5CS037 Concepts and Technologies of AI
Workshop-2
Pandas for Data Analysis.

Siman Giri

November 27, 2023

1. Pandas: Introduction.

1.1 What is pandas?

- Pandas is an open-source add-on modules to python which provides high-performance, easy-to-use data structure, and data analysis tools.

> [pandas] is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.
> -Wikipedia

- The pandas library contains several methods and functions for cleaning, manipulating and analyzing data.
- Though Pandas is built on top of the Numpy package, Numpy is suited for working with homogeneous numerical array data, Pandas is designed for working with tabular or heterogeneous data.

1.2 Use for Pandas!!!

Typically, the pandas library is used for:

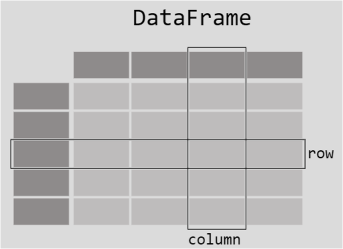
1. Data manipulation tasks such as missing values,filtering rows/columns, aggregating and mutating data.
2. Computing summary (Descriptive) statistics.
3. Computing correlation and distributions among columns in the data.
4. Visualizing the data with the help from the Matplotlib library.
5. Writing the cleaned and transformed data into CSV file or other database formats.

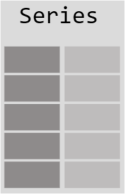
Importing the pandas:

```
import pandas as pd
```

1.3 Pandas Data Structure: Series and DataFrame

- There are two core components of the pandas library:
  - Series and DataFrame.
- A DataFrame is a two-dimensional object
  - comprising of tabular data organized in rows and columns
  - individual columns can be of different value types (numeric / string / Boolean etc.)
  - row indices: refers to individual rows (called index, usually integers if not defined otherwise).
  - column indices: refers to name(head) of each columns, if not defined otherwise.
- Each column in a DataFrame is a Series.





2.Creating, Reading and Writing Data with Pandas.

2.1 How do I Create, Read and Write Tabular Data?

- Creating a DataFrame/Series: A Pandas DataFrame can be created by converting the in-built python data structures such as lists, dictionaries etc. Example:

```python
#Transforming in-built data structures-DataFrame
#Style-1
import pandas as pd
pd.DataFrame({'Bob': ['I liked it.','It was awful'], 'Sue': ['Pretty good.', 'Bland.']})
#Style-2
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']},
             index=['Product A', 'Product B'])
```

2.1 How do I Create, Read and Write Tabular Data?

- Creating a DataFrame/Series: A Pandas DataFrame can be created by converting the in-built python data structures such as lists, dictionaries etc. Example:

```python
#Transforming in-built data structures-DataFrame
#Style-1
import pandas as pd
pd.DataFrame({'Bob': ['I liked it.','It was awful'
    ], 'Sue': ['Pretty good.', 'Bland.']})
#Style-2
pd.DataFrame({'Bob': ['I liked it.', 'It was awful
    .'], 'Sue': ['Pretty good.', 'Bland.']},
                index=['Product A', 'Product B'])
```

- Output for both the styles: Observe the Difference

	Bob	Sue
0	I liked it.	Pretty good.
1	It was awful.	Bland.

Figure: Output-Style:1

	Bob	Sue
Product A	I liked it.	Pretty good.
Product B	It was awful.	Bland.

Figure: Output-Style:2

2.1 How do I Create, Read and Write Tabular Data?

- Importing data from files: In the real world, a pandas DataFrame will typically be created by loading the datasets from CSV file, Excel file, etc.
  - Pandas provides the `read_csv()` function to read data stored as a csv file into a pandas DataFrame.
  - Pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, parquet, ...), each of them with the prefix `read_*`.
- The head/tail/info methods and the dtypes attribute are convenient for a first check.

```python
#Importing Data from file
import pandas as pd
# path to your dataset must be given to built in
    read_csv("Your path") function.
dataset = pd.read_csv("/data/Week02/bank.csv")
dataset.head()
dataset.tail()
dataset.info()
# Run the above code and observe the output.
```

2.1 How do I Create, Read and Write Tabular Data?

- Writing Data: Whereas `read_*` functions are used to read data to pandas, the `to_*` methods are used to store data.
- The `to_csv("path+file name", index=false)` method stores the data as an csv file.
  - `path`: Where you wan to store the created file.
  - `file name`: in the name you want to store the file.
  - `index:boolean`: store the index or not.
- Pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, parquet, . . .), each of them with the prefix `to_*`.

```
#Importing Data from file
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'],'City'
    : ['New York', 'San Francisco', 'Los Angeles'
    ]}
df = pd.DataFrame(data) # creating a DataFrame
#Writing DataFrame to csv.
df.to_csv('output.csv', index=False)
# Run the above code and observe the output.
```

3.Do something with a DataFrame or Series.

3.1 Attributes of Pandas DataFrame.

Some of the attributes of the Pandas DataFrame class are the following:

attributes	definition	Syntax
dtypes	data-types of columns	`dataset.dtypes`
columns	name of columns	`dataset.columns`
axes	row index $\times$ col index	`dataset.axes`
ndim	Dimension(2-DF and 1-Series)	`dataset.ndim`
size	number of elements - DataFrame	`dataset.size`
shape	tuple (rows, cols)	`dataset.shape`
values	NumPy Representation	`dataset.values`

Table: Attributes @ Pandas DataFrame

3.2 Methods of Pandas DataFrame.

Some of the popular methods of the Pandas DataFrame class are the following:

methods-Syntax	definition
head(n)/tail(n) `dataset.head(2)`	n rows from top or bottom
sample()-`dataset.sample(n)`	n random samples from dataset
max()/,min() `dataset["column"].max()`	maximum or minimum of numeric column
mean()/median()/std() `dataset["column"].mean()`	mean or median or std of numeric column
describe() `dataset.describe()`	summary statistics of numeric columns in dataset.

Table: (some)Methods @ Pandas DataFrame

3.2 Methods of Pandas DataFrame.

methods-Syntax	definition
unique() `dataset.column.unique()`	unique values of column
map(arg) `dataset["column"].map(arg)` {arg:function,dict,col.}	map distinct values of a column to another set of corresponding values.
apply() `dataset["col"].apply(func)`	takes a function and applies to all values of column

Table: (some)Methods @ Pandas DataFrame

For Associated examples and python implementation of all the attributes and methods also check provided code file.

3.2.1 Methods of Pandas DataFrame:drop()

drop(): Probably!!! the most important methods used in data manipulation.

Removing Rows from DataFrame

`dataset.dropna()|` : Removes all rows with (at least) one missing values.{used: shaldomly}

```
import pandas as pd
# Assuming df is your DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],'City': ['New York', 'San Francisco', 'Los Angeles']}
df = pd.DataFrame(data)
# Drop a specific row by index
df = df.drop(1)
# Drop rows based on a condition.
df = df[df['city'] = "New York"]
# Reset index after dropping rows
df = df.reset_index(drop=True)
```

3.2.2 Methods of Pandas DataFrame:drop()

Removing Columns from DataFrame

`dataset.dropna(axis=1)` | : Removes all columns with (at least) one missing values.{used shaldomly}

```python
import pandas as pd
# Assuming df is your DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],'City': ['
    New York', 'San Francisco', 'Los Angeles']}
df = pd.DataFrame(data)
# Drop a specific column by name
df = df.drop('city', axis=1)
# Drop multiple columns by names
df = df.drop(['Name', 'City'], axis=1)
# Drop columns by index
df = df.drop(df.columns[1], axis=1)
```

4.Data Cleaning and Preparation.

4.1 Missing Values

Missing values in a dataset can occur due to several reasons.

Types of Missing Values:

1. Missing Completely at Random (MCAR): The probability of being missing is the same for all cases,
   - missingness occurs with-out any systematic pattern or dependence on other varaibles.
   - Randomly deleting survey responses without considering content.
2. Missing at Random (MAR): If the probability of being missing is the same only within groups defined by the observed data,
   - missingness is related to other observed variables in the dataset.
   - For example, when placed on a soft surface, a weighing scale may produce more missing values than when placed on a hard surface. Such data are thus not MCAR.
3. Missing not at Random (MNAR): MNAR means that the probability of being missing varies for reasons that are unknown to us.

4.2 Handling the Missing Values:Identifying Missing Values

- Missing values in a Pandas DataFrame can be identified with the `dataset.isnull()` method.
- Total number of missing values in each column can be found using syntax `dataset.isnull().sum()`.
- The easiest fix for handling missing data might be using `dataset.dropna()` methods, which drops the observation that even have a single missing value.

```python
import pandas as pd
from sklearn.datasets import load_iris
import numpy as np
iris = load_iris() # Load the Iris dataset
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['
    target']], columns=iris['feature_names'] + ['
    target'])
np.random.seed(42) # Introduce missing values randomly
mask = np.random.rand(*iris_df.shape) < 0.1 # 10%
iris_df[mask] = np.nan
print("Missing Values in Iris Dataset:")
print(iris_df.isnull().sum())
```

4.2 Handling the Missing Values:Data Imputations

The easiest fix for handling missing data might be using `dataset.dropna()` methods, which drops the observation that even have a single missing value.

Data Imputations Techniques: The best way to impute the data will depend on the problem, and the assumptions taken. Below we present few techniques:

1. Naive Method: Filling the missing value of a column by coping the value of the previous non-missing observation.
   - Syntax: `dataset.fillna(method = "ffill")`
2. Imputing with the mean/median/constant: Missing values in the column can be imputed(filled) using the mean/median/constant of the non-missing values in the column.{constant can be any values such as 0
   - Syntax: `dataset.column.fillna(dataset.column.mean())`

{ Please check python documentation for more such imputations techniques}

4.3 Data Imputations: Code Example

We will try to fill missing values form slide 18.Dataset is `iris_df`. Run the following code and observe the output:

```
# Contd from code @ slide 18
# Filling missing values with forward fill (ffill),
    mean, median, and 0
iris_df_ffill = iris_df.ffill()
iris_df_mean = iris_df.fillna(iris_df.mean())
iris_df_median = iris_df.fillna(iris_df.median())
iris_df_zero = iris_df.fillna(0)
# Expand iris_df with filled columns
iris_df_expanded = pd.concat([iris_df, iris_df_ffill.
    add_suffix('_ffill'), iris_df_mean.add_suffix('
    _mean'),iris_df_median.add_suffix('_median'),
    iris_df_zero.add_suffix('_zero')], axis=1)
# Display the head of the expanded DataFrame
print("\nDataset after Filling Missing Values:")
print(iris_df_expanded.head())
```

5. Data Transformation

5.1 Data Transformations: Data Scaling

Standard Scaling: also known as z-score normalization) is a technique used to standardize the range of features by transforming them to have a mean of 0 and a standard deviation of 1.

$$z = \frac{x - \bar{x}}{SD}$$

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris() # Load the Iris dataset
iris_df = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
# Standard Scaling
iris_standard_scaled = (iris_df - iris_df.mean()) / iris_df.std()
print("Original Iris DataFrame:")
print(iris_df.head())
print("\nStandard Scaled Iris DataFrame:")
print(iris_standard_scaled.head()) # Display scaled data
```

5.1 Data Transformations: Data Scaling

Min-Max Scaling: Min-Max scaling (also known as feature scaling or min-max normalization) is a technique used to scale and center the values of a feature in a specific range, usually between 0 and 1.

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

```python
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris() # Load the Iris dataset
iris_df = pd.DataFrame(data=iris['data'], columns=iris
    ['feature_names'])
# Min-Max Scaling using Pandas
iris_minmax_scaled = (iris_df - iris_df.min()) / (
    iris_df.max() - iris_df.min())
print("Original Iris DataFrame:")
print(iris_df.head())
print("\nMin-Max Scaled Iris DataFrame:")
print(iris_minmax_scaled.head()) # Display scaled data
```

5.2 Data Transformations: Encoding

Ordinal Encoding:

- Ordinal encoding is used for categorical data with a meaningful order or ranking.
- Each category is assigned a numerical value based on its order.
- Example: Low, Medium, High can be encoded as 1, 2, 3.

```python
import pandas as pd
# Sample DataFrame with ordinal categories
df = pd.DataFrame({'Category': ['Low', 'Medium', 'High', 'Low', 'High']})
# Ordinal encoding using map
ordinal_mapping = {'Low': 1, 'Medium': 2, 'High': 3}
df['Category_Ordinal'] = df['Category'].map(
    ordinal_mapping)
print(df)
```

5.2 Data Transformations: Encoding

One-Hot Encoding:

- In one-hot encoding, each category is represented as a binary vector (0 or 1) in which all elements are zero except for the index that corresponds to the category.
- If there are *n* categories, each category is represented by a vector of length *n* with all zeros except for a 1 at the index corresponding to the category.

```
import pandas as pd
df_municipalities = pd.DataFrame({'Municipality': ['Kathmandu', 'Bhaktapur', 'Lalitpur', 'Madhyapur Thimi', 'Kirtipur']})
one_hot_encoding = pd.get_dummies(df_municipalities['Municipality'], prefix='Municipality')
df_encoded = pd.concat([df_municipalities, one_hot_encoding], axis=1)
print(df_encoded)# Display the result
```

6.Exercises.

Task Set-I: DataFrame Reading and Writing.

Answer the following:
Dataset: `"bank.csv"|.`

- `Load the provided dataset and import in pandas DataFrame.`
- `Check info of the DataFrame and identify following:`
  1. `columns with dtypes=object|`
  2. `unique values of those columns.`
  3. `check for the total number of null values in each column.`
- `Drop all the columns with dtypes int and store in new DataFrame, also write the DataFrame in ".csv" with name "banknumericdata.csv"`
- `Read "banknumericdata.csv" and Find the summary statistics.`

Task Set-II: Data Imputations

Answer the following:
Dataset: `"medical_Student.csv"`.

- Load the provided dataset and import in pandas DataFrame.
- Check info of the DataFrame and identify column with missing (null) values.
- For the column with missing values fill the values using various techniques we discussed above. Try to explain why did you select the particular methods for particular column.
- Check for any duplicate values present in Dataset and do necessary to manage the duplicate items.
  {Hint: `dataset.duplicated.sum()`}

Task Set-III: Data Transformations

Transform variables according to the following instructions:
Dataset: `"performance.csv"`.

- "School", "internet", "activities", into binary: 0 or 1 (create new columns without overwriting the existing ones).
- "Medu","reason", "guardian","studytime",, and "health" into ordinal numbers based on the number cases in the data set (create news columns without overwriting the existing ones).
- Convert column "age" to interval datatype. i.e. Create a new column name category_age whose values should be based on the frequency in the column "age", You can create categorical data with following interval.

  interval1: [15-17];interval2: [18-20]; interval3: [21-all]

- Create a new column name passed (yes or no) whose values should be based on the values present in the G3 column ($>= 8$–$yes, < -no$).