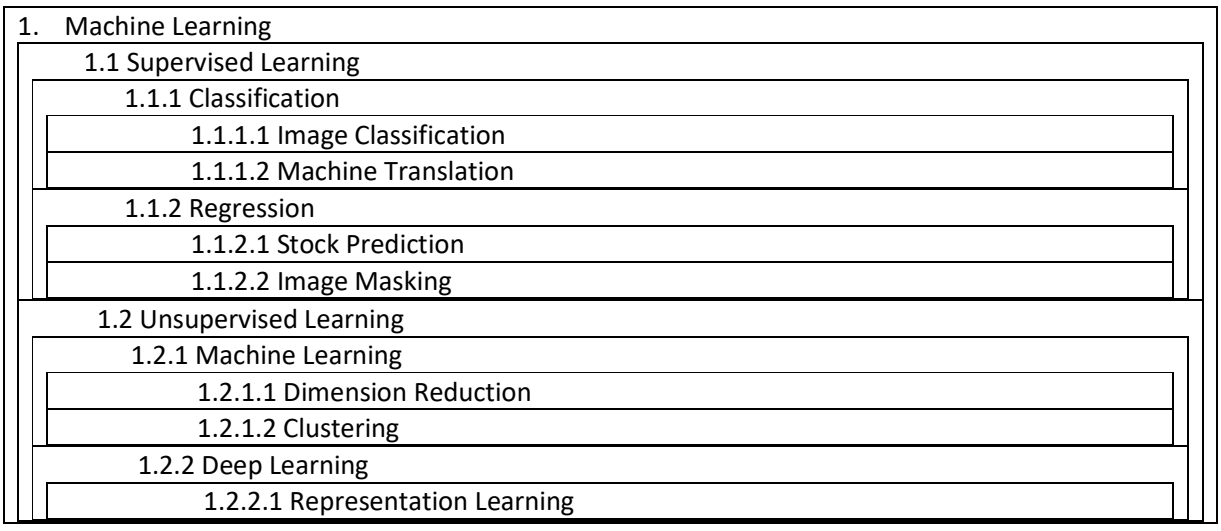
Introduction to Machine Learning

Machine learning is aimed at discovering patterns in data. It gives computers the ability to learn without being explicitly programmed. It involves using statistical methods to create programs that either improve performance over time, or detect patterns in massive amounts of data that humans would be unlikely to find. Machine Learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data driven predictions or decisions, rather than following strictly static program instructions. In short, we may say that Machine Learning is a collection of algorithms and techniques used to create computational systems that learn from data in order to make predictions and inferences.
Machine learning is a subset of Artificial Intelligence (AI).

Data used in analysis is about real-world phenomenon. For example, daily stock prices, earnings, and reviews are some of the indicators of stock market. This data could somehow provide the means for understanding the stock market and serve to make some predictions. Each piece of data describes an observation. The collection of these observations can assist us in serving some desired goal within a context, in this case that of the stock market. How do we process the data to find the answers we may be seeking? We use models of reality involving masses of data, look for relationships among the attributes that represent data and perform processing operations based on those relationships.

A partial classification of techniques used in machine learning are shown in Figure 1.

Figure 1: Machine Learning Techniques Classification



1.2.2.2 Generative Models

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples.

In supervised machine learning, labeled data set is used as the basis for predicting through the use of machine learning algorithms. Two of the common methods are classification and regression.

In the category of supervised learning, classification techniques focus on predicting a qualitative response by analyzing data and recognizing patterns. For example, this type of technique is used to classify whether or not a credit card transaction is fraudulent. Regression techniques begins with a choice of a plausible mathematical model that is likely to the given data. Appropriate learning algorithms are then used to try out and find a solution that best fits the model.

In unsupervised machine learning, the users do not need to supervise the model. The model works on its own to discover previously undetected patterns and information. Algorithms almost always require structured data, while deep learning networks rely on layers of artificial neural networks (ANN). If the outcome from machine learning produces an incorrect result, the algorithms go through a learning process. In deep learning, the mistakes are corrected through an automated process.

The following is a typical sequence of activities on data:

1. Take the raw data and identify the attributes inherently associated the data, making it information.
2. Take the information and explore the possibilities
3. Extract meaningful features relevant to problem solution and outcomes
4. Construct and try model based on perceived pattern in data.
5. Modify the model parameters or change to other models improved solutions

Some other examples of situations for machine learning are weather forecasting, email spam identification, fraud detection, probability of customer purchasing a product or renewal of insurance policy, predicting the chances of a person with a known illness, etc.

All situations can be described in terms data which may consist of text, categories or numerical values. Data is about objects in the world such as a person, customers, buyers, sellers, products, and any other situation of interest to us.

Observed data in a population may be expressed in terms of nominal, ordinal, interval, ration scales, etc.

The following are some examples.

Scale Type	Object Example	Attribute Name	Example of Measurement
Nominal	Flower Person Telephone	Color Gender Age Residence Brand	Red, Yellow, Purple, Orange Male, Female, other Infant, Child, Teen, Adult, Elderly Suburb, City, Town Apple, Samsung, LG
Ordinal	Person Service	Academic Rank Clothing Size Grade Satisfaction	Professor, Lecturer Small, Medium, large A, B, C Satisfied, Not satisfied, Neutral
Interval	Person	Income in thousands IQ	Below 50, 50-100, over 100 80-100, 100-120, 120-140
Ratio (values cab compared as double, half, etc.	Person	Height (ft) Age Weight (lbs)	5, 6, 7, etc. 1, 2, 3, etc. 10, 20, 30, etc.

In these notes, we will explore approaches to machine learning, with examples of problems and solutions using data analysis and data intelligence. Solutions are implemented in Python programming language. Python has been chosen because it a general-purpose programming language. It is open source meaning that it is available free of charge and allows access to programs and tools developed by other worldwide.

Anaconda Python Distribution is a good choice to download and install Python. It comes bundled with almost everything that you would need to start your data science journey. In addition to the core Python language, it includes many useful packages such as Numpy, Matplotlib, SciPy, Statsmodels, Pandas, Sklearn, and many more. Anaconda comes with an Integrated Development and Learning Environment (IDLE) with a built-in console for creating files of programs, running program in parts or completely, and provides a help facility.

Data values on such things as gender, marital status can be expressed in categories such as male, female for gender, and marital status as married, unmarried. For ease of processing in machine

learning they can be expressed as numerical such 1 for male, 2 for female, and likewise 1 married and 0 unmarried.

Consider the following artificially constructed trivial data example of using Pandas package.

```
import pandas as pd
df= pd.DataFrame({'A':['low','medium','high'], 'B':[10,20,30]}, index=[0,1,2])
print(df)
```

Data displayed by print statement is:

	A	B
0	low	10
1	medium	20
2	high	30

The data can factorized by assigning numeric values to categories of low, medium, and high as 0,1, 2.

```
df['A_pd_factorized'] = pd.factorize(df['A'])[0]
print (df)
```

	A	B	A_pd_factorized
0	low	10	0
1	medium	20	1
2	high	30	2

The factorization may be chosed as 10, 20, and 30 instead of 0, 1, and 2

Ex:

We can also convert textual data values to numbers by using the ‘Label Encoder’ function of Scikit-learn. If the number of levels is high (example zip code, state, etc.), then you may apply business logic to combine levels into groups. For example, a zip code or state can be combined to regions; however, in this method there is a risk of losing critical information. We may combine categories based on similar frequency (new category can be high, medium, low). Following are some examples of achieving this goal.

It is often desirable to normalize data i.e. transforming attribute values to roughly the same order of magnitude. Normalizing data can be achieved by Min-Max scaling, after removing the extreme outliers in the given data as follows.

$X_{normalized}=(X-X_{min})/(X_{max}-X_{min})$

The data may be standardized by making the values to have a zero mean.

Thus, $Z=(X-\mu)/\sigma$ where μ is the mean and σ is the standard deviation.

Generally, in machine learning exercises a large data set is used. Examples of such data sets can be found through the Internet.

Example 1

In this example, the data set known as IRIS from the UCI Machine Laboratory Repository at https://archive.ics.uci.edu/ml/datasets/iris. It is perhaps the best-known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

The data set attributes are:
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

Machine learning goal is to predict the class of iris plant.

In the details that follow, coding for each distinct step will be enclosed in a box. Comments on coding and the coding execution outcome will appear after the corresponding box.

Preparing the Environment for Coding

In these notes, coding statements are enclosed in a box for convenience. The coding environment uses pre-built relevant code in the form of available packages commonly known as Python libraries. These libraries provide standardized solutions for many problems that occur in

everyday programming. In Anaconda distribution of Python most of the libraries are included in the distribution installation. Selected libraries then simply imported as needed.

Package bought in the environment for this example are:
numpy: It is a library for array processing for numbers, strings, records, and objects.
sklearn (Scikit-learn) is a library for machine learning.

The import operations often assign a nickname for ease of use in coding. Here, the nickname used for numpy is np. This is a common practice.

```
from sklearn import datasets
import numpy as np
from sklearn import preprocessing
iris=datasets.load_iris()
x=iris.data[:,[2,3]]
y=iris.target
```

x is a matrix of two columns from iris data. The associated labels are petal length and petal width. y is the target array.
Petal Length and Petal width are selected as the control variables, x1 and x2. The target variable, y, consists of an array of zeroes, followed by ones, further followed by twos corresponding to categories of Iris Setosa, Iris Versicolour, and Iris Virginica.
We can display extracted data from iris for x and y, where x is a matrix of two columns of iris data with 150 rows and y is an array of the same size.

```
std_scale = preprocessing.StandardScaler().fit(x)
X_std = std_scale.transform(x)
x_std = std_scale.transform(x)
minmax_scale = preprocessing.MinMaxScaler().fit(x)
x_minmax = minmax_scale.transform(x)
```

```
print('Mean before standardization: petal length={:.1f}, petal width={:.1f}'
.format(x[:,0].mean(), x[:,1].mean()))
print('STD before standardization: petal length={:.1f}, petal width={:.1f}' .format(x[:,0].std(),
x[:,1].std()))
print('Mean after standardization: petal length={:.1f}, petal width={:.1f}'
.format(x_std[:,0].mean(), x_std[:,1].mean()))
print('STD after standardization: petal length={:.1f}, petal width={:.1f}'
.format(x_std[:,0].std(), x_std[:,1].std()))
```

Mean before standardization: petal length=3.8, petal width=1.2
STD before standardization: petal length=1.8, petal width=0.8
Mean after standardization: petal length=-0.0, petal width=-0.0
STD after standardization: petal length=1.0, petal width=1.0

The above printed outputs have some statistics of variables petal length and petal width data in iris before any standardization of the data.

```
print('\nMin value before min-max scaling: patel length={:.1f}, patel
width={:.1f}'.format(x[:,0].min(), x[:,1].min()))
print('Max value before min-max scaling: petal length={:.1f}, petal
width={:.1f}'.format(x[:,0].max(), x[:,1].max()))
print('Min value after min-max scaling: patel length={:.1f}, patel
width={:.1f}'.format(x_minmax[:,0].min(), x_minmax[:,1].min()))
print('Max value after min-max scaling: petal length={:.1f}, petal
width={:.1f}'.format(x_minmax[:,0].max(), x_minmax[:,1].max()))
```

Min value before min-max scaling: patel length=1.0, patel width=0.1
Max value before min-max scaling: petal length=6.9, petal width=2.5
Min value after min-max scaling: patel length=0.0, patel width=0.0
Max value after min-max scaling: petal length=1.0, petal width=1.0

The sklearn.preprocessing package provides several common utility functions to change raw feature vectors into a representation that is more suitable for the downstream estimators. In general, learning algorithms benefit from standardization of the data set. If some outliers are present in the set, robust scalers or transformers are more appropriate. The behaviors of the different scalers, transformers, and normalizers on a dataset containing marginal outliers is highlighted in Compare the effect of different scalers on data with outliers.

Feature Construction or Generation

Machine learning algorithms give best results only when we provide it the best possible features that structure the underlying form of the problem that you are trying to address. Often these features have to be manually created by spending a lot of time with actual raw data and trying to understand its relationship with all other data that you have collected to address a business problem.
It means thinking about aggregating, splitting, or combining features to create new features, or decomposing features. Often this part is talked about as an art form and is the key differentiator in competitive machine learning.

Feature construction is manual, slow, and requires subject-matter expert intervention heavily in order to create rich features that can be exposed to predictive modeling algorithms to produce best results. Summarizing the data is a fundamental technique to help us understand the data quality and issues/gaps. The following maps commonly use tabular and graphical data summarization methods for different data types.

Commonly Used Data Summarization Method			
Discrete/Qualitative		Quantitative/Quantitative	
Tabular Methods	Graphical Methods	Tabular Methods	Graphical Methods
• Frequency distribution • Relative frequency distribution • % Frequency distribution • Cross tabulation	• Bar Graph • Pie Chart • Other visualizations	• Frequency distribution • Relative frequency distribution • Cumulative Frequency distribution • Cumulative relative frequency distribution • Cross tabulation	• Line plot • Dot plot • Histogram • Scatter diagram

Exploratory Data Analysis (EDA)

In EDA understanding about the available data is done by summarizing and visualizing techniques. EDA allows both univariate and multivariate analysis. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important.

Example 2

Large sets of data are available from a variety of sources. We will make use of some of them. Consider, for example data on cars available from:
https://www.kaggle.com/CooperUnion/cardataset

We can import it for our work. This data contains more of 10, 000 rows and more than 10 columns which contains features of the car such as Engine Fuel Type, Engine HP, Transmission Type, highway MPG, city MPG and many more. We will explore data and make it ready to model machine learning.

The environment is adding pandas, seaborn, and matplotlib.

Seaborn is a statistical data visualization package, nicknamed sns.

Matplotlib provides publication quality figure, nicknamed plt.

```
import pandas as pd
import numpy as np
import seaborn as sns                    #visualisation
import matplotlib.pyplot as plt          #visualisation
%matplotlib inline
sns.set(color_codes=True)
df = pd.read_csv("data.csv")
```

%matplotlib inline sets the backend of matplotlib to the 'inline' backend. The output of plotting commands is displayed inline within frontends. The resulting plots can also be stored, copied and pasted.

The data frame, df is the name assigned to data loaded from the iris data worksheet. Some quick of loaded data is shown

```
df.count()
print(df.describe())
print(df.shape)
```

Count:

Make	11914
Model	11914
Year	11914
Engine Fuel Type	11911
Engine HP	11845
Engine Cylinders	11884
Transmission Type	11914
Driven_Wheels	11914
Number of Doors	11908
Market Category	8172
Vehicle Size	11914
Vehicle Style	11914
highway MPG	11914
city mpg	11914
Popularity	11914
MSRP	11914

dtype: int64

Description:

	Year	Engine HP	...	Popularity	MSRP
count	11914.000000	11845.00000	...	11914.000000	1.191400e+04

```
mean   2010.384338   249.38607  ...          1554.911197          4.059474e+04
std       7.579740   109.19187  ...          1441.855347          6.010910e+04
min    1990.000000    55.00000  ...             2.000000          2.000000e+03
25%    2007.000000   170.00000  ...           549.000000          2.100000e+04
50%    2015.000000   227.00000  ...          1385.000000          2.999500e+04
75%    2016.000000   300.00000  ...          2009.000000          4.223125e+04
max    2017.000000  1001.00000  ...          5657.000000          2.065902e+06
```

[8 rows x 8 columns]

Shape:
(11914, 16)

The call df.head(5) provides a partial view, five rows, and all sixteen columns, although not all are displayed.

```
print(df.head(5)) #print the top 5 rows
```

```
   Make       Model      Year  ...       city mpg  Popularity   MSRP
0  BMW    1 Series M  2011  ...      19        3916  46135
1  BMW    1 Series       2011  ...      19        3916  40650
2  BMW    1 Series       2011  ...      20        3916  36350
3  BMW    1 Series       2011  ...      18        3916  29450
4  BMW    1 Series       2011  ...      18        3916  34500
[5 rows x 16 columns]
```

The column labels in this database are:

1. Make
2. Model Year
3. Engine
4. Fuel Type
5. Engine HP
6. Engine Cylinders
7. Transmission Type
8. Driven_Wheels
9. Number of Doors
10. Market Category
11. Vehicle Size
12. Vehicle Style
13. highway MPG city
14. mpg
15. Popularity
16. MSRP

```
print(df.tail(5))
```

```
       Make    Model  Year  ... city mpg  Popularity   MSRP
11909  Acura    ZDX   2012  ...     16         204      46120
11910  Acura    ZDX   2012  ...     16         204      56670
11911  Acura    ZDX   2012  ...     16         204      50620
11912  Acura    ZDX    2013 ...      16          204     50920
11913  Lincoln  Zephyr 2006 ...      17          61      28995
[5 rows x 16 columns]
```

```
print(vars(df))
```

```
{'_is_copy': None, '_data': BlockManager
Items: Index(['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'],
      dtype='object')
Axis 1: RangeIndex(start=0, stop=11914, step=1)
FloatBlock: [4, 5, 8], 3 x 11914, dtype: float64
IntBlock: [2, 12, 13, 14, 15], 5 x 11914, dtype: int64
ObjectBlock: [0, 1, 3, 6, 7, 9, 10, 11], 8 x 11914, dtype: object, '_item_cache': {}, '_attrs': {}}
```

Selecting Attributes of Interest by Deleting Unwanted Columns for an EDA Session

```
df = df.drop(['Engine Fuel Type', 'Market Category', 'Vehicle Style', 'Popularity', 'Number of
Doors', 'Vehicle Size'], axis=1)
df.head(5)
```

```
Make        Model      Year  ... highway MPG  city mpg   MSRP
0  BMW  1 Series M   2011  ...      26        19  46135
1  BMW  1 Series     2011  ...      28        19  40650
2  BMW  1 Series      2011  ...     28        20  36350
3  BMW  1 Series      2011  ...     28        18  29450
4  BMW  1 Series     2011  ...      28        18  34500
[5 rows x 10 columns]
```

Of course, there is no observable difference in rows but observable the columns are different.

```
print(vars(df))
```

```
{'_is_copy': None, '_data': BlockManager
Items: Index(['Make', 'Model', 'Year', 'Engine HP', 'Engine Cylinders',
       'Transmission Type', 'Driven_Wheels', 'highway MPG', 'city mpg',
       'MSRP'],
      dtype='object')
```

Axis 1: RangeIndex(start=0, stop=11914, step=1)
ObjectBlock: [0, 1, 5, 6], 4 x 11914, dtype: object
IntBlock: [2, 7, 8, 9], 4 x 11914, dtype: int64
FloatBlock: slice(3, 5, 1), 2 x 11914, dtype: float64, '_item_cache': {}, '_attrs': {}}

The remaining columns are:

1. Make
2. Model
3. Year
4. Engine HP
5. Engine Cylinders
6. Transmission Type
7. Driven_Wheels
8. highway MPG
9. city mpg
10. MSRP

Renaming the Columns

If the attribute *column names), as it is in this instance, then change them in line with the current EDA.

```
df = df.rename(columns={"Engine HP": "HP", "Engine Cylinders": "Cylinders", "Transmission Type": "Transmission", "Driven_Wheels": "Drive Mode","highway MPG": "MPG-H", "city mpg": "MPG-C", "MSRP": "Price" })
df.head(5)
```

	Make	Model	Year	HP	…	Drive Mode	MPG-H	MPG-C	Price
0	BMW	1 Series M	2011	335.0	…	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	…	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	…	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	…	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	…	rear wheel drive	28	18	34500

[5 rows x 10 columns]

```
print(vars(df))
```

{'_is_copy': None, '_data': BlockManager
Items: Index(['Make', 'Model', 'Year', 'HP', 'Cylinders', 'Transmission',
'Drive Mode', 'MPG-H', 'MPG-C', 'Price'],
dtype='object')
Axis 1: RangeIndex(start=0, stop=11914, step=1)
ObjectBlock: [0, 1, 5, 6], 4 x 11914, dtype: object
IntBlock: [2, 7, 8, 9], 4 x 11914, dtype: int64

FloatBlock: slice(3, 5, 1), 2 x 11914, dtype: float64, '_item_cache': {}, '_attrs': {}}

Renamed columns can now be seen as:
1. Make
2. Model
3. Year
4. HP
5. Cylinders
6. Transmission
7. Drive Mode
8. MPG-H
9. MPG-C
10. Price

Dropping Duplicate Rows

A huge data set such as 10,000 records or more is like to have some duplicates as a cleanup operation.

```
df.shape
```

(11914, 10)

```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)

number of duplicate rows: (989, 10)

```
df.count()
```

df.count()

Make	11914
Model	11914
Year	11914
HP	11845
Cylinders	11884
Transmission	11914
Drive Mode	11914
MPG-H	11914
MPG-C	11914
Price	11914

dtype: int64

```
df = df.drop_duplicates()
```

From 11914 records, 989 have to be removed as duplicates.
[10925 rows x 10 columns]

```
df.count()
```

Make	10925
Model	10925
Year	10925
HP	10856
Cylinders	10895
Transmission	10925
Drive Mode	10925
MPG-H	10925
MPG-C	10925
Price	10925

dtype: int64

From 11914 records, 989 have to be removed as duplicates.
[10925 rows x 10 columns]

```
df.head(5)
```

Out[75]:

	Make	Model	Year	HP	...	Drive Mode	MPG-H	MPG-C	Price
0	BMW	1 Series M	2011	335.0	...	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	...	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	...	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	...	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	...	rear wheel drive	28	18	34500

[5 rows x 10 columns]

Dropping the Missing or Null values.

Similar to the previous step. Here all the missing values are detected and are dropped later, although some may replace missing values with the mean value.

```
print(df.isnull().sum())
```

```
Make          0
Model         0
Year          0
HP           69
Cylinders    30
Transmission  0
Drive Mode    0
MPG-H         0
MPG-C         0
Price         0
```

HP and Cylinders have null values. The corresponding rows are to be dropped.

```
df = df.dropna()    # Dropping the missing values.
df.count()
```

```
df = df.dropna()    # Dropping the missing values.
df.count()
Make          10827
Model         10827
Year          10827
HP            10827
Cylinders     10827
Transmission  10827
Drive Mode    10827
MPG-H         10827
MPG-C         10827
Price         10827
dtype: int64
```

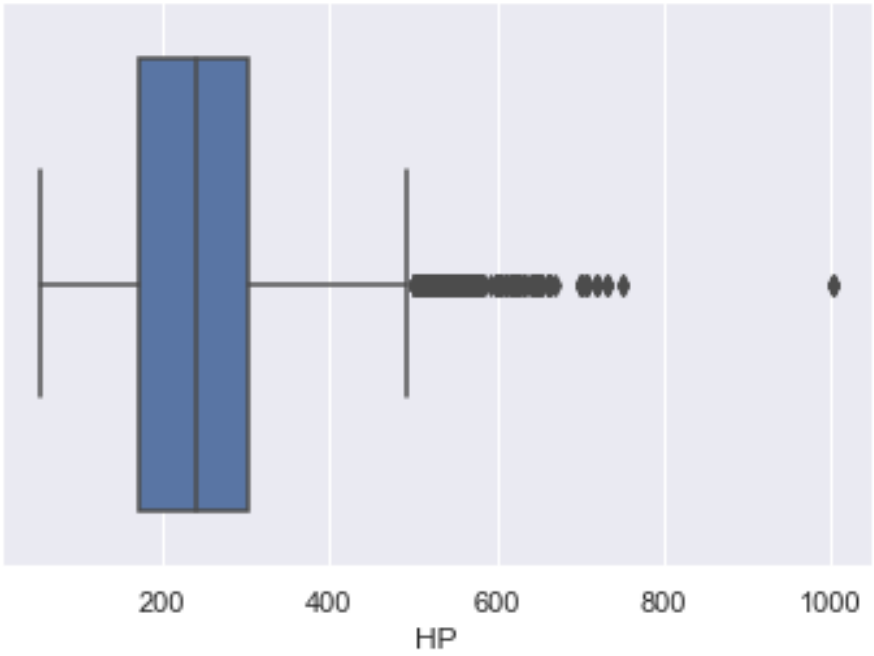
Now we have 10827 records remaining.

Detecting Outliers

An outlier is a point or set of points that are different from other points in terms of being very high or very low. It's often a good idea to detect and remove the outliers. Because outliers are one of the primary reasons for resulting in a less accurate model. The following example uses IQR score technique, https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba, for removing outliers. Often outliers can be seen with visualizations using a box plot. Shown below are the box plot of MSRP, Cylinders, Horsepower and EngineSize. Herein all the plots, you can find some points are outside the box they are none other than outliers.

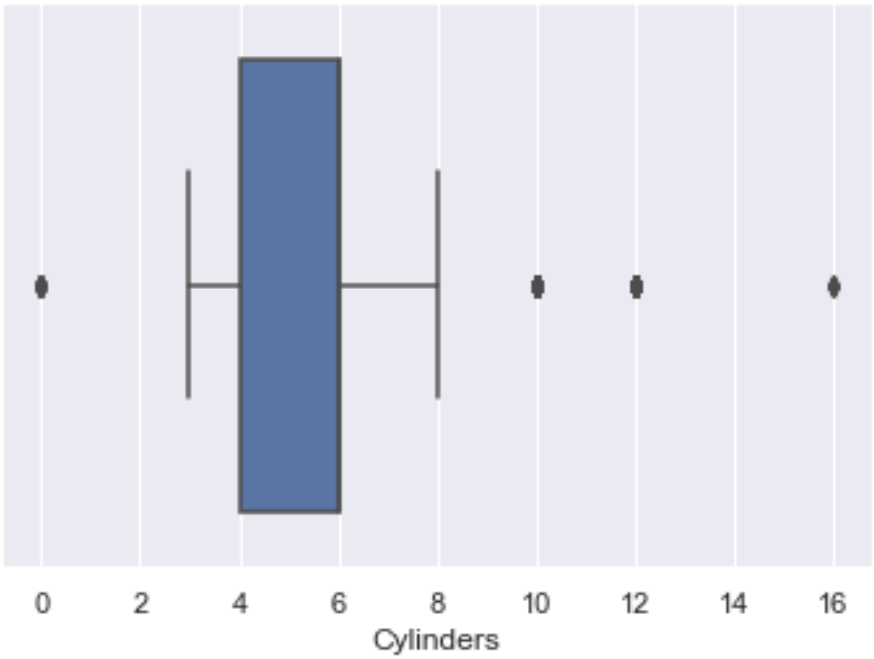
IQR (interquartile range) also called the midspread, middle 50%, or H-spread, is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles, IQR = $Q_3 - Q_1$.

```
sns.boxplot(x=df['HP'])
```




```
sns.boxplot(x=df['Cylinders'])
```

sns.boxplot(x=df['Cylinders'])




```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Year          9.0
HP          130.0
Cylinders     2.0
MPG-H         8.0
MPG-C         6.0
Price     21327.5
dtype: float64
```

Removing the outliers from the data:

```
df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape
```

(9191, 10)

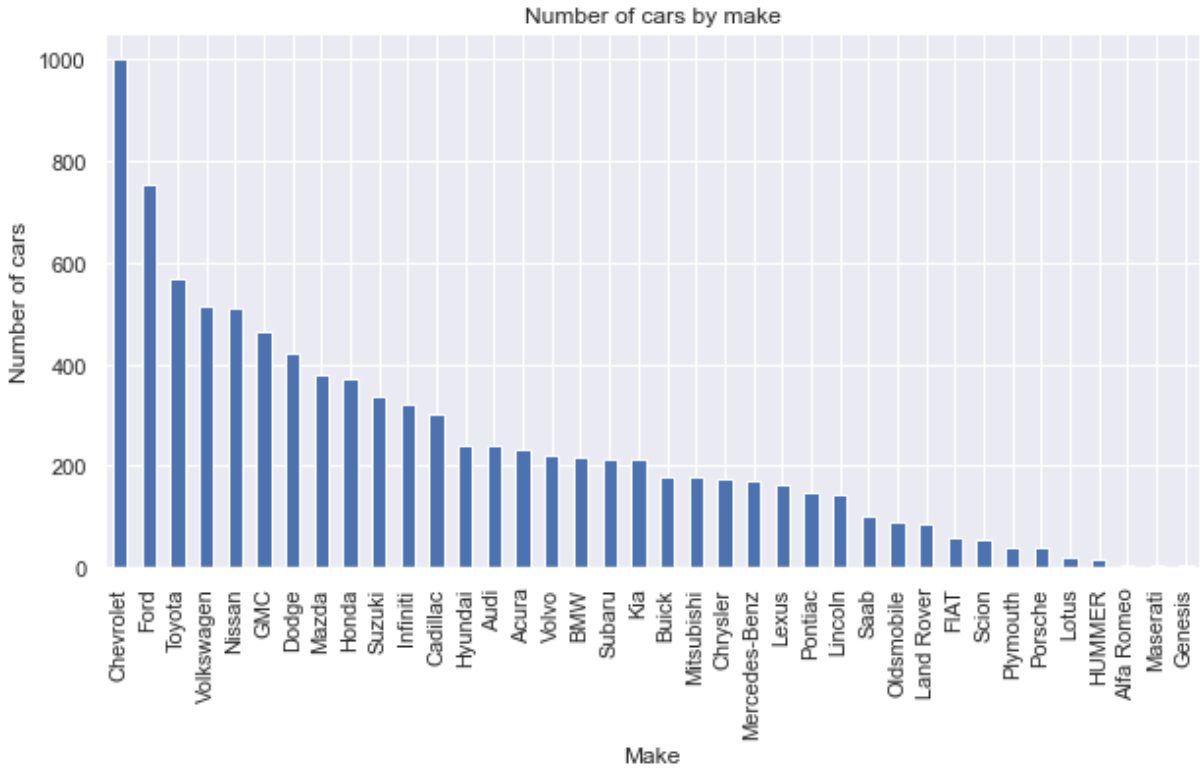
Now, we have 9191 records.

Plot Different Features against One Another (Scatter), against Frequency (Histogram)

Histogram refers to the frequency of occurrence of variables in an interval. In this case, there are mainly 10 different types of car manufacturing companies, but it is often important to know who

has the most number of cars. To do this histogram is one of the trivial solutions which lets us know the total number of cars manufactured by a different company.

```
df.Make.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title("Number of cars by make")
plt.ylabel('Number of cars')
plt.xlabel('Make');
```



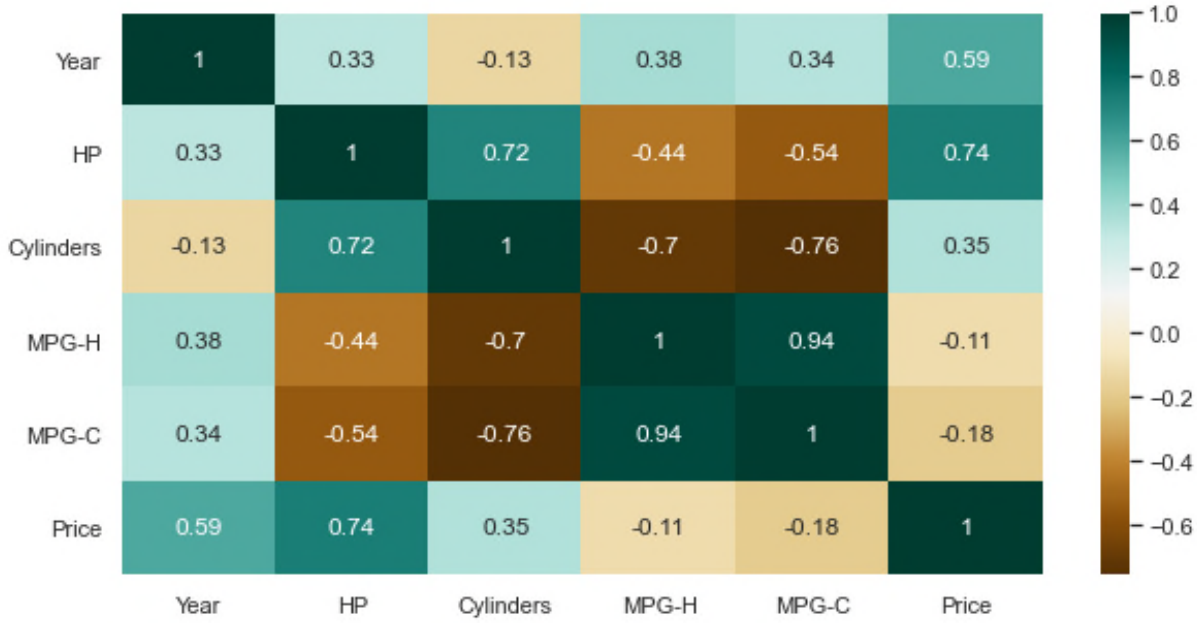
Heat Maps

Heat Maps is a type of plot which is necessary when we need to find the dependent variables. One of the best ways to find the relationship between the features can be done using heat maps. In the heat map figure shown below, we know that the price feature depends mainly on the Engine Size, Horsepower, and Cylinders.

```
plt.figure(figsize=(10,5))
c= df.corr()
sns.heatmap(c,cmap="BrBG",annot=True)
c
```

	Year	HP	Cylinders	MPG-H	MPG-C	Price
Year	1.000000	0.326726	-0.133920	0.378479	0.338145	0.592983
HP	0.326726	1.000000	0.715237	-0.443807	-0.544551	0.739042
Cylinders	-0.133920	0.715237	1.000000	-0.703856	-0.755540	0.354013
MPG-H	0.378479	-0.443807	-0.703856	1.000000	0.939141	-0.106320
MPG-C	0.338145	-0.544551	-0.755540	0.939141	1.000000	-0.180515
Price	0.592983	0.739042	0.354013	-0.106320	-0.180515	1.000000

The correlation coefficient values across the diagonal are all one naturally, an entity relates to itself perfectly. The next highest correlation is between MPG-H and MPG-C, which is quite reasonable. Negative correlation such between Cylinders and MPG-C mean when one goes up, the other goes down. The lowest correlation value is between Price and MPG-H which means one has hardly any effect on the other.



The heat map is a visualization of correlations.

Scatterplot

We generally use scatter plots to find the correlation between two variables. Here the scatter points are plotted between Horsepower and Price as show on the plot shown below. We can easily visualize a trend line. These features provide a good scattering of points.

```
fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(df['HP'], df['Price'])
```

```
ax.set_xlabel('HP')
ax.set_ylabel('Price')
plt.show()
```

The above examples show some of the steps involved in Exploratory Data Analysis (EDA).

There are many more. This is a good start on how to perform a good EDA given any data sets.

Useful Sources:

1. https://www.sciencedirect.com/topics/computer-science/supervised-learning
2. Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71. [Web Link]