In [ ]:
```
                                DATA SCIENCE - BASIC COMMANDS
                                    SKILL ASSESSMENT 1
```

In [ ]:
```
NAME  : PREETHI D
REG NO: 212224040250
```

In [1]:
```
pip install pandas
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib\site-pack
ages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from
pandas) (2022.7)
Requirement already satisfied: numpy>=1.21.0 in c:\programdata\anaconda3\lib\site-packages (fro
m pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from pyt
hon-dateutil>=2.8.1->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
#Creating an empty dataframe :
import pandas as pd
df = pd.DataFrame()
print (df)
```

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

In [6]:
```
#Creating a DataFrame from Lists:
data = [1,2,3,4]
df = pd.DataFrame(data)
print (df)
```

```
   0
0  1
1  2
2  3
3  4
```

In [7]:
```
data = [['Alex',10],['Bob',12]]
df = pd.DataFrame(data,columns=['Name','Age'])
print (df)
```

```
   Name  Age
0  Alex   10
1   Bob   12
```

In [9]:
```
# Creating DataFrame from dict of  lists:
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print(df)
```

```
    Name  Age
0    Tom   28
1   Jack   34
2  Steve   29
3  Ricky   42
```

In [10]:
```
mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

In [11]:
```
#Dealing with Rows and Columns-Column Addition, Deletion, Renaming :
```

In [12]:
```
#Column Addition:
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],  'Height': [5.1, 6.2, 5.1, 5.2], 'Qualifica
df = pd.DataFrame(data)
address = ['Delhi', 'Bangalore', 'Chennai', 'Patna']
df['Address'] = address
print(df)
```

```
     Name  Height Qualification    Address
0     Jai     5.1           Msc      Delhi
1  Princi     6.2            MA  Bangalore
2  Gaurav     5.1           Msc    Chennai
3    Anuj     5.2           Msc      Patna
```

In [18]:
```
#Column Deletion:
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd'],'address': ['Delhi', 'Bangalore', 'Chennai',
df = pd.DataFrame(data)
# using del function
del df['address']
df
```

Out[18]:

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

In [19]:
```
# using drop function
df.drop(['Address'],axis=1,inplace=True)
df
```

Out[19]:

	Name	Age	Qualification
0	Jai	27	Msc
1	Princi	24	MA
2	Gaurav	22	MCA
3	Anuj	32	Phd

In [21]:
```
# using pop function
df.pop('Age')
df
```

Out[21]:

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

In [29]:
```
#Column Renaming:
# Method 1: Using rename() function
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],  'Age':[27, 24, 22, 32],  'address':['Delhi'
df = pd.DataFrame(data)
print(df)
df.rename(columns={'address':'place'},inplace=True)
df
```

```
     Name  Age    address Qualification
0     Jai   27      Delhi           Msc
1  Princi   24     Kanpur            MA
2  Gaurav   22  Allahabad           MCA
3    Anuj   32    Kannauj           Phd
```

Out[29]:

	Name	Age	place	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

In [33]:
```
#2: By assigning a list of new column names
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],  'Age':[27, 24, 22, 32],  'Address':['Delhi'
df = pd.DataFrame(data)
print(df)
df.columns=['A','B','C','D']
df
```

```
     Name  Age    Address Qualification
0     Jai   27      Delhi           Msc
1  Princi   24     Kanpur            MA
2  Gaurav   22  Allahabad           MCA
3    Anuj   32    Kannauj           Phd
```

Out[33]:

	A	B	C	D
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

In [35]:
```
#Addition of Rows:
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df = pd.concat([df, df2], ignore_index=True)
df
```

Out[35]:

	a	b
0	1	2
1	3	4
2	5	6
3	7	8

In [36]:
```
# Deletion of Rows:
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
df = pd.DataFrame(data)
df
df.drop(0,axis=0,inplace=True)
df
```

Out[36]:

	Name	Age	Address	Qualification
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

In [37]:
```
#Indexing and Selecting Data
```

In [40]:
```
#Indexing a Dataframe using indexing operator [] :
#Column Selection:
data = {'name': ['Alice', 'Bob', 'Charlie', 'Dave'],
        'age': [25, 32, 18, 47],
        'gender': ['F', 'M', 'M', 'M'],
        'height': [1.62, 1.78, 1.65, 1.83]}
df = pd.DataFrame(data)
df = df['name']
df
```

Out[40]:
```
0      Alice
1        Bob
2    Charlie
3       Dave
Name: name, dtype: object
```

In [41]:
```
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32], 'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
df = pd.DataFrame(data)
print(df[['Name', 'Qualification']])
```

```
     Name Qualification
0     Jai           Msc
1  Princi            MA
2  Gaurav           MCA
3    Anuj           Phd
```

In [42]:
```
#drop_duplicates()
```

In [50]:
```
df = pd.DataFrame(data)
# Removing duplicate rows based on all columns
df = df.drop_duplicates()
df
# Removing duplicate rows based on a subset of columns
#df = df.drop_duplicates(subset=['name', 'age'])
# Keeping the last duplicate row based on a subset of columns
#df = df.drop_duplicates(subset=['name', 'age'], keep='last')
#df
```

Out[50]:

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

In [45]:
```
#sample()
df_sample = df.sample(n=2)
df_sample
```

Out[45]:

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
3	Anuj	32	Kannauj	Phd

In [53]:
```
# Randomly selecting 50% of the rows from the DataFrame
df_sample = df.sample(frac=0.5)
df_sample
```

Out[53]:

	Name	Age	Address	Qualification
2	Gaurav	22	Allahabad	MCA
1	Princi	24	Kanpur	MA

In [54]:
```
# Randomly selecting 2 columns from the DataFrame
df_sample = df.sample(n=2, axis=1)
df_sample
```

Out[54]:

	Qualification	Address
0	Msc	Delhi
1	MA	Kanpur
2	MCA	Allahabad
3	Phd	Kannauj

In [57]:
```
# nlargest() and nsmallest()
data = {'name': ['Alice', 'Bob', 'Charlie', 'David', 'Emily'],
        'age': [25, 30, 35, 40, 45],
        'salary': [50000, 60000, 70000, 80000, 90000]}
df = pd.DataFrame(data)
# Get the 2 rows with the largest salary
top_salaries = df.nlargest(2, columns='salary')
top_salaries
```

Out[57]:

	name	age	salary
4	Emily	45	90000
3	David	40	80000

In [58]:
```
data = {'name': ['Alice', 'Bob', 'Charlie', 'David', 'Emily'],
        'age': [25, 30, 35, 40, 45],
        'salary': [50000, 60000, 70000, 80000, 90000]}
df = pd.DataFrame(data)
# Get the 2 rows with the least salary
low_salaries = df.nsmallest(2, columns='salary')
print(low_salaries)
```

```
    name  age  salary
0  Alice   25   50000
1    Bob   30   60000
```

In [62]:
```
#query()
df.query('age >= 30')
```

Out[62]:

	name	age	salary
1	Bob	30	60000
2	Charlie	35	70000
3	David	40	80000
4	Emily	45	90000

In [63]:
```
df.query('name.str.contains("a") and salary>50000')
```

Out[63]:

	name	age	salary
2	Charlie	35	70000
3	David	40	80000

In [64]:
```
df.query('age<40 and salary<80000')
```

Out[64]:

	name	age	salary
0	Alice	25	50000
1	Bob	30	60000
2	Charlie	35	70000

In [67]:
```
#Indexing and Selecting
row = df.iloc[1]
print(row)
print()
# Select the row with the index label 2
row = df.loc[2]
row
```

```
name        Bob
age          30
salary    60000
Name: 1, dtype: object
```

Out[67]:
```
name      Charlie
age            35
salary      70000
Name: 2, dtype: object
```

In [68]:
```
# Select rows with index labels 1 and 3
subset = df.loc[[1,3]]
print(subset)
```

```
    name  age  salary
1    Bob   30   60000
3  David   40   80000
```

In [69]:
```
# Select the 2nd and 4th rows
subset = df.iloc[[1,3]]
print(subset)
```

```
    name  age  salary
1    Bob   30   60000
3  David   40   80000
```

In [70]:
```
#Indexing a DataFrame using .loc[ ] : Row Selection:
data = {'name': ['Alice', 'Bob', 'Charlie', 'David', 'Emily'],
        'age': [25, 32, 18, 47, 22], 'gender': ['F', 'M', 'M', 'M', 'F']}
df = pd.DataFrame(data)
print(df)
##select all rows for a specific column
df.loc[:,'age']
## Select all rows for multiple columns
df.loc[:,['name','age']]
```

```
      name  age gender
0    Alice   25      F
1      Bob   32      M
2  Charlie   18      M
3    David   47      M
4    Emily   22      F
```

Out[70]:

	name	age
0	Alice	25
1	Bob	32
2	Charlie	18
3	David	47
4	Emily	22

In [72]:
```
#Indexing a DataFrame using .iloc[ ]:
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22,2],  'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
df = pd.DataFrame(data)
df.iloc[:4]
```

Out[72]:

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	2	Kannauj	Phd

In [73]:
```
df.iloc[1:5, 2:4]
```

Out[73]:

	Address	Qualification
1	Kanpur	MA
2	Allahabad	MCA
3	Kannauj	Phd

In [74]:
```
df.iloc[[1, 3], [1, 3]]
```

Out[74]:

	Age	Qualification
1	24	MA
3	2	Phd

In [76]:
```
#Conditional Indexing:Conditional indexing is a way to filter a Pandas DataFrame based on certai
# create a DataFrame
data = {'name': ['Alice', 'Bob', 'Charlie', 'Dave'], 'age': [25, 32, 18, 47], 'gender': ['F', 'M
df = pd.DataFrame(data)
# select all rows where the age is greater than 30
df_filtered = df[df['age'] > 30]
# print the filtered DataFrame
df_filtered
```

Out[76]:

	name	age	gender	height
1	Bob	32	M	1.78
3	Dave	47	M	1.83

In [77]:
```
# create a sample DataFrame
data = {
    'Name': ['John', 'Sarah', 'Mike', 'Emily', 'David'],
    'Age': [25, 31, 29, 35, 27],
    'Gender': ['M', 'F', 'M', 'F', 'M'],
    'Salary': [50000, 70000, 60000, 80000, 55000]
}
df = pd.DataFrame(data)
```

In [78]:
```
# head method example
print(df.head(3)) # prints the first 3 rows of the DataFrame
```

```
    Name  Age Gender  Salary
0   John   25      M   50000
1  Sarah   31      F   70000
2   Mike   29      M   60000
```

In [79]:
```
# tail method example
print(df.tail(2)) # prints the last 2 rows of the DataFrame
```

```
    Name  Age Gender  Salary
3  Emily   35      F   80000
4  David   27      M   55000
```

In [80]:
```
# info method example
df.info() # prints the concise summary of the DataFrame, including data types and non-null value
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Name    5 non-null      object
 1   Age     5 non-null      int64 
 2   Gender  5 non-null      object
 3   Salary  5 non-null      int64 
dtypes: int64(2), object(2)
memory usage: 292.0+ bytes
```

In [81]:
```
# describe method example
print(df.describe()) # prints the descriptive statistics of the DataFrame
```

```
             Age        Salary
count   5.000000      5.000000
mean   29.400000  63000.000000
std     3.847077  12041.594579
min    25.000000  50000.000000
25%    27.000000  55000.000000
50%    29.000000  60000.000000
75%    31.000000  70000.000000
max    35.000000  80000.000000
```

In [83]:
```
# DATAFRAME SORTING
# sort DataFrame by 'age' column in descending order
data = {'name': ['Alice', 'Bob', 'Charlie', 'Dave'],
        'age': [25, 30, 35, 40],
        'score': [90, 80, 85, 95]}
df = pd.DataFrame(data)

df_sorted = df.sort_values(by='age', ascending=False)
print(df_sorted)
```

```
      name  age  score
3     Dave   40     95
2  Charlie   35     85
1      Bob   30     80
0    Alice   25     90
```

In [88]:
```
#groupby
import pandas as pd
data = {'name': ['Alice', 'Bob', 'Charlie', 'Dave', 'Emily', 'Frank'],
        'gender': ['F', 'M', 'M', 'M', 'F', 'M'],
        'age': [25, 35, 40, 28, 30, 45],
        'salary': [50000, 70000, 60000, 80000, 65000, 90000]}
df = pd.DataFrame(data)
grouped = df.groupby('gender').mean(numeric_only=True)['age']
print(grouped)
```

```
gender
F    27.5
M    37.0
Name: age, dtype: float64
```

In [89]:
```
#Data cleaning using pandas
df.dropna()
df
```

Out[89]:

	name	gender	age	salary
0	Alice	F	25	50000
1	Bob	M	35	70000
2	Charlie	M	40	60000
3	Dave	M	28	80000
4	Emily	F	30	65000
5	Frank	M	45	90000

In [90]:
```
df.fillna(1)
df
```

Out[90]:

	name	gender	age	salary
0	Alice	F	25	50000
1	Bob	M	35	70000
2	Charlie	M	40	60000
3	Dave	M	28	80000
4	Emily	F	30	65000
5	Frank	M	45	90000

In [91]:
```
# Create a sample DataFrame with missing values
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Dave', 'Eve'],
        'Age': [25, 32, None, 41, 28],
        'Salary': [50000, None, 70000, 90000, 60000]}
df = pd.DataFrame(data)
```

In [93]:
```
# Remove rows with missing values only in the 'Salary' column
df_cleaned = df.dropna(subset=['Salary'])
df_cleaned
```

Out[93]:

	Name	Age	Salary
0	Alice	25.0	50000.0
2	Charlie	NaN	70000.0
3	Dave	41.0	90000.0
4	Eve	28.0	60000.0

In [96]:
```
# Remove rows with all missing values
df_cleaned_all = df.dropna(how='all')
df_cleaned_all
```

Out[96]:

	Name	Age	Salary
0	Alice	25.0	50000.0
1	Bob	32.0	NaN
2	Charlie	NaN	70000.0
3	Dave	41.0	90000.0
4	Eve	28.0	60000.0

In [97]:
```
# Remove rows with less than 2 non-missing values
df_cleaned_thresh = df.dropna(thresh=2)
df_cleaned_thresh
```

Out[97]:

	Name	Age	Salary
0	Alice	25.0	50000.0
1	Bob	32.0	NaN
2	Charlie	NaN	70000.0
3	Dave	41.0	90000.0
4	Eve	28.0	60000.0

In [98]:
```
# Modify the original DataFrame by removing rows with missing values in any column
df.dropna(inplace=True)
```

In [102]:
```
# Create a sample DataFrame with missing values
import numpy as np
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Dave', 'Eve'],
        'Age': [25, np.nan, 35, 41, np.nan],
        'Salary': [50000, np.nan, 70000, np.nan, 60000]}
df = pd.DataFrame(data)
# Fill missing values with a constant value
df_filled = df.fillna(0)
df_filled
```

Out[102]:

	Name	Age	Salary
0	Alice	25.0	50000.0
1	Bob	0.0	0.0
2	Charlie	35.0	70000.0
3	Dave	41.0	0.0
4	Eve	0.0	60000.0

In [104]:
```
# Fill missing values using forward filling
df_ffilled = df.fillna(method='ffill')
df_ffilled
```

Out[104]:

	Name	Age	Salary
0	Alice	25.0	50000.0
1	Bob	25.0	50000.0
2	Charlie	35.0	70000.0
3	Dave	41.0	70000.0
4	Eve	41.0	60000.0

In [106]:
```
# Fill missing values using backward filling
df_bfilled = df.fillna(method='bfill')
df_bfilled
```

Out[106]:

	Name	Age	Salary
0	Alice	25.0	50000.0
1	Bob	35.0	70000.0
2	Charlie	35.0	70000.0
3	Dave	41.0	60000.0
4	Eve	NaN	60000.0

In [108]:
```
# Fill missing values with the mean value of the column
df_mean = df.fillna(df.mean(numeric_only=True))
df_mean
```

Out[108]:

	Name	Age	Salary
0	Alice	25.000000	50000.0
1	Bob	33.666667	60000.0
2	Charlie	35.000000	70000.0
3	Dave	41.000000	60000.0
4	Eve	33.666667	60000.0

In [109]:
```
# Modify the original DataFrame by filling missing values using forward filling
df.fillna(method='ffill', inplace=True)
df
```

Out[109]:

	Name	Age	Salary
0	Alice	25.0	50000.0
1	Bob	25.0	50000.0
2	Charlie	35.0	70000.0
3	Dave	41.0	70000.0
4	Eve	41.0	60000.0

In [112]:
```
#To check duplication
~df.duplicated()
df=df[~df.duplicated()]
```

In [113]:
```
#NumPy
#NumPy is a Python library.
#NumPy is short for "Numerical Python".
#NumPy is a Python library used for working with arrays.
```

In [114]:
```
#Installation of NumPy
import numpy as np
```

In [115]:
```
#SciPy
#SciPy is a scientific computation library that uses NumPy underneath.
#SciPy stands for Scientific Python.
```

In [117]:
```
#DATA VISUALIZATION
#   It is a language that allows you to make self-explanatory names of your data. You can simply
#Matplotlib, Seaborn & Datashader are some of the Python libraries that help you do this task.
```

In [118]:
```
### MACHINE LEARNING
#It is a language that allows you to make self-explanatory names of your data.
```