Pandas → for data Analysis.

Why Pandas?

i) simple to use

ii) integrated with many other data science & ML Python tools

iii) Helps you get your data ready for machine Learning

Things to cover:

→ most useful functions

→ pandas Datatypes

→ Importing & Exporting data

→ Describing data

→ viewing & selecting data

→ Manipulating data

where to get help?

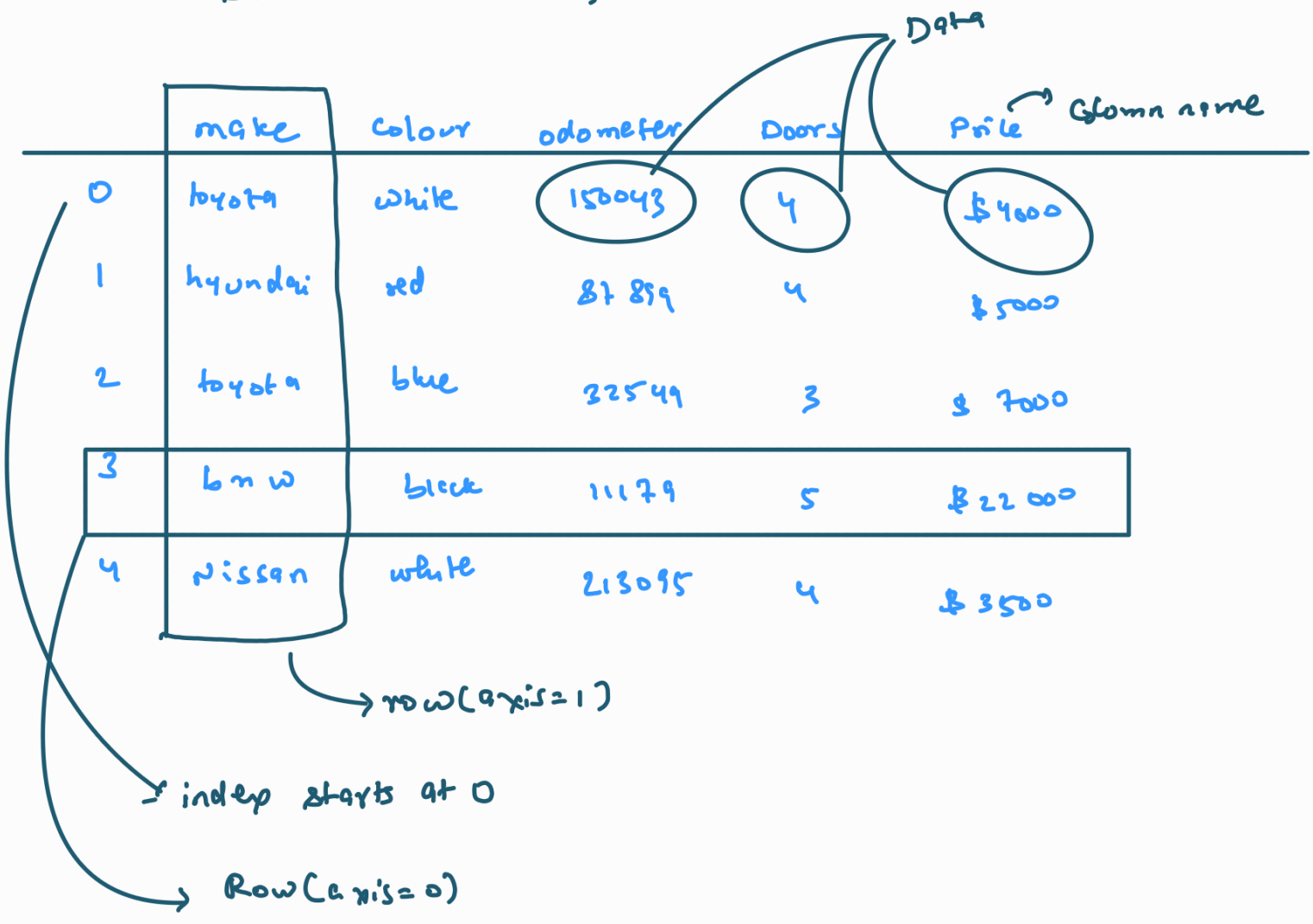
i) chatgpt

ii) Stack overflow

iii) pandas documentation

Series, Data frames and CSVs.

Pandas Data frame Anatomy



→ import pandas as pd

2 main data types.

i) **Series** = pd.Series(["bmw", "toyota", "honda"])

↳ 1-dimensional

Series

```
>>  0  bmw
    1  toyota
    2  honda
```

ii) Dataframe = pd.DataFrame({"key" : value , "key" : value})

↳ 2-dimensional

eg =

Colours = pd.Series(["red", "blue", "green"])

car_data = pd.DataFrame({"car_name" : series , "colour" : colour})

	car_name	colour
0	bmw	red
1	toyota	blue
2	honda	green

[row => axis=0
column => axis=1]

we can also create dataframes from csv, excels

→ import data

eg => df = pd.read_csv('car-sales.csv')

→ export data frame

→ df.to_excel('exported-car-sales.csv')

→ df.to_csv('exported_car_sales.csv', [index = false])

(by default it is True)

add an "unnamed" index col

importing data from URLs:

heart_disease = pd.read_csv("https://raw.URL")

↳ some csv url

Describing data with Pandas:

Attributes

i) .dtypes

eg → car_sales.dtypes

>> Make object
Colour object
odometer int64
Doors int64
Price object

ii) .columns

car_columns = car_sales.columns
car_columns

>> Index(['make', 'colour', 'odometer', 'Doors', 'price'],
dtype='object')

iii) index

car_index = car_sales.index
car_index

>> RangeIndex(start=0, stop=10, step=1)

iv) .shape → get dimensions

eg car_sales.shape()

→ (10,5

Functions

i) .to_csv()

ii) .describe()

functions

i) .describe()

car_sales.describe()

>>

	odometer	doors
count	10.0	10.0
mean	78.2	4.0
std	61.2	0.0
min	11	3.0
25%	35	4.0
50%	57	4.0
75%	96	4.0
Max	213	5.0

ii) .info() ⟶ (index combined with dtypes)

car_sales.info()

>> <class pandas.core.frame.DataFrame'>

RangeIndex : 10 entries 0 to 9

Data columns (total 5 columns):

#	Column	Non-null Count	Dtype
0	make	10 non-null	Object
1	Colour	10 non-null	object
2	odometer	10 non-null	int64
3	Doors	10 non null	int64
4	Price	10 non-null	object

dtypes : int64(2) , objects(3)

memory usage: 532.0+bytes

iii) .mean()

car_sales.mean()

>> Odometer : 78601.0

Doors : 4.0

dtype: float64

iv) .sum()

car_sales["Doors"].sum()

>> 40

v) .median()

vi) len() (no. of rows)

>> len(car_sales)

> 10

Selecting and viewing Data with Pandas.

i) .head([n])

↳ returns 5 top rows by default

ii) .tail([n])

↳ returns 5 last rows by default

iii) .loc , .iloc ⟶ integer based position (index)

↳ location label based

eg =

```
animals = pd.Series(["cat", "dog", "horse", "cow"],
                    index=[0, 1, 1, 2])

animals.loc[1]
```

>> 1 dog
1 horse

iloc

animals.iloc [1] → of index (ie 0, (1), 2)
cat dog horse

>> dog

we can slice with loc & iloc.

eg animals.iloc [:2]

>> ["cat", "dog"]

.loc slice

animals.loc [:1]

0 cat
1 dog
1 horse

---

selecting columns

dataframe ["ColumnName"] → preferred way

eg car_sales ["price"] OR car_sales.make
↳ causes issues if col-name has spaces.

selecting columns with filters

dataframe [dataframe ["column"] == "Toyota"]

eg car_sales [car_sales ["Make"] == "Toyota"]

>>

	make	Colour	odometer	Doors	Price
0	Toyota	white	1500	4	$ 4000
2	Toyota	blue	32	3	$ 7000
5	Toyota	green	99	4	$ 4500
8	Toyota	white	60	4	$ 6250

eg:-

```
car_sales[car_sales["odometer"] > 100]
```

>>

	make	Colour	odometer	Doors	Price
0	Toyota	white	1500	4	$ 4000

Regex :- regular expressions.

```
car_sales["Price"] = car_sales["price"].str.replace('[\$\,\.]', regex=True)
```

documentation:

pandas.Series.str.replace

.crosstab( ) ⟶ Compare two columns

```
p.crosstab(car_sales["Make"], car_sales["Doors"])
```

Doors	3	4	5
Make			
Bmw	0	0	1
Honda	0	3	0
Nissen	0	2	0
Toyota	1	3	0

.group by()

eg

```
car_sales.groupby(["Make"]).mean(numeric_only=True)
```

Make	odometer	doors
Bmw	111	5.00
Honda	627	4.00
Nissan	1123	4.00
Toyota	854	3.75

filtering groupby

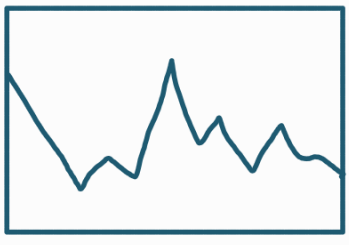
```
car_sales.groupby(["Make"]).mean(numeric_only=True).loc["Toyota"]
```

>>

```
odometer    854
Doors       3.75
Name: Toyota, dtype: float64
```

plot?

```
car_sales["odometer"].plot()
```



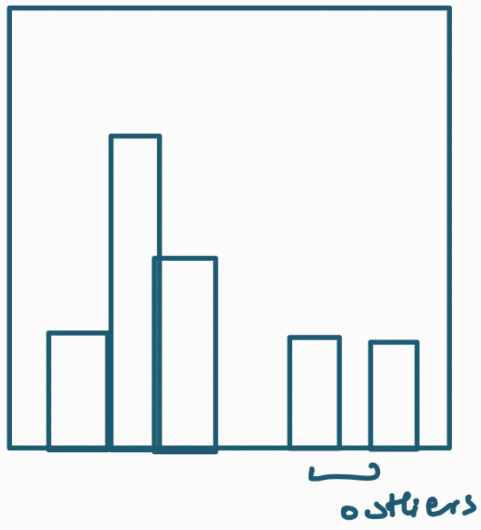
if matplot doesn't work

```
%matplotlib inline
import matplotlib.pyplot as plt
```

Histogram → a fancy word for Spread

eg

```
car_sales["odometer"].hist()
```



Manipulating Data:

i) .str.lower()

ii) .str.casefold()

iii) .str.upper()

iv) .fillna()

v) .dropna()

vi) .drop("Col_name", axis=1)

eg

```
car_sales["Make"] = car_sales["Make"].str.casefold()
```

eg: fill 0 inplace of Nan

→

```
car_sales["odometer"] = car_sales["odometer"].fillna(0.0).astype(float)
```

inplace parameter:

used to update data inplace instead of copying and re-assigning

eg 1- `car_sales["odometer"].fillna(0.0, inplace=True)` ~~.astype(float)~~

`car_sales.dropna(inplace=True)`

↳ removes all rows that has nan values

Create data from existing data.

Column from series

`seats_column = pd.Series([5,5,5,5,5])`

new column called seats

`car_sales["seats"] = seats_column`

`car_sales`

>>

	make	colour	odometer	Doors	Price	seats
0	toyota		100	4	1000	5
1	honda		99	4	2000	5
2	bmw		101	4	100000	5
3	honda		200	4	5000	5
4	nissan		100	4	3000	5

↳

`car_sales["seats"].fillna(5, inplace)`

inserting values into df with lists

eg

len(df) → 5

fuel_economy = [10, 15, 12, 13, 14]

car_sales["fuel economy"] = fuel_economy

car_sales

	make	colour	odometer	Doors	Price	Seats	fuel economy
0	toyota		100	4	1000	5	10
1	honda		99	4	2000	5	15
2	bmw		101	4	100000	5	12
3	honda		200	4	5000	5	13
4	nissan		100	4	3000	5	14

Creating a column from single value

eg

car_sales["wheels"] = 4

car_sales

>>

	make	colour	odometer	Doors	Price	Seats	fuel economy	wheels
0	toyota		100	4	1000	5	10	4
1	honda		99	4	2000	5	15	4
2	bmw		101	4	100000	5	12	4
3	honda		200	4	5000	5	13	4
4	nissan		100	4	3000	5	14	4

---

drop column:

car_sales = car_sales.drop("col_name", axis=1)

→ random samples

.sample (frac = 0-1)

eg :-

car_sales_shuffled = car_sales.sample (frac = 1)

only select 20% of data

car_sales_shuffled.sample (frac = 0.2)

→ reset index

car_sales_shuffled.reset_index (drop = False, inplace = True)

.apply() → applies a function to a column

car_sales ['odometer (km)'] = car_sales ["odometer (km)"].apply (lambda x : x / 1.6)

.rename → to rename columns

df = df.rename (column = {'odometer (km)': 'odometer (miles)'})
