AI/ML for Climate Workshop

International Livestock Research Institute (ILRI)

---

hide: - toc

📕 Pandas for Climate and Meteorology

🚀 Interactive Learning

launch binder

Click the Binder button above to launch an interactive Jupyter notebook for NumPy and Pandas climate data analysis!

A practical, beginner-friendly notebook that teaches Pandas using climate/weather examples.

Agenda

- Learn the basics of the Pandas library.
- Understand how to work with Series and DataFrames.
- Perform data manipulation tasks like filtering, sorting, and aggregating.
- Learn how to handle missing data.

Introduction to Pandas: - `Pandas` is a powerful library for data manipulation and analysis. - The two main data structures are: - `Series`: A one-dimensional labeled array. - `DataFrame`: A two-dimensional table-like data structure.

```
# Set working directory

import os
os.chdir("c:\\Users\\yonas\\Documents\\ICPAC\\python-climate")
```

```
processed_data_dir = os.path.join("data", "processed")
raw_data_dir = os.path.join("data", "raw")
```

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

```
# Check the version of Pandas
print(pd.__version__)
```

Output:

```
2.3.3
```

---

1. Pandas Series

- A `Series` is a one-dimensional array with labels (index).
- It is similar to a column in a spreadsheet.

```
# Create a Series from a list
data = [10, 20, 30, 40]
series = pd.Series(data, index=["A", "B", "C", "D"])
print(series)
```

Output:

```
A    10
B    20
C    30
D    40
dtype: int64
```

```
# Accessing elements in a Series
print(series["B"])  # Output: 20
```

Output:

```
20
```

---

2. Pandas DataFrame

- A DataFrame is a two-dimensional structure with rows and columns.
- Think of it as a table in a database or an Excel sheet.
- Create a DataFrame from a dictionary.

```
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35],
    "City": ["New York", "Los Angeles", "Chicago"]
}

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

> *Output:*

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
```

Viewing DataFrame

- Use DataFrame.head() and DataFrame.tail() to view the top and bottom rows of the frame respectively.

```
# Print the head of the DataFrame
df.head(2)
```

> *Output:*

```
    Name  Age         City
0  Alice   25     New York
1    Bob   30  Los Angeles
```

```
# Print the tail of the DataFrame
df.tail(2)
```

> *Output:*

```
      Name  Age         City
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
```

Display DataFrame information

```
df.info()
```

> *Output:*

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Name    3 non-null      object
 1   Age     3 non-null      int64 
 2   City    3 non-null      object
dtypes: int64(1), object(2)
memory usage: 204.0+ bytes
```

Display the data types

```
df.dtypes
```

> *Output:*

```
Name    object
Age      int64
City    object
dtype: object
```

Display statistical summary of DataFrame

```
df.describe()
```

> *Output:*

```
        Age
count   3.0
mean   30.0
```

```
std      5.0
min     25.0
25%     27.5
50%     30.0
75%     32.5
max     35.0
```

Display the DataFrame index and columns

- Use `DataFrame.index` or `DataFrame.columns` to get the DataFrame index and columns
- Get the DataFrame index

```
df.index
```

> *Output:*

```
RangeIndex(start=0, stop=3, step=1)
```

Get the DataFrame columns

```
df.columns
```

> *Output:*

```
Index(['Name', 'Age', 'City'], dtype='object')
```

Accessing a DataFrame

```
# Accessing a column
df["Name"]
```

> *Output:*

```
0      Alice
1        Bob
2    Charlie
Name: Name, dtype: object
```

```
# Accessing rows using loc
```

```
df.loc[1]
```

> *Output:*

```
Name              Bob
Age                30
City      Los Angeles
Name: 1, dtype: object
```

```
# Accessing rows using iloc
df.iloc[2]
```

> *Output:*

```
Name      Charlie
Age            35
City      Chicago
Name: 2, dtype: object
```

Adding New Columns

```
df['Experience'] = np.random.randint(1, 5, size=len(df))

df.head()
```

> *Output:*

```
      Name  Age         City  Experience
0    Alice   25     New York           2
1      Bob   30  Los Angeles           2
2  Charlie   35      Chicago           2
```

Dropping Columns

- Use the drop() method to remove columns.
- Specify axis=1 to indicate columns.
- Use inplace=True to modify the DataFrame directly (optional).

```
# Dropping a Single Column
df_dropped = df.drop("City", axis=1)
```

```
df_dropped
```

Output:

```
      Name  Age  Experience
0    Alice   25           2
1      Bob   30           2
2  Charlie   35           2
```

```
# Alternatively, drop in place (modifies the original DataFrame)
df.drop("City", axis=1, inplace=True)
df
```

Output:

```
      Name  Age  Experience
0    Alice   25           2
1      Bob   30           2
2  Charlie   35           2
```

```
# Drop multiple rows
df = pd.DataFrame(data)  # Recreate the original DataFrame
df_dropped = df.drop([0, 2], axis=0)
df_dropped
```

Output:

```
  Name  Age         City
1  Bob   30  Los Angeles
```

---

4. Synthetic climate dataset (multi-station, 5 years daily)

```
# Initializing random number generator and station data
rng = np.random.default_rng(42)

# Station metadata: (Station ID, Name, Latitude, Longitude)
stations = [
    ("ADD", "Addis Ababa",      9.03,   38.74),
    ("NBO", "Nairobi",         -1.286,  36.817),
    ("KRT", "Khartoum",        15.500,  32.560),
    ("DAR", "Dar es Salaam",   -6.792,  39.208),
```

```
    ("MOG", "Mogadishu",        2.046,  45.318),
]
```

```
dates = pd.date_range("2015-01-01", "2019-12-31", freq="D")
dates
```

Output:

```
DatetimeIndex(['2015-01-01', '2015-01-02', '2015-01-03', '2015-01-04',
               '2015-01-05', '2015-01-06', '2015-01-07', '2015-01-08',
               '2015-01-09', '2015-01-10',
               ...
               '2019-12-22', '2019-12-23', '2019-12-24', '2019-12-25',
               '2019-12-26', '2019-12-27', '2019-12-28', '2019-12-29',
               '2019-12-30', '2019-12-31'],
              dtype='datetime64[ns]', length=1826, freq='D')
```

```
rows = []
for sid, name, lat, lon in stations:
    doy = dates.dayofyear.values
    season = 6*np.sin(2*np.pi*(doy-1)/365.0)
    lat_grad = 0.2*(10 - lat)
    temp = 24 + season + lat_grad + rng.normal(0,1.0,len(dates))
    season_r = 2 + 2*np.sin(2*np.pi*(doy-30)/365.0) + 1.5*np.sin(4*np.pi*(doy-30)/365.(
    precip = rng.gamma(1.2, np.clip(season_r,0.2,None), size=len(dates))
    precip = np.maximum(precip - 1.5, 0)
    wind = np.abs(rng.normal(4.0,1.5,len(dates)))
    rh = np.clip(65 + 10*np.sin(2*np.pi*(doy-100)/365.0) + rng.normal(0,5,len(dates)),
    df_s = pd.DataFrame({
        "date": dates, "station_id": sid, "station_name": name, "lat": lat, "lon": lon,
        "t2m_c": temp, "precip_mm": precip, "wind_ms": wind, "rh_pct": rh
    })
    rows.append(df_s)
df = pd.concat(rows, ignore_index=True).set_index("date").sort_index()
df.head()
```

Output:

```
           station_id  station_name     lat     lon      t2m_c  precip_mm  \
date
2015-01-01        ADD   Addis Ababa   9.030  38.740  23.492607        0.0
2015-01-01        MOG     Mogadishu   2.046  45.318  25.061257        0.0
2015-01-01        NBO       Nairobi  -1.286  36.817  26.761467        0.0
2015-01-01        KRT      Khartoum  15.500  32.560  23.538721        0.0
2015-01-01        DAR  Dar es Salaam  -6.792  39.208  27.539051        0.0

             wind_ms     rh_pct
```

```
date
2015-01-01  3.828286  62.610573
2015-01-01  1.512537  59.372599
2015-01-01  5.396700  46.460131
2015-01-01  2.196587  51.063869
2015-01-01  2.585994  55.125804
```

Pandas Series

```
precip_add = df.loc[df["station_id"]=="ADD", "precip_mm"]
precip_add.describe(), precip_add.head()
```

Output:

```
(count    1826.000000
 mean        1.598806
 std         3.167576
 min         0.000000
 25%         0.000000
 50%         0.000000
 75%         1.880291
 max        39.516660
 Name: precip_mm, dtype: float64,
 date
 2015-01-01    0.0
 2015-01-02    0.0
 2015-01-03    0.0
 2015-01-04    0.0
 2015-01-05    0.0
 Name: precip_mm, dtype: float64)
```

Pandas DataFrame

```
df.sample(5)
```

Output:

```
           station_id   station_name  latitude  longitude      t2m_c  \
date
2017-03-10        DAR  Dar es Salaam    -6.792     39.208  33.341576
2019-03-24        ADD    Addis Ababa     9.030     38.740  30.115611
2016-09-04        KRT       Khartoum    15.500     32.560  17.238317
2019-11-05        ADD    Addis Ababa     9.030     38.740  19.031074
2017-01-06        NBO        Nairobi    -1.286     36.817  27.216106
```

```
            precip_mm   wind_ms     rh_pct
date
2017-03-10   2.085262  5.681299  56.673766
2019-03-24   4.235970  6.746884  46.397766
2016-09-04   0.445245  3.434693  79.932452
2019-11-05   0.000000  4.880244  64.469575
2017-01-06   0.000000  5.269936  53.219116
```

Viewing DataFrame

```
df.head(3)
```

Output:

```
           station_id station_name  latitude  longitude      t2m_c  precip_mm  \
date
2015-01-01        ADD  Addis Ababa     9.030     38.740  23.492607        0.0
2015-01-01        MOG    Mogadishu     2.046     45.318  25.061257        0.0
2015-01-01        NBO      Nairobi    -1.286     36.817  26.761467        0.0

             wind_ms     rh_pct
date
2015-01-01  3.828286  62.610573
2015-01-01  1.512537  59.372599
2015-01-01  5.396700  46.460131
```

```
df.tail(2)
```

Output:

```
           station_id station_name  latitude  longitude      t2m_c  precip_mm  \
date
2019-12-31        KRT     Khartoum    15.500     32.560  23.972350        0.0
2019-12-31        MOG    Mogadishu     2.046     45.318  25.181895        0.0

             wind_ms     rh_pct
date
2019-12-31  3.355706  53.422156
2019-12-31  2.094387  45.893028
```

Accessing a DataFrame

```
df.loc["2017-07":"2017-07-10", ["station_id","t2m_c","precip_mm"]].head()
```

Output:

```
           station_id      t2m_c  precip_mm
date
2017-07-01        MOG  26.639599   0.000000
2017-07-01        DAR  27.402184   0.000000
2017-07-01        KRT  25.944065   0.801203
2017-07-01        ADD  24.904493   1.306163
2017-07-01        NBO  26.941912   1.541067
```

Adding New Columns

```
df2 = df.copy()
df2["t2m_k"] = df2["t2m_c"] + 273.15
e = (df2["rh_pct"]/100.0)*6.105*np.exp(17.27*df2["t2m_c"]/(237.7+df2["t2m_c"]))
df2["apparent_c"] = df2["t2m_c"] + 0.33*e - 0.70*df2["wind_ms"] - 4.0
```

```
df2
```

Output:

```
           station_id   station_name  latitude  longitude      t2m_c  \
date
2015-01-01        ADD    Addis Ababa     9.030     38.740  23.492607
2015-01-01        MOG      Mogadishu     2.046     45.318  25.061257
2015-01-01        NBO        Nairobi    -1.286     36.817  26.761467
2015-01-01        KRT       Khartoum    15.500     32.560  23.538721
2015-01-01        DAR  Dar es Salaam    -6.792     39.208  27.539051
...               ...            ...       ...        ...        ...
2019-12-31        DAR  Dar es Salaam    -6.792     39.208  27.647587
2019-12-31        NBO        Nairobi    -1.286     36.817  27.582417
2019-12-31        ADD    Addis Ababa     9.030     38.740  24.332698
2019-12-31        KRT       Khartoum    15.500     32.560  23.972350
2019-12-31        MOG      Mogadishu     2.046     45.318  25.181895

            precip_mm   wind_ms     rh_pct       t2m_k  apparent_c
date
2015-01-01        0.0  3.828286  62.610573  296.642607   22.775580
2015-01-01        0.0  1.512537  59.372599  298.211257   26.213101
2015-01-01        0.0  5.396700  46.460131  299.911467   24.357177
2015-01-01        0.0  2.196587  51.063869  296.688721   22.877733
2015-01-01        0.0  2.585994  55.125804  300.689051   28.401328
...               ...       ...        ...         ...         ...
2019-12-31        0.0  5.313968  53.584144  300.797587   26.454868
```

```
2019-12-31        0.0  5.854557  59.454385  300.732417   26.698865
2019-12-31        0.0  7.754628  47.303117  297.482698   19.642227
2019-12-31        0.0  3.355706  53.422156  297.122350   22.859794
2019-12-31        0.0  2.094387  45.893028  298.331895   24.550967

[9130 rows x 10 columns]
```

Dropping Columns

```
df2.drop(columns=["apparent_c"]).head(2)
```

Output:

```
           station_id station_name    lat     lon      t2m_c  precip_mm  \
date
2015-01-01        ADD  Addis Ababa  9.030  38.740  24.498717        0.0
2015-01-01        MOG    Mogadishu  2.046  45.318  26.021727        0.0

             wind_ms     rh_pct       t2m_k
date
2015-01-01  6.011673  51.363205  297.648717
2015-01-01  3.413349  47.676047  299.171727
```

```
### Renaming Columns
df.rename(columns={"lon": "longitude", "lat": "latitude"}, inplace=True)
df.head(2)
```

Output:

```
           station_id station_name  latitude  longitude      t2m_c  precip_mm  \
date
2015-01-01        ADD  Addis Ababa     9.030     38.740  23.492607        0.0
2015-01-01        MOG    Mogadishu     2.046     45.318  25.061257        0.0

             wind_ms     rh_pct
date
2015-01-01  3.828286  62.610573
2015-01-01  1.512537  59.372599
```

Dropping Rows

```
tmp = df.copy()
mask = (tmp["station_id"]=="MOG") & (tmp.index<"2015-03-01")
```

```
tmp = tmp.drop(index=tmp[mask].index)
tmp.loc[tmp["station_id"]=="MOG"].head()
```

Output:

```
           station_id station_name    lat     lon      t2m_c  precip_mm  \
date
2015-03-01        MOG    Mogadishu  2.046  45.318  31.029596   3.475400
2015-03-02        MOG    Mogadishu  2.046  45.318  32.013884   6.166426
2015-03-03        MOG    Mogadishu  2.046  45.318  29.216900   1.129071
2015-03-04        MOG    Mogadishu  2.046  45.318  30.966538   6.251592
2015-03-05        MOG    Mogadishu  2.046  45.318  28.853817   1.759359

             wind_ms     rh_pct
date
2015-03-01  4.783702  62.084746
2015-03-02  3.307198  53.210218
2015-03-03  5.781958  78.253268
2015-03-04  2.453640  58.970132
2015-03-05  3.803125  50.789886
```

---

5. Reading and Writing Data

```
df.to_csv("data/processed/climate_data.csv")
```

```
df_csv = pd.read_csv("data/processed/climate_data.csv", parse_dates=["date"], index_col
df_csv.head(2)
```

Output:

```
           station_id station_name    lat     lon      t2m_c  precip_mm  \
date
2015-01-01        ADD  Addis Ababa  9.030  38.740  24.498717        0.0
2015-01-01        MOG    Mogadishu  2.046  45.318  26.021727        0.0

             wind_ms     rh_pct
date
2015-01-01  6.011673  51.363205
2015-01-01  3.413349  47.676047
```

Writing Excel Files

- To save a DataFrame to an Excel file, use the to_excel() function.

```
# Save DataFrame to an Excel file
df.to_excel("data/processed/climate_data.xlsx", index=False)  # Save without including
```

```
# Read excel file
df_excel = pd.read_excel("data/processed/climate_data.xlsx", index_col=None)
df_excel.head(2)
```

Output:

```
    Name  Age         City
0  Alice   25     New York
1    Bob   30  Los Angeles
```

Reading JSON Files

```
df_json = pd.read_json("data/raw/weather.json")
df_json
```

Output:

```
                                             weather
0  {'cityId': 1, 'cityName': 'London', 'currentCo...
1  {'cityId': 2, 'cityName': 'Newcastle', 'curren...
2  {'cityId': 3, 'cityName': 'Edinburgh', 'curren...
3  {'cityId': 4, 'cityName': 'Manchester', 'curre...
4  {'cityId': 5, 'cityName': 'Glasgow', 'currentC...
5  {'cityId': 6, 'cityName': 'Bristol', 'currentC...
6  {'cityId': 7, 'cityName': 'Liverpool', 'curren...
7  {'cityId': 8, 'cityName': 'Sheffield', 'curren...
```

```
df_json.head(1)
```

Output:

```
                                             weather
0  {'cityId': 1, 'cityName': 'London', 'currentCo...
```

```
df_json["weather"].values[0]
```

Output:

```
{'cityId': 1,
 'cityName': 'London',
 'currentConditions': 'Cloud',
 'temperature': 25,
 'windSpeed': 27,
 'windDirection': 'Easterly',
 'windChillFactor': 11}
```

Write JSON file

```
data = {
    "Station": ["ADD", "NBO", "KRT"],
    "Temperature_C": [20.5, 22.3, 30.1],
    "Precipitation_mm": [5.0, 0.0, 2.5]
}

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

Output:

```
  Station  Temperature_C  Precipitation_mm
0     ADD           20.5               5.0
1     NBO           22.3               0.0
2     KRT           30.1               2.5
```

```
df_climate.to_json("data/processed/climate_data.json", orient="records", indent=4)
```

---

6. Data Manipulation: Filtering

```
july = df[df.index.month==7]
july.head()
```

Output:

```
           station_id   station_name     lat     lon      t2m_c  precip_mm  \
date
2015-07-01        MOG      Mogadishu   2.046  45.318  25.298206   0.000000
2015-07-01        KRT       Khartoum  15.500  32.560  23.247284   0.000000
2015-07-01        DAR  Dar es Salaam  -6.792  39.208  27.521499   0.853249
2015-07-01        ADD    Addis Ababa   9.030  38.740  24.568293   1.168959
2015-07-01        NBO        Nairobi  -1.286  36.817  26.394597   0.000000
```

```
              wind_ms     rh_pct
date
2015-07-01  2.166290  83.351931
2015-07-01  3.881973  73.649941
2015-07-01  4.865813  77.754760
2015-07-01  1.748411  77.138895
2015-07-01  5.213327  73.447242
```

```
wet_nbo = df[(df["station_id"]=="NBO") & (df["precip_mm"]>=1.0)]
wet_nbo.head(2)
```

Output:

```
           station_id station_name    lat     lon      t2m_c  precip_mm  \
date
2015-01-10        NBO      Nairobi -1.286  36.817  26.365741   1.420092
2015-01-13        NBO      Nairobi -1.286  36.817  27.920302   1.275217

              wind_ms     rh_pct
date
2015-01-10  2.993146  59.633344
2015-01-13  5.455883  43.237625
```

Data Manipulation: Sorting

```
df[df["station_id"]=="ADD"].sort_values("precip_mm", ascending=False).head(5)[["statio
```

Output:

```
           station_id  precip_mm
date
2018-03-08        ADD  39.516660
2016-03-30        ADD  30.544245
2016-03-11        ADD  26.025224
2016-04-26        ADD  23.269092
2017-05-04        ADD  21.152791
```

Data Manipulation: Aggregations

```
# Compute annual mean temperature for each station
annual_mean = df.groupby("station_id").resample("YE")["t2m_c"].mean().rename("t_ann").r
```

```
annual_mean.head()
```

Output:

```
C:\Users\yonas\AppData\Local\Temp\ipykernel_58044\4160454045.py:1: FutureWarning: 'A' i
  annual_mean = df.groupby("station_id").resample("A")["t2m_c"].mean().rename("t_ann").

  station_id       date      t_ann
0        ADD 2015-12-31  24.185020
1        ADD 2016-12-31  24.110184
2        ADD 2017-12-31  24.210580
3        ADD 2018-12-31  24.210628
4        ADD 2019-12-31  24.042635
```

7. Aggregations: Monthly Climatology & Anomalies

```
df_an = df.copy()

df_an["t_mon_clim"] = (
    df.groupby(["station_id", df.index.month])["t2m_c"]
      .transform("mean")
)

# Compute anomalies
df_an["month"] = df_an.index.month
df_an["t2m_anom"] = df_an["t2m_c"] - df_an["t_mon_clim"]

df_an[["station_id","t2m_c","t_mon_clim","t2m_anom"]].head()
```

Output:

```
           station_id      t2m_c  t_mon_clim  t2m_anom
date
2015-01-01        ADD  23.492607   25.658658 -2.166051
2015-01-01        MOG  25.061257   27.194053 -2.132796
2015-01-01        NBO  26.761467   27.848914 -1.087447
2015-01-01        KRT  23.538721   24.385507 -0.846786
2015-01-01        DAR  27.539051   28.999603 -1.460551
```

Selecting Specific Rows and Columns

```
df.loc[df["station_id"]=="KRT", ["t2m_c","precip_mm"]].tail(5)
```

Output:

```
                t2m_c  precip_mm
date
2019-12-27  21.434750        0.0
2019-12-28  22.750775        0.0
2019-12-29  22.606622        0.0
2019-12-30  23.431164        0.0
2019-12-31  23.972350        0.0
```

Exercise

Exercises

1) **Filtering & Selection**:
- Extract JJA for Khartoum; compute mean `t2m_c`.
- List top-10 daily rainfall events for Nairobi.

2) **Aggregations & Resampling**:
- Compute **monthly climatology** (calendar-month means) of temperature for each station.
- Compute **7-day rolling precipitation sum** for Addis and find the maximum value and its date.

3) **Anomalies**:
- Build daily temperature anomalies relative to station-wise monthly climatology; plot a 60-day window (optional).
- Compute the **wet-day fraction** (precip >= 1 mm) per month per station.

4) **Missing Data**:
- Introduce a 10-day missing block in 2018 for Dar es Salaam; compare `dropna`, `ffill/bfill`, and time interpolation (`interpolate(method='time')`) for temperature.
- Recompute monthly precipitation sums after filling and compare to baseline.

5) **Joins & Regions**:
- Add a `region` column and compute **region monthly** temperature means.
- Compute **area-weighted** (cos(lat)) region means using station latitudes.

6) **I/O**:
- Save anomalies to `anoms.parquet` (if supported) and re-load.
- Export monthly climatology to CSV.