1. Pandas Series and DataFrame for Machine Learning

August 18, 2024

1 Introduction to the Pandas

Pandas is an open-source data manipulation and analysis library for Python, offering data structures and functions specifically designed to work with structured data seamlessly. Created by Wes McKinney in 2008, Pandas has become one of the most popular tools for data analysis in Python due to its ease of use, performance, and versatility.

At its core, Pandas provides two primary data structures:

1. **Series**: A one-dimensional array-like structure that can hold various data types (integers, floats, strings, etc.). Each element in a Series has an associated label, known as its index.
2. **DataFrame**: A two-dimensional table-like structure with labeled axes (rows and columns). It can be thought of as a collection of Series objects sharing the same index, allowing for a more complex data organization similar to a spreadsheet or SQL table.

1.1 Key Features of Pandas

- **Data Cleaning and Preparation**: Pandas makes it easy to handle missing data, filter unwanted entries, and transform data formats, ensuring that datasets are ready for analysis.
- **Data Alignment**: One of Pandas' most powerful features is automatic data alignment. When performing operations on data from different sources, Pandas automatically aligns the data based on the index, making it easier to combine and compare different datasets.
- **Efficient Data Manipulation**: Pandas provides a wide range of functions for data manipulation, such as merging, joining, reshaping, and pivoting datasets. These operations are optimized for performance, allowing you to work with large datasets efficiently.
- **Time Series Analysis**: Pandas has robust support for working with time series data, including date and time indexing, resampling, and rolling window calculations. This makes it an excellent tool for financial and economic data analysis.
- **Input/Output Tools**: Pandas can read and write data in various formats, including CSV, Excel, SQL databases, JSON, and more. This versatility makes it easy to integrate Pandas into existing data workflows.
- **Group By Operations**: Pandas allows you to split your data into groups based on specific criteria and then apply aggregate functions or transformations to each group independently.

1.2 Use Cases for Pandas

Pandas is widely used in various fields, including finance, economics, social sciences, engineering, and more. Some common use cases include:

- **Data Analysis**: Pandas is frequently used for exploring, cleaning, and analyzing datasets, whether for academic research, business intelligence, or machine learning.
- **Data Visualization**: While Pandas itself does not provide advanced plotting capabilities, it integrates well with libraries like Matplotlib and Seaborn, making it easier to visualize data trends and patterns.
- **Data Wrangling**: Pandas is often used for preparing and transforming data before feeding it into machine learning models or other statistical tools.

In summary, Pandas is a powerful and flexible tool that simplifies many aspects of data analysis and manipulation in Python. Whether you're working with small datasets or large, complex data, Pandas provides the tools you need to analyze, clean, and visualize your data efficiently.

1.3 Installation of Pandas

[ ]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in
c:\users\vishwajeet\appdata\local\programs\python\python312\lib\site-packages
(2.2.2)
Requirement already satisfied: numpy>=1.26.0 in
c:\users\vishwajeet\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (2.0.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\vishwajeet\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\vishwajeet\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\vishwajeet\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in
c:\users\vishwajeet\appdata\local\programs\python\python312\lib\site-packages
(from python-dateutil>=2.8.2->pandas) (1.16.0)
```

1.4 Pandas Series Object

A **Pandas Series** is a one-dimensional labeled array capable of holding data of any type (integers, strings, floats, Python objects, etc.). It is similar to a column in a spreadsheet or a database table, but with more flexibility and functionality.

1.4.1 Key Characteristics of a Pandas Series

1. **One-Dimensional**:

- A Series is essentially a single column of data, making it one-dimensional. Unlike a list or a NumPy array, each element in a Series is associated with a label, known as an index.

2. **Index**:
    - The index in a Series is a key feature that distinguishes it from other data structures like lists or arrays. Each element in a Series is indexed, meaning it has a label that allows you to access data based on a unique identifier rather than just its position. By default, Pandas assigns an integer index starting from 0, but you can customize the index to use labels, dates, or other identifiers.
3. **Homogeneous Data**:
    - A Series can hold data of only one type, similar to an array in NumPy. However, it is more flexible because it can contain any data type, including numbers, strings, or even Python objects.
4. **Data Alignment**:
    - One of the powerful features of a Series is automatic alignment based on the index labels. This means that when performing operations on two Series objects, Pandas automatically aligns them by their index, facilitating more intuitive data manipulation.

[ ]:
```
# Importing all the libraries
import numpy as np
import pandas as pd
```

1.5 Creating a Series from Python Objects

[ ]:
```
help(pd.Series)
```

1.6 Data Lists and Indexing

We can create a series from Python lists(also from NumPy arrays)

[ ]:
```
city = ["Delhi", "Mumbai", "Kolkata", "Pune", "Bengaluru"]
```

[ ]:
```
num = [10,20,30,40,50]
```

[ ]:
```
series1 = pd.Series(data=city)
series1
```

[ ]:
```
0        Delhi
1       Mumbai
2      Kolkata
3         Pune
4    Bengaluru
dtype: object
```

[ ]:
```
series2 = pd.Series(data=num)
series2
```

[ ]:
```
0    10
1    20
```

```
2    30
3    40
4    50
dtype: int64
```

[ ]:
```
series3 = pd.Series(data=num, index=city)
series3
```

[ ]:
```
Delhi        10
Mumbai       20
Kolkata      30
Pune         40
Bengaluru    50
dtype: int64
```

Let's take another example

[ ]:
```
age = np.random.randint(0,100,4)
age
```

[ ]:
```
array([40, 60, 92, 64], dtype=int32)
```

[ ]:
```
names = ["Raj", "Vishal", "Vinay", "Shekhar"]
names
```

[ ]:
```
['Raj', 'Vishal', 'Vinay', 'Shekhar']
```

[ ]:
```
ages = pd.Series(age, names)
```

[ ]:
```
ages
```

[ ]:
```
Raj        40
Vishal     60
Vinay      92
Shekhar    64
dtype: int32
```

1.7 Pandas Series Object from Python Dictionary

[ ]:
```
ages = {'Sammy':5,'Frank':10,'Spike':7}
```

[ ]:
```
ages
```

[ ]:
```
{'Sammy': 5, 'Frank': 10, 'Spike': 7}
```

[ ]:
```
pd.Series(ages)
```

```
[ ]: Sammy     5
     Frank    10
     Spike     7
     dtype: int64
```

1.8 Named Index in Pandas Series

```
[ ]: # Imaginary Sales Data for 1st and 2nd Quarters for Global Company
     q1 = {'Japan': 80, 'China': 450, 'India': 200, 'USA': 250}
     q2 = {'Brazil': 100,'China': 500, 'India': 210,'USA': 260}
```

```
[ ]: # Convert into Pandas Series
     sales_Q1 = pd.Series(q1)
     sales_Q2 = pd.Series(q2)
```

```
[ ]: sales_Q1
```

```
[ ]: Japan     80
     China    450
     India    200
     USA      250
     dtype: int64
```

```
[ ]: # Call values based on Named Index
     sales_Q1['Japan']
```

```
[ ]: np.int64(80)
```

```
[ ]: # Integer Based Location information also retained!
     sales_Q1[0]
```

```
C:\Users\Vishwajeet\AppData\Local\Temp\ipykernel_14716\3172792608.py:2:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  sales_Q1[0]
```

```
[ ]: np.int64(80)
```

1.8.1 Be careful with potential errors

```
[ ]: # Wrong Name
     sales_Q1['France']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:
↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\inc
↪py:3805, in Index.get_loc(self, key)
```

```
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas\\_libs\\hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas\\_libs\\hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'France'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[112], line 2
      1 # Wrong Name
----> 2 sales_Q1['France']

File c:
 ↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ser
 ↪py:1121, in Series.__getitem__(self, key)
   1118     return self._values[key]
   1120 elif key_is_scalar:
-> 1121     return self._get_value(key)
   1123 # Convert generator to list before going through hashable part
   1124 # (We will iterate through the generator there to check for slices)
   1125 if is_iterator(key):

File c:
 ↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ser
 ↪py:1237, in Series._get_value(self, label, takeable)
   1234     return self._values[label]
   1236 # Similar to Index.get_value, but we do not fall back to positional
-> 1237 loc = self.index.get_loc(label)
   1239 if is_integer(loc):
   1240     return self._values[loc]

File c:
 ↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ind
 ↪py:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
```

```
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'France'
```

[ ]:
```
# Accidental Extra Space
sales_Q1['USA ']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:
↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ind
↪py:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas\\_libs\\hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

File pandas\\_libs\\hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

KeyError: 'USA '

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[113], line 2
      1 # Accidental Extra Space
----> 2 sales_Q1['USA ']

File c:
↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ser
↪py:1121, in Series.__getitem__(self, key)
   1118     return self._values[key]
   1120 elif key_is_scalar:
```

```
-> 1121     return self._get_value(key)
   1123 # Convert generator to list before going through hashable part
   1124 # (We will iterate through the generator there to check for slices)
   1125 if is_iterator(key):

File c:
 ↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ser
 ↪py:1237, in Series._get_value(self, label, takeable)
   1234     return self._values[label]
   1236 # Similar to Index.get_value, but we do not fall back to positional
-> 1237 loc = self.index.get_loc(label)
   1239 if is_integer(loc):
   1240     return self._values[loc]

File c:
 ↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\inc
 ↪py:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'USA '
```

[ ]:
```
# Capitalization Mistake
sales_Q1['usa']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:
 ↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\inc
 ↪py:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()
```

```
File pandas\\_libs\\hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

File pandas\\_libs\\hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

KeyError: 'usa'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[114], line 2
      1 # Capitalization Mistake
----> 2 sales_Q1['usa']

File c:
↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ser
↪py:1121, in Series.__getitem__(self, key)
   1118     return self._values[key]
   1120 elif key_is_scalar:
-> 1121     return self._get_value(key)
   1123 # Convert generator to list before going through hashable part
   1124 # (We will iterate through the generator there to check for slices)
   1125 if is_iterator(key):

File c:
↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\ser
↪py:1237, in Series._get_value(self, label, takeable)
   1234     return self._values[label]
   1236 # Similar to Index.get_value, but we do not fall back to positional
-> 1237 loc = self.index.get_loc(label)
   1239 if is_integer(loc):
   1240     return self._values[loc]

File c:
↪\Users\Vishwajeet\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\inc
↪py:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)
```

```
KeyError: 'usa'
```

1.9 Basic Operations with Pandas Series

[ ]:
```
# Grab just the index keys
sales_Q1.keys()
```

[ ]:
```
Index(['Japan', 'China', 'India', 'USA'], dtype='object')
```

[ ]:
```
# Can Perform Operations Broadcasted across entire Series
sales_Q1 * 2
```

[ ]:
```
Japan    160
China    900
India    400
USA      500
dtype: int64
```

[ ]:
```
sales_Q2 / 100
```

[ ]:
```
Brazil    1.0
China     5.0
India     2.1
USA       2.6
dtype: float64
```

1.10 Operations Between Two Series

[ ]:
```
# Notice how Pandas informs you of mismatch with NaN
sales_Q1 + sales_Q2
```

[ ]:
```
Brazil      NaN
China     950.0
India     410.0
Japan       NaN
USA       510.0
dtype: float64
```

[ ]:
```
# You can fill these with any value you want
sales_Q1.add(sales_Q2,fill_value=0)
```

[ ]:
```
Brazil    100.0
China     950.0
India     410.0
Japan      80.0
USA       510.0
```

```
dtype: float64
```

1.10.1 Pandas DataFrame

A **Pandas DataFrame** is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns). It is one of the most widely used data structures in Pandas, providing a convenient way to store and manipulate data similar to a table in a relational database or an Excel spreadsheet.

Key Characteristics of a DataFrame

1. **Two-Dimensional**:
   - A DataFrame is a two-dimensional structure, meaning it has rows and columns. Each column in a DataFrame is a Pandas Series, allowing you to work with data in a structured and organized manner.
2. **Labeled Axes**:
   - The rows and columns in a DataFrame are labeled with an index (for rows) and column names (for columns). This labeling allows for intuitive data selection, filtering, and manipulation based on labels rather than just integer positions.
3. **Heterogeneous Data**:
   - Unlike a Series, which holds data of a single type, a DataFrame can hold multiple data types across different columns. For example, one column could store integers, another could store strings, and yet another could store floating-point numbers.
4. **Size-Mutable**:
   - DataFrames can grow or shrink as needed, meaning you can add or remove rows and columns without having to redefine the entire structure.
5. **Alignment**:
   - Like Series, DataFrames automatically align data based on the labels when performing operations between different DataFrames or between a DataFrame and a Series. This alignment is done on both rows and columns.

Creating a DataFrame You can create a DataFrame in several ways:

- **From a Dictionary of Lists**:

  ```python
  import pandas as pd

  data = {
      'Name': ['Alice', 'Bob', 'Charlie'],
      'Age': [25, 30, 35],
      'City': ['New York', 'San Francisco', 'Los Angeles']
  }
  df = pd.DataFrame(data)
  print(df)
  ```

- **From a List of Dictionaries**:

  ```python
  data = [
      {'Name': 'Alice', 'Age': 25, 'City': 'New York'},
      {'Name': 'Bob', 'Age': 30, 'City': 'San Francisco'},
  ```

```
    {'Name': 'Charlie', 'Age': 35, 'City': 'Los Angeles'}
]
df = pd.DataFrame(data)
print(df)
```

- **From a 2D NumPy Array**:

```
import numpy as np

data = np.array([[25, 'Alice', 'New York'],
                 [30, 'Bob', 'San Francisco'],
                 [35, 'Charlie', 'Los Angeles']])
df = pd.DataFrame(data, columns=['Age', 'Name', 'City'])
print(df)
```

- **From a CSV File**:

```
df = pd.read_csv('data.csv')
print(df)
```

Accessing Data in a DataFrame You can access the data in a DataFrame using various methods:

- **Accessing Columns**:

```
print(df['Name'])  # Returns the 'Name' column as a Series
```

- **Accessing Rows by Label**:

```
print(df.loc[0])  # Returns the first row as a Series
```

- **Accessing Rows by Integer Location**:

```
print(df.iloc[0])  # Same as above, using integer location
```

- **Accessing a Subset of Rows and Columns**:

```
print(df.loc[0:1, ['Name', 'Age']])  # Returns first two rows and 'Name' and 'Age' columns
```

Operations on a DataFrame DataFrames support a wide range of operations, including arithmetic operations, aggregation functions, and more complex data manipulations:

- **Element-Wise Operations**:

```
df['Age'] = df['Age'] + 1  # Increase each age by 1
print(df)
```

- **Filtering Data**:

```
filtered_df = df[df['Age'] > 30]  # Filter rows where 'Age' > 30
print(filtered_df)
```

- **Aggregation Functions**:

```
mean_age = df['Age'].mean()  # Calculate the mean of the 'Age' column
print(mean_age)
```

- **Adding and Dropping Columns**:

```
df['Country'] = 'USA'  # Add a new column 'Country'
df = df.drop('City', axis=1)  # Drop the 'City' column
print(df)
```

DataFrame Methods Pandas DataFrames come with numerous built-in methods for common data tasks:

- **Sorting**:

```
df = df.sort_values(by='Age', ascending=False)  # Sort by 'Age' in descending order
```

- **Grouping**:

```
grouped = df.groupby('City').mean()  # Group by 'City' and calculate the mean for each gro
```

- **Merging and Joining**:

```
df1 = pd.DataFrame({'key': ['A', 'B', 'C'], 'value': [1, 2, 3]})
df2 = pd.DataFrame({'key': ['B', 'C', 'D'], 'value': [4, 5, 6]})
merged_df = pd.merge(df1, df2, on='key', how='inner')  # Merge on 'key' with an inner joi
print(merged_df)
```

The Pandas DataFrame is an essential tool for anyone working with structured data in Python. Its versatility, ease of use, and powerful built-in functions make it an invaluable resource for tasks ranging from simple data exploration to complex data analysis. Whether you are dealing with small datasets or large, complex data, Pandas DataFrames provide the tools you need to manage, manipulate, and analyze your data efficiently.

1.11 Import Libraries

[ ]:
```
import numpy as np
import pandas as pd
```

1.12 Creating a DataFrame from Python Objects

[ ]:
```
help(pd.DataFrame)
```

[ ]:
```
## Let's Create our first Pandas DataFrame
# Make sure the seed is in the same cell as the random call
np.random.seed(101)
df1 = np.random.randint(0,101,(4,3))
```

[ ]:
```
df1
```

[ ]:
```
array([[95, 11, 81],
       [70, 63, 87],
       [75,  9, 77],
       [40,  4, 63]], dtype=int32)
```

[ ]:
```
indx = ['CA','NY','AZ','TX']
```

[ ]:
```
cols = ['Jan', 'Feb', 'Mar']
```

[ ]:
```
df = pd.DataFrame(data=df1)
df
```

[ ]:

	0	1	2
0	95	11	81
1	70	63	87
2	75	9	77
3	40	4	63

[ ]:
```
df = pd.DataFrame(data=df1,index=indx)
df
```

[ ]:

	0	1	2
CA	95	11	81
NY	70	63	87
AZ	75	9	77
TX	40	4	63

[ ]:
```
df = pd.DataFrame(data=df1,index=indx,columns=cols)
df
```

[ ]:

	Jan	Feb	Mar
CA	95	11	81
NY	70	63	87
AZ	75	9	77
TX	40	4	63

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, CA to TX
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Jan     4 non-null      int32
 1   Feb     4 non-null      int32
 2   Mar     4 non-null      int32
dtypes: int32(3)
memory usage: 80.0+ bytes
```

2 Reading a .csv file for a DataFrame

2.0.1 What is a CSV File?

A **CSV file** (Comma-Separated Values file) is a plain text file that stores tabular data (numbers and text) in a simple, structured format. Each line in a CSV file corresponds to a row in a table, and the values in each row are separated by a comma or another delimiter, such as a semicolon or tab.

Key Features of a CSV File

1. **Plain Text Format**:
   - CSV files are human-readable plain text files. This makes them easy to create, edit, and share using basic text editors or more advanced spreadsheet programs like Microsoft Excel or Google Sheets.
2. **Simple Structure**:
   - CSV files are structured in a straightforward manner. Each line represents a row in the table, and within each row, commas (or another delimiter) separate the values that correspond to the columns.
3. **No Data Types**:
   - Unlike more complex file formats (like Excel or SQL databases), CSV files do not store information about data types, formatting, or complex structures. All values are stored as plain text, and it is up to the software reading the file to interpret the data types (e.g., integers, floats, strings).
4. **Compatibility**:
   - CSV files are widely supported across different platforms, programming languages, and software tools. This makes them a popular choice for data exchange between different systems.
5. **Lack of Metadata**:
   - CSV files typically do not contain metadata, such as information about the data source, column data types, or formatting details. This simplicity is one of the reasons CSV is so widely used, but it also means that additional context or processing may be required when working with the data.

Example of a CSV File A CSV file might look like this:

```
Name,Age,City
Alice,25,New York
Bob,30,San Francisco
Charlie,35,Los Angeles
```

In this example: - The first line contains the column headers: "Name," "Age," and "City." - Each subsequent line represents a row in the table, with values corresponding to the columns.

Working with CSV Files **Opening a CSV File**: - You can open CSV files with text editors like Notepad, or spreadsheet programs like Excel or Google Sheets, which will display the data in a tabular format.

Saving Data as CSV: - Most spreadsheet applications allow you to save data as a CSV file, making it easy to export data for use in other programs.

Reading CSV Files in Python: - Python's `pandas` library is often used to read CSV files into a DataFrame, making it easy to analyze and manipulate the data:

```
import pandas as pd

df = pd.read_csv('data.csv')
print(df)
```

Writing to a CSV File: - You can also write data to a CSV file using `pandas`:

```
df.to_csv('output.csv', index=False)
```

Common Uses of CSV Files

- **Data Exchange**: CSV is commonly used for exporting data from one system and importing it into another, especially in situations where different software tools are used.
- **Data Storage**: CSV files are often used to store simple datasets that do not require complex formatting or data types.
- **Data Processing**: Many data processing tasks, especially in scripting or programming, involve reading data from CSV files, processing it, and then saving the results back to a CSV file.

CSV files are a simple, effective way to store and exchange tabular data. Their ease of use, compatibility with a wide range of software, and human-readable format make them a popular choice for data storage and transfer, especially when working with data across different platforms and programming environments.

Print your current directory file path with pwd

```
[ ]: %pwd
```

```
[ ]: 'e:\\Tutorials\\Python_for_ML\\2. Pandas'
```

List the files in your current directory with ls

```
[ ]: %ls
```

```
 Volume in drive E has no label.
 Volume Serial Number is 3CDB-88F2

 Directory of e:\Tutorials\Python_for_ML\2. Pandas

08/18/2024  12:38 PM    <DIR>          .
08/18/2024  12:34 PM    <DIR>          ..
08/18/2024  12:38 PM    <DIR>          Datasets
08/18/2024  02:05 PM           763,414 Pandas for Machine Learning.ipynb
               1 File(s)        763,414 bytes
               3 Dir(s)  168,625,639,424 bytes free
```

2.1 Let's read the data from .csv file

[ ]:
```python
df = pd.read_csv('Datasets/tips.csv')
```

[ ]:
```python
df
```

[ ]:
```
     total_bill   tip     sex smoker   day    time  size  price_per_person  \
0         16.99  1.01  Female     No   Sun  Dinner     2              8.49
1         10.34  1.66    Male     No   Sun  Dinner     3              3.45
2         21.01  3.50    Male     No   Sun  Dinner     3              7.00
3         23.68  3.31    Male     No   Sun  Dinner     2             11.84
4         24.59  3.61  Female     No   Sun  Dinner     4              6.15
..          …     …       …      …     …      …    …                …
239       29.03  5.92    Male     No   Sat  Dinner     3              9.68
240       27.18  2.00  Female    Yes   Sat  Dinner     2             13.59
241       22.67  2.00    Male    Yes   Sat  Dinner     2             11.34
242       17.82  1.75    Male     No   Sat  Dinner     2              8.91
243       18.78  3.00  Female     No  Thur  Dinner     2              9.39

             Payer Name         CC Number Payment ID
0     Christy Cunningham  3560325168603410    Sun2959
1         Douglas Tucker  4478071379779230    Sun4608
2         Travis Walters  6011812112971322    Sun4458
3       Nathaniel Harris  4676137647685994    Sun5260
4           Tonya Carter  4832732618637221    Sun2251
..                   …                …          …
239        Michael Avila  5296068606052842    Sat2657
240       Monica Sanders  3506806155565404    Sat1766
241           Keith Wong  6011891618747196    Sat3880
242         Dennis Dixon     4375220550950      Sat17
243       Michelle Hardin  3511451626698139    Thur672

[244 rows x 11 columns]
```

About this DataSet (in case you are interested)

- Description
  - One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:
- Format
  - A data frame with 244 rows and 7 variables
- Details
  - tip in dollars,
  - bill in dollars,
  - sex of the bill payer,
  - whether there were smokers in the party,
  - day of the week,
  - time of day,
  - size of the party.

In all he recorded 244 tips. The data was reported in a collection of case studies for business statistics (Bryant & Smith 1995).

- References
  - Bryant, P. G. and Smith, M (1995) Practical Data Analysis: Case Studies in Business Statistics. Homewood, IL: Richard D. Irwin Publishing:
- Note: We created some additional columns with Fake data, including Name, CC Number, and Payment ID.

2.2 Obtaining Basic Information About DataFrame

[ ]:
```
df.columns
```

[ ]:
```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size',
       'price_per_person', 'Payer Name', 'CC Number', 'Payment ID'],
      dtype='object')
```

[ ]:
```
df.index
```

[ ]:
```
RangeIndex(start=0, stop=244, step=1)
```

[ ]:
```
df.head(3)
```

[ ]:
```
   total_bill   tip     sex smoker  day    time  size  price_per_person  \
0       16.99  1.01  Female     No  Sun  Dinner     2              8.49
1       10.34  1.66    Male     No  Sun  Dinner     3              3.45
2       21.01  3.50    Male     No  Sun  Dinner     3              7.00

           Payer Name         CC Number Payment ID
0  Christy Cunningham  3560325168603410    Sun2959
1      Douglas Tucker  4478071379779230    Sun4608
2      Travis Walters  6011812112971322    Sun4458
```

[ ]:
```
df.tail(3)
```

[ ]:
```
     total_bill   tip     sex smoker   day    time  size  price_per_person  \
241       22.67  2.00    Male    Yes   Sat  Dinner     2             11.34
242       17.82  1.75    Male     No   Sat  Dinner     2              8.91
243       18.78  3.00  Female     No  Thur  Dinner     2              9.39

          Payer Name         CC Number Payment ID
241       Keith Wong  6011891618747196    Sat3880
242     Dennis Dixon     4375220550950      Sat17
243  Michelle Hardin  3511451626698139    Thur672
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
```

```
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   total_bill        244 non-null    float64
 1   tip               244 non-null    float64
 2   sex               244 non-null    object
 3   smoker            244 non-null    object
 4   day               244 non-null    object
 5   time              244 non-null    object
 6   size              244 non-null    int64
 7   price_per_person  244 non-null    float64
 8   Payer Name        244 non-null    object
 9   CC Number         244 non-null    int64
 10  Payment ID        244 non-null    object
dtypes: float64(3), int64(2), object(6)
memory usage: 21.1+ KB
```

[ ]:
```
## length of the dataframe
len(df)
```

[ ]:
```
244
```

[ ]:
```
df.describe()
```

[ ]:
```
       total_bill         tip        size  price_per_person     CC Number
count  244.000000  244.000000  244.000000        244.000000  2.440000e+02
mean    19.785943    2.998279    2.569672          7.888197  2.563496e+15
std      8.902412    1.383638    0.951100          2.914234  2.369340e+15
min      3.070000    1.000000    1.000000          2.880000  6.040679e+10
25%     13.347500    2.000000    2.000000          5.800000  3.040731e+13
50%     17.795000    2.900000    2.000000          7.255000  3.525318e+15
75%     24.127500    3.562500    3.000000          9.390000  4.553675e+15
max     50.810000   10.000000    6.000000         20.270000  6.596454e+15
```

[ ]:
```
df.describe().transpose()
```

[ ]:
```
                  count          mean           std           min  \
total_bill        244.0  1.978594e+01  8.902412e+00  3.070000e+00
tip               244.0  2.998279e+00  1.383638e+00  1.000000e+00
size              244.0  2.569672e+00  9.510998e-01  1.000000e+00
price_per_person  244.0  7.888197e+00  2.914234e+00  2.880000e+00
CC Number         244.0  2.563496e+15  2.369340e+15  6.040679e+10

                           25%           50%           75%           max
total_bill        1.334750e+01  1.779500e+01  2.412750e+01  5.081000e+01
tip               2.000000e+00  2.900000e+00  3.562500e+00  1.000000e+01
size              2.000000e+00  2.000000e+00  3.000000e+00  6.000000e+00
price_per_person  5.800000e+00  7.255000e+00  9.390000e+00  2.027000e+01
```

```
CC Number        3.040731e+13  3.525318e+15  4.553675e+15  6.596454e+15
```

2.3 Selection and Indexing

Let's learn how to retrieve information from a DataFrame.

2.3.1 COLUMNS

We will begin be learning how to extract information based on the columns

```
[ ]: df.head()
```

```
[ ]:    total_bill   tip     sex smoker  day    time  size  price_per_person  \
0        16.99  1.01  Female     No  Sun  Dinner     2              8.49
1        10.34  1.66    Male     No  Sun  Dinner     3              3.45
2        21.01  3.50    Male     No  Sun  Dinner     3              7.00
3        23.68  3.31    Male     No  Sun  Dinner     2             11.84
4        24.59  3.61  Female     No  Sun  Dinner     4              6.15

            Payer Name         CC Number Payment ID
0  Christy Cunningham  3560325168603410    Sun2959
1      Douglas Tucker  4478071379779230    Sun4608
2      Travis Walters  6011812112971322    Sun4458
3    Nathaniel Harris  4676137647685994    Sun5260
4        Tonya Carter  4832732618637221    Sun2251
```

Grab a Single Column

```
[ ]: df['total_bill']
```

```
[ ]: 0      16.99
1      10.34
2      21.01
3      23.68
4      24.59
       …
239    29.03
240    27.18
241    22.67
242    17.82
243    18.78
Name: total_bill, Length: 244, dtype: float64
```

```
[ ]: type(df['total_bill'])
```

```
[ ]: pandas.core.series.Series
```

Grab Multiple Columns

```
[ ]: # Note how its a python list of column names! Thus the double brackets.
df[['total_bill','tip']]
```

```
[ ]:    total_bill   tip
0         16.99  1.01
1         10.34  1.66
2         21.01  3.50
3         23.68  3.31
4         24.59  3.61
..          …     …
239       29.03  5.92
240       27.18  2.00
241       22.67  2.00
242       17.82  1.75
243       18.78  3.00

[244 rows x 2 columns]
```

Create New Columns

```
[ ]: df['tip_percentage'] = 100* df['tip'] / df['total_bill']
```

```
[ ]: df.head()
```

```
[ ]:   total_bill   tip     sex smoker  day    time  size  price_per_person  \
0       16.99  1.01  Female     No  Sun  Dinner     2              8.49
1       10.34  1.66    Male     No  Sun  Dinner     3              3.45
2       21.01  3.50    Male     No  Sun  Dinner     3              7.00
3       23.68  3.31    Male     No  Sun  Dinner     2             11.84
4       24.59  3.61  Female     No  Sun  Dinner     4              6.15

           Payer Name         CC Number Payment ID  tip_percentage
0  Christy Cunningham  3560325168603410    Sun2959        5.944673
1      Douglas Tucker  4478071379779230    Sun4608       16.054159
2      Travis Walters  6011812112971322    Sun4458       16.658734
3    Nathaniel Harris  4676137647685994    Sun5260       13.978041
4        Tonya Carter  4832732618637221    Sun2251       14.680765
```

```
[ ]: df['price_per_person'] = df['total_bill'] / df['size']
```

```
[ ]: df.head()
```

```
[ ]:   total_bill   tip     sex smoker  day    time  size  price_per_person  \
0       16.99  1.01  Female     No  Sun  Dinner     2          8.495000
1       10.34  1.66    Male     No  Sun  Dinner     3          3.446667
2       21.01  3.50    Male     No  Sun  Dinner     3          7.003333
3       23.68  3.31    Male     No  Sun  Dinner     2         11.840000
4       24.59  3.61  Female     No  Sun  Dinner     4          6.147500
```

```
            Payer Name         CC Number Payment ID  tip_percentage
0  Christy Cunningham  3560325168603410    Sun2959        5.944673
1      Douglas Tucker  4478071379779230    Sun4608       16.054159
2      Travis Walters  6011812112971322    Sun4458       16.658734
3    Nathaniel Harris  4676137647685994    Sun5260       13.978041
4        Tonya Carter  4832732618637221    Sun2251       14.680765
```

[ ]:
```
help(np.round)
```

```
Help on _ArrayFunctionDispatcher in module numpy:

round(a, decimals=0, out=None)
    Evenly round to the given number of decimals.

    Parameters
    ----------
    a : array_like
        Input data.
    decimals : int, optional
        Number of decimal places to round to (default: 0).  If
        decimals is negative, it specifies the number of positions to
        the left of the decimal point.
    out : ndarray, optional
        Alternative output array in which to place the result. It must have
        the same shape as the expected output, but the type of the output
        values will be cast if necessary. See :ref:`ufuncs-output-type`
        for more details.

    Returns
    -------
    rounded_array : ndarray
        An array of the same type as `a`, containing the rounded values.
        Unless `out` was specified, a new array is created.  A reference to
        the result is returned.

        The real and imaginary parts of complex numbers are rounded
        separately.  The result of rounding a float is a float.

    See Also
    --------
    ndarray.round : equivalent method
    around : an alias for this function
    ceil, fix, floor, rint, trunc


    Notes
    -----
```

```
For values exactly halfway between rounded decimal values, NumPy
rounds to the nearest even value. Thus 1.5 and 2.5 round to 2.0,
-0.5 and 0.5 round to 0.0, etc.

``np.round`` uses a fast but sometimes inexact algorithm to round
floating-point datatypes. For positive `decimals` it is equivalent to
``np.true_divide(np.rint(a * 10**decimals), 10**decimals)``, which has
error due to the inexact representation of decimal fractions in the IEEE
floating point standard [1]_ and errors introduced when scaling by powers
of ten. For instance, note the extra "1" in the following:

    >>> np.round(56294995342131.5, 3)
    56294995342131.51

If your goal is to print such values with a fixed number of decimals, it is
preferable to use numpy's float printing routines to limit the number of
printed decimals:

    >>> np.format_float_positional(56294995342131.5, precision=3)
    '56294995342131.5'

The float printing routines use an accurate but much more computationally
demanding algorithm to compute the number of digits after the decimal
point.

Alternatively, Python's builtin `round` function uses a more accurate
but slower algorithm for 64-bit floating point values:

    >>> round(56294995342131.5, 3)
    56294995342131.5
    >>> np.round(16.055, 2), round(16.055, 2)  # equals 16.0549999999999997
    (16.06, 16.05)


References
----------
.. [1] "Lecture Notes on the Status of IEEE 754", William Kahan,
       https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF

Examples
--------
>>> np.round([0.37, 1.64])
array([0., 2.])
>>> np.round([0.37, 1.64], decimals=1)
array([0.4, 1.6])
>>> np.round([.5, 1.5, 2.5, 3.5, 4.5]) # rounds to nearest even value
array([0., 2., 2., 4., 4.])
>>> np.round([1,2,3,11], decimals=1) # ndarray of ints is returned
```

```
array([ 1,  2,  3, 11])
>>> np.round([1,2,3,11], decimals=-1)
array([ 0,  0,  0, 10])
```

Adjust Existing Columns

[ ]:
```
# Because pandas is based on numpy, we get awesome capabilities with numpy's␣
↪universal functions!
df['price_per_person'] = np.round(df['price_per_person'],2)
```

[ ]:
```
df.head()
```

[ ]:
```
   total_bill   tip     sex smoker  day    time  size  price_per_person  \
0       16.99  1.01  Female     No  Sun  Dinner     2              8.49
1       10.34  1.66    Male     No  Sun  Dinner     3              3.45
2       21.01  3.50    Male     No  Sun  Dinner     3              7.00
3       23.68  3.31    Male     No  Sun  Dinner     2             11.84
4       24.59  3.61  Female     No  Sun  Dinner     4              6.15

          Payer Name         CC Number Payment ID  tip_percentage
0  Christy Cunningham  3560325168603410    Sun2959        5.944673
1      Douglas Tucker  4478071379779230    Sun4608       16.054159
2      Travis Walters  6011812112971322    Sun4458       16.658734
3    Nathaniel Harris  4676137647685994    Sun5260       13.978041
4        Tonya Carter  4832732618637221    Sun2251       14.680765
```

Remove Columns

[ ]:
```
df = df.drop("tip_percentage",axis=1)
```

[ ]:
```
df.head()
```

[ ]:
```
   total_bill   tip     sex smoker  day    time  size  price_per_person  \
0       16.99  1.01  Female     No  Sun  Dinner     2              8.49
1       10.34  1.66    Male     No  Sun  Dinner     3              3.45
2       21.01  3.50    Male     No  Sun  Dinner     3              7.00
3       23.68  3.31    Male     No  Sun  Dinner     2             11.84
4       24.59  3.61  Female     No  Sun  Dinner     4              6.15

          Payer Name         CC Number Payment ID
0  Christy Cunningham  3560325168603410    Sun2959
1      Douglas Tucker  4478071379779230    Sun4608
2      Travis Walters  6011812112971322    Sun4458
3    Nathaniel Harris  4676137647685994    Sun5260
4        Tonya Carter  4832732618637221    Sun2251
```

3 Index Basics

Before going over the same retrieval tasks for rows, let's build some basic understanding of the pandas DataFrame Index.

[ ]:
```
df.head()
```

[ ]:
```
   total_bill   tip     sex smoker  day    time  size  price_per_person  \
0       16.99  1.01  Female     No  Sun  Dinner     2              8.49
1       10.34  1.66    Male     No  Sun  Dinner     3              3.45
2       21.01  3.50    Male     No  Sun  Dinner     3              7.00
3       23.68  3.31    Male     No  Sun  Dinner     2             11.84
4       24.59  3.61  Female     No  Sun  Dinner     4              6.15

           Payer Name         CC Number Payment ID
0  Christy Cunningham  3560325168603410    Sun2959
1      Douglas Tucker  4478071379779230    Sun4608
2      Travis Walters  6011812112971322    Sun4458
3    Nathaniel Harris  4676137647685994    Sun5260
4        Tonya Carter  4832732618637221    Sun2251
```

[ ]:
```
df.index
```

[ ]:
```
RangeIndex(start=0, stop=244, step=1)
```

[ ]:
```
df.set_index('Payment ID')
```

[ ]:
```
            total_bill   tip     sex smoker   day    time  size  \
Payment ID
Sun2959          16.99  1.01  Female     No   Sun  Dinner     2
Sun4608          10.34  1.66    Male     No   Sun  Dinner     3
Sun4458          21.01  3.50    Male     No   Sun  Dinner     3
Sun5260          23.68  3.31    Male     No   Sun  Dinner     2
Sun2251          24.59  3.61  Female     No   Sun  Dinner     4
…                    …     …       …      …     …       …
Sat2657          29.03  5.92    Male     No   Sat  Dinner     3
Sat1766          27.18  2.00  Female    Yes   Sat  Dinner     2
Sat3880          22.67  2.00    Male    Yes   Sat  Dinner     2
Sat17            17.82  1.75    Male     No   Sat  Dinner     2
Thur672          18.78  3.00  Female     No  Thur  Dinner     2

            price_per_person          Payer Name         CC Number
Payment ID
Sun2959                 8.49  Christy Cunningham  3560325168603410
Sun4608                 3.45      Douglas Tucker  4478071379779230
Sun4458                 7.00      Travis Walters  6011812112971322
Sun5260                11.84    Nathaniel Harris  4676137647685994
Sun2251                 6.15        Tonya Carter  4832732618637221
```

```
…                          …              …                 …
Sat2657                 9.68      Michael Avila  5296068606052842
Sat1766                13.59    Monica Sanders  3506806155565404
Sat3880                11.34        Keith Wong  6011891618747196
Sat17                   8.91      Dennis Dixon     4375220550950
Thur672                 9.39   Michelle Hardin  3511451626698139

[244 rows x 10 columns]
```

[ ]:
```
df.head()
```

[ ]:
```
   total_bill   tip     sex smoker  day    time  size  price_per_person  \
0       16.99  1.01  Female     No  Sun  Dinner     2              8.49
1       10.34  1.66    Male     No  Sun  Dinner     3              3.45
2       21.01  3.50    Male     No  Sun  Dinner     3              7.00
3       23.68  3.31    Male     No  Sun  Dinner     2             11.84
4       24.59  3.61  Female     No  Sun  Dinner     4              6.15

           Payer Name         CC Number Payment ID
0  Christy Cunningham  3560325168603410    Sun2959
1      Douglas Tucker  4478071379779230    Sun4608
2      Travis Walters  6011812112971322    Sun4458
3    Nathaniel Harris  4676137647685994    Sun5260
4        Tonya Carter  4832732618637221    Sun2251
```

[ ]:
```
df = df.set_index('Payment ID')
```

[ ]:
```
df.head()
```

[ ]:
```
            total_bill   tip     sex smoker  day    time  size  \
Payment ID
Sun2959          16.99  1.01  Female     No  Sun  Dinner     2
Sun4608          10.34  1.66    Male     No  Sun  Dinner     3
Sun4458          21.01  3.50    Male     No  Sun  Dinner     3
Sun5260          23.68  3.31    Male     No  Sun  Dinner     2
Sun2251          24.59  3.61  Female     No  Sun  Dinner     4

            price_per_person          Payer Name         CC Number
Payment ID
Sun2959                 8.49  Christy Cunningham  3560325168603410
Sun4608                 3.45      Douglas Tucker  4478071379779230
Sun4458                 7.00      Travis Walters  6011812112971322
Sun5260                11.84    Nathaniel Harris  4676137647685994
Sun2251                 6.15        Tonya Carter  4832732618637221
```

[ ]:
```
## With the help of reset_index() we can reset the index
df = df.reset_index()
```

[ ]:
```
df.head()
```

[ ]:
```
  Payment ID  total_bill   tip     sex smoker  day    time  size  \
0    Sun2959       16.99  1.01  Female     No  Sun  Dinner     2
1    Sun4608       10.34  1.66    Male     No  Sun  Dinner     3
2    Sun4458       21.01  3.50    Male     No  Sun  Dinner     3
3    Sun5260       23.68  3.31    Male     No  Sun  Dinner     2
4    Sun2251       24.59  3.61  Female     No  Sun  Dinner     4

   price_per_person          Payer Name         CC Number
0              8.49  Christy Cunningham  3560325168603410
1              3.45      Douglas Tucker  4478071379779230
2              7.00      Travis Walters  6011812112971322
3             11.84    Nathaniel Harris  4676137647685994
4              6.15        Tonya Carter  4832732618637221
```

3.0.1 ROWS

Let's now explore these same concepts but with Rows.

[ ]:
```
df.head()
```

[ ]:
```
  Payment ID  total_bill   tip     sex smoker  day    time  size  \
0    Sun2959       16.99  1.01  Female     No  Sun  Dinner     2
1    Sun4608       10.34  1.66    Male     No  Sun  Dinner     3
2    Sun4458       21.01  3.50    Male     No  Sun  Dinner     3
3    Sun5260       23.68  3.31    Male     No  Sun  Dinner     2
4    Sun2251       24.59  3.61  Female     No  Sun  Dinner     4

   price_per_person          Payer Name         CC Number
0              8.49  Christy Cunningham  3560325168603410
1              3.45      Douglas Tucker  4478071379779230
2              7.00      Travis Walters  6011812112971322
3             11.84    Nathaniel Harris  4676137647685994
4              6.15        Tonya Carter  4832732618637221
```

[ ]:
```
df = df.set_index('Payment ID')
```

[ ]:
```
df.head()
```

[ ]:
```
            total_bill   tip     sex smoker  day    time  size  \
Payment ID
Sun2959          16.99  1.01  Female     No  Sun  Dinner     2
Sun4608          10.34  1.66    Male     No  Sun  Dinner     3
Sun4458          21.01  3.50    Male     No  Sun  Dinner     3
Sun5260          23.68  3.31    Male     No  Sun  Dinner     2
Sun2251          24.59  3.61  Female     No  Sun  Dinner     4
```

```
             price_per_person          Payer Name         CC Number
Payment ID
Sun2959                  8.49  Christy Cunningham  3560325168603410
Sun4608                  3.45      Douglas Tucker  4478071379779230
Sun4458                  7.00      Travis Walters  6011812112971322
Sun5260                 11.84   Nathaniel Harris  4676137647685994
Sun2251                  6.15        Tonya Carter  4832732618637221
```

Grab a Single Row

[ ]:
```
# Index Number
df.iloc[0]
```

[ ]:
```
total_bill                          16.99
tip                                  1.01
sex                                Female
smoker                                 No
day                                   Sun
time                               Dinner
size                                    2
price_per_person                     8.49
Payer Name             Christy Cunningham
CC Number                3560325168603410
Name: Sun2959, dtype: object
```

[ ]:
```
# Name Based
df.loc['Sun2959']
```

[ ]:
```
total_bill                          16.99
tip                                  1.01
sex                                Female
smoker                                 No
day                                   Sun
time                               Dinner
size                                    2
price_per_person                     8.49
Payer Name             Christy Cunningham
CC Number                3560325168603410
Name: Sun2959, dtype: object
```

Grab Multiple Rows

[ ]:
```
df.iloc[0:4]
```

[ ]:
```
             total_bill   tip     sex smoker  day    time  size  \
Payment ID
Sun2959           16.99  1.01  Female     No  Sun  Dinner     2
Sun4608           10.34  1.66    Male     No  Sun  Dinner     3
Sun4458           21.01  3.50    Male     No  Sun  Dinner     3
```

```
Sun5260           23.68  3.31    Male     No  Sun  Dinner     2

             price_per_person          Payer Name         CC Number
Payment ID
Sun2959                  8.49  Christy Cunningham  3560325168603410
Sun4608                  3.45      Douglas Tucker  4478071379779230
Sun4458                  7.00      Travis Walters  6011812112971322
Sun5260                 11.84    Nathaniel Harris  4676137647685994
```

```
[ ]: df.loc[['Sun2959','Sun5260']]
```

```
[ ]:             total_bill   tip     sex smoker  day    time  size  \
Payment ID
Sun2959           16.99  1.01  Female     No  Sun  Dinner     2
Sun5260           23.68  3.31    Male     No  Sun  Dinner     2

             price_per_person          Payer Name         CC Number
Payment ID
Sun2959                  8.49  Christy Cunningham  3560325168603410
Sun5260                 11.84    Nathaniel Harris  4676137647685994
```

Remove Row Typically are datasets will be large enough that we won't remove rows like this since we won't know thier row location for some specific condition, instead, we drop rows based on conditions such as missing data or column values. The next lecture will cover this in a lot more detail.

```
[ ]: df.head()
```

```
[ ]:             total_bill   tip     sex smoker  day    time  size  \
Payment ID
Sun2959           16.99  1.01  Female     No  Sun  Dinner     2
Sun4608           10.34  1.66    Male     No  Sun  Dinner     3
Sun4458           21.01  3.50    Male     No  Sun  Dinner     3
Sun5260           23.68  3.31    Male     No  Sun  Dinner     2
Sun2251           24.59  3.61  Female     No  Sun  Dinner     4

             price_per_person          Payer Name         CC Number
Payment ID
Sun2959                  8.49  Christy Cunningham  3560325168603410
Sun4608                  3.45      Douglas Tucker  4478071379779230
Sun4458                  7.00      Travis Walters  6011812112971322
Sun5260                 11.84    Nathaniel Harris  4676137647685994
Sun2251                  6.15        Tonya Carter  4832732618637221
```

```
[ ]: df.drop('Sun2959',axis=0).head()
```

```
[ ]:               total_bill   tip     sex smoker  day    time  size  \
Payment ID
Sun4608             10.34  1.66    Male     No  Sun  Dinner     3
Sun4458             21.01  3.50    Male     No  Sun  Dinner     3
Sun5260             23.68  3.31    Male     No  Sun  Dinner     2
Sun2251             24.59  3.61  Female     No  Sun  Dinner     4
Sun9679             25.29  4.71    Male     No  Sun  Dinner     4

            price_per_person        Payer Name        CC Number
Payment ID
Sun4608                 3.45    Douglas Tucker  4478071379779230
Sun4458                 7.00    Travis Walters  6011812112971322
Sun5260                11.84  Nathaniel Harris  4676137647685994
Sun2251                 6.15      Tonya Carter  4832732618637221
Sun9679                 6.32        Erik Smith   213140353657882
```

Insert a New Row Pretty rare to add a single row like this. Usually you use pd.concat() to add many rows at once. You could use the .append() method with a list of pd.Series() objects, but you won't see us do this with realistic real-world data.

```
[ ]: one_row = df.iloc[0]
```

```
[ ]: one_row
```

```
[ ]: total_bill                          16.99
     tip                                  1.01
     sex                                Female
     smoker                                 No
     day                                   Sun
     time                               Dinner
     size                                    2
     price_per_person                     8.49
     Payer Name             Christy Cunningham
     CC Number                3560325168603410
     Name: Sun2959, dtype: object
```

```
[ ]: type(one_row)
```

```
[ ]: pandas.core.series.Series
```

```
[ ]: df.tail()
```

```
[ ]:               total_bill   tip     sex smoker  day    time  size  \
Payment ID
Sat2657             29.03  5.92    Male     No  Sat  Dinner     3
Sat1766             27.18  2.00  Female    Yes  Sat  Dinner     2
Sat3880             22.67  2.00    Male    Yes  Sat  Dinner     2
Sat17               17.82  1.75    Male     No  Sat  Dinner     2
```

```
Thur672          18.78  3.00  Female     No  Thur  Dinner     2

            price_per_person       Payer Name         CC Number
Payment ID
Sat2657                 9.68    Michael Avila  5296068606052842
Sat1766                13.59   Monica Sanders  3506806155565404
Sat3880                11.34       Keith Wong  6011891618747196
Sat17                   8.91     Dennis Dixon     4375220550950
Thur672                 9.39  Michelle Hardin  3511451626698139
```

4 Happy Programming!!!

```
[ ]:
```