Python: Data Cleaning and Transformation

The goal of this project is to clean and transform a **car dataset** with over 2 million rows containing information on various attributes such as the car brand, model, year of manufacture, engine and transmission info, and fuel type. The dataset had several challenges and issues, to tackle these issues, various **Python** data cleaning techniques were employed such as **filling in missing values, standardizing categories, correcting string inconsistencies, limiting the data range, removing duplicates, creating columns,** and **sorting the data**. The end result is a clean, transformed dataset that is ready for further analysis.

To effectively clean the dataset, a column-by-column approach was taken. This was necessary to focus on specific issues in each column and to address them in a targeted manner, rather than trying to clean the entire dataset at once.
This method makes it easier to detect issues and allows for better control over the cleaning process and improved accuracy.

The dataset is a sample of a large dataset in Kaggle with the following [link]
The Dataset consists of the following columns:
"**maker**" - The manufacturer of the car.
"**model**" - The specific model of the car.
"**manufacture_year**" - The year in which the car was manufactured.
"**engine_displacement**" - The engine size in cc
"**engine_power**" - The maximum power output of the engine, measured in kilowatts.
"**transmission**" - The type of gearbox used in the car, such as manual or automatic.
"**fuel_type**" - The type of fuel used by the car, such as gasoline or diesel.
"**door_count**" - The number of doors on the car.
"**seat_count**" - The number of seats in the car.

Loading Essential Python Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from fuzzywuzzy import fuzz
```

Importing the data

```
df = pd.read_csv(
    r'/Users/wsm/Downloads/dirty_car_data')

df_backup = pd.read_csv(
    r'/Users/wsm/Downloads/dirty_car_data')
```

In order to ensure that the original data was not altered permanently during the cleaning process, a backup of the data was created and stored in a separate Dataframe named 'df_backup'. This backup was used in the rare case where a mistake was made while cleaning a column in the primary Dataframe, 'df'. In such instances, the original data was retrieved by reassigning the relevant column from 'df_backup' to 'df'.

Exploring the data

```
df.shape
(2096218, 9)
```

```
df.info(show_counts=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2096218 entries, 0 to 2096217
Data columns (total 9 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   maker                1790224 non-null  object
 1   model                1427254 non-null  object
 2   manufacture_year     1877484 non-null  float64
 3   engine_displacement  2096218 non-null  object
 4   engine_power         1768185 non-null  float64
 5   transmission         1658458 non-null  object
 6   fuel_type            1114209 non-null  object
 7   door_count           1733718 non-null  object
 8   seat_count           1653954 non-null  object
dtypes: float64(2), object(7)
memory usage: 143.9+ MB
```

```
df.head()
           maker        model  manufacture_year engine_displacement  \
0     volkswagen  transporter            2005.0              2500cm
1  mercedes-benz          NaN               NaN              2143cm
2            NaN          NaN            2003.0              1390cm
3            bmw          NaN            2002.0              1995cm
4         nissan       patrol               NaN              3000cm

engine_power transmission fuel_type door_count seat_count
0         96.0       manual    diesel        4.0        5.0
1        130.0    automatic       NaN        4.0        5.0
2         55.0       manual  gasoline          2          5
3        105.0       manual  gasoline          2          5
4        116.0          NaN  gasoline       None       None
```

Let's start cleaning! First column 'maker'

We will start by replacing missing values in this column with 'unknown', then we will deal with string inconsistency by converting all values to lowercase and removing leading/trailing whitespaces.

```
# Handling missing values
df['maker'].fillna('unknown', inplace=True)
# lowercase
df['maker'] = df['maker'].str.lower()
# Remove leading/trailing whitespaces
df['maker'] = df['maker'].str.strip()
```

String inconsistencies in the data can occur due to manual entry errors or variations in the data source. To thoroughly assess the column, we will examine its unique values.

```
df['maker'].unique()
array(['volkswagen', 'mercedes-benz', 'unknown', 'bmw', 'nissan', 'audi',
       'opel', 'renault', 'skoda', 'ford', 'peugeot', 'seat', 'smart',
       'citroen', 'fiat', 'toyota', 'mazda', 'suzuki', 'volvo', 'kia',
       'honda', 'chevrolet', 'oudi', 'hyundai', 'porsche', 'subaru',
       'rover', 'audii', 'mini', 'jaguar', 'lancia', 'aude','mitsubishi',
       'volzwagen', 'jeep', 'isuzu', 'alfa-romeo', 'dacia', 'lexus',
       'chrysler', 'dodge', 'porshe', 'maserati', 'tesla', 'infinity',
       'hummer', 'volksvagen', 'peugeo', 'bentley', 'lotus', 'porche',
       'land-rover', 'lamborghini', 'peugot', 'prsche', 'aston-martin',
       'rolls-royce'], dtype=object)
```

Some maker names are misspelled, such as 'peugeo' instead of 'peugeot', and 'volksvagen' instead of 'volkswagen', and many more. Manually correcting these errors can be time-consuming and error-prone. To streamline this process, we will use the fuzz.token_sort_ratio method in Python.
The method compares each entry in the 'maker' column to a list of the correct maker names and based on a predefined similarity threshold, it will identify and correct any misspelled names.

```
# First define the list of correct maker names
categories = ['volkswagen', 'mercedes-benz', 'unknown', 'bmw', 'nissan',
              'audi', 'opel', 'renault', 'skoda', 'ford', 'peugeot',
              'seat','smart', 'citroen', 'fiat', 'toyota', 'mazda',
              'suzuki','volvo', 'kia', 'honda', 'chevrolet', 'hyundai',
              'porsche', 'subaru', 'rover', 'mini', 'jaguar', 'lancia',
              'mitsubishi', 'jeep', 'isuzu', 'alfa-romeo', 'dacia',
              'lexus', 'chrysler', 'dodge', 'maserati', 'tesla',
              'infinity', 'hummer', 'bentley', 'lotus', 'land-rover',
              'lamborghini', 'aston-martin', 'rolls-royce']

# Calculate the similarity score between each value in the 'maker' column and
each name in 'categories'
scores = df['maker'].apply(
    lambda x: [fuzz.token_sort_ratio(x, name) for name in categories])

# Find the index of the highest similarity score for each value in 'maker'
highest_scores = scores.apply(lambda x: categories[np.argmax(x)])

# Replace values in 'maker' where the similarity score is at least 80
df['maker'] = np.where(scores.apply(
    lambda x: np.max(x) >= 70), highest_scores, df['maker'])

# Check unique values and if the number of unique values is the same as in
categories
df['maker'].unique()
len(df['maker'].unique()) == len(categories)
```

Explanation of the method used:
First, we define a list of the correct maker names in a variable called categories.
Next, we use the `fuzz.token_sort_ratio` method to calculate the similarity score between each value in the 'maker' column and each name in `'categories'`. We store the scores in a variable called scores.

Then, we find the index of the highest similarity score for each value in 'maker' by using the np.argmax function. We store the results in a variable called highest_scores.
We then use np.where to replace values in 'maker' where the similarity score is at least 70. The replaced values are those in highest_scores.
Finally, we check the unique values in 'maker' and compare them to the length of categories to see if all of the correct maker names are present.

Column ‘model’

Same as the first column we will start by replacing missing values in this column with ‘unknown’, then we will deal with string inconsistency by converting all values to lowercase and removing leading/trailing whitespaces.

```
# Handling missing values
df['model'].fillna('unknown', inplace=True)
# lowercase
df['model'] = df['model'].str.lower()
# Remove leading/trailing whitespaces
df['model'] = df['model'].str.strip()
```

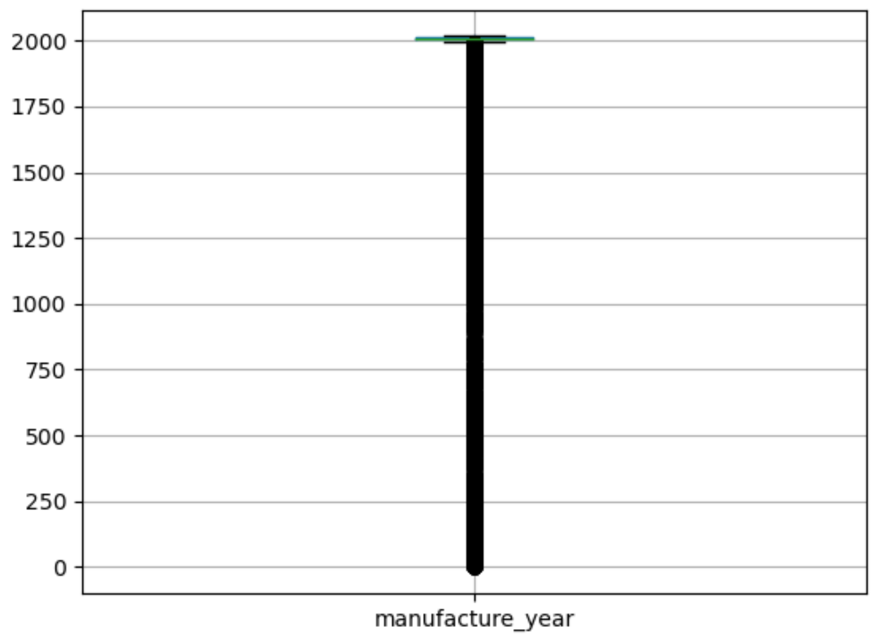
Column ‘manufacture_year’

The manufacture year column is a float type and requires a different approach than the first two columns. We will start by replacing the missing values with the median manufacture year of the corresponding maker, for example, if a Porsche car has a missing value in the manifacture_year column, the missing value will be replaced by the median manufacture year of Porsche.

```
# Replace NA with the median manufacture year of the maker
median_manufacture_year = df.groupby('maker')['manufacture_year'].median()
df['manufacture_year'].fillna(
    df['maker'].map(median_manufacture_year), inplace=True)
```

Next, we will verify the validity of the "manifacture_year" column. The range of this column should fall between the start of car manufacturing in the 1900s and the date the data was collected (2017). Any value outside this range will be considered invalid.

```
# Let's check the range
df.boxplot(column=['manufacture_year'])
plt.show()
```



The minimum value in the Manufacture Year column is 0, which is unrealistic as the first car was manufactured in the late 1800s. To address this issue, we will replace any value less than 1900 with the median manufacture year of the corresponding manufacturer. This will ensure that the values in the Manufacture Year column are realistic and within the range of the historical data available on car manufacturing. Lastly we will convert the data type to integer.

```
# Replace <1900 years with the median of their maker
df.loc[df['manufacture_year'] < 1900,
       'manufacture_year'] = df['maker'].map(median_manufacture_year)
# manufacture_year has a float data type, let's make it int
df['manufacture_year'] = df['manufacture_year'].astype('int')
```

Column ‘engine_displacement’

The engine displacement column has a unit ‘cm’, To ensure consistency and clarity in the data, it is recommended to remove the unit 'cm' from the column and store the values in the appropriate numerical format.
The engine displacement column appears to have non-null values, however, upon closer inspection, it is observed that the values of "0cm" were used to fill in the missing data. Removing the unit "cm" leaves many values as simply "0". These "0" values will be treated as missing data and will be replaced with the median engine displacement value for each respective manufacturer.

```
# Remove unit
df['engine_displacement'] = df['engine_displacement'].str.strip('cm')
# Change data type
df['engine_displacement'] = df['engine_displacement'].astype('int')
# Replace 0 with NA
df['engine_displacement'].replace(0, np.nan, inplace=True)
# Replace NA with the median engine_displacement of the maker
median_engine_displacement = df.groupby(
    'maker')['engine_displacement'].median()
df['engine_displacement'].fillna(
    df['maker'].map(median_engine_displacement), inplace=True)
```

The range of this column should fall between the smallest possible car engine size (49cc) and the largest possible car engine size (15000cc). Any value outside this range will be considered invalid.
To address this issue, we will replace any value less than 49 and more than 15000 with the median engine displacement of the corresponding manufacturer. This will ensure that the values in this column are realistic and within the range of the historical data available on car manufacturing. Lastly, we will convert the data type to an integer.

```
# Replace <49 & >15000 engine_displacement with the mode of their maker
df.loc[(df['engine_displacement'] < 49) |
       (df['engine_displacement'] > 15000),
       'engine_displacement'] = df['maker'].map(median_engine_displacement)
# Change data type
df['engine_displacement'] = df['engine_displacement'].astype('int')
```

Column ‘engine_power’

To clean this column we will start by replacing its missing values and values exceeding 750 kw with the median engine power of the corresponding manufacturer, then we will convert the column to the appropriate data type.

```
# Replace NA with the median maker power
median_engine_power = df.groupby('maker')['engine_power'].median()
df['engine_power'].fillna(
    df['maker'].map(median_engine_power), inplace=True)
# Replace >750 engine_power with the median of their maker
df.loc[df['engine_power'] > 750,
       'engine_power'] = df['maker'].map(median_engine_power)
# Replace data type
df['engine_power'] = df['engine_power'].astype('int')
```

Column ‘transmission’

The transmission column is in form of a string representing the transmission category (automatic or manual). To clean this column we will first replace NA with ‘unknown’, then we will deal with string inconsistency by converting all values to lowercase and removing leading/trailing whitespaces.
The transmission column can be either automatic or manual, but due to misspelling, this column has string inconsistency, to address this issue we will use the fuzz.token_sort_ratio method in Python same as the first column ‘maker’.

```
# Replace NA with unknown
df['transmission'].fillna('unknown', inplace=True)
# lowercase
df['maker'] = df['maker'].str.lower()
# Remove leading/trailing whitespaces
df['maker'] = df['maker'].str.strip()
# Check unique values
df['transmission'].unique()
# Some transmissions are misspelled
# First define the list of correct transmissions
transmissions_categories = ['manual', 'automatic', 'unknown']

# Calculate the similarity score between each value in the 'transmission' column
and each tr in 'transmissions_categories'
scores = df['transmission'].apply(
    lambda x: [fuzz.token_sort_ratio(x, tr) for tr in transmissions_categories])

# Find the index of the highest similarity score for each value in
'transmission'
highest_scores = scores.apply(lambda x: transmissions_categories[np.argmax(x)])

# Replace values in 'transmission' where the similarity score is at least 80
df['transmission'] = np.where(scores.apply(
    lambda x: np.max(x) >= 70), highest_scores, df['transmission'])
# Check
len(df['transmission'].unique()) == len(transmissions_categories)
```

Column ‘fuel_type’

The fuel type column will follow the same steps as the previous column, replacing NA with ‘unknown’, converting values to lowercase and removing leading/trailing whitespaces, then checking for unique values as the only possible values for this column should be 'diesel', 'unknown', 'gasoline', 'cng', 'lpg', and 'electric'

```python
# Replace NA with unknown
df['fuel_type'].fillna('unknown', inplace=True)
# lowercase
df['fuel_type'] = df['fuel_type'].str.lower()
# Remove leading/trailing whitespaces
df['fuel_type'] = df['fuel_type'].str.strip()
# Check unique values
df['fuel_type'].unique()
# Some rows have the string 'nan' referring to NA, we will replace 'nan' with unknown
df['fuel_type'].replace('nan', 'unknown', inplace=True)
```

Column ‘door_count’

The door_count column was found to have an object data type, upon inspection of its unique values, it was discovered that the string "None" was present, leading to the object type classification for the column. To clean this column we will first replace the ‘None’ values with NA and then we will replace NA values with the median door_count of the corresponding manufacturer. Lastly, we will limit the range to 2 – 6 as the number of doors in a car cannot differ from that range.

```python
# Replace 'None' with NA
df['door_count'].replace('None', np.nan, inplace=True)
# Replace NA with the median maker door count
median_door_count = df.groupby('maker')['door_count'].median()
df['door_count'].fillna(
    df['maker'].map(median_door_count), inplace=True)
# Change data type
df['door_count'] = df['door_count'].astype('float').astype('int')
# Replace <2 & >7 dour cunt with the mode of their maker
df.loc[(df['door_count'] < 2) | (df['door_count'] > 6),
       'door_count'] = df['maker'].map(median_door_count)
```

Column ‘seat_count’

The seat_count column has exactly the same issues as the door_count column, so it will be cleaned following the same steps of replacing ‘None’ with NA, NA with the median seat count of the corresponding manufacturer, and lastly, we will limit the range to 2 – 27 taking into account that transportation vehicles can have up to 27 seats.

```
# Replace 'None' with NA
df['seat_count'].replace('None', np.nan, inplace=True)
# Replace NA with the median maker seat count
median_seat_count = df.groupby('maker')['seat_count'].median()
df['seat_count'].fillna(
    df['maker'].map(median_seat_count), inplace=True)
# Change data type
df['seat_count'] = df['seat_count'].astype('float').astype('int')
# Replace <2 & >27 seat cunt with the mode of their maker
df.loc[(df['seat_count'] < 2) | (df['seat_count'] > 27),
       'seat_count'] = df['maker'].map(median_seat_count)
```

That was the last column in the dataset, now we will deal with duplicate rows.

Duplicate values

To deal with duplicates in this data we will first drop rows that are completely duplicated, then we will drop rows where only the seat count or door count is different.

```
# Dropping complete duplicate rows
df.drop_duplicates(inplace=True)
# Dropping duplicate rows having a difference only in seat_count or door_count
df.drop_duplicates(subset=['maker', 'model', 'manufacture_year',
'engine_displacement',
                        'engine_power', 'transmission', 'fuel_type'], keep=False,
inplace=True)
```

Irrelevent rows

It is suggested to remove rows from the data that have an unknown maker and model as they are not relevant to the analysis.

```
# Dropping rows with no maker and no model
df = df[~((df['maker'] == 'unknown') & (df['model'] == 'unknown'))]
```

Final cleaning steps

The dataset is now clean, let’s add columns for engine size in liters and power in HP, rename columns, and then sort the data and reset the index.

```
# Adding engine size in liters and engine power in hp
df['engine_size_liter'] = round(df['engine_displacement']/1000, 1)
df['engine_power_hp'] = round(df['engine_power'] * 1.34102).astype('int')

# Renaming columns
df = df.rename(columns={'engine_power': 'engine_power_kw',
               'engine_displacement': 'engine_displacement_cc'})

# Reordering columns
df = df[['maker', 'model', 'manufacture_year', 'engine_displacement_cc',
         'engine_size_liter', 'engine_power_kw', 'engine_power_hp',
         'transmission', 'fuel_type', 'door_count', 'seat_count']]

# Sorting the data by maker, model, and manufacture year ascending
df.sort_values(by=['maker', 'model', 'manufacture_year'],
               ascending=True, inplace=True)

# index
df.reset_index(drop=True, inplace=True)

# Exporting cleaned data
df.to_csv('car_data_clean', index=False)
```

A look at the the first 5 rows

```
       maker model  manufacture_year  engine_displacement_cc  \
0  alfa-romeo   145              1995                    1996
1  alfa-romeo   145              1995                    1910
2  alfa-romeo   145              1997                    1910
3  alfa-romeo   145              1997                    1910
4  alfa-romeo   145              1998                    1929

   engine_size_liter  engine_power_kw  engine_power_hp transmission fuel_type  \
0                2.0              110              148       manual   unknown
1                1.9              106              142      unknown   unknown
2                1.9               76              102      unknown   unknown
3                1.9               76              102      unknown    diesel
4                1.9               66               89       manual   unknown

   door_count  seat_count
0           3           5
1           5           5
2           5           5
3           5           5
4           3           5
```

Download the cleaned data [here]