KIET GROUP OF INSTITUTIONS
Connecting Life with Learning
www.kiet.edu
Delhi-NCR, Ghaziabad

Assessment Report

on

"COVID-19 Case Prediction"

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE(AIML)

SEC-A

By : Group 02

Sr. No.	Name	Roll No.
1.	Amit Kumar	202401100400030
2.	Amit Kumar	202401100400031
3.	Ananya Bharti	202401100400035
4.	Anushka Jaiswal	202401100400044
5.	Akshat Saxena	202401100400024

Under the supervision of

"BIKKI KUMAR"

KIET Group of Institutions, Ghaziabad

May, 2025

1. Introduction

The COVID-19 pandemic, caused by the SARS-CoV-2 virus, created a global health crisis. Analyzing COVID-19 data using artificial intelligence (AI) can help identify trends, predict future cases, and assist in resource allocation. AI models have proven effective in extracting meaningful patterns from large datasets, enabling data-driven decisions in public health.

---

2. Problem Statement

The sudden and rapid spread of COVID-19 led to uncertainties in healthcare planning and policy decisions. There was a crucial need to analyze the ongoing data to track infection trends, predict future cases, and optimize healthcare resources using AI techniques.

---

3. Objectives

- To collect and preprocess COVID-19 datasets.
- To analyze patterns and trends in infection and recovery rates.
- To build AI models for predicting future COVID-19 cases.
- To evaluate model performance using appropriate metrics.
- To visualize and interpret the results for actionable insights.

4. Methodology

- **Model Evaluation**:

- Data collection from reliable sources (e.g., WHO, Kaggle datasets).
- Data preprocessing to handle missing values, normalization, and feature selection.
- Splitting data into training and testing sets.
- Applying AI algorithms (e.g., Linear Regression, Decision Trees, or LSTM).
- Evaluating model accuracy using metrics like MAE, RMSE, or R^2 score.
- Visualization of predictions and trends.

---

5. Data Preprocessing

- The dataset is cleaned and prepared as follows:
- Data cleaning: Handling missing/null values.
- Feature engineering: Creating new variables from date or cumulative data.
- Normalization: Scaling features to improve model performance.
- Splitting: Training (80%) and testing (20%) datasets.

---

6. Model Implementation

- Python with libraries like Pandas, NumPy, Matplotlib, and Scikit-learn is used.
- Models implemented: Linear Regression, Random Forest, or LSTM (for time series).
- Jupyter Notebook or Google Colab as the IDE for implementation.

7. Evaluation Metrics

The following metrics are used to evaluate the model:

- **Mean Absolute Error (MAE)**
- **Root Mean Square Error (RMSE)**
- **R^2 Score**
- **Mean Squared Error (MSE)**

  These metrics help assess the model's prediction capability and error rates.

---

8. Results and Analysis

- Visual graphs showing actual vs predicted cases.
- Statistical results indicating model performance.
- Insights from trends, such as peaks, decline, and recovery periods.
- Comparison of different model performances.

---

9. Conclusion

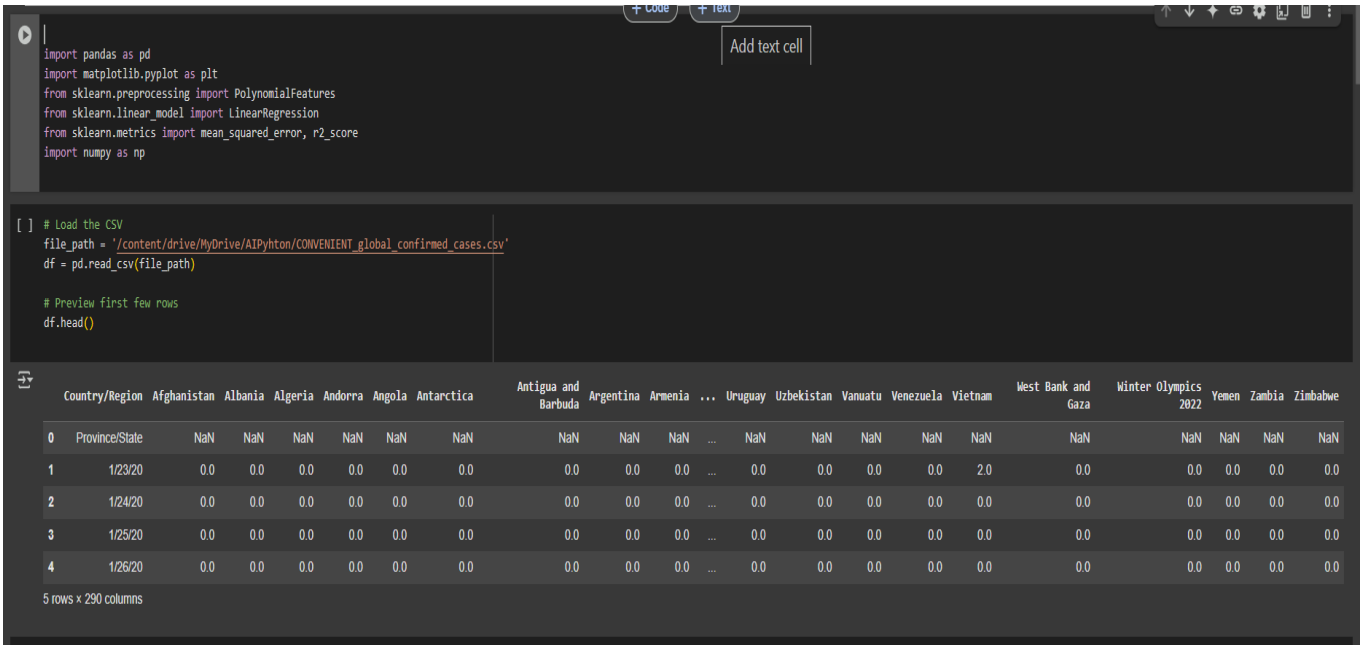
AI-based analysis of COVID-19 data provides valuable insights into the pandemic’s progression. The developed model can predict upcoming trends, aiding in proactive

decision-making. Continuous updating of models with real-time data can further enhance accuracy.

---

10. References

- World Health Organization (WHO) – https://www.who.int
- Kaggle COVID-19 datasets – https://www.kaggle.com/datasets
- Scikit-learn Documentation – https://scikit-learn.org
- COVID-19 Open Research Dataset (CORD-19) – https://www.semanticscholar.org/cord19



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

```
# Load the CSV
file_path = '/content/drive/MyDrive/AIPyhton/CONVENIENT_global_confirmed_cases.csv'
df = pd.read_csv(file_path)

# Preview first few rows
df.head()
```

	Country/Region	Afghanistan	Albania	Algeria	Andorra	Angola	Antarctica	Antigua and Barbuda	Argentina	Armenia	...	Uruguay	Uzbekistan	Vanuatu	Venezuela	Vietnam	West Bank and Gaza	Winter Olympics 2022	Yemen	Zambia	Zimbabwe
0	Province/State	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1/23/20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0
2	1/24/20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1/25/20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1/26/20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

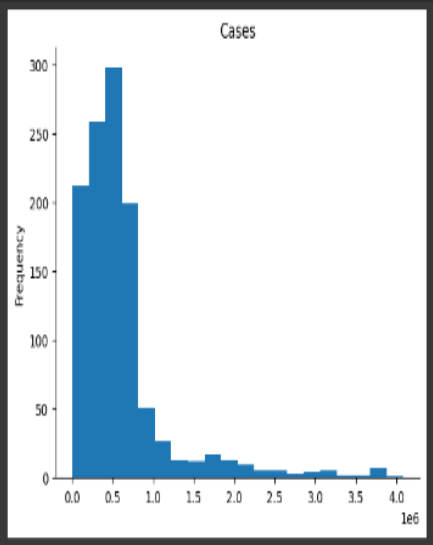
5 rows × 290 columns

```
df = pd.read_csv(file_path, skiprows=1)

# Rename first column to 'Date'
df.rename(columns={df.columns[0]: "Date"}, inplace=True)

# Convert 'Date' to datetime
df["Date"] = pd.to_datetime(df["Date"], infer_datetime_format=True)

# Sum across all countries/regions for global total (row-wise)
df["Global_Cases"] = df.drop(columns="Date").sum(axis=1)

# Final DataFrame for modeling
cases_df = df[["Date", "Global_Cases"]].copy()
cases_df.rename(columns={"Global_Cases": "Cases"}, inplace=True)

cases_df.head()
```

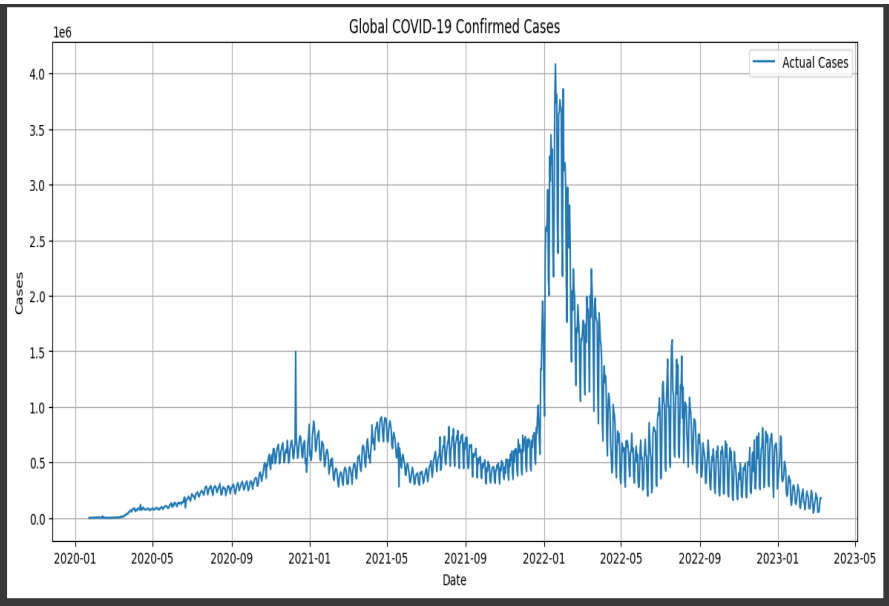
```
<ipython-input-33-911af7e4b1a9>:7: UserWarning: The argument 'infer_datetime_format' is deprecated and will be removed in a future version. A strict version of it is now the default, see https://pandas.pydata.org/pdeps/0004-consisten
  df["Date"] = pd.to_datetime(df["Date"], infer_datetime_format=True)
<ipython-input-33-911af7e4b1a9>:7: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df["Date"] = pd.to_datetime(df["Date"], infer_datetime_format=True)
```

	Date	Cases
0	2020-01-23	100.0
1	2020-01-24	287.0
2	2020-01-25	493.0
3	2020-01-26	683.0
4	2020-01-27	809.0

```
[ ] from matplotlib import pyplot as plt
    cases_df['Cases'].plot(kind='hist', bins=20, title='Cases')
    plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
plt.figure(figsize=(14, 6))
plt.plot(cases_df["Date"], cases_df["Cases"], label="Actual Cases")
plt.title("Global COVID-19 Confirmed Cases")
plt.xlabel("Date")
plt.ylabel("Cases")
plt.grid(True)
plt.legend()
plt.show()
```




```
[ ] # Convert dates to ordinal (numerical format)
    cases_df["Days"] = cases_df["Date"].map(pd.Timestamp.toordinal)

    # Features and labels
    X = cases_df["Days"].values.reshape(-1, 1)
    y = cases_df["Cases"].values
```

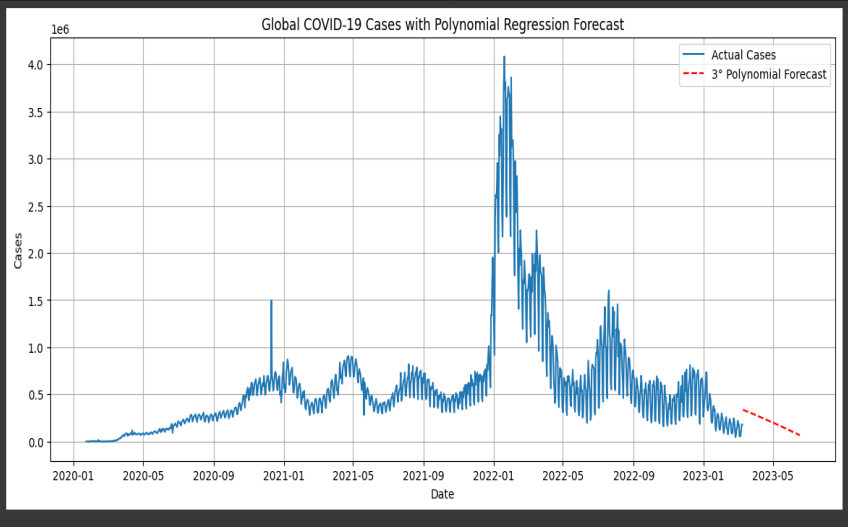
```
[ ] # Polynomial transformation
    degree = 3
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(X)
```

```
[ ] # Train the model
    model = LinearRegression()
    model.fit(X_poly, y)

    # Predict on training data
    y_pred = model.predict(X_poly)

    # Evaluate
    print("Mean Squared Error:", mean_squared_error(y, y_pred))
    print("R² Score:", r2_score(y, y_pred))
```

```
Mean Squared Error: 253881359266.90707
R² Score: 0.2838111285522747
```

```
# Predict future 30 days
future_days = 100
last_day = cases_df["Days"].max()
future_X = np.array([last_day + i for i in range(1, future_days + 1)]).reshape(-1, 1)
future_X_poly = poly.transform(future_X)
future_y_pred = model.predict(future_X_poly)

# Convert future ordinals back to dates
future_dates = [pd.Timestamp.fromordinal(int(day[0])) for day in future_X]

# Create future DataFrame
future_df = pd.DataFrame({
    "Date": future_dates,
    "Predicted_Cases": future_y_pred
})
```

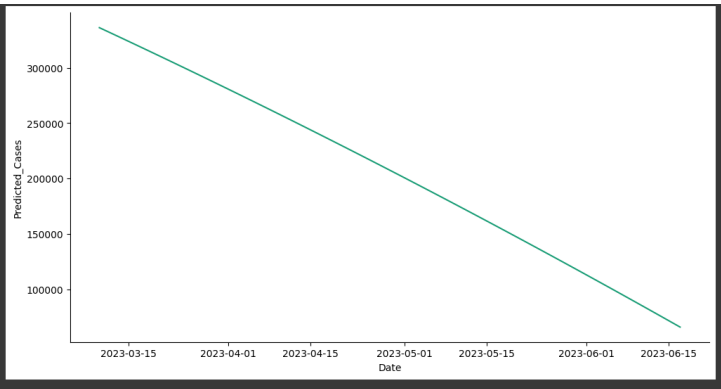
```
plt.title("Global COVID-19 Cases with Polynomial Regression Forecast")
plt.xlabel("Date")
plt.ylabel("Cases")
plt.legend()
plt.grid(True)
plt.show()
```



	Date	Predicted_Cases
0	2023-03-10	336236.191345
1	2023-03-11	333766.360046
2	2023-03-12	331291.223450
3	2023-03-13	328810.781189
4	2023-03-14	326325.033630

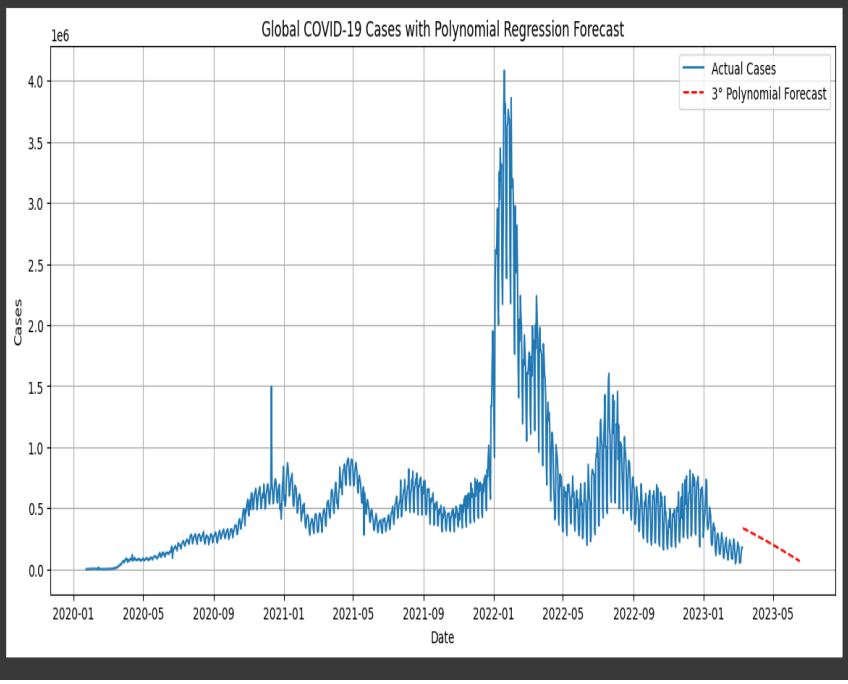
Date vs Predicted_Cases

```
# @title Date vs Predicted_Cases

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Predicted_Cases']

  plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = future_df.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Predicted_Cases')
```




```
plt.figure(figsize=(14, 6))
plt.plot(cases_df["Date"], cases_df["Cases"], label="Actual Cases")
plt.plot(future_df["Date"], future_df["Predicted_Cases"], label=f"{degree}° Polynomial Forecast", linestyle="dashed", color="red")
plt.title("Global COVID-19 Cases with Polynomial Regression Forecast")
plt.xlabel("Date")
plt.ylabel("Cases")
plt.legend()
plt.grid(True)
plt.show()
```

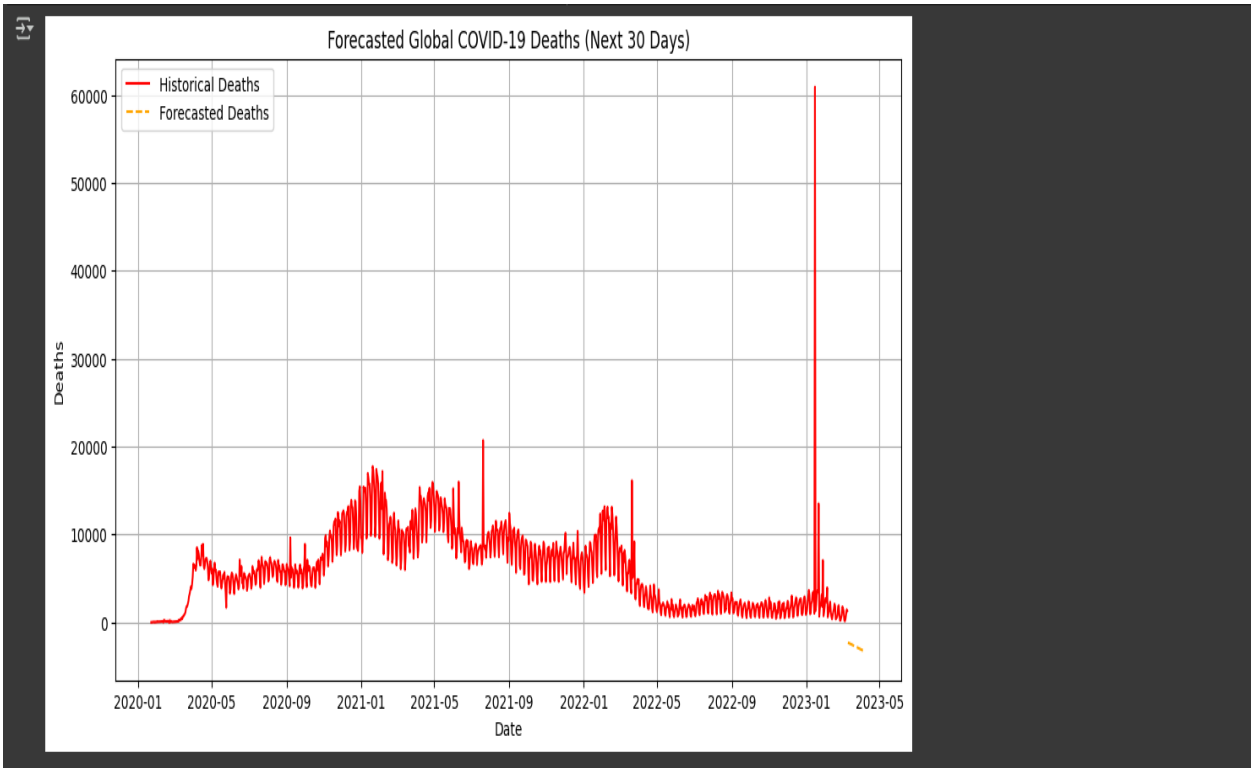
```
plt.title("Global COVID-19 Cases with Polynomial Regression Forecast")
plt.xlabel("Date")
plt.ylabel("Cases")
plt.legend()
plt.grid(True)
plt.show()
```




```
[ ]
deaths_path = '/content/drive/MyDrive/AIPyhton/CONVENIENT_global_deaths.csv'

# Skip metadata row
deaths_df = pd.read_csv(deaths_path, skiprows=1)

# Rename first column to "Date"
deaths_df.rename(columns={deaths_df.columns[0]: "Date"}, inplace=True)

# Convert Date column to datetime format
deaths_df["Date"] = pd.to_datetime(deaths_df["Date"], infer_datetime_format=True)
```

```
<ipython-input-41-fd5094bb10cb>:10: UserWarning: The argument 'infer_datetime_format' is deprecated and will be removed in a future version. A strict version of it is now the default, see https://pandas.pydata.org/pdeps/0004-consiste
  deaths_df["Date"] = pd.to_datetime(deaths_df["Date"], infer_datetime_format=True)
<ipython-input-41-fd5094bb10cb>:10: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  deaths_df["Date"] = pd.to_datetime(deaths_df["Date"], infer_datetime_format=True)
```

```
# Sum deaths across all countries (columns)
deaths_df["Global_Deaths"] = deaths_df.drop(columns="Date").sum(axis=1)

# Prepare final DataFrame
df = pd.DataFrame({
    "Date": deaths_df["Date"],
    "Deaths": deaths_df["Global_Deaths"]
})

# Convert Date to ordinal for regression
df["Days"] = df["Date"].map(pd.Timestamp.toordinal)

# Preview
df.head()
```


Forecasted Global COVID-19 Deaths (Next 30 Days)
Historical Deaths
Forecasted Deaths
Deaths
60000
50000
40000
30000
20000
10000
0
2020-01
2020-05
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05
2022-09
2023-01
2023-05
Date