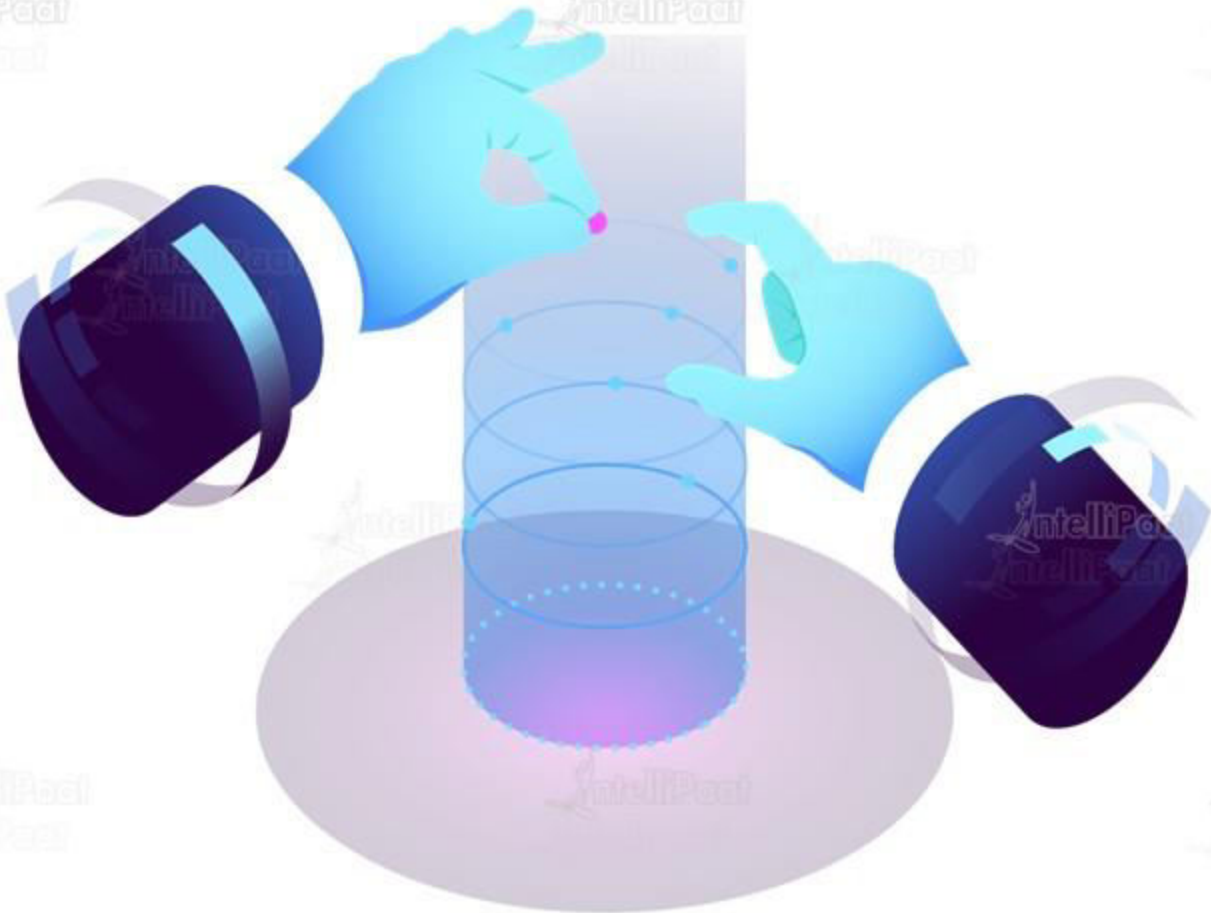
Data Science with Python

Data Manipulation



Agenda

01 Why Pandas?

02 Introduction to Pandas

03 Features of Pandas

04 Pandas vs NumPy

05 Data Structures in Pandas

06 Pandas Series Object and DataFrames

07 Merge, Join and Concatenate using DataFrames

08 Importing Datasets

09 Analyzing Datasets

10 Cleaning the Datasets

11 Manipulating the Dataset

12 Visualizing the Dataset

Why Pandas?

Why Pandas?

If you are using Python for Data Science, then Pandas is a very popular Library which is used for Manipulation and Analysis of data

You can achieve the same result writing 1-2 lines using Pandas compared to Native Python. There are also a lot of pre-defined functions which will reduce your time typing

Why Pandas?

Working with large datasets?

To clean the messy data and analyze & manipulate it according to your needs, Pandas is where you go.

Data Cleansing

Data analysis and Manipulation

Introduction to Pandas

Introduction to Pandas

Pandas is a simple yet powerful and open source data analysis and manipulation tool which is built on top of the Python language

According to the official Pandas Documentation, the points below is the vision of this library:

- Accessible to everyone
- Free for users to use and modify
- Flexible
- Powerful
- Easy to use
- Fast

Introduction to Pandas

The name Pandas is derived from **Panel Data**.

Panel Data is multi-dimensional data involving measurements over time

Person	Year	Income	Age	Sex
1	2013	20,000	23	F
1	2014	25,000	24	F
2	2013	35,000	27	M
2	2014	42,500	28	M
2	2015	50,000	29	M
3	2014	46,000	25	F

Panel Data

Introduction to Pandas

Who created Pandas?

Created in 2015 by Wes McKinney

Introduction to Pandas

Import the Pandas module ----->

```
import pandas as pd
```

Suitable data to use Pandas on ------>

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	X0	X1	X2	X3

Arbitrary Matrix

CUSTOMER		
NAME	DATATYPE	NULLABLE?
CUSTOMER_ID	VARCHAR	NO
FIRST_NAME	VARCHAR	NO
LAST_NAME	VARCHAR	NO
BIRTH_DAY	TIMESTAMP	NO
ADDRESS	VARCHAR	NO
ADDRESS2	VARCHAR	YES
STATE	VARCHAR	NO
ZIP_CODE	INTEGER	NO

Tabular data

	date	data
0	2018-01-01 00:00:00	52
1	2018-01-01 01:00:00	69
2	2018-01-01 02:00:00	23
3	2018-01-01 03:00:00	89
4	2018-01-01 04:00:00	53
5	2018-01-01 05:00:00	95
6	2018-01-01 06:00:00	19
7	2018-01-01 07:00:00	79
8	2018-01-01 08:00:00	33
9	2018-01-01 09:00:00	2
10	2018-01-01 10:00:00	0
11	2018-01-01 11:00:00	44
12	2018-01-01 12:00:00	45
13	2018-01-01 13:00:00	16
14	2018-01-01 14:00:00	38

Time Series Data

Features of Pandas

Features of Pandas

1. Series object and DataFrame
2. Handling of missing data
3. Data alignment
4. Group by functionality
5. Slicing, Indexing, Subseting
6. Merging, joining, reshaping
7. Hierarchical labeling of axes
8. Time series-specific functionality

Pandas vs NumPy

Pandas vs NumPy

1. Pandas performs better than numpy for 500k rows or more
2. Pandas Series Object is more flexible as you can define your own labeled index to index and access elements of an array

1. Numpy performs better for 50k rows or less
2. Elements in NumPy arrays are accessed by their default integer position

Data Structures in Pandas

Data Structures in Pandas

01 SERIES

- One-dimensional labelled array
- Can have any data type, but all elements of the array should be of same type

```
Out[4]: 0    1
        1    2
        2    3
        3    4
        dtype: int64
```

02 DATA FRAME

- Two-dimensional tabular structure
- Columns can hold different data types
- Size is mutable

	Player	Points	Title
0	Player1	8	Game1
1	Player2	9	Game2
2	Player3	5	Game3

03 PANEL

- Three-dimensional size mutable array
- Holds different data types
- Size is mutable

person	year	income	age	sex
1	2016	1600	23	1
1	2017	1500	24	1
2	2016	1900	41	2
2	2017	2000	42	2
2	2018	2100	43	2
3	2017	3300	34	1

Pandas Series Object and DataFrames

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

What is a Series Object?

It is a one-dimensional array which can contain the same or different data types. But in a series, the data types should be the same

Creating an empty Series:

```
empty = pd.Series()
print(empty)
```

```
Series([], dtype: float64)
```

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

Different ways of creating a Series

- An empty Series
- Converting an Array, List or a Dictionary to a Series
- From a Scalar value
- Using NumPy functions

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

Basic syntax of Series

Series([data, index, dtype])

- **data** – This is where you enter the data
- **index** – The index of the series
- **dtype** – Returns the data type object of the data

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

How to check the type?

```
type(ser)
```

```
pandas.core.series.Series
```

How to change the index name?

```
series = pd.Series(['1','2','3','4'],index=['a','b','c','d'])
print(series)
```

```
a    1
b    2
c    3
d    4
dtype: object
```

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

Few examples of Series

Converting an Array to a Series

```
arr = np.array(['p','y','t','h','o','n'])
ser = pd.Series(arr)
print(ser)
```

```
0    p
1    y
2    t
3    h
4    o
5    n
dtype: object
```

Dictionary to a Series

```
dict1 = {'py':100,
         'th':200,
         'on':300}
series1 = pd.Series(dict1)
print(series1)
```

```
py    100
th    200
on    300
dtype: int64
```

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

What is a DataFrame?

DataFrame is a two-dimensional, labelled data structure

- Can have columns with the same or different data types
- The size of a DataFrame is mutable
- Axes can be labelled which makes it easily readable
- Arithmetic operations on rows and columns are possible

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

Basic syntax of DataFrame

DataFrame([data, index, columns, dtype])

- **data** – This is where you enter the data
- **index** – The index of the series
- **columns** – Column labels of the DataFrame
- **dtype** – Returns the data type object of the data

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

Creating an empty DataFrame

```
dataf = pd.DataFrame()
print(dataf)
```

```
Empty DataFrame
Columns: []
Index: []
```

Checking the type

```
type(dataf)
```

```
pandas.core.frame.DataFrame
```

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

Different ways of creating a DataFrame

- Create an empty DataFrame
- Converting an Array, List, Dictionary or a Series to a DataFrame

```
dataf = pd.DataFrame(series1)
dataf
```

Converting a Series into a DataFrame →

	0
py	100
th	200
on	300

Pandas Series Object & DataFrames

Pandas Series

Pandas DataFrames

Few examples of Series

2D Array to a DataFrame

```
arr1 = [['joe',25], ['ashok',115], ['adbul',53]]
dataf1 = pd.DataFrame(arr1,columns=['Name','Age'])
dataf1
```

	Name	Age
0	joe	25
1	ashok	115
2	adbul	53

Dictionary to a DataFrame

```
dict2 = {'Show':['Brooklyn99','Breaking Bad','GOT'],
         'Rating':['8.4','9.5','9.3']}
dataf2 = pd.DataFrame(dict2)
dataf2
```

	Show	Rating
0	Brooklyn99	8.4
1	Breaking Bad	9.5
2	GOT	9.3

Demo: Pandas Series and Dataframe

Merge, Join and Concatenate using DataFrames

Merge, Join and Concatenate using DataFrames

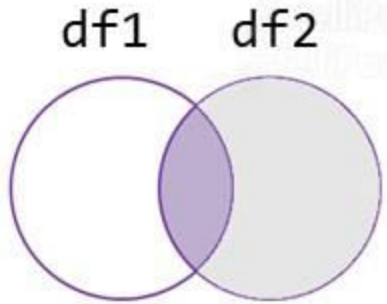
Merge and Join combines the given data into new columns.
Concatenation combines data in multiple DataFrames without any gaps

When you join/merge two DataFrames together, the df1 data is shown in one column alongside the df2's column in the same row

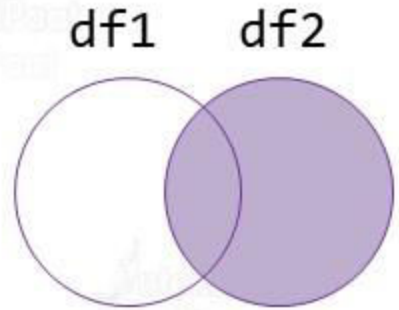
Function name
merge()
join()
concat()

Merge, Join and Concatenate using DataFrames

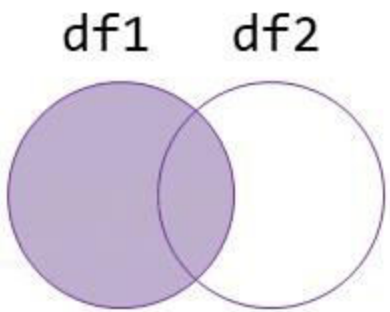
Types of Merges/Joins



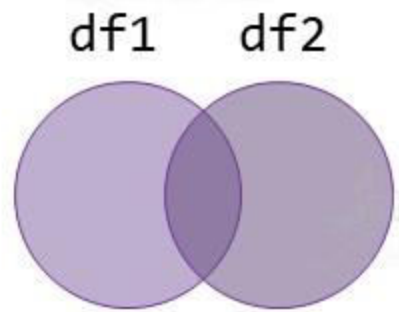
Inner Merge/
Inner join



Right Merge/
Right Join



Left Merge/
Left Join



Outer Merge/
Outer Join

Merge, Join and Concatenate using DataFrames

Merge Example

```
leftdf = pd.DataFrame({'Alphabet':['a','b','c','d'],
                       'number':['1','2','3','4'],
                       'alphanumeric':['a1','b2','c3','d4']})
```

```
rightdf = pd.DataFrame({'Alphabet':['a','b','c','d'],
                        'number':['5','6','7','8'],
                        'alphanumeric':['e5','f6','g7','h8']})
```

```
mergeddf = pd.merge(leftdf, rightdf, on='Alphabet')
mergeddf
```

	Alphabet	number_x	alphanumeric_x	number_y	alphanumeric_y
0	a	1	a1	5	e5
1	b	2	b2	6	f6
2	c	3	c3	7	g7
3	d	4	d4	8	h8

Merge, Join and Concatenate using DataFrames

Join Example

```
In [11]: df1 = pd.DataFrame({'key': ['0', '1', '2', '3'],
                             'A': ['A0', 'A1', 'A2', 'A3']})
         df2 = pd.DataFrame({'key': ['0', '1'],
                             'A': ['A0', 'A1',]})
         joined_df = df1.join(df2, lsuffix='_first_df', rsuffix='_second_df')
```

```
In [12]: joined_df
```

Out[12]:

	key_first_df	A_first_df	key_second_df	A_second_df
0	0	A0	0	A0
1	1	A1	1	A1
2	2	A2	NaN	NaN
3	3	A3	NaN	NaN

Merge, Join and Concatenate using DataFrames

Concatenate Example

```
import pandas as pd
```

```
first_df = pd.DataFrame({
    'name': ['Jane', 'John', 'Jacob'],
    'age': [15, 25, 25]
}, index=[0, 1, 2])
```

```
second_df = pd.DataFrame({
    'name': ['June', 'Julia', 'Julie'],
    'age': [28, 18, 20]
}, index=[3, 4, 5])
```

```
third_df = pd.DataFrame({
    'name': ['Jean', 'Jenny', 'Jessie'],
    'age': [17, 28, 30]
}, index=[6, 7, 8])
```

```
joined_df = pd.concat([first_df, second_df, third_df])
```

```
In [6]: joined_df
```

Out[6]:

	name	age
0	Jane	15
1	John	25
2	Jacob	25
3	June	28
4	Julia	18
5	Julie	20
6	Jean	17
7	Jenny	28
8	Jessie	30

Demo: Merge, Join and Concatenate using DataFrame

Importing Datasets

Importing Datasets

To start analyzing and manipulating data you need a dataset first. You can import datasets using pandas

Importing a CSV file

variable = pd.read_csv("filename.csv")

```
In [1]: #import pandas library
        import pandas as pd
        #raed dataset and store into a dataframe
        cars=pd.read_csv("mtcars2.csv")
        #print
        cars
```

Importing Datasets

Other format types and their read & write functions

Format Type	Description	Reader	Writer
Text	CSV	read_csv	to_csv
Text	HTML	read_html	to_html
Text	JSON	read_json	to_json
Binary	Python pickle format	read_pickle	to_pickle
Binary	MS Excel	read_excel	to_excel
SQL	SQL	read_sql	to_sql

Analysing Datasets

Analyzing Datasets

Once you have imported a dataset, you can start analyzing the DataFrame using the pre-defined functions available in Pandas

Getting the information about a DataFrame

```
df1.info(null_counts=True)
```

Using **null_counts=True** is to display all the information about every column available

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 13 columns):
S.No           32 non-null int64
Unnamed: 1     32 non-null object
mpg            32 non-null object
cyl            32 non-null int64
disp           32 non-null float64
hp             32 non-null int64
drat           32 non-null float64
wt             32 non-null float64
qsec           29 non-null float64
vs             32 non-null int64
am             32 non-null int64
gear           32 non-null int64
carb           32 non-null int64
dtypes: float64(4), int64(7), object(2)
memory usage: 3.3+ KB
```

Analyzing Datasets

IntelliPaat

Other Analysis functions

Function	Description
.count()	Non-null records in each column
.describe()	Descriptive statistical summary of the DataFrame
.shape	Number of rows and columns of the DataFrame
.mean()	Mean of the column
.median()	Median of the column
.std()	Standard deviation of the column
.min()	Minimum of each attribute (column)
.max()	Minimum of each attribute (column)

Analyzing Datasets

Few examples of Analyzing datasets

The number of rows and columns of a dataset

```
In [7]: #view number of rows and columns in the dataframe
        cars.shape

Out[7]: (32, 13)
```

Statistics Summary

```
In [60]: #descriptive statistics summary
         cars.describe()
```

Out[60]:

	S.No	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	29.000000	32.000000	32.000000	32.000000	32.0000
mean	16.500000	6.187500	230.721875	146.687500	3.596563	3.217250	17.674828	0.437500	0.406250	3.687500	2.8125
std	9.380832	1.785922	123.938694	68.562868	0.534679	0.978457	1.780394	0.504016	0.498991	0.737804	1.6152
min	1.000000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.0000
25%	8.750000	4.000000	120.825000	96.500000	3.080000	2.581250	16.870000	0.000000	0.000000	3.000000	2.0000
50%	16.500000	6.000000	196.300000	123.000000	3.695000	3.325000	17.420000	0.000000	0.000000	4.000000	2.0000
75%	24.250000	8.000000	326.000000	180.000000	3.920000	3.610000	18.600000	1.000000	1.000000	4.000000	4.0000
max	32.000000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8.0000

Mean of the Dataset

```
In [9]: #mean
        cars.mean()

Out[9]: S.No     16.500000
        cyl       6.187500
        disp    230.721875
        hp      146.687500
        drat      3.596563
        wt        3.217250
        qsec     17.674828
        vs        0.437500
        am        0.406250
        gear      3.687500
        carb      2.812500
        dtype: float64
```

Demo: Analysing Dataset

Cleaning the Datasets

Cleaning the Datasets

Data cleaning/cleansing is the process of removing unwanted or inaccurate records from a table or a dataset. Analysis made on clean data is more accurate.

Few pointers on Data cleansing

01 Treat all the Null or blank cells, fill them up with relevant data

02 Convert all the numbers stored as text to a number data type

03 Get rid of extra spaces which will be a problem while analyzing

04 Keep the Cases in check – Lowercase, Uppercase or Proper case

Cleaning the Datasets

Cleansing Functions

Function	Description
.rename()	Rename a column name
.fillna()	Fill the null or empty cells with the mean value
.drop()	Drop the mentioned column
.astype()	Change the data type of a column

Cleaning the Datasets

Few examples of Analyzing datasets

Changing the type of a column

```
In [66]: #change mpg from string to float
         cars.mpg = cars.mpg.astype(float)
         #see the change
         cars.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 13 columns):
S.No      32 non-null int64
model     32 non-null object
mpg       32 non-null float64
cyl       32 non-null int64
disp      32 non-null float64
hp        32 non-null int64
drat      32 non-null float64
wt        32 non-null float64
qsec      32 non-null float64
vs        32 non-null int64
am        32 non-null int64
gear      32 non-null int64
carb      32 non-null int64
dtypes: float64(5), int64(7), object(1)
memory usage: 3.3+ KB
```

Filling the empty cells with mean value

```
In [63]: #Fill the null values with mean of the column
         cars.qsec=cars.qsec.fillna(cars.qsec.mean())
         cars
```

Out[63]:

	S.No	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.460000	0	1	4	4
1	2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	1	4	4
2	3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.610000	1	1	4	1
3	4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.440000	1	0	3	1
4	5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.020000	0	0	3	2
5	6	Valiant	18.1	6	225.0	105	2.76	3.460	17.674828	1	0	3	1
6	7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.840000	0	0	3	4
7	8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.000000	1	0	4	2
8	9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.900000	1	0	4	2
9	10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.300000	1	0	4	4
10	11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.900000	1	0	4	4
11	12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.400000	0	0	3	3
12	13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.600000	0	0	3	3
13	14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.000000	0	0	3	3
14	15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.980000	0	0	3	4
15	16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.820000	0	0	3	4
16	17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.420000	0	0	3	4
17	18	Fiat 128	32.4	4	78.7	66	4.08	2.200	17.674828	1	1	4	1
18	19	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.520000	1	1	4	2
19	20	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.900000	1	1	4	1
20	21	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.010000	1	0	3	1
21	22	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.870000	0	0	3	2
22	23	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.300000	0	0	3	2
23	24	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.410000	0	0	3	4
24	25	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.050000	0	0	3	2

Cleaning the Datasets

Drop an unwanted Column

```
In [18]: #drop unwanted column
         cars = cars.drop(columns=['S.No'])
         cars
```

Before

Out[63]:

	S.No	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.460000	0	1	4	4
1	2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	1	4	4
2	3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.610000	1	1	4	1
3	4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.440000	1	0	3	1
4	5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.020000	0	0	3	2
5	6	Valiant	18.1	6	225.0	105	2.76	3.460	17.674828	1	0	3	1
6	7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.840000	0	0	3	4
7	8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.000000	1	0	4	2
8	9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.900000	1	0	4	2
9	10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.300000	1	0	4	4
10	11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.900000	1	0	4	4
11	12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.400000	0	0	3	3
12	13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.600000	0	0	3	3
13	14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.000000	0	0	3	3
14	15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.980000	0	0	3	4
15	16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.820000	0	0	3	4
16	17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.420000	0	0	3	4
17	18	Fiat 128	32.4	4	78.7	66	4.08	2.200	17.674828	1	1	4	1
18	19	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.520000	1	1	4	2
19	20	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.900000	1	1	4	1
20	21	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.010000	1	0	3	1
21	22	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.870000	0	0	3	2
22	23	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.300000	0	0	3	2
23	24	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.410000	0	0	3	4
24	25	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.050000	0	0	3	2

After

Out[18]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.460000	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.610000	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.440000	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.020000	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	17.674828	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.840000	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.000000	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.900000	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.300000	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.900000	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.400000	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.600000	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.000000	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.980000	0	0	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.820000	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.420000	0	0	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	17.674828	1	1	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.520000	1	1	4	2

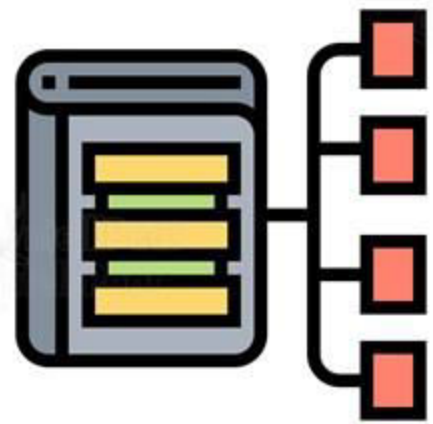
Demo: Cleaning Dataset

Manipulating the Dataset

Manipulating the Dataset

What is the need to manipulate data?

Any real world dataset will have millions of rows and 100s of columns. To make the dataset more readable and organized, you will have to manipulate it. For example, ascending order in accordance with the date.



Manipulating the Dataset

Different ways to manipulate datasets

01 Indexing by position

02 Indexing by label

03 Setting value

04 Applying function

05 Sorting

06 Filtering

Manipulating the Dataset

01 Indexing by position

Indexing by position is selecting a set of rows & columns using their index.
The below example is for selecting all rows:

```
In [25]: #all rows, all columns
         cars.iloc[:,:]
```

Out[25]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.460000	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.610000	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.440000	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.020000	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	17.674828	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.840000	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.000000	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.900000	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.300000	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.900000	1	0	4	4

Manipulating the Dataset

More examples of indexing by position

First 5 records of the 4th column

```
In [70]: #first five records of hp column
         cars.iloc[0:5,4]

Out[70]: 0    160.0
         1    160.0
         2    108.0
         3    258.0
         4    360.0
         Name: disp, dtype: float64
```

All rows of the 1st column

```
In [73]: #Now we want to look at all the rows and only the first column
         cars.iloc[:,1]

Out[73]: 0              Mazda RX4
         1          Mazda RX4 Wag
         2             Datsun 710
         3         Hornet 4 Drive
         4      Hornet Sportabout
         5                Valiant
         6             Duster 360
         7              Merc 240D
         8               Merc 230
         9               Merc 280
         10             Merc 280C
         11            Merc 450SE
         12            Merc 450SL
         13           Merc 450SLC
```

Manipulating the Dataset

02 Indexing by label

Indexing by label is selecting a set of rows & columns using the label of the rows and columns
The below example is for selecting the **mpg** column:

```
In [75]: #see all the record of mpg column
         cars.loc[:,"mpg"]
```

```
Out[75]: 0     21.0
         1     21.0
         2     22.8
         3     21.4
         4     18.7
         5     18.1
         6     14.3
         7     24.4
         8     22.8
         9     19.2
         10    17.8
         11    16.4
         12    17.3
         13    15.2
         14    10.4
         15    10.4
         16    14.7
         17    32.4
         18    30.4
         19    33.9
```

Manipulating the Dataset

More examples of indexing by label

Records of **mpg** column from 0 to 6

```
In [78]: #display the records from index 0 to index 6 from mpg column
         cars.loc[:6,"mpg"]

Out[78]: 0    21.0
         1    21.0
         2    22.8
         3    21.4
         4    18.7
         5    18.1
         6    14.3
         Name: mpg, dtype: float64
```

First 7 records from **mpg to qsec**

```
In [77]: #see the first 7 records from mpg to qsec column
         cars.loc[:6,"mpg":"qsec"]
```

Out[77]:

	mpg	cyl	disp	hp	drat	wt	qsec
0	21.0	6	160.0	110	3.90	2.620	16.460000
1	21.0	6	160.0	110	3.90	2.875	17.020000
2	22.8	4	108.0	93	3.85	2.320	18.610000
3	21.4	6	258.0	110	3.08	3.215	19.440000
4	18.7	8	360.0	175	3.15	3.440	17.020000
5	18.1	6	225.0	105	2.76	3.460	17.674828
6	14.3	8	360.0	245	3.21	3.570	15.840000

Manipulating the Dataset

03 Setting value

Setting a single value to one or more columns

```
In [31]: #set value 1 to column 'am'
         cars['am'] = 1
         cars
```

Out[31]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.460000	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.610000	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.440000	1	1	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.020000	0	1	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	17.674828	1	1	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.840000	0	1	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.000000	1	1	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.900000	1	1	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.300000	1	1	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.900000	1	1	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.400000	0	1	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.600000	0	1	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.000000	0	1	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.980000	0	1	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.820000	0	1	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.420000	0	1	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	17.674828	1	1	4	1

Manipulating the Dataset

04 Applying function

The apply() function can be used when you want to pass a function to an entire row or column

```
In [80]: #double up records in 'am' using lambda fxn
         f = lambda x: x*2
         cars['am']= cars['am'].apply(f)
         cars
```

Out[80]:

	S.No	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.460000	0	2	4	4
1	2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	2	4	4
2	3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.610000	1	2	4	1
3	4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.440000	1	0	3	1
4	5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.020000	0	0	3	2
5	6	Valiant	18.1	6	225.0	105	2.76	3.460	17.674828	1	0	3	1
6	7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.840000	0	0	3	4
7	8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.000000	1	0	4	2
8	9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.900000	1	0	4	2
9	10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.300000	1	0	4	4
10	11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.900000	1	0	4	4
11	12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.400000	0	0	3	3
12	13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.600000	0	0	3	3
13	14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.000000	0	0	3	3
14	15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.980000	0	0	3	4
15	16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.820000	0	0	3	4
16	17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.420000	0	0	3	4

Manipulating the Dataset

05 Sorting

You can sort the dataset in an ascending or a descending order in accordance to a column

Manipulating the Dataset

Sorting **cyl** in ascending order

```
In [33]: #sorting cyl column ascending order
         cars.sort_values(by='cyl')
```

Out[33]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.600000	1	2	4	2
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.610000	1	2	4	1
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.900000	1	2	5	2
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.700000	0	2	5	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	17.674828	1	2	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.010000	1	2	3	1
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.000000	1	2	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.900000	1	2	4	2
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.900000	1	2	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.520000	1	2	4	2
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	17.674828	1	2	4	1
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.500000	0	2	5	6
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.460000	0	2	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	2	4	4
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.440000	1	2	3	1
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.900000	1	2	4	4
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.300000	1	2	4	4
5	Valiant	18.1	6	225.0	105	2.76	3.460	17.674828	1	2	3	1
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.000000	0	2	3	3

Sorting **cyl** in descending order

```
In [79]: #sort cyl in descending order
         cars.sort_values(by='cyl', ascending=False)
```

Out[79]:

	S.No	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
16	17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.420000	0	0	3	4
30	31	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.600000	0	1	5	8
4	5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.020000	0	0	3	2
28	29	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.500000	0	1	5	4
6	7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.840000	0	0	3	4
24	25	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.050000	0	0	3	2
23	24	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.410000	0	0	3	4
22	23	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.300000	0	0	3	2
21	22	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.870000	0	0	3	2
11	12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.400000	0	0	3	3
12	13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.600000	0	0	3	3
13	14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.000000	0	0	3	3
14	15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.980000	0	0	3	4
15	16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.820000	0	0	3	4
1	2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.020000	0	1	4	4
29	30	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.500000	0	1	5	6

Manipulating the Dataset

06 Filtering

You can apply filters on certain columns and get the filtered values.
Below is an example of filtering:

```
In [37]: #filter records with more than 6 cyl and hp more than 300
         filter2 = (cars["cyl"] > 6) & (cars["hp"] > 300)
         #apply filter to dataframe
         filtered_review = cars[filter2]
         #display filtered data
         filtered_review
```

Out[37]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
30	Maserati Bora	15.0	8	301.0	335	3.54	3.57	14.6	0	2	5	8

Manipulating the Dataset

More than 6 cylinders or not

```
In [35]: #filter records with more than 6 cylinders
         cars['cyl'] > 6
```

```
Out[35]: 0     False
         1     False
         2     False
         3     False
         4      True
         5     False
         6      True
         7     False
         8     False
         9     False
         10    False
         11     True
         12     True
         13     True
         14     True
         15     True
         16     True
         17    False
         18    False
         19    False
         20    False
         21     True
         22     True
         23     True
         24     True
         25    False
         26    False
         27    False
         28     True
```

Print the rows with more than 6 cylinders

```
In [36]: #filter records with more than 6 cylinders
         filter1 = cars['cyl'] > 6
         #apply filter to dataframe
         filtered_new = cars[filter1]
         #view filtered dataframe
         filtered_new
```

Out[36]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	2	3	2
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	2	3	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	2	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	2	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	2	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	2	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	2	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	2	3	4
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	2	3	2
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	2	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	2	3	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	2	3	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	2	5	4
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	2	5	8

Demo: Manipulating Dataset

Visualizing Dataset

Visualizing the Dataset

To visualize a dataset, you will have to use matplotlib library. We will just look at the available plots in this module.

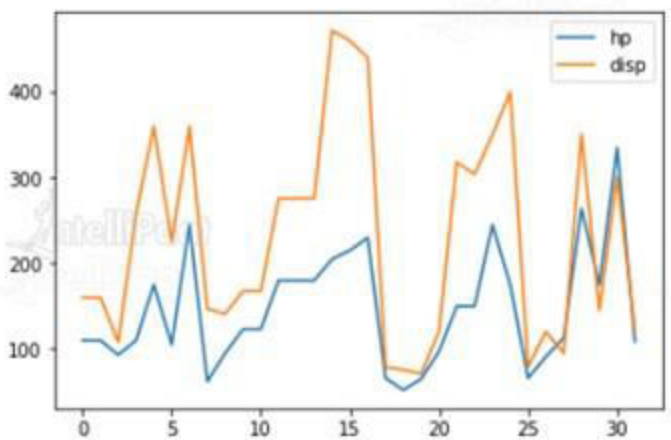
Once you have manipulated and analyzed the dataset, and you want to visualize the analyzed data, matplotlib is where you go.
Few possible plots using matplotlib:

- Line plot
- Area plot
- Line and Area plot
- Bar plot
- Horizontal Bar plot

Visualizing the Dataset

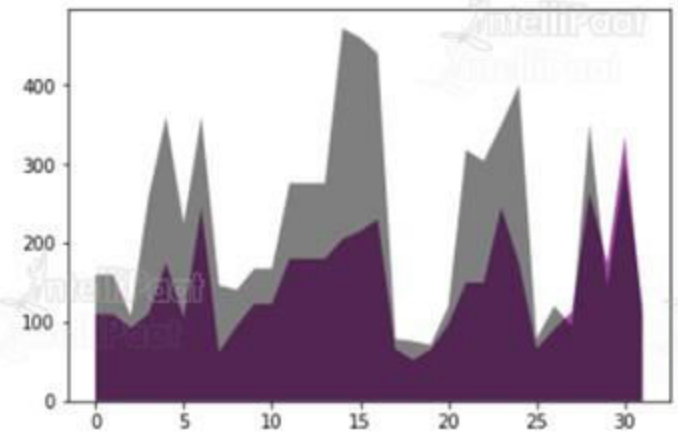
Line plot

```
In [4]: #import matplotlib
        import matplotlib.pyplot  as plt
        %matplotlib inline
        y1 = cars['hp']
        y2 = cars['disp']
        #see how both hp and disp varies
        x  = range(32)
        plt.plot(x,y1)
        plt.plot(x,y2)
        plt.legend()

Out[4]: <matplotlib.legend.Legend at 0x2da9cd1b470>
```



Area plot

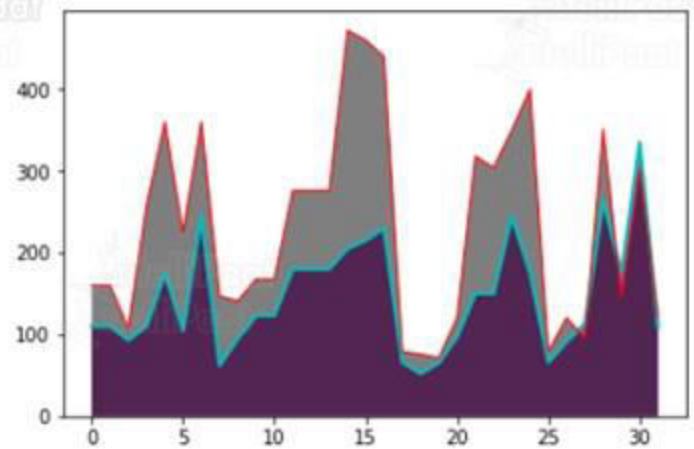
```
In [6]: #import matplotlib
        import matplotlib.pyplot  as plt
        %matplotlib inline
        y1 = cars['hp']
        y2 = cars['disp']
        x  = range(32)
        #area plot of hp and disp
        plt.stackplot(x,y1,colors = 'purple', alpha = 0.7)
        plt.stackplot(x,y2,colors = 'black', alpha = 0.5)

Out[6]: [<matplotlib.collections.PolyCollection at 0x2da9ce11940>]
```



Visualizing the Dataset

Area and Line plot

```
In [107]: import matplotlib.pyplot  as plt
          %matplotlib inline
          y1 = cars['hp']
          y2 = cars['disp']
          x  = range(32)
          #plot both line plot and area plot to see the margin
          plt.plot(x,y1, linewidth = 2.0, color = 'c')
          plt.stackplot(x,y1,colors = 'purple', alpha = 0.7)
          plt.plot(x,y2, linewidth = 1.0, color = 'r')
          plt.stackplot(x,y2,colors = 'black', alpha = 0.5)

Out[107]: [<matplotlib.collections.PolyCollection at 0x1cbfe64bd30>]
```

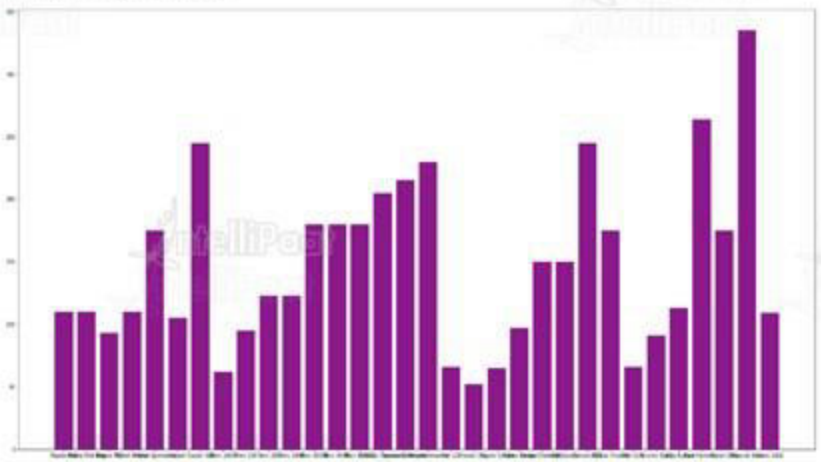


Visualizing the Dataset

Bar plot

```
In [121]: #import matplotlib
          import matplotlib.pyplot  as plt
          %matplotlib inline
          y = cars['hp']
          x  = range(32)
          #model to list
          x1 = cars['model'].tolist()
          #adding figure to adjust figsize
          fig = plt.figure(figsize = (30,15))
          #see how hp changes with bar plot
          plt.bar(x1,y,color="purple", alpha=0.9)
```

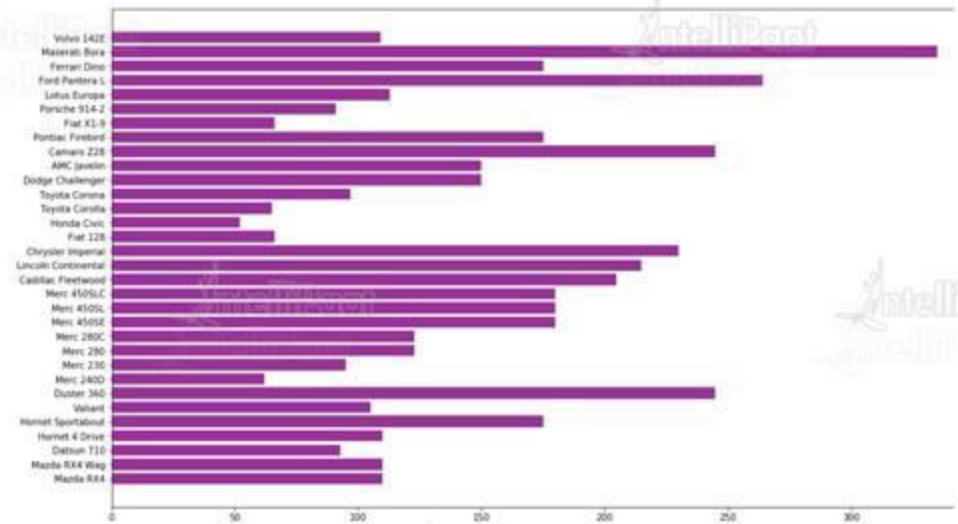
```
Out[121]: <BarContainer object of 32 artists>
```



Horizontal Bar plot

```
In [117]: #import matplotlib
          import matplotlib.pyplot  as plt
          %matplotlib inline
          y = cars['hp']
          x  = range(32)
          x1 = cars['model'].tolist()
          fig = plt.figure(figsize = (17,10))
          #to avoid the overlapping issue plot horizontal bar plot
          plt.barh(x1,y, color="purple", alpha=0.8)
```

```
Out[117]: <BarContainer object of 32 artists>
```



Demo: Visualizing Dataset

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor