PES University, Bangalore

Established under Karnataka Act No. 16 of 2013

UE22AM343AB4 - Advanced Data Analytics

Designed by Sathwik HJ

Student Details

- Name : **Shusrith S**
- SRN : **PES1UG22AM155**

Data cleaning and Preprocessing

Context

As an analyst at Torque Titans, you've been given an exciting opportunity to work with a comprehensive dataset that spans the motorcycle market from 1894 to 2022. Your primary responsibility is to clean and preprocess this data to ensure its quality and readiness for analysis. By doing so, you'll enable your team to extract valuable insights that will drive Torque Titans forward in a competitive market. This critical task will set the foundation for innovative, data-driven strategies that will fuel the company's success in the industry.

Let's dive in!

```
!wget https://raw.githubusercontent.com/MBUYt0n/ada/refs/heads/main/ADA_Workshe
```

```
--2024-09-20 17:56:38--  https://raw.githubusercontent.com/MBUYt0n/ada/refs
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199
HTTP request sent, awaiting response... 200 OK
Length: 32644306 (31M) [text/plain]
Saving to: 'all_bikes.csv.1'

all_bikes.csv.1     100%[===================>]  31.13M   183MB/s    in 0.2s

2024-09-20 17:56:38 (183 MB/s) - 'all_bikes.csv.1' saved [32644306/32644306
```

About the dataset

About the dataset

- "all_bikes.csv"
- Each record of the dataset represents a bike model which contains whereas details about it.

```
%pip install matplotlib pandas
%pip install numpy
%pip install scikit-learn
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dis
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/di
Requirement already satisfied: numpy<2.0,>=1.17.3 in /usr/local/lib/python3
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pytho
```

```
# Step 1: Import the required modules
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('all_bikes.csv')  # Replace with your dataset path
```

```
<ipython-input-31-ef01ff8d2199>:6: DtypeWarning: Columns (80,81) have mixed
  df = pd.read_csv('all_bikes.csv')  # Replace with your dataset path
```

Note:

Give reasons/explanations/reasoning for each question

Step 1: Understanding the Dataset Start by closely examining the dataset to understand its structure. Are there any discrepancies or inconsistencies? Are there columns that may not provide valuable insights? Begin by importing the required modules, such as Pandas, and let's gather some initial insights from the data:

- Analyze the number of columns, data types, and the number of values in each column.
- Calculate basic statistics like averages, minimums, and maximums for numerical data.

```
#Pandas Describe
stats = df.describe(include='all')
stats
```

	Model	Year	Category	Rating	Displacement	Engine type	Engin detail
count	38472	38472.000000	38472	38472	37461	38461	639
unique	18597	NaN	18	255	1330	30	130
top	Harley-Davidson Servi-Car GE	NaN	Scooter	Do you know this bike? Click here to rate it. W...	125.0 ccm (7.63 cubic inches)	Single cylinder, four-stroke	Titaniur valve
freq	38	NaN	6669	13018	1481	14703	16
mean	NaN	2003.195883	NaN	NaN	NaN	NaN	Nal
std	NaN	20.083372	NaN	NaN	NaN	NaN	Nal
min	NaN	1894.000000	NaN	NaN	NaN	NaN	Nal
25%	NaN	2000.000000	NaN	NaN	NaN	NaN	Nal
50%	NaN	2010.000000	NaN	NaN	NaN	NaN	Nal
75%	NaN	2016.000000	NaN	NaN	NaN	NaN	Nal
max	NaN	2022.000000	NaN	NaN	NaN	NaN	Nal

11 rows × 85 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38472 entries, 0 to 38471
Data columns (total 85 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Model                                           38472 non-null  object
 1   Year                                            38472 non-null  int64
 2   Category                                        38472 non-null  object
 3   Rating                                          38472 non-null  object
 4   Displacement                                    37461 non-null  object
 5   Engine type                                     38461 non-null  object
 6   Engine details                                  6390 non-null   object
```

```
 7   Power                                   26110 non-null  object
 8   Top speed                               12730 non-null  object
 9   Bore x stroke                           28689 non-null  object
 10  Fuel system                             27844 non-null  object
 11  Ignition                                16529 non-null  object
 12  Cooling system                          34258 non-null  object
 13  Gearbox                                 32675 non-null  object
 14  Transmission type                       32861 non-null  object
 15  Driveline                               8324 non-null   object
 16  Frame type                              17303 non-null  object
 17  Front suspension                        26107 non-null  object
 18  Rear suspension                         25540 non-null  object
 19  Wheels                                  9387 non-null   object
 20  Seat                                    4846 non-null   object
 21  Dry weight                              22483 non-null  object
 22  Power/weight ratio                      15213 non-null  object
 23  Clutch                                  15152 non-null  object
 24  Overall width                           18738 non-null  object
 25  Fuel capacity                           31704 non-null  object
 26  Comments                                13700 non-null  object
 27  Exhaust system                          5900 non-null   object
 28  Compression                             23405 non-null  object
 29  Fuel control                            22008 non-null  object
 30  Lubrication system                      10158 non-null  object
 31  Front tire                              31982 non-null  object
 32  Rear tire                               32008 non-null  object
 33  Front brakes                            36889 non-null  object
 34  Rear brakes                             36696 non-null  object
 35  Weight incl. oil, gas, etc              14994 non-null  object
 36  Overall length                          22242 non-null  object
 37  Ground clearance                        14205 non-null  object
 38  Wheelbase                               25493 non-null  object
 39  Oil capacity                            3786 non-null   object
 40  Color options                           24327 non-null  object
 41  Starter                                 26845 non-null  object
 42  Electrical                              3085 non-null   object
 43  Valves per cylinder                     16173 non-null  float64
 44  Diameter                                18816 non-null  object
 45  Carrying capacity                       2702 non-null   object
 46  Modifications compared to previous model  344 non-null    object
 47  Seat height                             24182 non-null  object
 48  Overall height                          16635 non-null  object
 49  Fuel consumption                        6176 non-null   object
 50  Greenhouse gases                        6176 non-null   object
 51  Torque                                  16634 non-null  object
 52  Max RPM                                 663 non-null    float64
```

Step 2: Handling Missing Values Next, let's address missing values in the dataset. For numeric columns, we'll fill the missing values with the column's average. For categorical or other columns, choose an appropriate strategy based on what makes sense for the data—this could be filled with a placeholder like "Unknown" or the most frequent value.

```
null_percentage = df.isna().sum() / len(df)
null_percentage
```

```
0
```

	0
Model	0.000000
Year	0.000000
Category	0.000000
Rating	0.000000
Displacement	0.026279
...	...
Belt width	0.999064
Pulley teeth	0.999116
Chain size	0.996985
Factory warranty	0.859196
Service interval	0.998414

85 rows × 1 columns

dtype: float64

```
a = df.columns[np.where(df.isna().sum() > 0)]
a
```

```
Index(['Displacement', 'Engine type', 'Engine details', 'Power', 'Top
speed',
       'Bore x stroke', 'Fuel system', 'Ignition', 'Cooling system',
'Gearbox',
       'Transmission type', 'Driveline', 'Frame type', 'Front suspension',
       'Rear suspension', 'Wheels', 'Seat', 'Dry weight', 'Power/weight
ratio',
       'Clutch', 'Overall width', 'Fuel capacity', 'Comments',
       'Exhaust system', 'Compression', 'Fuel control', 'Lubrication
system',
       'Front tire', 'Rear tire', 'Front brakes', 'Rear brakes',
       'Weight incl. oil, gas, etc', 'Overall length', 'Ground clearance',
       'Wheelbase', 'Oil capacity', 'Color options', 'Starter',
'Electrical',
       'Valves per cylinder', 'Diameter', 'Carrying capacity',
       'Modifications compared to previous model', 'Seat height',
       'Overall height', 'Fuel consumption', 'Greenhouse gases', 'Torque',
       'Max RPM', 'Light', 'Alternate seat height', 'Rake (fork angle)',
       '0-100 km/h (0-62 mph)', 'Front wheel travel', 'Rear wheel travel',
       'Engine oil', 'Instruments', '60-140 km/h (37-87 mph), highest
gear',
       'Front percentage of weight', 'Trail', 'Brake fluid', 'Coolant',
       'Spark plugs', 'Idle speed', 'Tire pressure front',
       'Tire pressure rear', 'Fork tube size', 'Chain links', 'Sprockets',
       'Reserve fuel capacity', '1/4 mile (0.4 km)', 'Emission details',
       'Rear percentage of weight', 'Oil filter', 'Battery', 'Belt teeth',
       'Belt width', 'Pulley teeth', 'Chain size', 'Factory warranty',
       'Service interval'],
      dtype='object')
```

dtype='object')

```
numerical = df.describe()
means = numerical.loc["mean"]
df[numerical.columns] = df[numerical.columns].fillna(means)
df[numerical.columns].isna().sum()
```

	0
Year	0
Valves per cylinder	0
Max RPM	0
Front percentage of weight	0
Chain links	0
Rear percentage of weight	0
Belt teeth	0
Chain size	0

dtype: int64

```
non_numeric = list(set(a) - set(numerical.columns))
non_numeric_df = df[non_numeric]
non_numeric_modes = non_numeric_df.describe().loc["top"]
non_numeric_modes
```

	top
Seat	Dual seat
Fuel control	Double Overhead Cams/Twin Cam (DOHC)
Tire pressure rear	36 PSI (2.5 Bar or 250 kPa)
Engine type	Single cylinder, four-stroke
Reserve fuel capacity	4.00 litres (1.06 US gallons)
...	...
Top speed	45.0 km/h (28.0 mph)
Greenhouse gases	129.9 CO2 g/km. (CO2 - Carbon dioxide emission)
Spark plugs	NGK DCPR7E, NGK DCPR7EIX
Driveline	CVT
Lubrication system	Wet sump

74 rows × 1 columns

dtype: object

dtype: object

```
df[non_numeric] = df[non_numeric].fillna(non_numeric_modes)
df.isna().sum()
```

	0
Model	0
Year	0
Category	0
Rating	0
Displacement	0
...	...
Belt width	0
Pulley teeth	0
Chain size	0
Factory warranty	0
Service interval	0

85 rows × 1 columns

dtype: int64

Step 3: Eliminating Redundancies Take a look at the 0-100 column—do we really need speed in two different units? Let's clean up this redundancy. Be mindful, though; this might not be the only column with unnecessary duplication.

```
df.columns
```

```
Index(['Model', 'Year', 'Category', 'Rating', 'Displacement', 'Engine 
type',
       'Engine details', 'Power', 'Top speed', 'Bore x stroke', 'Fuel 
system',
       'Ignition', 'Cooling system', 'Gearbox', 'Transmission type',
       'Driveline', 'Frame type', 'Front suspension', 'Rear suspension',
       'Wheels', 'Seat', 'Dry weight', 'Power/weight ratio', 'Clutch',
       'Overall width', 'Fuel capacity', 'Comments', 'Exhaust system',
       'Compression', 'Fuel control', 'Lubrication system', 'Front tire',
       'Rear tire', 'Front brakes', 'Rear brakes',
       'Weight incl. oil, gas, etc', 'Overall length', 'Ground clearance',
       'Wheelbase', 'Oil capacity', 'Color options', 'Starter', 
'Electrical',
       'Valves per cylinder', 'Diameter', 'Carrying capacity',
       'Modifications compared to previous model', 'Seat height',
       'Overall height', 'Fuel consumption', 'Greenhouse gases', 'Torque',
```

```
       'Overall height', 'Fuel consumption', 'Greenhouse gases', 'Torque',
       'Max RPM', 'Light', 'Alternate seat height', 'Rake (fork angle)',
       '0-100 km/h (0-62 mph)', 'Front wheel travel', 'Rear wheel travel',
       'Engine oil', 'Instruments', '60-140 km/h (37-87 mph), highest
gear',
       'Front percentage of weight', 'Trail', 'Brake fluid', 'Coolant',
       'Spark plugs', 'Idle speed', 'Tire pressure front',
       'Tire pressure rear', 'Fork tube size', 'Chain links', 'Sprockets',
       'Reserve fuel capacity', '1/4 mile (0.4 km)', 'Emission details',
       'Rear percentage of weight', 'Oil filter', 'Battery', 'Belt teeth',
       'Belt width', 'Pulley teeth', 'Chain size', 'Factory warranty',
       'Service interval'],
      dtype='object')
```

```
a = len(df) - len(df[df["Tire pressure rear"] == df["Tire pressure front"]])
b = len(df) - len(df[df["Front brakes"] == df["Rear brakes"]])
c = len(df) - len(df[df["Rear percentage of weight"] == df["Front percentage of
d = len(df) - len(df[df["Front wheel travel"] == df["Rear wheel travel"]])
print(a, b, c, d)
```

```
156 21499 38467 36349
```

```
a = df[["Gearbox", "Transmission type"]]
a.value_counts()
```

Gearbox	Transmission type	count
6-speed	Chain (final drive)	14595
5-speed	Chain (final drive)	7915
Automatic	Belt (final drive)	3828
4-speed	Chain (final drive)	2258
5-speed	Shaft drive (cardan) (final drive)	1990
6-speed	Belt (final drive)	1974
Automatic	Chain (final drive)	1599
6-speed	Shaft drive (cardan) (final drive)	1132
5-speed	Belt (final drive)	1087
Automatic	Shaft drive (cardan) (final drive)	595
4-speed	Shaft drive (cardan) (final drive)	419
3-speed	Chain (final drive)	406
1-speed	Chain (final drive)	224
4-speed	Belt (final drive)	119
3-speed	Shaft drive (cardan) (final drive)	57
2-speed	Chain (final drive)	53

1-speed	**Belt**	**(final drive)**	51
4-speed with reverse	**Shaft drive (cardan)**	**(final drive)**	49
2-speed	**Shaft drive (cardan)**	**(final drive)**	25
	Belt	**(final drive)**	15
3-speed	**Belt**	**(final drive)**	15
7-speed	**Shaft drive (cardan)**	**(final drive)**	13
	Chain	**(final drive)**	11
100-speed	**Belt**	**(final drive)**	7
	Shaft drive (cardan)	**(final drive)**	6
2-speed automatic	**Shaft drive (cardan)**	**(final drive)**	5
1-speed	**Shaft drive (cardan)**	**(final drive)**	5
5-speed with reverse	**Shaft drive (cardan)**	**(final drive)**	5
2-speed automatic	**Chain**	**(final drive)**	3
10-speed	**Shaft drive (cardan)**	**(final drive)**	3
8-speed	**Chain**	**(final drive)**	3
100-speed	**Chain**	**(final drive)**	2
3-speed automatic	**Chain**	**(final drive)**	1
6-speed with reverse	**Shaft drive (cardan)**	**(final drive)**	1
10-speed	**Chain**	**(final drive)**	1

dtype: int64

```
df.drop(
    [
        "Weight incl. oil, gas, etc",
        "1/4 mile (0.4 km)",
        "60-140 km/h (37-87 mph), highest gear",
        "Tire pressure front",
        "Greenhouse gases"
    ], axis=1, inplace=True
)
```

Step 4: Duplicates Now, check for any duplicate records in the dataset. If duplicates are found, remove them to avoid any skewed analysis.

```
df.drop_duplicates(inplace=True)
```

Step 5: Engine Details Column Examine the Engine Details column carefully. Will this column be useful in providing insights, or is it redundant or irrelevant to the analysis? Decide whether to keep or drop it.

```
df["Engine details"].value_counts() / len(df)
```

	count
Engine details	
Titanium valves	0.838090
Reed intake.	0.003613
90° V-twin	0.002495
Reed valve.	0.002079
Balancer shaft	0.002027
...	...
Permanent magnet synchronous motor in a disc armature design.	0.000026
Permanent magnet synchronous motorin a disc armature design	0.000026
48 V BLDC motor with outer rotor	0.000026
16 valves with variable valve timing	0.000026
Fuel injection: ø42 mm x 2	0.000026

1301 rows × 1 columns

dtype: float64

```
df.drop("Engine details", axis=1, inplace=True)
```

```
Start coding or generate with AI.
```

Step 6: Preparing for Future Text Processing Torque Titans might explore text processing on some of the data in the future, so let's be proactive! We can tokenize the strings in the relevant columns to ensure we're ready for text analysis down the line. This involves splitting text into individual tokens (words) and storing them for future use.

There are a lot of tokeniser available, note: Torque Titans are potentially looking to integrate with openai.

```
!pip install tiktoken
```

```
Requirement already satisfied: tiktoken in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.1
```

```
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/p
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
```

```
import tiktoken

enc = tiktoken.get_encoding("o200k_base")
non_numeric = df.select_dtypes(exclude=np.number)

tokenized_df = non_numeric.apply(lambda x: x.apply(lambda y: enc.encode(y)))
tokenized_df
```

	Model	Category	Rating	Displacement	Engine type	Power	Top speed	Bore x stroke
0	[32, 41166, 128298, 47, 220, 1434, 6437, 2884]	[4764, 2884, 820, 1277, 12086]	[220, 18, 13, 22, 220, 5310, 4383, 842, 290, 1...	[3796, 13, 24, 274, 7871, 350, 18, 13, 3000, 6...	[13301, 49938, 11, 1920, 6321, 11472]	[24, 13, 20, 21979, 350, 21, 13, 24, 220, 372,...	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15,...	[1723, 13, 15, 1215, 220, 3255, 13, 22, 8957, ...
1	[32, 41166, 128298, 47, 220, 1434, 460]	[50837]	[220, 18, 13, 21, 220, 5310, 4383, 842, 290, 1...	[3796, 13, 24, 274, 7871, 350, 18, 13, 3000, 6...	[13301, 49938, 11, 1920, 6321, 11472]	[24, 13, 20, 21979, 350, 21, 13, 24, 220, 372,...	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15,...	[1723, 13, 15, 1215, 220, 3255, 13, 22, 8957, ...
2	[32, 41166, 128298, 47, 220, 1434, 8220, 46121...	[17260, 3598, 597]	[220, 16, 13, 24, 220, 5310, 4383, 842, 290, 1...	[3796, 13, 24, 274, 7871, 350, 18, 13, 3000, 6...	[13301, 49938, 11, 1920, 6321, 11472]	[24, 13, 20, 21979, 350, 21, 13, 24, 220, 372,...	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15,...	[1723, 13, 15, 1215, 220, 3255, 13, 22, 8957, ...
3	[32, 41166, 11448, 18, 220, 10676, 6437, 2884]	[4764, 2884, 820, 1277, 12086]	[220, 17, 13, 16, 220, 5310, 4383, 842, 290, 1...	[16059, 13, 15, 274, 7871, 350, 22, 13, 5085, ...	[13301, 49938, 11, 4242, 6321, 11472]	[899, 13, 20, 21979, 350, 24, 13, 16, 220, 372...	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15,...	[6733, 13, 15, 1215, 220, 6733, 13, 15, 8957, ...

4	[32, 41166, 11448, 18, 220, 10676, 6437, 2884,...	[4764, 2884, 820, 1277, 12086]	[6449, 481, 1761, 495, 17431, 30, 3524, 2105, ...	[16059, 13, 15, 274, 7871, 350, 22, 13, 5085, ...	[13301, 49938, 11, 4242, 6321, 11472]	[899, 13, 20, 21979, 350, 24, 13, 16, 220, 372...	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15,...	[6733, 13, 15, 1215, 220, 6733, 13, 15, 8957, ...
...	...	...	...	...	...	...	...	...
38467	[347, 1191, 15928, 5239, 394, 336, 220, 4095, ...	[188210, 8708]	[6449, 481, 1761, 495, 17431, 30, 3524, 2105, ...	[10676, 13, 15, 274, 7871, 350, 22, 13, 8876, ...	[78080]	[19, 13, 15, 21979, 350, 17, 13, 24, 220, 372,...	[3898, 13, 15, 8571, 14174, 350, 1723, 13, 19,...	[6733, 13, 15, 1215, 220, 6733, 13, 15, 8957, ...
38468	[347, 1191, 15928, 5239, 394, 336, 220, 3234, 15]	[188210, 8708]	[6449, 481, 1761, 495, 17431, 30, 3524, 2105, ...	[10676, 13, 15, 274, 7871, 350, 22, 13, 8876, ...	[78080]	[21, 13, 22, 21979, 350, 19, 13, 24, 220, 372,...	[10116, 13, 15, 8571, 14174, 350, 6231, 13, 15...	[6733, 13, 15, 1215, 220, 6733, 13, 15, 8957, ...
38469	[823, 130874, 113621]	[188210, 8708]	[6449, 481, 1761, 495, 17431, 30, 3524, 2105, ...	[10676, 13, 15, 274, 7871, 350, 22, 13, 8876, ...	[78080]	[17, 13, 22, 21979, 350, 17, 13, 15, 220, 372,...	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15,...	[6733, 13, 15, 1215, 220, 6733, 13, 15, 8957, ...
38470	[823, 130874, 122050]	[188210, 8708]	[6449, 481, 1761, 495, 17431, 30, 3524, 2105, ...	[10676, 13, 15, 274, 7871, 350, 22, 13, 8876, ...	[78080]	[17, 13, 22, 21979, 350, 17, 13, 15, 220, 372,...	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15,...	[6733, 13, 15, 1215, 220, 6733, 13, 15, 8957, ...
38471	[823, 130874, 45558]	[188210, 8708]	[6449, 481, 1761, 495, 17431, 30, 3524, 2105,	[10676, 13, 15, 274, 7871, 350, 22, 13, 8876, ...	[78080]	[17, 13, 22, 21979, 350, 17, 13, 15, 220, 372	[2548, 13, 15, 8571, 14174, 350, 2029, 13, 15	[6733, 13, 15, 1215, 220, 6733, 13, 15, 8957,

...

38472 rows × 71 columns

```
df.columns
```

```
Index(['Model', 'Year', 'Category', 'Rating', 'Displacement', 'Engine
type',
       'Power', 'Top speed', 'Bore x stroke', 'Fuel system', 'Ignition',
       'Cooling system', 'Gearbox', 'Transmission type', 'Driveline',
       'Frame type', 'Front suspension', 'Rear suspension', 'Wheels',
'Seat',
       'Dry weight', 'Power/weight ratio', 'Clutch', 'Overall width',
       'Fuel capacity', 'Comments', 'Exhaust system', 'Compression',
       'Fuel control', 'Lubrication system', 'Front tire', 'Rear tire',
       'Front brakes', 'Rear brakes', 'Overall length', 'Ground
clearance',
       'Wheelbase', 'Oil capacity', 'Color options', 'Starter',
'Electrical',
       'Valves per cylinder', 'Diameter', 'Carrying capacity',
       'Modifications compared to previous model', 'Seat height',
       'Overall height', 'Fuel consumption', 'Torque', 'Max RPM', 'Light',
       'Alternate seat height', 'Rake (fork angle)', '0-100 km/h (0-62
mph)',
       'Front wheel travel', 'Rear wheel travel', 'Engine oil',
'Instruments',
       'Front percentage of weight', 'Trail', 'Brake fluid', 'Coolant',
       'Spark plugs', 'Idle speed', 'Tire pressure rear', 'Fork tube
size',
       'Chain links', 'Sprockets', 'Reserve fuel capacity', 'Emission
details',
       'Rear percentage of weight', 'Oil filter', 'Battery', 'Belt teeth',
       'Belt width', 'Pulley teeth', 'Chain size', 'Factory warranty',
       'Service interval'],
      dtype='object')
```

Step 7: Designing a Rating System Here comes the fun part—Torque Titans wants to roll out a rating system for their motorcycles! You'll take into account the following factors:

● Speed

● Engine type (feel free to assign weight based on your opinion of which engines are superior)

● 0-100 acceleration

● Power

● Torque

● Weight

● RPM (you might want to cross-reference with the torque column).

Using these factors, create a new column called Rating, which is scaled from 1 to 4 (no decimals). You don't need to give all columns equal weight—experiment to find the ideal balance for what you think makes a great motorcycle!

Make sure to normalise the values before considering them for the rating column, as they can add a bias to the calculations, Lets try using MaxAbsScaler and try keep the values between [-1,1]

```
A = ['Engine type', 'Top speed', 'Power', 'Torque', 'Dry weight']
features = df[A]
```

```
for i in A:
  print(features[i].value_counts())
```

```
    Engine type
    Single cylinder, four-stroke       14714
    V2, four-stroke                     7405
    Single cylinder, two-stroke         5982
    In-line four, four-stroke           3152
    Twin, four-stroke                   2888
    Electric                             979
    Two cylinder boxer, four-stroke      862
    In-line three, four-stroke           794
    Twin, two-stroke                     500
    V4, four-stroke                      452
    Six cylinder boxer, four-stroke      137
    In-line six, four-stroke             111
    V8, four-stroke                       79
    Two cylinder boxer, two-stroke        74
    In-line three, two-stroke             58
    Four cylinder boxer, four-stroke      52
    V6, four-stroke                       42
    V2, two-stroke                        39
    Diesel                                37
    Square four cylinder                  33
    Gas turbine                           19
    In-line four, two-stroke              17
    Dual disk Wankel                      13
    Radial                                10
    Single disk Wankel                     8
    V4, two-stroke                         7
    In-line six, two-stroke                3
    V3, two-stroke                         3
    V10, four-stroke                       1
    Four cylinder boxer, two-stroke        1
    Name: count, dtype: int64
    Top speed
    45.0 km/h (28.0 mph)      26278
    90.0 km/h (55.9 mph)        445
    100.0 km/h (62.1 mph)       422
    110.0 km/h (68.4 mph)       401
    95.0 km/h (59.0 mph)        342
                              ...
    267.0 km/h (165.9 mph)        1
    67.0 km/h (41.6 mph)          1
    224.5 km/h (139.5 mph)        1
    32.2 km/h (20.0 mph)          1
```

```
32.2 km/h (20.0 mph)          1
133.6 km/h (83.0 mph)         1
Name: count, Length: 416, dtype: int64
Power
27.0 HP (19.7  kW)) @ 6000 RPM    12458
50.0 HP (36.5  kW)) @ 6500 RPM       90
27.0 HP (19.7  kW)) @ 6500 RPM       85
15.0 HP (10.9  kW))                  83
2.7 HP (2.0  kW))                    76
                                 ...
65.7 HP (48.0  kW)) @ 4700 RPM        1
71.0 HP (51.8  kW)) @ 5500 RPM        1
63.0 HP (46.0  kW)) @ 4700 RPM        1
64.0 HP (46.7  kW)) @ 4900 RPM        1
136.0 HP (99.3  kW))                  1
Name: count, Length: 4914, dtype: int64
```

```
# Remove commas before converting to float
features["Top speed"] = features["Top speed"].apply(lambda x: float(x.split()[0
features["Dry weight"] = features["Dry weight"].apply(lambda x: float(x.split()
features["RPM"] = features["Torque"].apply(lambda x: float(x.split()[-2].replac
features["Power"] = features["Power"].apply(lambda x: float(x.split()[0].replac
features["Torque"] = features["Torque"].apply(lambda x: float(x.split()[0].repl
```

```
<ipython-input-82-390fbd474cf6>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  features["Top speed"] = features["Top speed"].apply(lambda x: float(x.spl
<ipython-input-82-390fbd474cf6>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  features["Dry weight"] = features["Dry weight"].apply(lambda x: float(x.s
<ipython-input-82-390fbd474cf6>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  features["RPM"] = features["Torque"].apply(lambda x: float(x.split()[-2].
<ipython-input-82-390fbd474cf6>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  features["Power"] = features["Power"].apply(lambda x: float(x.split()[0].
<ipython-input-82-390fbd474cf6>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  features["Torque"] = features["Torque"].apply(lambda x: float(x.split()[0
```

```
features
```

	Engine type	Top speed	Power	Torque	Dry weight	RPM
0	Single cylinder, two-stroke	45.0	9.5	52.0	78.0	4000.0
1	Single cylinder, two-stroke	45.0	9.5	52.0	78.0	4000.0
2	Single cylinder, two-stroke	45.0	9.5	52.0	78.0	4000.0
3	Single cylinder, four-stroke	45.0	12.5	8.5	110.0	8000.0
4	Single cylinder, four-stroke	45.0	12.5	8.5	110.0	8000.0
...	...	...	...	...	...	...
38467	Electric	65.0	4.0	52.0	144.0	4000.0
38468	Electric	82.0	6.7	52.0	130.0	4000.0
38469	Electric	45.0	2.7	52.0	110.0	4000.0
38470	Electric	45.0	2.7	52.0	110.0	4000.0

Next steps: Generate code with features | View recommended plots | New interactive sheet

```
for i in features:
  print(features[i].value_counts())
```

```
Engine type
Single cylinder, four-stroke        14714
V2, four-stroke                      7405
Single cylinder, two-stroke          5982
In-line four, four-stroke            3152
Twin, four-stroke                    2888
Electric                              979
Two cylinder boxer, four-stroke       862
In-line three, four-stroke            794
Twin, two-stroke                      500
V4, four-stroke                       452
Six cylinder boxer, four-stroke       137
In-line six, four-stroke              111
V8, four-stroke                        79
Two cylinder boxer, two-stroke         74
In-line three, two-stroke              58
Four cylinder boxer, four-stroke       52
V6, four-stroke                        42
V2, two-stroke                         39
Diesel                                 37
Square four cylinder                   33
Gas turbine                            19
In-line four, two-stroke               17
Dual disk Wankel                       13
Radial                                 10
Single disk Wankel                      8
V4, two-stroke                          7
```

```
In-line six, two-stroke                  3
V3, two-stroke                           3
V10, four-stroke                         1
Four cylinder boxer, two-stroke          1
Name: count, dtype: int64
Top speed
45.0     26278
90.0       445
100.0      422
110.0      401
95.0       342
         ...
267.0        1
67.0         1
224.5        1
32.2         1
133.6        1
Name: count, Length: 416, dtype: int64
Power
27.0     13207
50.0       582
17.0       512
100.0      455
15.0       369
         ...
40.4         1
82.9         1
93.8         1
19.1         1
194.5        1
Name: count, Length: 849, dtype: int64
```

```
d = {"Single" : 1, "single": 1, "two" : 2, "three" : 3, "four" : 4, "five" : 5,
e = features["Engine type"]
s = []
for i in e:
  c = 0
  p = i.split(",")
  for j in d:
    for k in p:
      if j in k:
        c += d[j]
  s.append(c)
features["Engine type"] = s
features
```

```
<ipython-input-85-125ba2a9ffae>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  features["Engine type"] = s
```

	Engine type	Top speed	Power	Torque	Dry weight	RPM
0	3	45.0	9.5	52.0	78.0	4000.0
1	3	45.0	9.5	52.0	78.0	4000.0

2	3	45.0	9.5	52.0	78.0	4000.0
3	5	45.0	12.5	8.5	110.0	8000.0
4	5	45.0	12.5	8.5	110.0	8000.0
...	...	...	...	...	...	...
38467	0	65.0	4.0	52.0	144.0	4000.0
38468	0	82.0	6.7	52.0	130.0	4000.0
38469	0	45.0	2.7	52.0	110.0	4000.0
38470	0	45.0	2.7	52.0	110.0	4000.0
38471	0	45.0	2.7	52.0	110.0	4000.0

38472 rows × 6 columns

Next steps: Generate code with `features` | View recommended plots | New interactive sheet

```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
normalized_df = pd.DataFrame(scaler.fit_transform(features), columns=features.co
normalized_df
```

	Engine type	Top speed	Power	Torque	Dry weight	RPM
0	0.3	0.069231	0.011807	0.073034	0.078	0.275862
1	0.3	0.069231	0.011807	0.073034	0.078	0.275862
2	0.3	0.069231	0.011807	0.073034	0.078	0.275862
3	0.5	0.069231	0.015536	0.011938	0.110	0.551724
4	0.5	0.069231	0.015536	0.011938	0.110	0.551724
...	...	...	...	...	...	...
38467	0.0	0.100000	0.004971	0.073034	0.144	0.275862
38468	0.0	0.126154	0.008327	0.073034	0.130	0.275862
38469	0.0	0.069231	0.003356	0.073034	0.110	0.275862
38470	0.0	0.069231	0.003356	0.073034	0.110	0.275862
38471	0.0	0.069231	0.003356	0.073034	0.110	0.275862

38472 rows × 6 columns

Next steps: Generate code with `normalized_df` | View recommended plots | New interactive sheet

```
weights = [0.8, 0.8, 0.8, 0.8, -0.8, 0.8]
def map_number(n):
    if n < 0.25:
        return 1
    elif n < 0.5:
        return 2
    elif n < 0.75:
        return 3
    else:
        return 4

normalized_df["Rating"] = (normalized_df * weights).sum(axis=1)
normalized_df["Rating"] = normalized_df["Rating"].apply(map_number)
normalized_df
```

	Engine type	Top speed	Power	Torque	Dry weight	RPM	Rating
0	0.3	0.069231	0.011807	0.073034	0.078	0.275862	3
1	0.3	0.069231	0.011807	0.073034	0.078	0.275862	3
2	0.3	0.069231	0.011807	0.073034	0.078	0.275862	3
3	0.5	0.069231	0.015536	0.011938	0.110	0.551724	4
4	0.5	0.069231	0.015536	0.011938	0.110	0.551724	4
...	...	...	...	...	...	...	...
38467	0.0	0.100000	0.004971	0.073034	0.144	0.275862	1
38468	0.0	0.126154	0.008327	0.073034	0.130	0.275862	2
38469	0.0	0.069231	0.003356	0.073034	0.110	0.275862	1
38470	0.0	0.069231	0.003356	0.073034	0.110	0.275862	1
38471	0.0	0.069231	0.003356	0.073034	0.110	0.275862	1

38472 rows × 7 columns

Next steps: Generate code with `normalized_df` | View recommended plots | New interactive sheet

Double-click (or enter) to edit

By completing these steps, you'll not only ensure that the dataset is ready for insightful analysis but also set the stage for exciting innovations at Torque Titans. Let’s get started and have fun while we shape the future of motorcycles!