

Fundamentals of Machine Learning

Lab Assignment-8

Name: Hitha Choudhary G

USN: 22BTRAD015

Branch: CSE in AI&DE

Implementing Pandas

- Pandas is a Python library used for working with data sets.
- It has functions for analysing, cleaning, exploring, and manipulating data.
- Pandas allows us to analyse big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.
- The data produced by Pandas are often used as input for plotting functions of Matplotlib, statistical analysis in SciPy, and machine learning algorithms in Scikit-learn.
- Here is a list of things that we can do using Pandas.

Creating a Series: A one-dimensional array capable of storing a variety of data types is how it is defined. The term "index" refers to the row labels of a series.

In the below code we imported pandas as pd and created a series using “pd.Series()”.

Code:

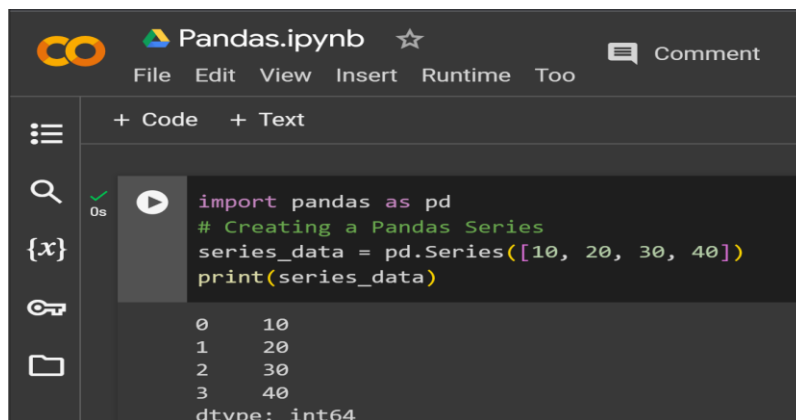
```
import pandas as pd
```

```
# Creating a Pandas Series
```

```
series_data = pd.Series([10, 20, 30, 40])
```

```
print(series_data)
```

Output:



The screenshot shows a Jupyter Notebook titled "Pandas.ipynb". The code cell contains the following Python code:

```
import pandas as pd
# Creating a Pandas Series
series_data = pd.Series([10, 20, 30, 40])
print(series_data)
```

The output of the code is displayed below the code cell:

```
0    10
1    20
2    30
3    40
dtype: int64
```

Creating a DataFrame: As a standard method for storing data, DataFrame has two distinct indexes-row index and column index. It has the following characteristics: It can be thought of as a series structure dictionary with indexed rows and columns. It is referred to as "columns" for rows and "index" for columns.

In the below code we are creating a dictionary which is later represented as a DataFrame using "pd.DataFrame()".

Code:

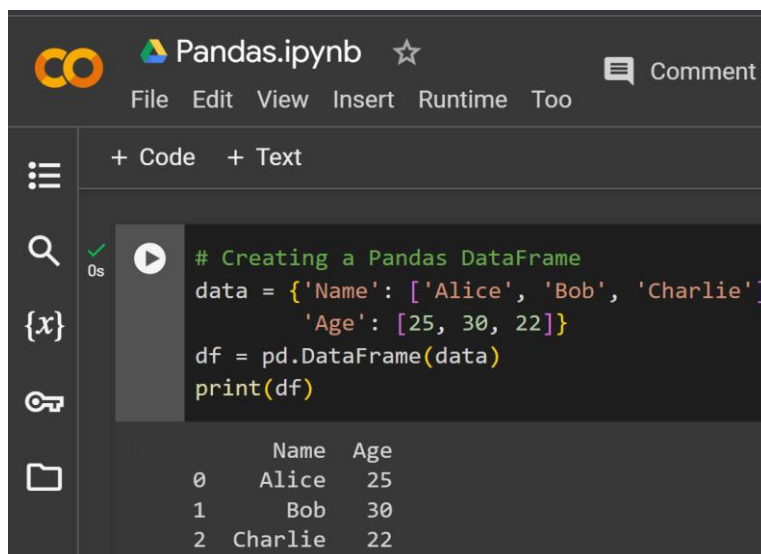
```
# Creating a Pandas DataFrame
```

```
data = {'Name': ['Alice', 'Bob', 'Charlie'],  
        'Age': [25, 30, 22]}
```

```
df = pd.DataFrame(data)
```

```
print(df)
```

Output:



The screenshot shows a Jupyter Notebook titled "Pandas.ipynb". The code cell contains the following Python code:

```
# Creating a Pandas DataFrame  
data = {'Name': ['Alice', 'Bob', 'Charlie'],  
        'Age': [25, 30, 22]}  
df = pd.DataFrame(data)  
print(df)
```

The output of the code is a DataFrame with three rows and two columns:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	22

Operations in a DataFrame: Selecting a particular column by label and selecting rows based on a condition.

Code:

```
# Selecting a column by label
```

```
ages = df['Age']
```

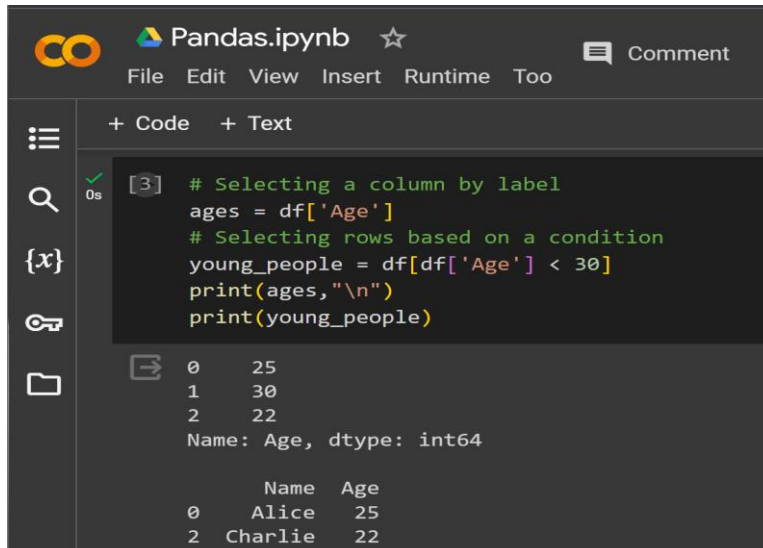
```
# Selecting rows based on a condition
```

```
young_people = df[df['Age'] < 30]
```

```
print(ages, "\n")
```

```
print(young_people)
```

Output:



```
# Selecting a column by label
ages = df['Age']
# Selecting rows based on a condition
young_people = df[df['Age'] < 30]
print(ages, "\n")
print(young_people)
```

0 25
1 30
2 22
Name: Age, dtype: int64

	Name	Age
0	Alice	25
2	Charlie	22

In the above code we are accessing the column age from the above mentioned dataframe and we are also filtering the age < 30 and storing it in the variable “young_people”.

Handling Missing Values in a DF: We use “df.dropna()” to find out the missing values if any present in the DataFrame. We also use “df.fillna()” to fill in a missing value.

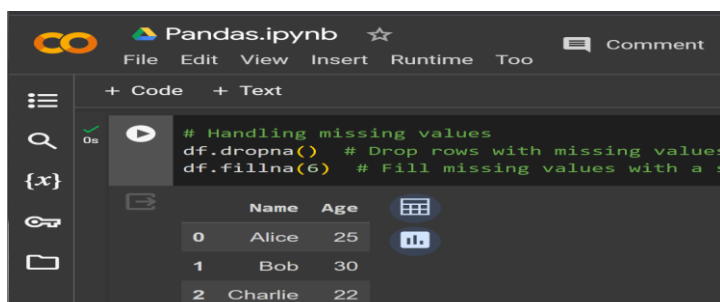
Code:

```
# Handling missing values
```

```
df.dropna() # Drop rows with missing values
```

```
df.fillna(6) # Fill missing values with a specified value
```

Output:



```
# Handling missing values
df.dropna() # Drop rows with missing values
df.fillna(6) # Fill missing values with a specified value
```

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	22

Since the above DataFrame does not contain any missing values there are no changes made.

Transformation: We can transform a particular column in the dataframe. For example in the age column we can transform the column by incrementing all the age values by 1.

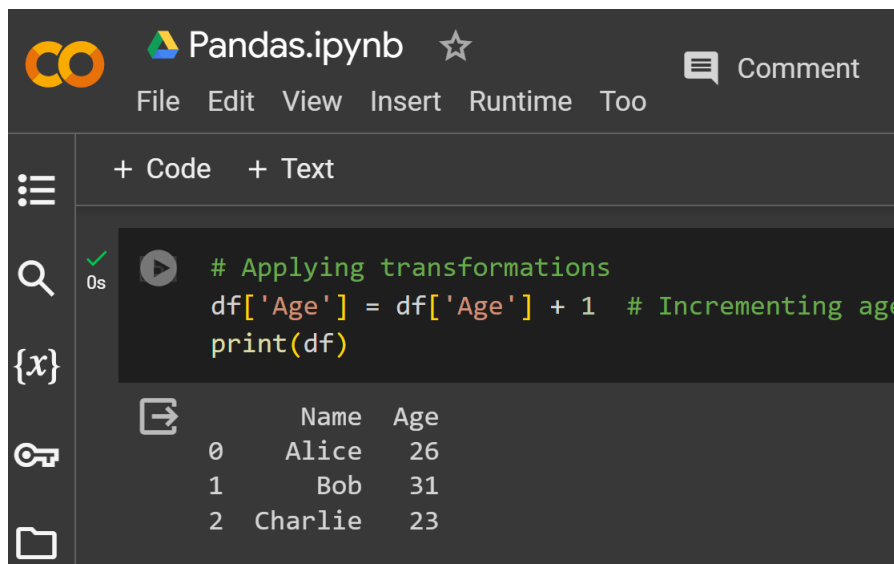
Code:

```
# Applying transformations
```

```
df['Age'] = df['Age'] + 1 # Incrementing ages by 1
```

```
print(df)
```

Output:

The screenshot shows a Jupyter Notebook interface with a dark theme. At the top, there's a header with the 'CO' logo, 'Pandas.ipynb', a star icon, and a 'Comment' button. Below the header is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', and 'Tools'. A sidebar on the left contains icons for a menu, search, variables, keys, and files. The main area shows a code cell with a play button icon, a green checkmark, and a '0s' timer. The code in the cell is:

```
# Applying transformations
df['Age'] = df['Age'] + 1 # Incrementing ages by 1
print(df)
```

 Below the code, the output is displayed as a table with three columns: an index column, 'Name', and 'Age'. The data rows are: 0 Alice 26, 1 Bob 31, and 2 Charlie 23.

	Name	Age
0	Alice	26
1	Bob	31
2	Charlie	23

GroupBy: We can group different columns and calculate the mean of it by using “df.groupby()” for grouping and “.mean()” for finding the mean.

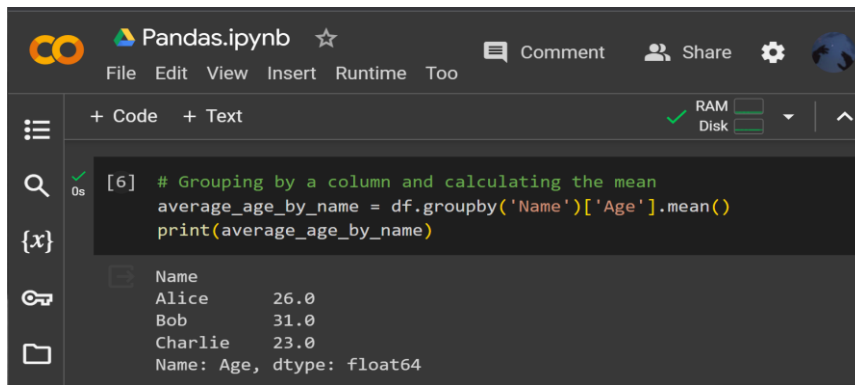
Code:

```
# Grouping by a column and calculating the mean
```

```
average_age_by_name = df.groupby('Name')['Age'].mean()
```

```
print(average_age_by_name)
```

Output:



The screenshot shows a Jupyter Notebook interface with the title 'Pandas.ipynb'. The code cell contains the following Python code:

```
[6] # Grouping by a column and calculating the mean
average_age_by_name = df.groupby('Name')['Age'].mean()
print(average_age_by_name)
```

The output of the code is displayed below the code cell:

```
Name
Alice      26.0
Bob        31.0
Charlie    23.0
Name: Age, dtype: float64
```

We are grouping the columns “name” and “age” and finding the mean of it. The result is shown above.

Read CSV files using Pandas: A simple way to store big data sets is to use CSV files (comma separated files). CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

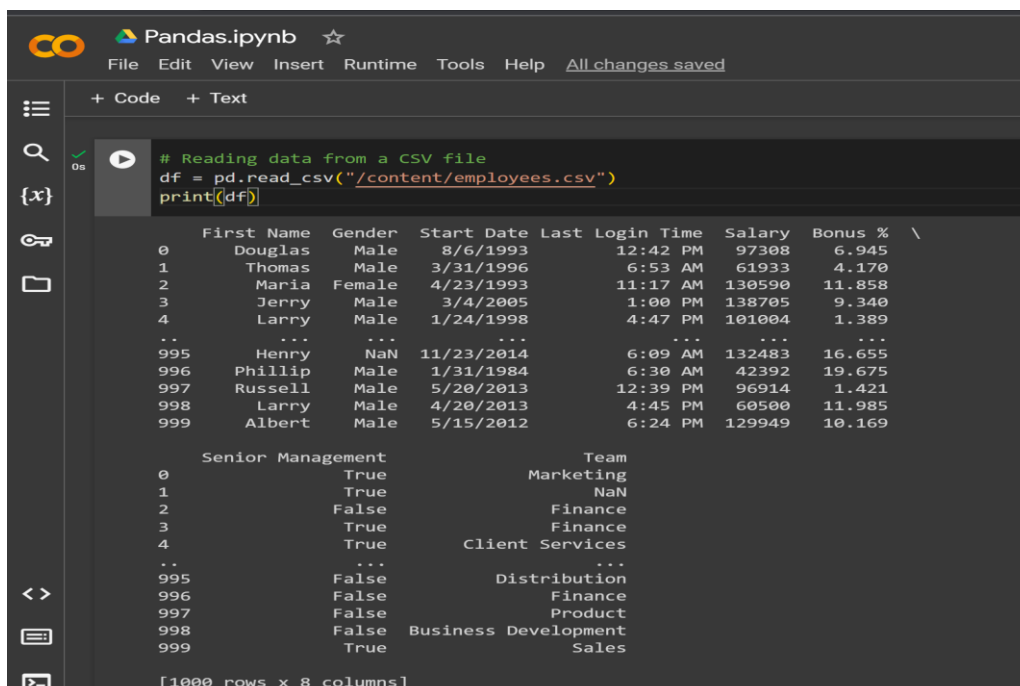
Code:

```
# Reading data from a CSV file

df = pd.read_csv("/content/employees.csv")

print(df)
```

Output:



The screenshot shows a Jupyter Notebook interface with the title 'Pandas.ipynb'. The code cell contains the following Python code:

```
# Reading data from a CSV file
df = pd.read_csv("/content/employees.csv")
print(df)
```

The output of the code is displayed below the code cell, showing a DataFrame with 1000 rows and 8 columns. The columns are: First Name, Gender, Start Date, Last Login Time, Salary, Bonus %, and \. The output is truncated to show the first 10 rows and the last 10 rows.

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	\
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	
...
995	Henry	NaN	11/23/2014	6:09 AM	132483	16.655	
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	

The output is truncated to show the first 10 rows and the last 10 rows. The last row of the output is:

```
[1000 rows x 8 columns]
```

Printing the first five rows from the csv file: Using “df.head()” we can print only the first five rows of data from the CSV file.

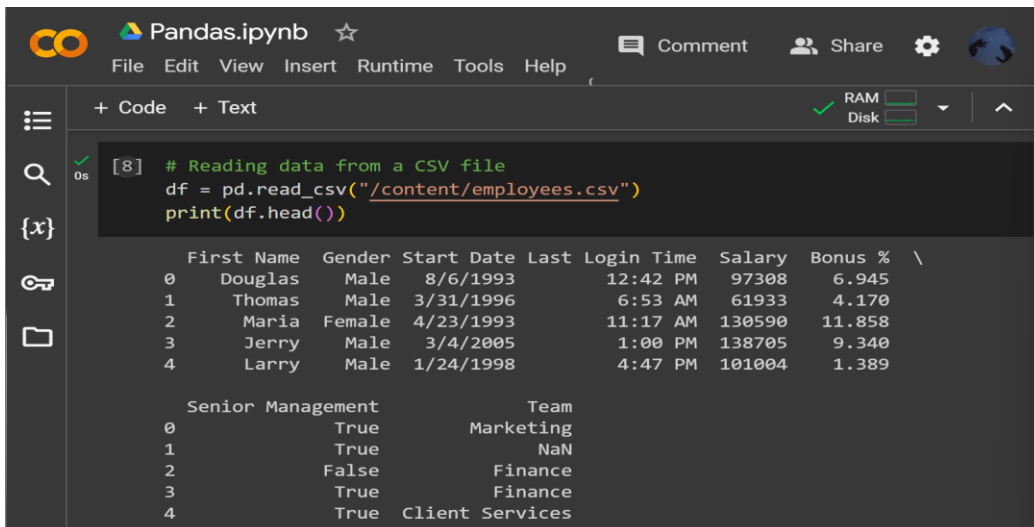
Code:

```
# Reading data from a CSV file
```

```
df = pd.read_csv("/content/employees.csv")
```

```
print(df.head())
```

Output:



	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	\
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	

	Senior Management	Team
0	True	Marketing
1	True	NaN
2	False	Finance
3	True	Finance
4	True	Client Services

Github Link: <https://github.com/hithachoudhary/machinelearning>