Subject: Solar Radiation Prediction Using Machine Learning

Objectives

The main objective of this project is to build a machine learning model that accurately predicts solar radiation. One of the key challenges in predicting solar radiation is weather variation, as each region has its own unique weather patterns, precipitation amounts and types, and temperature ranges.

Target:

- Solar radiation (shortwave/global): watts per square meter

Features (Select the most relevant ones based on your analysis):

- **Temperature**: Air temperature at 2 meters above ground
- **Humidity**: Relative humidity at 2 meters above ground
- **Pressure**: Surface pressure
- **Precipitation**: Total precipitation (rain, snow, etc.) over the preceding hour
- **Cloud cover**: Total cloud cover as a fraction of the area
- **Wind direction**: Wind direction
- **Wind speed**: Wind speed at 10 meters above ground
- **Wind gusts**: Maximum gust speed at 10 meters above ground over the preceding hour
- **Sunrise/Sunset**: Local time
- **GPS coordinates**: Latitude, longitude, elevation

Data Sources:

1. **Solar Energy Production Data**: https://data.calgary.ca/Environment/Solar-Energy-Production/ytdn-2qsp/data_preview
2. **Solar Farms Sites Data**: https://data.calgary.ca/Environment/Solar-Photovoltaic-Sites/csgq-e555/data_preview

---

Part One (Jupyter Notebook file expected):

1. Merge both datasets to create a dataframe containing the locations and solar production for each site. Select only the relevant columns.
2. Data cleaning and preprocessing.
3. Data analysis.

4. Feature selection: Use the API function to extract weather data from the historical weather API: https://open-meteo.com/en/docs/historical-weather-api#hourly=dew_point_2m,apparent_temperature,precipitation,rain
5. Model selection.
6. Hyperparameter selection and model training.
7. Predict on the test set.
8. Save the model as a `.pickle` file for later use.

Part Two (Airflow Pipeline):

1. Install Docker and the Astro CLI following these steps: https://www.astronomer.io/docs/astro/cli/install-cli
2. Run Airflow locally.
3. Create a Python file in the generated Astro project.
4. Create a DAG (Directed Acyclic Graph) for the steps mentioned in Part One. You can design your tasks and pipeline schema as you see fit; there is no strict requirement for how it should look.

Related Articles:

1. American-CSE 2023 Paper on Solar Radiation
2. MDPI Solar Radiation Prediction Paper

Additional Resources:

- Airflow book: https://biconsult.ru/files/Data_warehouse/Bas_P_Harenslak%2C_Julian_Rutger_de_Ruiter_Data_Pipelines_with_Apache.pdf
- Machine learning book: https://nessie.ilab.sztaki.hu/~kornai/2020/AdvancedMachineLearning/Ng_MachineLearningYearning.pdf

---

Important Notes:

- The main objective of the project is to demonstrate creativity and explain your approach clearly in English. Each group member should speak for at least 5 minutes during the presentation.
- The steps listed above are not strict; you are encouraged to develop your own approach to the project.

- Groups should have a maximum of 5 members. For new students, it is recommended to join a group that has participated in previous sessions to avoid difficulties. Exceptions can be made for groups with more than 5 people.

API function :

```
def api(start_date, end_date, latitude, longitude):
    """api call function.

    Keyword arguments:
    time -- desired date in "%Y-%m-%d format
    latitude -- gps coordinate
    longitude -- gps coordinate
    """

    date_string_1 = start_date + "T" + "00:00"
    date_string_2 = end_date + "T" + "00:00"

    date_object1 = datetime.strptime(date_string_1, "%Y-%m-%dT%H:%M")
    date_object2 = datetime.strptime(date_string_2, "%Y-%m-%dT%H:%M")

    date_only1 = date_object1.date()
    date_only2 = date_object2.date()
    date_string_1 = str(date_only1)
    date_string_2 = str(date_only2)
    # import dataset from API
    x = [(latitude, longitude)]

    li = []
    for i in x:
        params = {
            "latitude": i[0],
            "longitude": i[1],
            "start_date": date_string_1,
            "end_date": date_string_2,
            "timezone": "auto",
            "temperature_unit": "fahrenheit",  #  units
            "windspeed_unit": "mph",
            "precipitation_unit ": "inch",
            "hourly": {
                "precipitation",
```

```
                "snowfall",
                "temperature_2m",
                "relativehumidity_2m",
                "surface_pressure",
                "windspeed_10m",
                "winddirection_10m",
                "windgusts_10m",
                "cloudcover",
            },
            "daily": {"sunrise", "sunset"},
        }

        response = requests.get(
            "https://archive-api.open-meteo.com/v1/era5", params=params
        )
        res = response.json()

        df3 = pd.DataFrame.from_dict(res["daily"], orient="index").T
        df3 = df3.loc[df3.index.repeat(24)].reset_index(drop=True)
        df2 = pd.DataFrame.from_dict(res["hourly"], orient="index").T
        df2 = df2.assign(
            elevation=res["elevation"],
            latitude=res["latitude"],
            longitude=res["longitude"],
            timezone=res["timezone_abbreviation"],
        )
        df2["sunrise"] = df3["sunrise"]
        df2["sunset"] = df3["sunset"]
        li.append(df2)
    frame = pd.concat(li, axis=0, ignore_index=True)
    return frame, res

test = merged_df[merged_df['id'] == 577650    ]  #this is a test on a
single location , you can adapt the code to choose multiple ones

grouped_conc = test.groupby("id")
frames = []
for id, group in grouped_conc:


    start_time = str(group["date"].iloc[0].date())
    print(start_time)
    end_time = str(group["date"].iloc[len(group) - 1].date())
```

```
    print(end_time)
    lat = group["latitude"].iloc[0]
    print(lat)
    lon = group["longitude"].iloc[0]
    print(lon)
    frame ,res = api(start_time, end_time, lat, lon)

    frames.append(frame)

frame['time'] = pd.to_datetime(frame['time'], format='%Y-%m-%dT%H:%M')
test['time'] = pd.to_datetime(test['date'], format='%Y-%m-%d %H:%M:%S')
frame = frame[['time', 'surface_pressure', 'snowfall',
'temperature_2m',
       'winddirection_10m', 'relativehumidity_2m', 'windgusts_10m',
       'windspeed_10m', 'precipitation', 'cloudcover', 'elevation',
'timezone', 'sunrise', 'sunset']]
df_merged = pd.merge(test, frame, on='time', how='left')
```