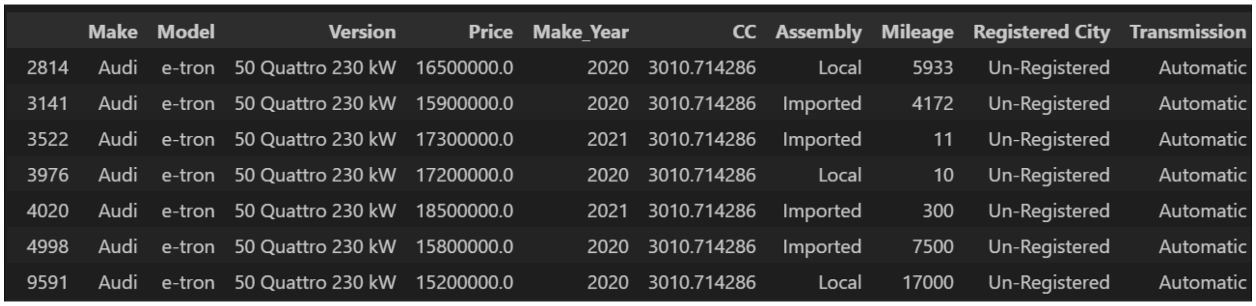
We observe that the Audi e-tron has a lot of variations in the CC values. Some of the values are

- 300
- 1000
- 1200
- 500

We would expect if the version of a car is the same then it should have close values of CC. Outliers here might suggest that the data is incorrect. Let us plot the CC of e-tron as compared to its price



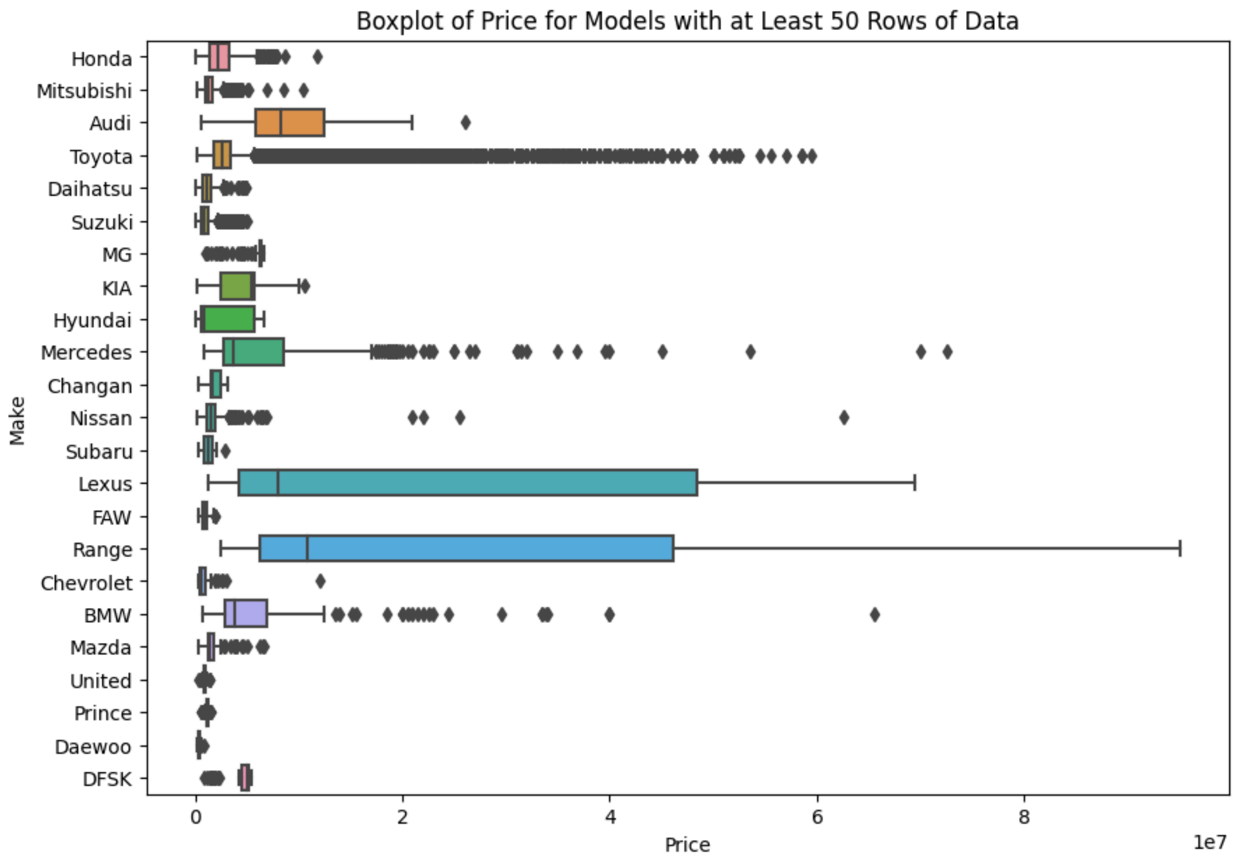
We can observe that even though there is a lot of variation in CC values, the price of a car has remained relatively constant. Considering a car with the same variant should have the same value of CC we set the value of CC as the average of the group of cars that have the same variant.



	Make	Model	Version	Price	Make_Year	CC	Assembly	Mileage	Registered City	Transmission
2814	Audi	e-tron	50 Quattro 230 kW	16500000.0	2020	3010.714286	Local	5933	Un-Registered	Automatic
3141	Audi	e-tron	50 Quattro 230 kW	15900000.0	2020	3010.714286	Imported	4172	Un-Registered	Automatic
3522	Audi	e-tron	50 Quattro 230 kW	17300000.0	2021	3010.714286	Imported	11	Un-Registered	Automatic
3976	Audi	e-tron	50 Quattro 230 kW	17200000.0	2020	3010.714286	Local	10	Un-Registered	Automatic
4020	Audi	e-tron	50 Quattro 230 kW	18500000.0	2021	3010.714286	Imported	300	Un-Registered	Automatic
4998	Audi	e-tron	50 Quattro 230 kW	15800000.0	2020	3010.714286	Imported	7500	Un-Registered	Automatic
9591	Audi	e-tron	50 Quattro 230 kW	15200000.0	2020	3010.714286	Local	17000	Un-Registered	Automatic

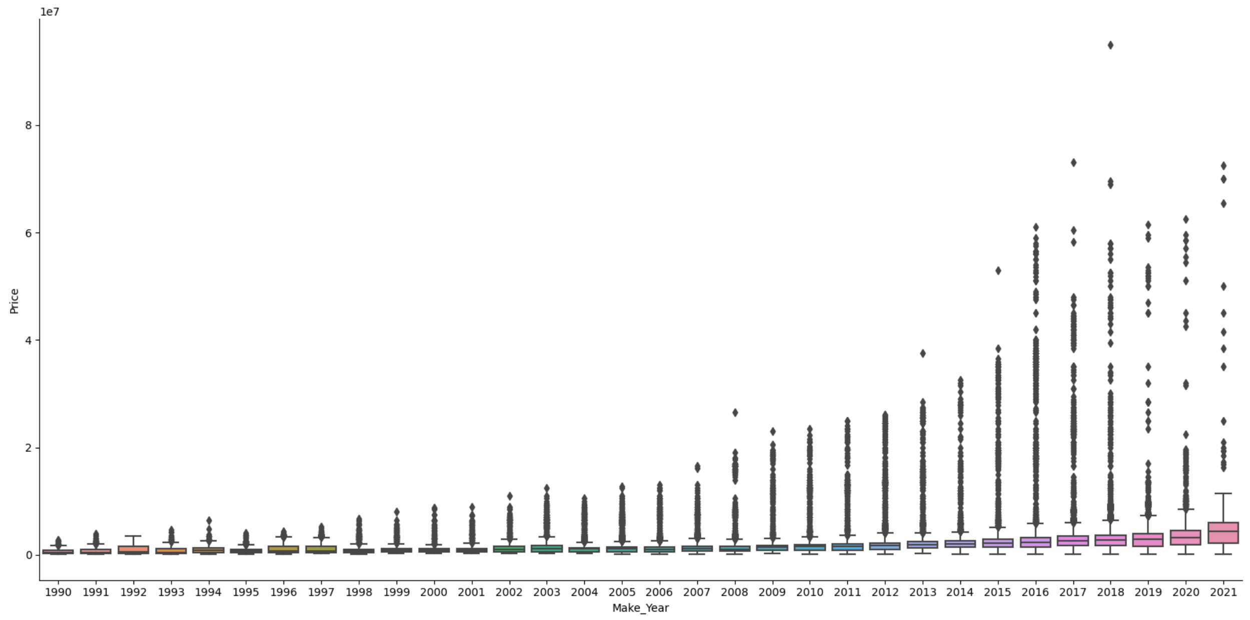
3. Price vs Car brand

We observe that Lexus and Range Rover are generally more expensive while Toyota has a lot of outliers. This is because Toyota has a lot of variety when it comes to cars. Some of them are quite cheap while others fall in the pricey region just as we saw above. We can conclude that when it comes to predicting a car's price, the Make and model adds a lot of information. Now let us take a look at the remaining columns and see if we can find any insights from them



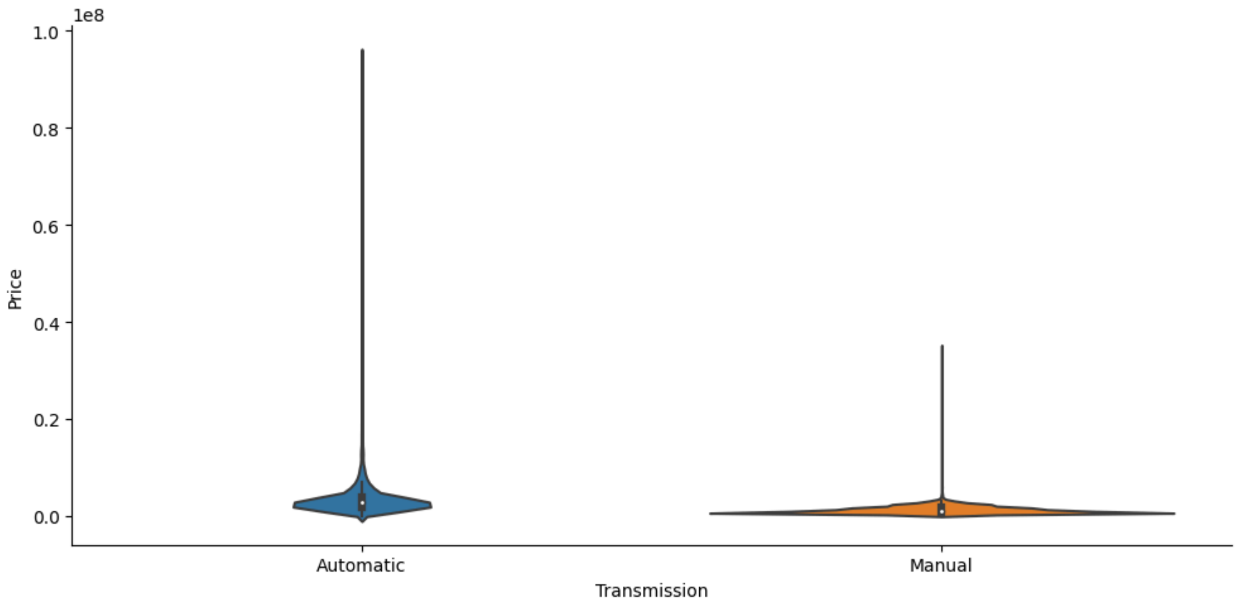
4. Price vs Make_year

Here we can see that even though the average price of cars has remained similar over the years. But the upper limit of the price has increased. This suggests that the price of luxury cars has increased over the years.

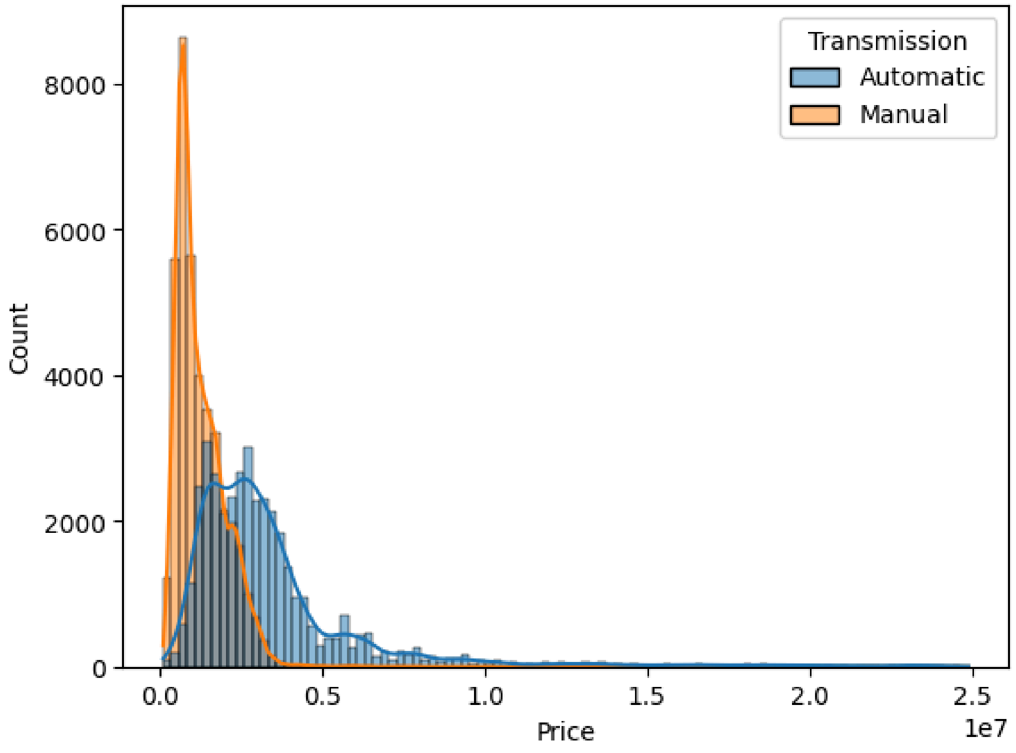


5. Price vs Transmission

So the price of cars with automatic transmission is generally higher than that of manual transmission. Moreover, most manual cars have similar prices while automatic cars have a lot of variation in price.

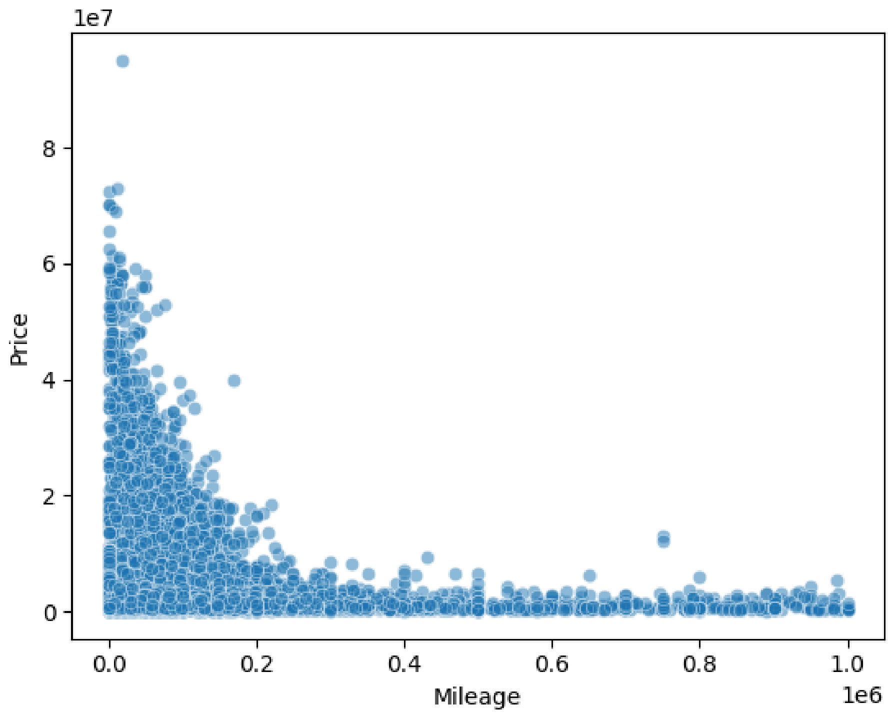


Moreover, the median value of manual cars is lesser as compared to automatic ones.



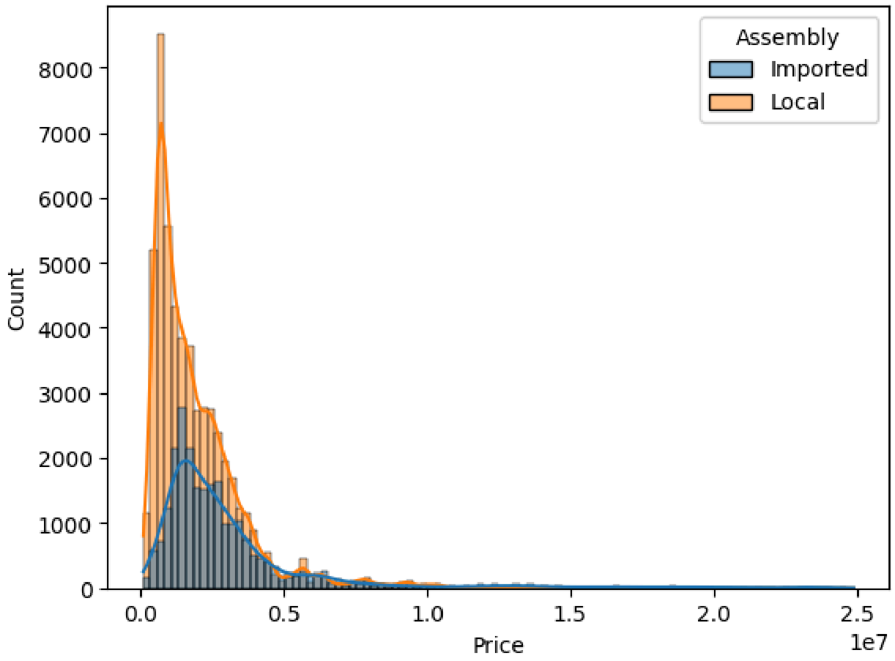
6. Price vs Mileage

The more a car is driven the lower its price becomes.



7. Price vs Assembly

We can see that most cars are locally assembled but the median price of imported cars is higher than that of locally assembled cars.



Training the Model

We have defined a utility function ***train_and_evaluate_model*** that automates the process of taking a DataFrame, preprocessing its columns into numerical values so it can be accepted by an ML algorithm. Then it trains the model and evaluates it. The methodology behind the function is as follows:

- Separating the data into features and target variables
- Separating the numerical and categorical data so we can do their pre processing separately.

```
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numerical_cols),
            ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_cols)
        ])
```

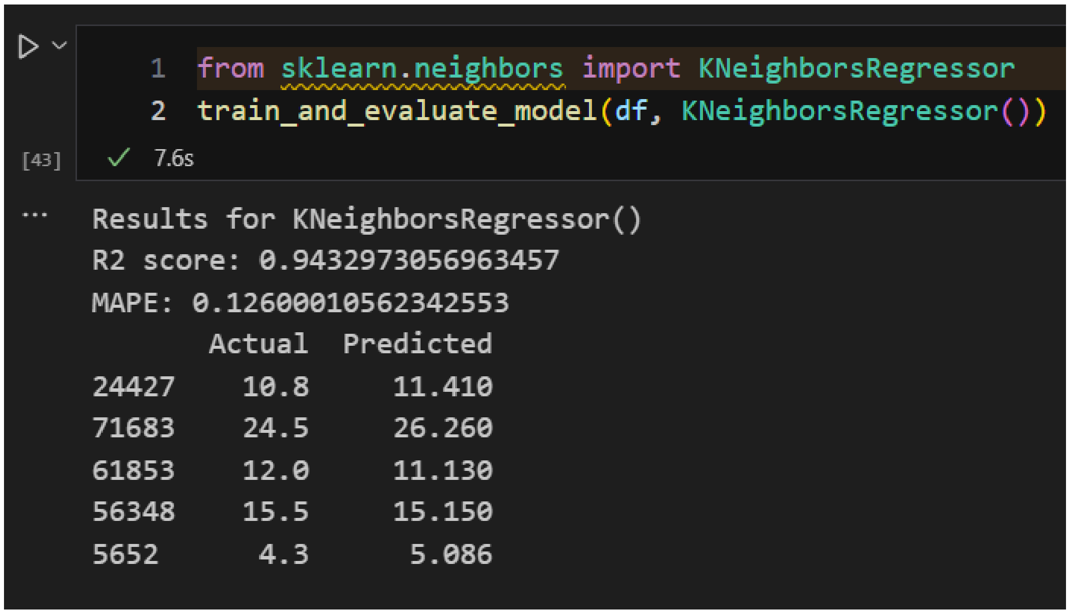
- Splitting our data into train and test so we can evaluate our model
- Fitting the model that is given as an argument to the function.

```
# Create the pipeline with the specified model
model_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', model)
])

# Fit the model
model_pipeline.fit(X_train, y_train)
```

- Evaluating our model using R2 score and Mean absolute percentage error. We have chosen R2 score as a metric because it works well with regression data. Moreover we have preferred MAPE over mean squared error as the value of prices is quite large and MSE will give really large errors. On the other hand, MAPE is easier to interpret as it is scaled to the magnitude of the variable. For example a MAPE of 20% suggests our prediction is 20% more or less than the actual value.

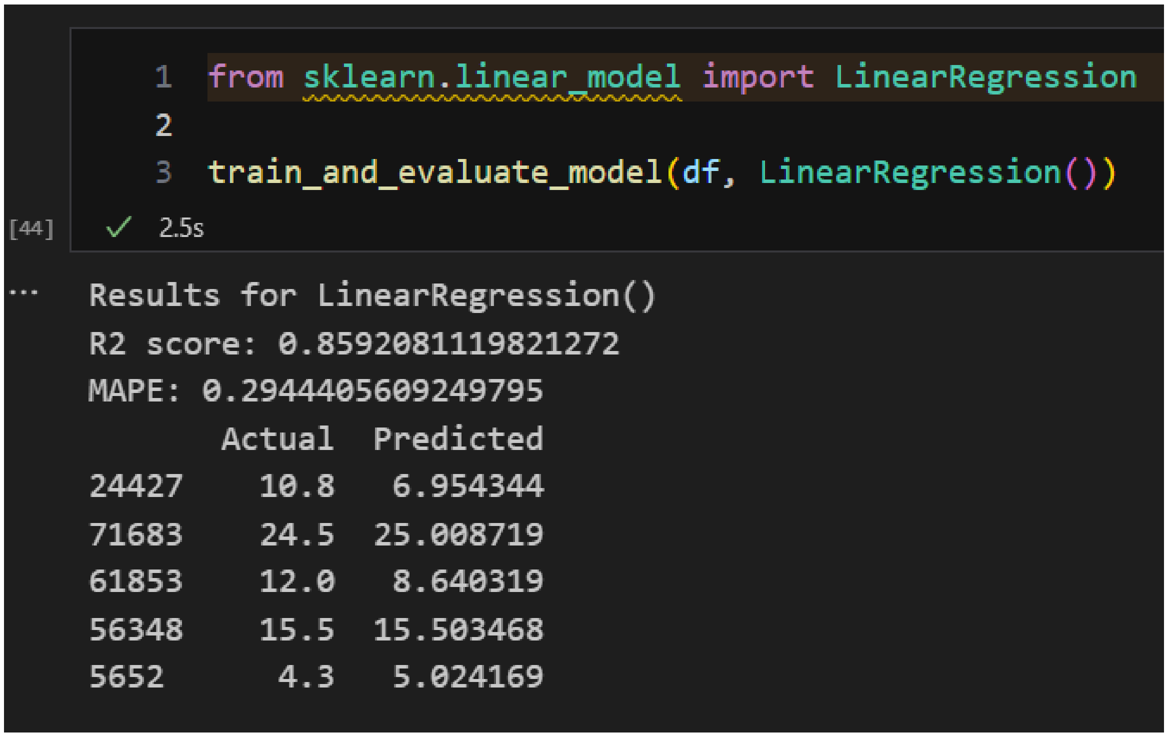
Performance of Models



```
from sklearn.neighbors import KNeighborsRegressor
train_and_evaluate_model(df, KNeighborsRegressor())
```

[43] ✓ 7.6s

```
Results for KNeighborsRegressor()
R2 score: 0.9432973056963457
MAPE: 0.12600010562342553
       Actual  Predicted
24427    10.8     11.410
71683    24.5     26.260
61853    12.0     11.130
56348    15.5     15.150
5652      4.3      5.086
```

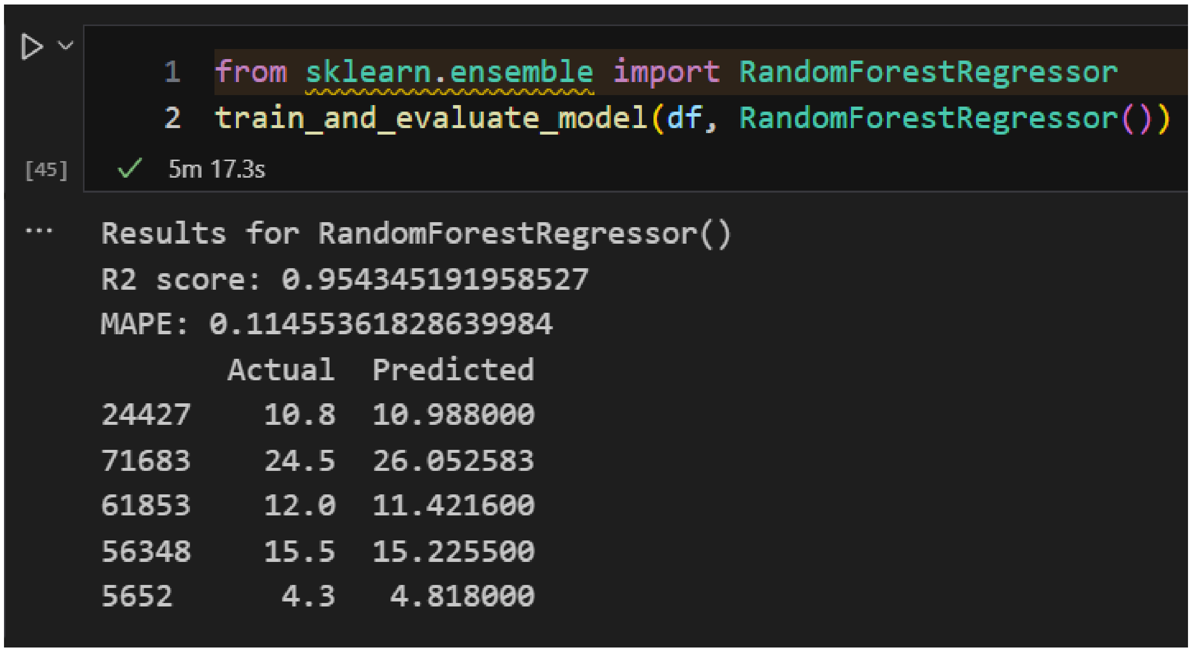


```
from sklearn.linear_model import LinearRegression

train_and_evaluate_model(df, LinearRegression())
```

[44] ✓ 2.5s

```
Results for LinearRegression()
R2 score: 0.8592081119821272
MAPE: 0.2944405609249795
       Actual  Predicted
24427    10.8   6.954344
71683    24.5  25.008719
61853    12.0   8.640319
56348    15.5  15.503468
5652      4.3   5.024169
```



```
from sklearn.ensemble import RandomForestRegressor
train_and_evaluate_model(df, RandomForestRegressor())
```

[45] ✓ 5m 17.3s

```
Results for RandomForestRegressor()
R2 score: 0.954345191958527
MAPE: 0.11455361828639984
       Actual  Predicted
24427    10.8  10.988000
71683    24.5  26.052583
61853    12.0  11.421600
56348    15.5  15.225500
5652      4.3   4.818000
```

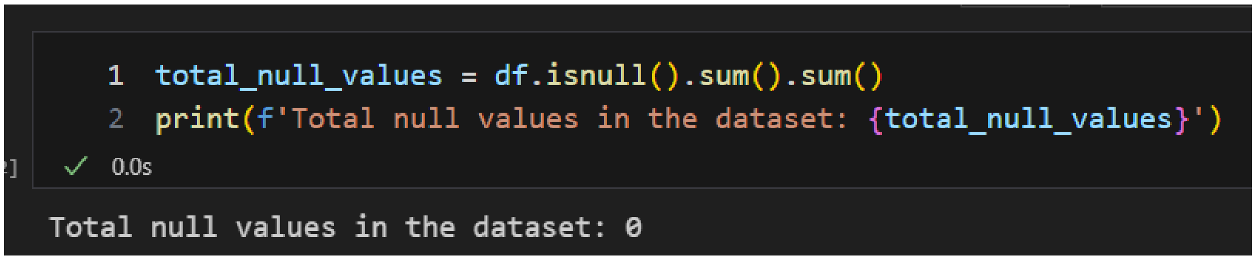
We observe that linear regression performs quite badly as compared to KNN and random forest. Maybe we could adjust the hyper-parameters to boost its accuracy. But it is unnecessary as we are getting extremely accurate results from the other two models. Random forest outperforms KNN slightly. The R2 value of 0.954 suggests that our model's accuracy is close to perfect.

Task: 2

The second dataset is about movies. We have a movie and the rating the movie was given by its viewer out of 5. We need to use a movie watched by a person as input to predict what movie we should watch next. We can do this by finding the movie that has the rating most similar to the movie the user just viewed.

Data Cleaning

1. Checking for Null values

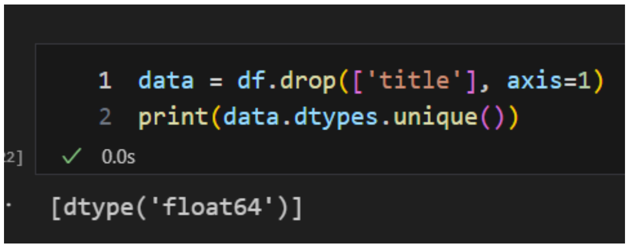


```
total_null_values = df.isnull().sum().sum()
print(f'Total null values in the dataset: {total_null_values}')
```

✓ 0.0s

```
Total null values in the dataset: 0
```

2. Checking data types



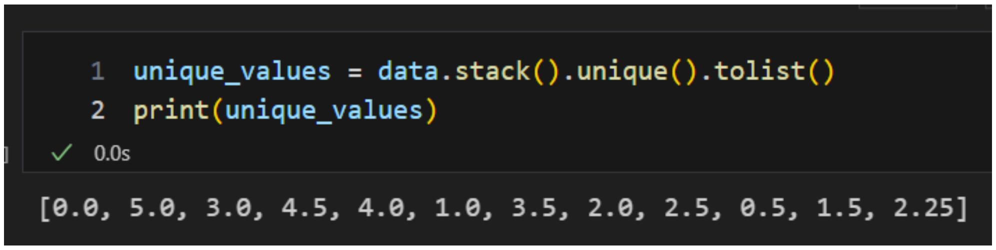
```
data = df.drop(['title'], axis=1)
print(data.dtypes.unique())
```

✓ 0.0s

```
[dtype('float64')]
```

We can see that the data types are correct

3. Checking for incorrect values

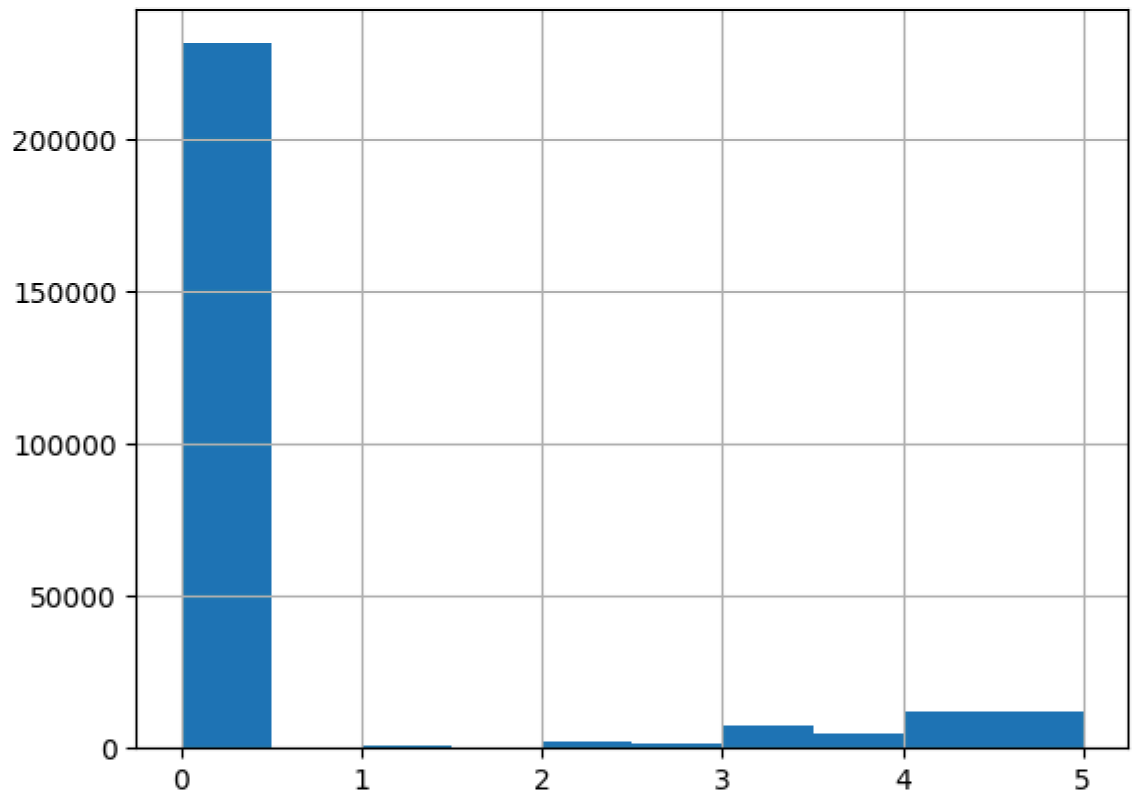


```
unique_values = data.stack().unique().tolist()
print(unique_values)
```

✓ 0.0s

```
[0.0, 5.0, 3.0, 4.5, 4.0, 1.0, 3.5, 2.0, 2.5, 0.5, 1.5, 2.25]
```

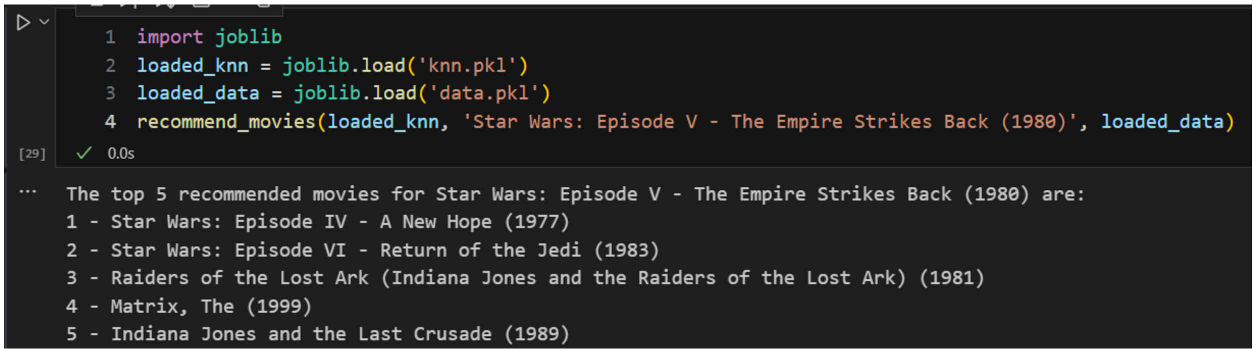
So we can see that the ratings are spread from 0 to 5. Thus there are no errors in the data. We can draw the histogram for ratings to check the count for each rating. We observe that most values are 0 showing that most user's have not watched a particular movie.



Training a Model

We can use the K nearest neighbor algorithm to find the movies that are most similar to movies the user has just watched. We can use cosine similarity as a metric. This will work good for our data because:

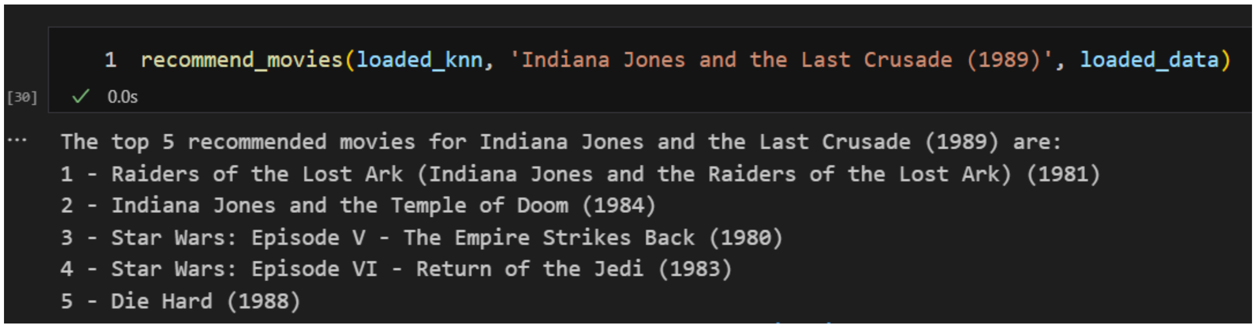
- KNN with cosine similarity is well-suited for sparse data because it focuses on the angular separation between vectors rather than their magnitudes.
- KNN models store the entire training dataset during inference. In the case of sparse matrices, where many entries are zero, memory usage can be more efficient since only non-zero entries need to be stored.
- KNN is a local method that relies on the similarity of data points within the neighborhood of a given point. In sparse matrices, local relationships might be more meaningful than global relationships, and KNN can capture these local structures effectively.



```python
import joblib
loaded_knn = joblib.load('knn.pkl')
loaded_data = joblib.load('data.pkl')
recommend_movies(loaded_knn, 'Star Wars: Episode V - The Empire Strikes Back (1980)', loaded_data)
```

```
The top 5 recommended movies for Star Wars: Episode V - The Empire Strikes Back (1980) are:
1 - Star Wars: Episode IV - A New Hope (1977)
2 - Star Wars: Episode VI - Return of the Jedi (1983)
3 - Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
4 - Matrix, The (1999)
5 - Indiana Jones and the Last Crusade (1989)
```

So we can see that people that like Star wars are recommended other movies from the franchise. This is a good sign that our model is working well. Moreover we can see that the movies that are recommended are also of the same genre. This is a good sign that our model is working well.



```python
recommend_movies(loaded_knn, 'Indiana Jones and the Last Crusade (1989)', loaded_data)
```

```
The top 5 recommended movies for Indiana Jones and the Last Crusade (1989) are:
1 - Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
2 - Indiana Jones and the Temple of Doom (1984)
3 - Star Wars: Episode V - The Empire Strikes Back (1980)
4 - Star Wars: Episode VI - Return of the Jedi (1983)
5 - Die Hard (1988)
```

Here we can see that the Indiana Jones movies are being recommended more as compared to Star wars movies. This is a good sign that our model is working well.