DAY 14 **[10 - JULY - 2025]**

DATA PRE-PROCESSING

Introduction

There are various formats for a dataset, .csv, .json, .xlsx etc. The dataset can be stored in different places, on your local machine or sometimes online. In this section, you will learn how to load a dataset into our Jupyter Notebook. In our case, the Automobile Dataset is an online source, and it is in CSV (comma separated value) format. Let's use this dataset as an example to practice data reading.

- **Dataset name**: dataset_1.data

The Pandas Library is a useful tool that enables us to read various datasets into a data frame; our Jupyter notebook platforms have a built-in **Pandas Library** so that all we need to do is import Pandas without installing.

Importing Libraries

```
import pandas as pd
import numpy as np
```

Importing Datasets

We use pandas.read_csv() function to read the csv file. In the bracket, we put the file path along with a quotation mark, so that pandas will read the file into a data frame from that address. The file path can be either an URL or your local file address.

Name of Dataset is "dataset_1.data"

```
headers_data = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style",
        "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine-type",
        "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",
        "peak-rpm","city-mpg","highway-mpg","price"]
print("headers\n", type(headers_data))
```

Output:

```
headers
 <class 'list'>
# Read the dataset in "df" variable
df = pd.read_csv('Dataset_1.data', names=headers_data)
```

After reading the dataset, we can use the dataframe.head(n) method to check the top n rows of the dataframe; where n is an integer. Contrary to dataframe.head(n), dataframe.tail(n) will show you the bottom n rows of the dataframe.

# show the first 5 rows using dataframe.head() method
print("The first 5 rows of the dataframe")
df.head(5)

Output:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

# show the last 10 rows of the dataframe
df.tail(10)

Output:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
1	-1	74	vo	ga	std	fo	wa	rw	fro	10	.	141	mp	3.	3.1	9.5	114	54	2	28	13

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
95			lvo	s		ur	gon	d	nt	4.3	..		fi	78	5			00	3		415
196	-2	103	volvo	gas	std	four	sedan	rwd	front	104.3	...	141	mpfi	3.78	3.15	9.5	114	5400	24	28	15985
197	-1	74	volvo	gas	std	four	wagon	rwd	front	104.3	...	141	mpfi	3.78	3.15	9.5	114	5400	24	28	16515
198	-2	103	volvo	gas	turbo	four	sedan	rwd	front	104.3	...	130	mpfi	3.62	3.15	7.5	162	5100	17	22	18420
199	-1	74	volvo	gas	turbo	four	wagon	rwd	front	104.3	...	130	mpfi	3.62	3.15	7.5	162	5100	17	22	18950
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	23	28	16845
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	160	5300	19	25	19045
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	134	5500	18	23	21485
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	106	4800	26	27	22470
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	19	25	22625

Data Types

Data has a variety of types. The main types stored in Pandas dataframes are **object**, **float**, **int**, **bool**, and **datetime64**. In order to better learn about each attribute, it is always good for us to know the data type of each column. In Pandas:

Syntax:

```
dataframe.dtypes
```

Returns a Series with the data type of each column.

```
# check the data type of data frame "df" by .dtypes
print(df.dtypes)
```

Output:

```
symboling            int64
normalized-losses    object
make                 object
fuel-type            object
aspiration           object
num-of-doors         object
body-style           object
drive-wheels         object
engine-location      object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight          int64
engine-type          object
num-of-cylinders     object
engine-size          int64
fuel-system          object
bore                 object
stroke               object
compression-ratio    float64
horsepower           object
peak-rpm             object
city-mpg             int64
highway-mpg          int64
price                object
dtype: object
```

As a result, as shown above, it is clear to see that the data type of "symboling" and "curb-weight" are int64, "normalized-losses" is object, and "wheel-base" is float64, etc. These data types can be changed; we will learn how to accomplish this in a later module.

Dealing With Null Values with Scikit-Learn

Identify and Handle Missing Values

Identify Missing Values

Convert "?" to NaN

In the car dataset, missing data comes with the question mark "?". We replace "?" with NaN (Not a Number), which is Python's default missing value marker, for reasons of computational speed and convenience. Here we use the function:

```
.replace(A, B, inplace=True)
```

to replace A by B.

```
# replace "?" to NaN
df.replace("?", np.nan, inplace=True)
df.head(5)
```

Output:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

Note: The notebook appears to have a section where the dataset columns are reassigned to a different dataset with columns ['Australia', 'Canada', 'Dubai', 'USA', 'Salary', 'YearsExperience', 'Purchased']. This seems inconsistent with the earlier automobile dataset. For clarity, I will include this section as it appears in the notebook, assuming it is part of the intended content.

```
df.columns = ['Australia','Canada','Dubai','USA','Salary', 'YearsExperience', 'Purchased']
df.head(10)
```

Output:

	Australia	Canada	Dubai	USA	Salary	YearsExperience	Purchased
0	0.0	0.0	1.0	0.0	39343.0	1.1	0.0
1	0.0	1.0	0.0	0.0	46205.0	1.3	1.0

	Australia	Canada	Dubai	USA	Salary	YearsExperience	Purchased
2	0.0	1.0	0.0	0.0	37731.0	1.5	0.0
3	0.0	1.0	0.0	0.0	43525.0	2.0	0.0
4	0.0	0.0	0.0	1.0	39891.0	2.2	0.0
5	0.0	0.0	1.0	0.0	56642.0	2.9	0.0
6	0.0	1.0	0.0	0.0	60150.0	3.0	1.0
7	1.0	0.0	0.0	0.0	54445.0	3.2	0.0
8	0.0	0.0	1.0	0.0	64445.0	3.2	1.0
9	0.0	0.0	1.0	0.0	57189.0	3.7	0.0

```
# You can find the sorted list of column names of encoding variable by this method.
print(sorted(list(dataset['country'].unique())))
# To find the name of all the columns of dataset.
dataset.columns
```

Output:

```
['Australia', 'Canada', 'Dubai', 'USA']
Index(['country', 'Salary', 'YearsExperience', 'Purchased'], dtype='object')
```

Splitting of Dataset

Dividing into Target and Predictor Variables

```
# In X we store predictor variables.
# Here X will be dataframe because it has more than one features(columns).
X = df.iloc[:,:-1]
# In y we store target variables.
# Here y will be pandas series as it has only one feature(column).
y = df.iloc[:,-1]

# Here we are using X,y as dataframe because it has nice representation so you can understand it without confusion.
# But you can also use X,y as numpy array by this method.
# X = df.iloc[:,:-1].values
# y = df.iloc[:,-1].values
```

Divide Data as Training Set and Testing Set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=90)
```

X_train Output:

	Australia	Canada	Dubai	USA	Salary	YearsExperience
29	0	1	0	0	121872	10.5
6	0	1	0	0	60150	3.0
20	0	1	0	0	91738	6.8
23	1	0	0	0	113812	8.2
25	1	0	0	0	105582	9.0

	Australia	Canada	Dubai	USA	Salary	YearsExperience
...	...	...	...	...	...	...

X_test Output:

	Australia	Canada	Dubai	USA	Salary	YearsExperience
17	0	1	0	0	83088	5.3
24	0	0	1	0	109431	8.7
26	0	0	1	0	116969	9.5
2	0	1	0	0	37731	1.5
1	0	1	0	0	46205	1.3
12	0	0	1	0	56957	4.0

y_train Output:

```
29    1
6     1
20    0
23    1
25    1
...
Name: Purchased, dtype: int64
```

y_test Output:

```
17    0
24    0
26    0
2     0
1     1
12    1
Name: Purchased, dtype: int64
```

Feature Scaling

Splitting Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Perform Feature Scaling

```
# Importing StandardScaler
from sklearn.preprocessing import StandardScaler
# Creating Instance of StandardScaler
sc = StandardScaler()
# Perform scaling in X_train with fit_transform.
# Here we are applying fit_transform because,
#       fit will calculate mean and standard deviation of X_train
#       transform will actually perform scaling with calculated mean and std.
#       fit_transform method does this both thing in one line of code.
```

```
X_train.iloc[:, 4:] = sc.fit_transform(X_train.iloc[:, 4:])
# Here we will only use transform because we have already calculated mean and std.
# Another reason is we don't want to know the mean and std of our test dataset As it
# Lead to information leakage.
X_test.iloc[:, 4:] = sc.transform(X_test.iloc[:, 4:])
```

Check It

X_train Output:

	Australia	Canada	Dubai	USA	Salary	YearsExperience
26	0	0	1	0	1.461305	1.391080
3	0	1	0	0	-1.067603	-1.058963
24	0	0	1	0	1.201748	1.129742
22	1	0	0	0	0.921841	0.868404
23	1	0	0	0	1.352600	0.966406
...	...	...	...	...	...	...

X_test Output:

	Australia	Canada	Dubai	USA	Salary	YearsExperience
17	0	1	0	0	0.294676	0.019056
21	1	0	0	0	0.817543	0.607066
10	0	0	1	0	-0.389511	-0.438286
19	0	1	0	0	0.668344	0.247727
14	1	0	0	0	-0.462061	-0.242282
20	0	1	0	0	0.592523	0.509065