Analyzing and Forecasting COVID-19 Trends

Done by:

Meggna Karthikeyan

Abstract:

The objective of this project is to leverage advanced machine learning techniques and time series analysis methods to develop robust forecasting models for predicting COVID-19 cases with precision. By exploring the performance of various models, including linear regression, decision tree regression, random forest regression, and ARIMA, the project aims to identify the most effective approach for capturing the intricate dynamics and non-linear patterns inherent in COVID-19 case data. Through rigorous evaluation and comparison of R-squared values, the project seeks to enhance understanding and provide actionable insights into the forecasting of COVID-19 cases, thereby facilitating informed decision-making, efficient resource allocation, and effective public health interventions in response to the pandemic.

Data:

The project utilized publicly available COVID-19 case data from reliable sources such as the World Health Organization (WHO) and Johns Hopkins University's COVID-19 Data Repository. The dataset included daily confirmed cases, recoveries, and deaths for a specific region or country over the course of the pandemic.

- Date: Date of reporting the COVID-19 data.
- Country/Region: Name of the country or region for which the COVID-19 data is reported.
- Cases: Number of confirmed COVID-19 cases reported on the given date.
- Deaths: Number of COVID-19 related deaths reported on the given date.

Importing necessary libraries:

I aim to develop a COVID-19 data analysis system using Python, with the objective of understanding trends and making predictions. Through this project, I will explore various Python libraries and techniques to analyze COVID-19 data effectively.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
```

```
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from pmdarima import auto_arima
```

NumPy: A Python library for array manipulation, linear algebra, and matrix operations, essential for numerical computing.

Pandas: A Python library for data manipulation and analysis, offering powerful data structures and operations for handling tabular data.

scikit-learn: A comprehensive Python library for machine learning tasks, including classification, regression, clustering, and model selection. It includes sub-libraries for data splitting (train_test_split), linear regression (LinearRegression), decision tree regression (DecisionTreeRegressor), and random forest regression (RandomForestRegressor).

sklearn.metrics.r2_score: A sub-library in scikit-learn for computing the R-squared score, a metric used to assess the performance of regression models.

pmdarima: A Python library specialized in time series forecasting using ARIMA models, featuring functions for automatic hyperparameter tuning and model selection.

Part-1: Analyzing the Dataset

To read the dataset from the csv file, I use

```
desktop_path_cases = os.path.join(r"C:\Users\meggn\OneDrive\Documents\CIS 550-AML\Final Project\CONVENIENT_global_deaths.csv")
desktop_path_deaths = os.path.join(r"C:\Users\meggn\OneDrive\Documents\CIS 550-AML\Final Project\CONVENIENT_global_confirmed_case
df_cases = pd.read_csv(desktop_path_cases,delimiter=',', na_values="?", header=None)
df_deaths = pd.read_csv(desktop_path_deaths,delimiter=',', na_values="?", header=None)
```

The pd.read_csv function from the Pandas library is used to read the CSV files into DataFrames.
df_cases is assigned the result of reading the file specified by desktop_path_cases.
similarly, df_deaths is assigned to the result of reading the file specified by desktop_path_deaths.

```
df_cases.shape
```

```
(1144, 290)
```

```
df_cases.head()
```

	0	1	2	3	4	5	6	7	8	9	...	280	281	282	283	284	2
0	Country/Region	Afghanistan	Albania	Algeria	Andorra	Angola	Antarctica	Antigua and Barbuda	Argentina	Armenia	...	Uruguay	Uzbekistan	Vanuatu	Venezuela	Vietnam	We Ba a Ga
1	Province/State	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na
2	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	0.0	(
3	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	(
4	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	(

To display the basic information of data such as datatype, columns, non-null values count. The df.head() function is used to retrieve the first few rows of the DataFrame [pandas.pydata.org]. By default, it retrieves the top five rows.

Part-2 Data Cleaning

```
world = pd.DataFrame({"Country":[],"Cases":[]})
world["Country"] = df_cases.iloc[0,1:].index
cases = []
for i in world["Country"]:
    cases.append(pd.to_numeric(df_cases[2:][i]).sum())
world["Cases"]=cases
world["Country"] = df_cases.iloc[0,1:].values
```

```
world.head()
```

	Country	Cases
0	Afghanistan	7896.0
1	Albania	3598.0
2	Algeria	6881.0
3	Andorra	165.0
4	Angola	1933.0

```
world.Country.unique
```

```
<bound method Series.unique of 0              Afghanistan
1                  Albania
2                  Algeria
3                  Andorra
4                   Angola
               ...
284      West Bank and Gaza
285    Winter Olympics 2022
286                   Yemen
287                  Zambia
288                Zimbabwe
```

This data cleaning step helps transform the raw data from df_cases into a format that's more suitable for further analysis focused on total cases per country. It improves data clarity and consistency; the above code shows the cleaning data from the df_cases DataFrame into a ***new DataFrame named "world***". It extracts country names, converts relevant data into numeric format, calculates the sum of cases for each country, and stores the results in a new DataFrame.

```
world.describe()
```

	Cases
count	2.890000e+02
mean	2.381240e+04
std	9.355568e+04
min	0.000000e+00
25%	6.600000e+01
50%	9.520000e+02
75%	8.727000e+03
max	1.123836e+06

The code snippet utilizes the **describe()** method from the Pandas library [pandas.pydata.org] to generate summary statistics of a dataset. In the code, world is likely a Pandas DataFrame containing data. The .describe() method calculates various statistics about the numerical columns in the DataFrame.

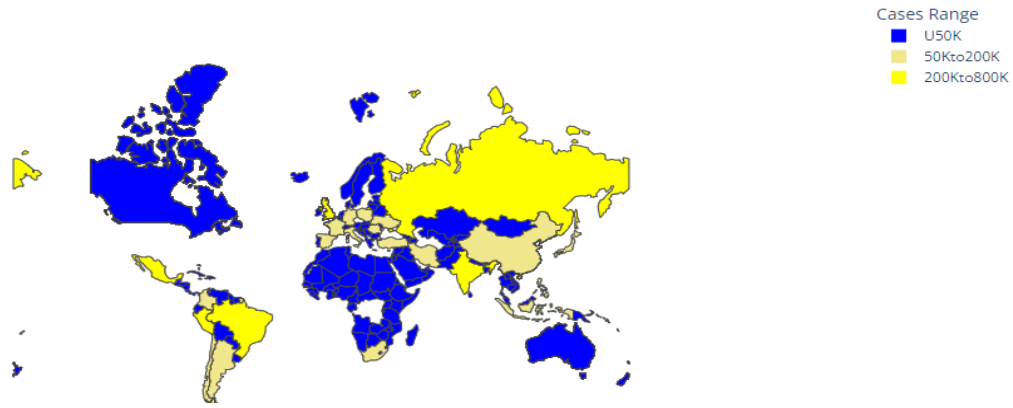
```
alpha =[]
for i in world["Country"].str.upper().values:
    if len(continent[continent["Country"]==i]["alpha"].values)==0:
        alpha.append(np.nan)
    else:
        alpha.append(continent[continent["Country"]==i]["alpha"].values[0])
world["alpha"]=alpha
```

```
world.head()
```

	Country	Cases	Cases Range	alpha
0	Afghanistan	7896.0	U50K	AFG
1	Albania	3598.0	U50K	ALB
2	Algeria	6881.0	U50K	DZA
3	Andorra	165.0	U50K	AND
4	Angola	1933.0	U50K	AGO

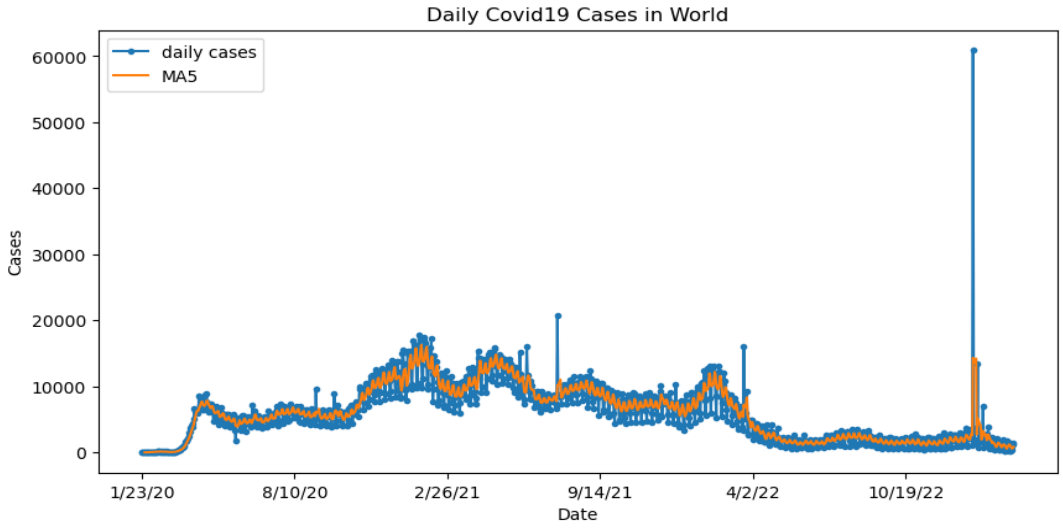
The code adds a new column named "alpha" to the DataFrame world. It iterates through each country in the "Country" column and checks if there's a corresponding value in another DataFrame, likely named continent. If a matching country is found in the continent DataFrame, the code retrieves the value from the "alpha" column and assigns it to the new "alpha" column in the world DataFrame. If there's no match, the code adds np.nan (which represents Not a Number) to the new column.

```python
import pandas as pd
import plotly.express as px

# Create a DataFrame with the missing value '800Kto1.5M'
new_row = {'alpha': '800Kto1.5M', 'Cases Range': '800Kto1.5M'}
new_data = pd.DataFrame([new_row])

# Assuming you have your data loaded into the 'world' DataFrame
# Concatenate the new row with the existing data
world = pd.concat([world, new_data], ignore_index=True)

# Create the choropleth plot
fig = px.choropleth(world.dropna(),
                    locations="alpha",
                    color="Cases Range",
                    projection="mercator",
                    color_discrete_sequence=["blue", "khaki", "yellow", "orange", "red"])
fig.update_geos(fitbounds="locations", visible=False)
fig.update_layout(margin={"r": 0, "t": 0, "l": 0, "b": 0})
fig.show()
```



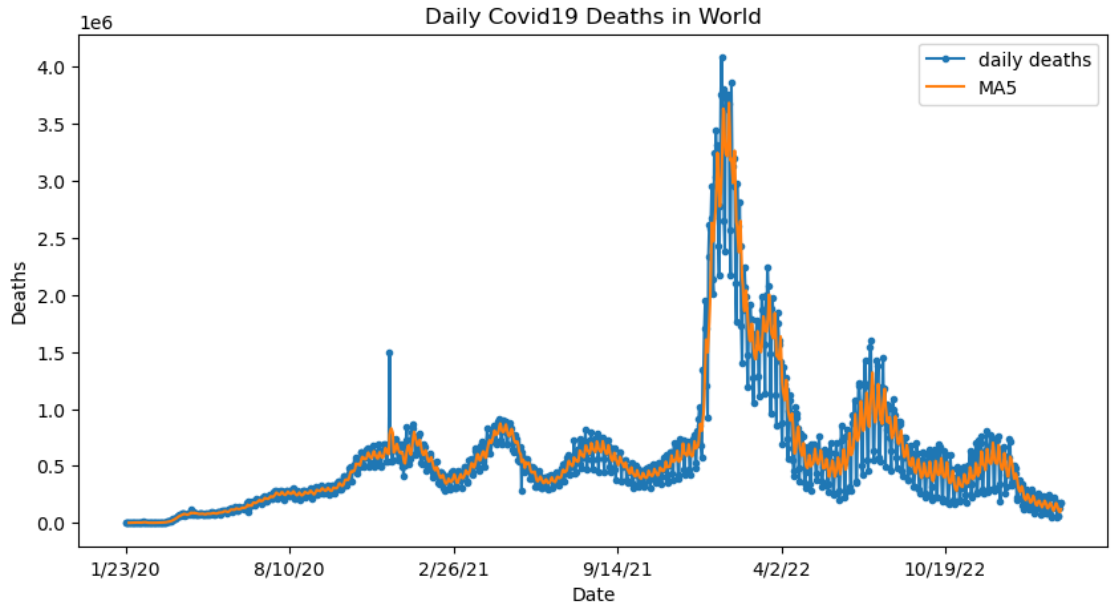
The code creates a choropleth map where countries are colored based on their COVID-19 cases range data from the "Cases Range" column. The color scale goes from blue (low cases) to red (high cases).

```python
world_cases = world_cases.set_index("Date")
world_cases.Cases.plot(title="Daily Covid19 Cases in World",marker=".",figsize=(10,5),label="daily cases")
world_cases.Cases.rolling(window=5).mean().plot(figsize=(10,5),label="MA5")
plt.ylabel("Cases")
plt.legend()
plt.show()
```



The code snippet appears to be visualizing daily Covid-19 cases over time. It uses Matplotlib, a popular Python library, for data visualization. The code imports the world_cases DataFrame, which contains a date column ("Date") and a cases column ("Cases"). This visualization helps compare the daily cases with a smoothed trend to potentially identify patterns over time.

```
world_deaths = world_deaths.set_index("Date")
world_deaths.Deaths.plot(title="Daily Covid19 Deaths in World",marker=".",figsize=(10,5),label="daily deaths")
world_deaths.Deaths.rolling(window=5).mean().plot(figsize=(10,5),label="MA5")
plt.ylabel("Deaths")
plt.legend()
plt.show()
```

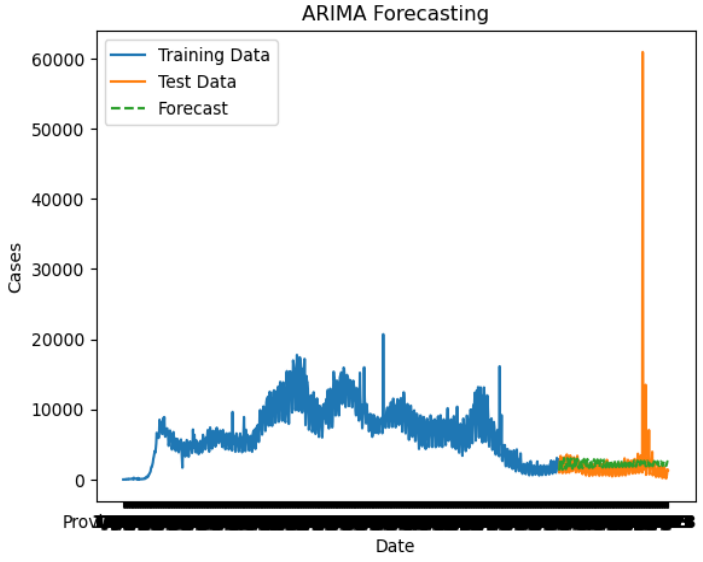


The code snippet appears to be visualizing daily Covid-19 deaths over time. It likely uses Matplotlib [matplotlib.org], a popular Python library for data visualization. The code imports the world_deaths DataFrame, which seems to contain a date column ("Date") and a deaths column ("Deaths"). This visualization helps compare the daily deaths with a smoothed trend to potentially identify patterns over time.

Part -3 Model Selection and Training:

The result below shows the fitting of an ARIMA model on the df_cases_daily DataFrame. This DataFrame likely contains a date index, and a column named "Cases". The code uses auto_arima from pmdarima to automatically select the best hyperparameters for the ARIMA model based on the data in df_cases_daily. Once the hyperparameters are identified, the code creates an ARIMA model instance and fits it to the training data.

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=15755.068, Time=2.19 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=16270.594, Time=0.05 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=16269.726, Time=0.10 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=16266.411, Time=0.12 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=16268.595, Time=0.04 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=16025.452, Time=1.96 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=15971.086, Time=1.87 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=15694.216, Time=2.41 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=15909.141, Time=1.25 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=15541.056, Time=2.62 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=15820.883, Time=1.60 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=15402.261, Time=3.35 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=15605.044, Time=1.24 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=15410.542, Time=3.70 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=15522.213, Time=3.14 sec
 ARIMA(5,1,2)(0,0,0)[0]             : AIC=15400.253, Time=1.92 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=15538.956, Time=1.77 sec
 ARIMA(5,1,1)(0,0,0)[0]             : AIC=15603.124, Time=0.65 sec
 ARIMA(5,1,3)(0,0,0)[0]             : AIC=15408.585, Time=1.81 sec
 ARIMA(4,1,1)(0,0,0)[0]             : AIC=15818.877, Time=0.80 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=15520.156, Time=2.04 sec

Best model:  ARIMA(5,1,2)(0,0,0)[0]
Total fit time: 34.676 seconds
```




```
R2 Score: 0.0030433088712231715
```

R2 Score: This value (0.0030433) is a metric used to evaluate how well the ARIMA model fits the historical data. An R-squared score closer to 1 indicates a better fit. In this case, the value is very low, which suggests the model does not explain much of the variance in the actual data.

Overall, the ARIMA model fit to the COVID-19 cases data seems to have a poor R-squared score, which means the model isn't very accurate in capturing the patterns in the historical data. Consequently, the forecast for future cases may not be reliable.

```
# Assuming you have a DataFrame named world_cases with columns 'Date' and 'Cases'

# Assuming your 'Date' column is of datetime type, if not, convert it
# world_cases['Date'] = pd.to_datetime(world_cases['Date'])

# Extracting features and target variable
X = np.array(world_cases.index).reshape(-1, 1)  # Assuming the index represents the date as numeric values
y = world_cases['Cases'].values

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Models initialization
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree Regression': DecisionTreeRegressor(random_state=42),
    'Random Forest Regression': RandomForestRegressor(random_state=42)
}

# Training and evaluation
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    results[name] = r2

# Finding the best model
best_model = max(results, key=results.get)

print("R-squared values:")
for name, r2 in results.items():
    print(f"{name}: {r2}")

print(f"\nThe best model is: {best_model} with R-squared value of {results[best_model]}")

# Predicting values for the next 30 days using the best model
best_model_instance = models[best_model]
future_dates = np.array(range(len(world_cases), len(world_cases) + 30)).reshape(-1, 1)
future_predictions = best_model_instance.predict(future_dates)

# Printing the future predictions
print("\nFuture predictions for the next 30 days:")
for i, pred in enumerate(future_predictions):
    print(f"Day {i+1}: {pred}")
```

The code splits the data into training and testing sets using train_test_split from sklearn.model_selection. Then, it trains three different models:

- Linear Regression model (LinearRegression())
- Decision Tree Regression model (DecisionTreeRegressor())
- Random Forest Regression model (RandomForestRegressor())

After training the models, the code uses metrics like r2_score to evaluate their performance on the testing data. The R-squared score indicates how well the model fits the data, with a value closer to 1 representing a better fit.

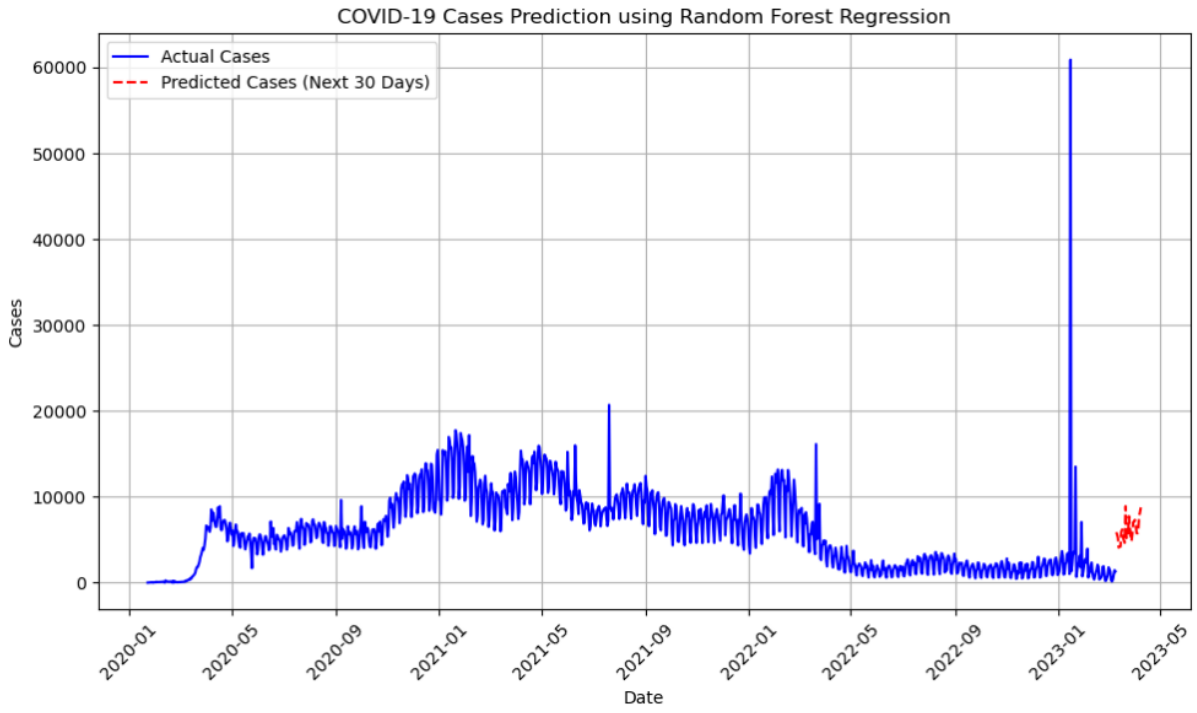
```
R-squared values:
Linear Regression: -0.011058330578216147
Decision Tree Regression: 0.37587654223471856
Random Forest Regression: 0.4176927356402138

The best model is: Random Forest Regression with R-squared value of 0.4176927356402138

Future predictions for the next 30 days:
Day 1: 980.82
Day 2: 1143.18
Day 3: 1143.18
Day 4: 1143.18
Day 5: 1143.18
Day 6: 1143.18
Day 7: 1143.18
Day 8: 1143.18
Day 9: 1143.18
Day 10: 1143.18
Day 11: 1143.18
Day 12: 1143.18
Day 13: 1143.18
Day 14: 1143.18
Day 15: 1143.18
Day 16: 1143.18
Day 17: 1143.18
Day 18: 1143.18
Day 19: 1143.18
Day 20: 1143.18
Day 21: 1143.18
Day 22: 1143.18
Day 23: 1143.18
Day 24: 1143.18
Day 25: 1143.18
Day 26: 1143.18
Day 27: 1143.18
Day 28: 1143.18
Day 29: 1143.18
Day 30: 1143.18
```

The R-squared score is used to measure how well each model fits the data, with a value closer to 1 indicating a better fit. In this case, the Random Forest Regression model achieved the highest R-squared score of 0.4177, followed by Decision Tree Regression (0.3759) and Linear Regression (-0.0111).

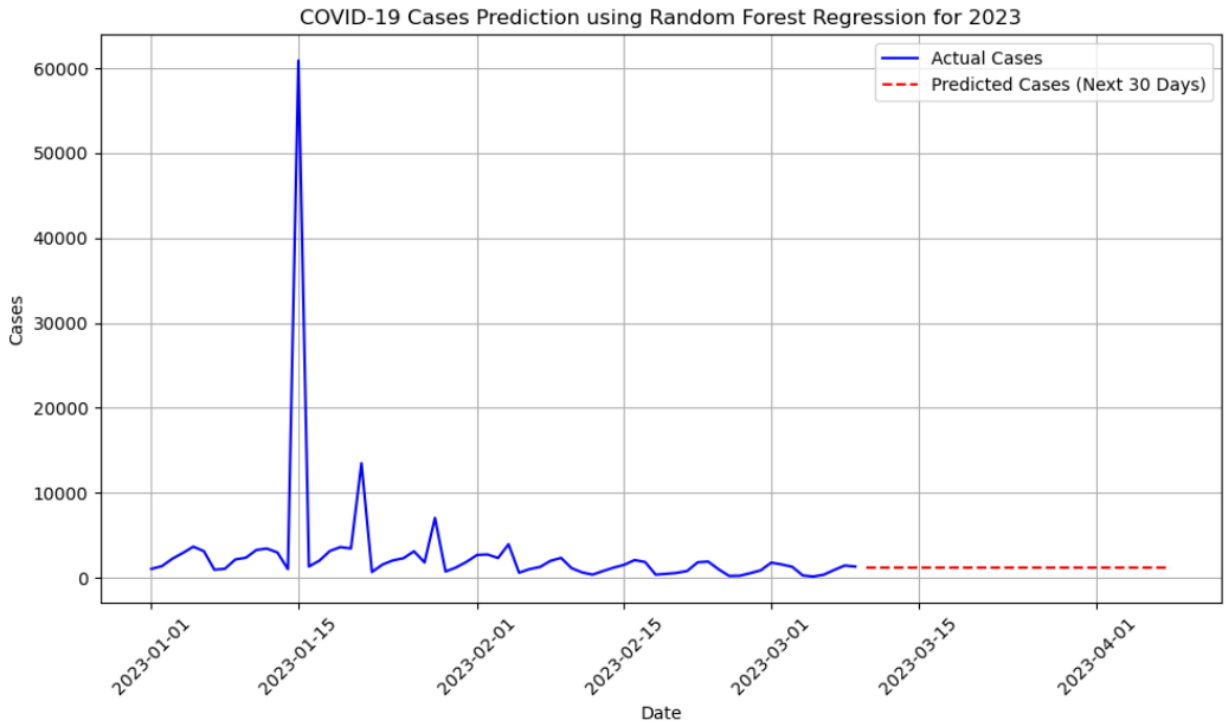
Based on these results, the Random Forest Regression model appears to perform best at capturing the patterns in the COVID-19 cases data.

Part-4 Training and Validation Accuracy Calculation:

The below plot shows us the model predicting future COVID-19 cases using a Random Forest Regression model. Random Forest Regression is a machine learning technique that ensembles multiple decision trees to improve prediction accuracy. The code imports the sklearn.ensemble library from scikit-learn, which provides functions for ensemble methods.

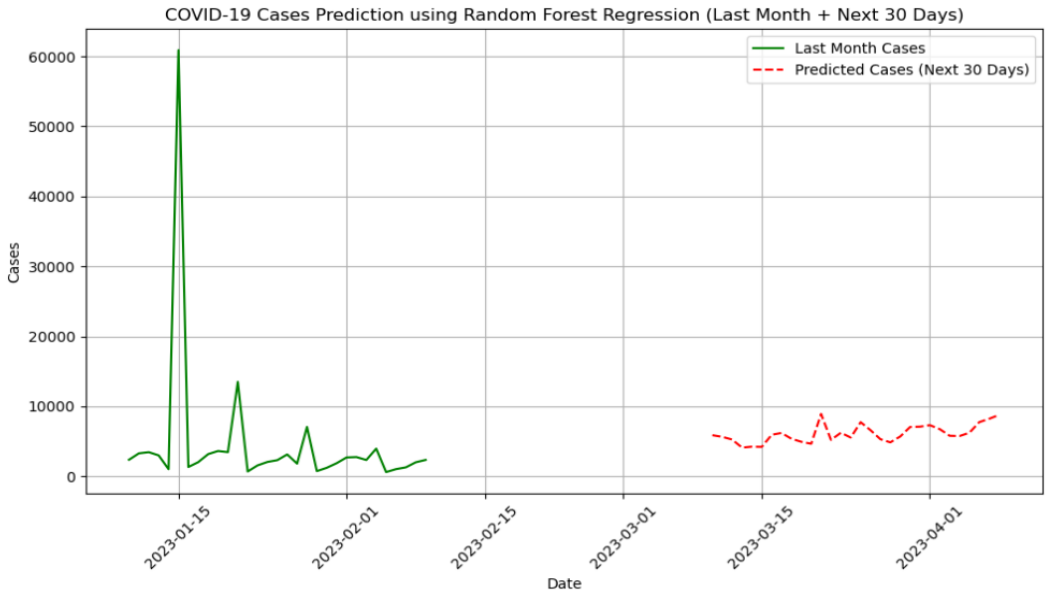


The screenshot below shows the model plotting the actual cases data for 2023 with a blue line. Plots the predicted cases for the next 30 days with a red dashed line. Includes labels, title, legend, and formatting elements for clarity.



Overall, this code demonstrates a basic approach for daily case prediction using a Random Forest Regression model. The image shows the results of a Random Forest Regression model predicting future daily COVID-19 cases. The forecast is visualized over a time series plot. The x-axis represents the date, and the y-axis represents the number of cases.

```
# Convert 'Date' column to datetime type
world_cases['Date'] = pd.to_datetime(world_cases['Date'])

# Extracting features and target variable
X = world_cases['Date'].dt.dayofyear.values.reshape(-1, 1)  # Using day of year as numeric feature
y = world_cases['Cases'].values

# Model initialization
model = RandomForestRegressor(random_state=42)

# Fitting the model
model.fit(X, y)

# Predicting values for the next 30 days
future_dates = pd.date_range(start=world_cases['Date'].max() + pd.Timedelta(days=1), periods=30)
future_dates_numeric = future_dates.dayofyear.values.reshape(-1, 1)
future_predictions = model.predict(future_dates_numeric)

# Getting the last month's data
last_month_end = world_cases['Date'].max() - pd.DateOffset(months=1)
last_month_start = last_month_end - pd.DateOffset(days=30)
last_month_data = world_cases[(world_cases['Date'] >= last_month_start) & (world_cases['Date'] <= last_month_end)]

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(last_month_data['Date'], last_month_data['Cases'], color='green', label='Last Month Cases')
plt.plot(future_dates, future_predictions, color='red', linestyle='--', label='Predicted Cases (Next 30 Days)')
plt.title('COVID-19 Cases Prediction using Random Forest Regression (Last Month + Next 30 Days)')
plt.xlabel('Date')
plt.ylabel('Cases')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



A **RandomForestRegressor** model is initialized with a specified random seed (**random_state=42**). The model is trained (**model.fit**) using the extracted features and target variable. Future dates for the next 30 days are generated using **pd.date_range**, and their day of the year values are calculated.

The trained model is used to predict the number of cases for the next 30 days.

Finally, a plot is created using **matplotlib.pyplot**, displaying the COVID-19 cases from the last month (**last_month_data**) and the predicted cases for the next 30 days The plot is annotated with appropriate labels, titles, and legends for clarity.

This predictive modelling approach aims to provide insights into the trend of COVID-19 cases based on historical data, enabling stakeholders to anticipate future developments and plan interventions accordingly.