Data governance project (Phase 1&2)

Anas mahmoud mohamed ashour
202201304

Dataset Description:

I used two data sets for different car companies to illustrate the importance of car components in making it more competitive in the marketing market, where the price of the car is determined based on

1- Arabic brand “make”

2-Year of publication

3- Its components, such as: the car’s engine, type of fuel, extent of use, and number of doors

Problem Definition:

first data has nulls in column (`Engine Cylinders`, `Market Category`, `Engine HP`, `Engine Fuel Type`)

```
[ ]  1  x=xxxx.isnull().sum()
```

```
1  x
```

```
Make                    0
Model                   0
Year                    0
Engine Fuel Type        3
Engine HP              69
Engine Cylinders       30
Transmission Type       0
Driven_Wheels           0
Number of Doors         6
Market Category      3742
Vehicle Size            0
Vehicle Style           0
highway MPG             0
city mpg                0
Popularity              0
MSRP                    0
dtype: int64
```

First data has nulls in column (`make`, `model`, `trim`, `body`, `condition`, `odometer`, `color`, `interior`, `mmr`, `Selling price`, `saledate`).

Methodology:

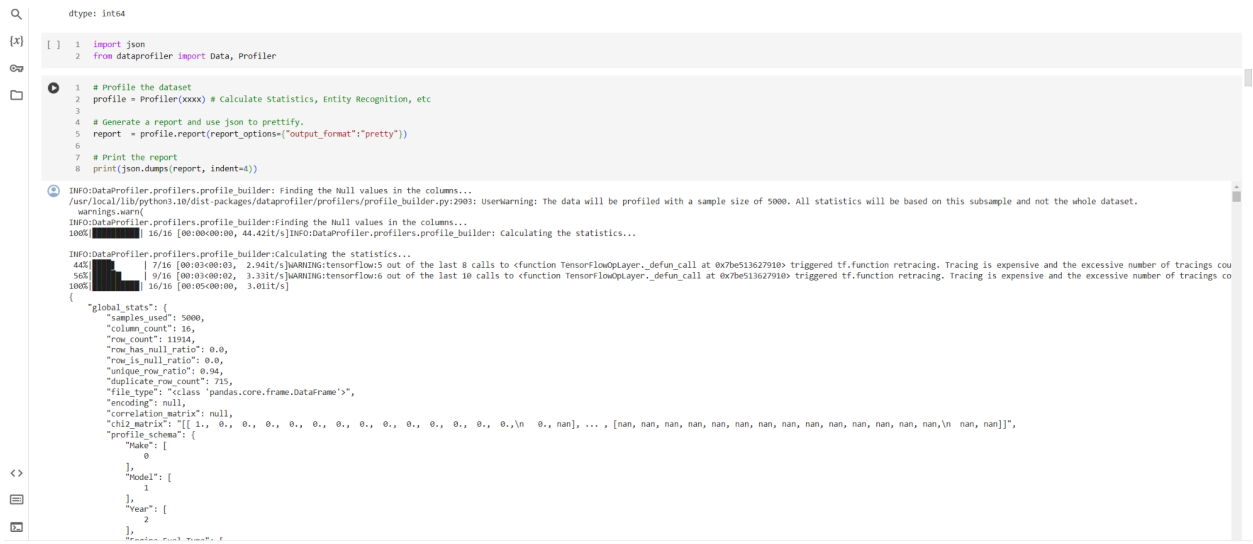
To solve this problem of duplicates:

We will drop the duplicate values.

To solve the problem of null values:

Fixed by replacing the null by the mean as the data that contains the null value is integer data type.

Then we made a profiling for the data by using “dataprofiler” library:



```
dtype: int64
```

```
import json
from dataprofiler import Data, Profiler
```

```
# Profile the dataset
profile = Profiler(xxxx) # Calculate Statistics, Entity Recognition, etc

# Generate a report and use json to prettify.
report  = profile.report(report_options={"output_format":"pretty"})

# Print the report
print(json.dumps(report, indent=4))
```

```
INFO:DataProfiler.profilers.profile_builder: Finding the Null values in the columns...
/usr/local/lib/python3.10/dist-packages/dataprofiler/profilers/profile_builder.py:2903: UserWarning: The data will be profiled with a sample size of 5000. All statistics will be based on this subsample and not the whole dataset.
  warnings.warn(
INFO:DataProfiler.profilers.profile_builder:Finding the Null values in the columns...
100%|██████████| 16/16 [00:00<00:00, 44.42it/s]INFO:DataProfiler.profilers.profile_builder: Calculating the statistics...

INFO:DataProfiler.profilers.profile_builder:Calculating the statistics...
 44%|████      | 7/16 [00:03<00:03,  2.94it/s]WARNING:tensorflow:5 out of the last 8 calls to <function TensorFlowOpLayer._defun_call at 0x7be513627910> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings cou
 56%|█████▋    | 9/16 [00:03<00:02,  3.33it/s]WARNING:tensorflow:6 out of the last 10 calls to <function TensorFlowOpLayer._defun_call at 0x7be513627910> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings co
100%|██████████| 16/16 [00:05<00:00,  3.01it/s]
{
    "global_stats": {
        "samples_used": 5000,
        "column_count": 16,
        "row_count": 11914,
        "row_has_null_ratio": 0.0,
        "row_is_null_ratio": 0.0,
        "unique_row_ratio": 0.94,
        "duplicate_row_count": 715,
        "file_type": "<class 'pandas.core.frame.DataFrame'>",
        "encoding": null,
        "correlation_matrix": null,
        "chi2_matrix": "[[ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,\n   0., nan], ... , [nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,\n  nan, nan]]",
        "profile_schema": {
            "Make": [
                0
            ],
            "Model": [
                1
            ],
            "Year": [
                2
            ],
```

Then we made a schema for the data by using “cerberus” library:

```
from cerberus import Validator
```

```
import pandas as pd
from cerberus import Validator

# Define the validation schema
schema = {
    'Make': {'type': 'string', 'nullable': False},
    'Model': {'type': 'string', 'nullable': False},
    'Year': {'type': 'integer', 'nullable': False},
    'Engine Fuel Type': {'type': 'string', 'nullable': False},
    'Engine HP': {'type': 'float', 'nullable': False},
    'Engine Cylinders': {'type': 'float', 'nullable': False},
    'Transmission Type': {'type': 'string', 'nullable': False},
    'Driven_Wheels': {'type': 'string', 'nullable': False},
    'Number of Doors': {'type': 'float', 'nullable': False},
    'Market Category': {'type': 'string', 'nullable': True},
    'Vehicle Size': {'type': 'string', 'nullable': False},
    'Vehicle Style': {'type': 'string', 'nullable': False},
    'highway MPG': {'type': 'integer', 'nullable': False},
    'city mpg': {'type': 'integer', 'nullable': False},
    'Popularity': {'type': 'integer', 'nullable': False},
    'MSRP': {'type': 'integer', 'nullable': False}
}

# Create a Validator object
validator = Validator(schema)

# Suppose df is your DataFrame
# df = pd.read_csv('your_dataset.csv')

# Validate each row (example for DataFrame 'df')
invalid_rows = []
for index, row in xxxx.iterrows():
    if not validator.validate(row.to_dict()):
        invalid_rows.append((index, validator.errors))

# Output the invalid rows and their errors
for index, errors in invalid_rows:
    print(f"Row {index} has validation errors: {errors}")
```

Then we made the inferred schema by using “pandera” library: \

ode + Text Copy to Drive

```
import pandera as pa
from pandera import Column, DataFrameSchema, Check, Index

inferred_schema = pa.infer_schema(xxxx).to_script()
print(inferred_schema)
```

```
        regex=False,
        description=None,
        title=None,
    ),
    "Model": Column(
        dtype="object",
        checks=None,
        nullable=False,
        unique=False,
        coerce=False,
        required=True,
        regex=False,
        description=None,
        title=None,
    ),
    "Year": Column(
        dtype="int64",
        checks=[
            Check.greater_than_or_equal_to(min_value=1990.0),
            Check.less_than_or_equal_to(max_value=2017.0),
        ],
        nullable=False,
        unique=False,
        coerce=False,
        required=True,
        regex=False,
        description=None,
        title=None,
    ),
    "Engine Fuel Type": Column(
        dtype="object",
        checks=None,
        nullable=False,
        unique=False,
        coerce=False,
        required=True,
        regex=False,
        description=None,
        title=None,
    ),
    "Engine HP": Column(
        dtype="float64",
        checks=[
            Check.greater_than_or_equal_to(min_value=55.0),
            Check.less_than_or_equal_to(max_value=1001.0),
```

To secure the dataset:

```
import numpy as np

dfp = pd.read_csv('cleaned_data.csv')

dfp
def add_noise(dfp, column, std = None):   #std=standaard deviation
    if std == None:
        std = dfp[column].std()

    withNoise = dfp[column].add(np.random.normal(0, std, dfp.shape[0]))
    copy = dfp.copy()
    copy[column] = withNoise
    return copy

perturbation = add_noise(dfp, 'Popularity', std=100)
perturbation
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3977.734060	46135
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3771.232152	40650
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916.250396	36350
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3835.922184	29450
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	4093.181204	34500
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11909	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	146.591807	46120
11910	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	118.904566	56670
11911	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	306.610144	50620
11912	Acura	ZDX	2013	premium unleaded (recommended)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	334.196861	50920
11913	Lincoln	Zephyr	2006	regular unleaded	221.0	6.0	AUTOMATIC	front wheel drive	4.0	Luxury	Midsize	Sedan	26	17	96.452223	28995

11914 rows × 16 columns

11914 rows × 16 columns

```
df= pd.read_csv('cleaned_data.csv')

# Swapping data in a specific column (e.g., swapping Age values)
df_swp = df.copy()
df_swp['Driven_Wheels'] = df['Market Category'].sample(frac=1).reset_index(drop=True)

df_swp
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	Luxury	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916	46135
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	Luxury	2.0	Luxury,Performance	Compact	Convertible	28	19	3916	40650
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	Luxury	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916	36350
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	Luxury	2.0	Luxury,Performance	Compact	Coupe	28	18	3916	29450
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	Luxury	2.0	Luxury	Compact	Convertible	28	18	3916	34500
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11909	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	Luxury	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	46120
11910	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	Luxury	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	56670
11911	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	Luxury	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	50620
11912	Acura	ZDX	2013	premium unleaded (recommended)	300.0	6.0	AUTOMATIC	Factory Tuner,Luxury,High-Performance	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	50920
11913	Lincoln	Zephyr	2006	regular unleaded	221.0	6.0	AUTOMATIC	Exotic,High-Performance	4.0	Luxury	Midsize	Sedan	26	17	61	28995

11914 rows × 16 columns

```
import hashlib
```

```
import random


# Define a function to scramble data in a specific column
def scramble_data(data):
    data_list = list(data)
    random.shuffle(data_list)
    return ''.join(data_list)

# Scramble the 'Medical Condition' column
df['Scrambled_Engine Fuel Type'] = df['Engine Fuel Type'].apply(scramble_data)

# Display the result
print(df[['Engine Fuel Type', 'Scrambled_Engine Fuel Type']].head())
```

```
           Engine Fuel Type   Scrambled_Engine Fuel Type
0  premium unleaded (required)   rpdrdeqnureimimle)uedaue (
1  premium unleaded (required)  ap )lrndiemuq eeeemd(druiur
2  premium unleaded (required)  mueueuipmiqln  derarr()eedd
3  premium unleaded (required)  ne)dar( meqieuduiupdrr elme
4  premium unleaded (required)   )endderduu ieepmlrq(rauiem
```

```
df
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP	Scrambled_Engine Fuel Type
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916	46135	rpdrdeqnureimimle)uedaue (
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3916	40650	ap)lrndiemuq eeeemd(druiur
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916	36350	mueueuipmiqln derarr()eedd
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3916	29450	ne)dar(meqieuduiupdrr elme
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	3916	34500	)endderduu ieepmlrq(rauiem
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11909	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	46120	dlq(ddaiermem)lee puruma

```python
from cryptography.fernet import Fernet


# Generate a key
key = Fernet.generate_key()

# Initialize cipher
cipher = Fernet(key)

# Encrypt the 'Date of Treatment' column using a lambda function
df['Encrypted_Date_of_Treatment'] = df['Year'].astype(str).apply(lambda x: cipher.encrypt(x.encode()).decode())
df['Encrypted_Market Category'] = df['Market Category'].astype(str).apply(lambda x: cipher.encrypt(x.encode()).decode())
df['Encrypted_Engine Fuel Type'] = df['Engine Fuel Type'].astype(str).apply(lambda x: cipher.encrypt(x.encode()).decode())
df
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	...	city mpg	Popularity	MSRP	Scrambled_Engine Fuel Type	Hashed_Treatment	
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	...	19	3916	46135	rpdrdeqnurelmimie)uedaue (	25e91e6b12a5331dfbcdfabba9c10159c0e458575e60e1...	440b1d6150cc3e7abb17e
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	...	19	3916	40650	ap)jrndlemuq eeeemd(druiur	25e91e6b12a5331dfbcdfabba9c10159c0e458575e60e1...	440b1d6150cc3e7abb17e
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	...	20	3916	36350	mueueulpmiqln derarr()eedd	25e91e6b12a5331dfbcdfabba9c10159c0e458575e60e1...	440b1d6150cc3e7abb17e
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	...	18	3916	29450	ne)dar(meqleudulupdrr elme	25e91e6b12a5331dfbcdfabba9c10159c0e458575e60e1...	440b1d6150cc3e7abb17e
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	...	18	3916	34500	)endderduu ieepmlrq(rauiem	25e91e6b12a5331dfbcdfabba9c10159c0e458575e60e1...	440b1d6150cc3e7abb17e
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
11909	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	...	16	204	46120	dlq(ddaierrmem)lee pururnue	25e91e6b12a5331dfbcdfabba9c10159c0e458575e60e1...	fb81b42f629f0588981948
11910	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	...	16	204	56670	lemdepuiedra re)(dn urmieuq	25e91e6b12a5331dfbcdfabba9c10159c0e458575e60e1...	fb81b42f629f0588981948

Results:

2 data sets were cleaned not having nulls and duplicates and all columns including data have data type as same as column data type and after clean nulls and duplicates data become more informative.

```
[ ]
    car.duplicated().sum()

    0
```

```
xxxx.isnull().sum()

Make                0
Model               0
Year                0
Engine Fuel Type    0
Engine HP           0
Engine Cylinders    0
Transmission Type   0
Driven_Wheels       0
Number of Doors     0
Market Category     0
Vehicle Size        0
Vehicle Style       0
highway MPG         0
city mpg            0
Popularity          0
MSRP                0
dtype: int64
```

```
[ ] cleaned_car.isnull().sum()

    year           0
    make           0
    model          0
    trim           0
    body           0
    transmission   0
    vin            0
    state          0
    condition      0
    odometer       0
    color          0
    interior       0
    seller         0
    mmr            0
    sellingprice   0
    saledate       0
    dtype: int64
```

Conclusion:

Through rigorous data cleaning processes, both datasets were transformed to eliminate null values and duplicates, enhancing their informativeness. This meticulous approach ensures accurate analysis and modeling, thereby facilitating informed decision-making for car companies seeking competitive positioning in the market.