Optimizing Train Services: Predictive Modeling and Capacity Planning Amidst COVID-19 Dynamics

Setup environment and install requirements

This code snippet imports the essential libraries for data manipulation and visualization.

In [9]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import sklearn
```

```
Matplotlib is building the font cache; this may take a moment.
```

Data loader

This code snippet loads the dataset into a pandas DataFrames. It reads the dataset files from the specified file paths using the read_csv function.

- df contains covid case and weather data per country

In [10]:
```
df = pd.read_csv('data/Data.csv')
df.sample(10)
```

Out[10]:

	Id	Province/State	Country/Region	Lat	Long	Date	ConfirmedC
7742	11403	NaN	Guyana	5.0000	-58.7500	2020-03-18	
9097	13418	NaN	Latvia	56.8796	24.6032	2020-02-16	
12093	17824	NaN	Senegal	14.4974	-14.4524	2020-03-22	
15293	22554	Missouri	US	38.4561	-92.2884	2020-03-09	
4192	6173	Liaoning	China	41.2956	122.6085	2020-02-25	
7106	10467	NaN	Georgia	42.3154	43.3569	2020-03-12	
10798	15929	NaN	Norway	60.4720	8.4689	2020-02-16	
8784	12955	NaN	Kenya	-0.0236	37.9062	2020-02-18	
15861	23392	North Dakota	US	47.5289	-99.7840	2020-03-10	
4643	6834	Shanxi	China	37.5777	112.2922	2020-03-06	

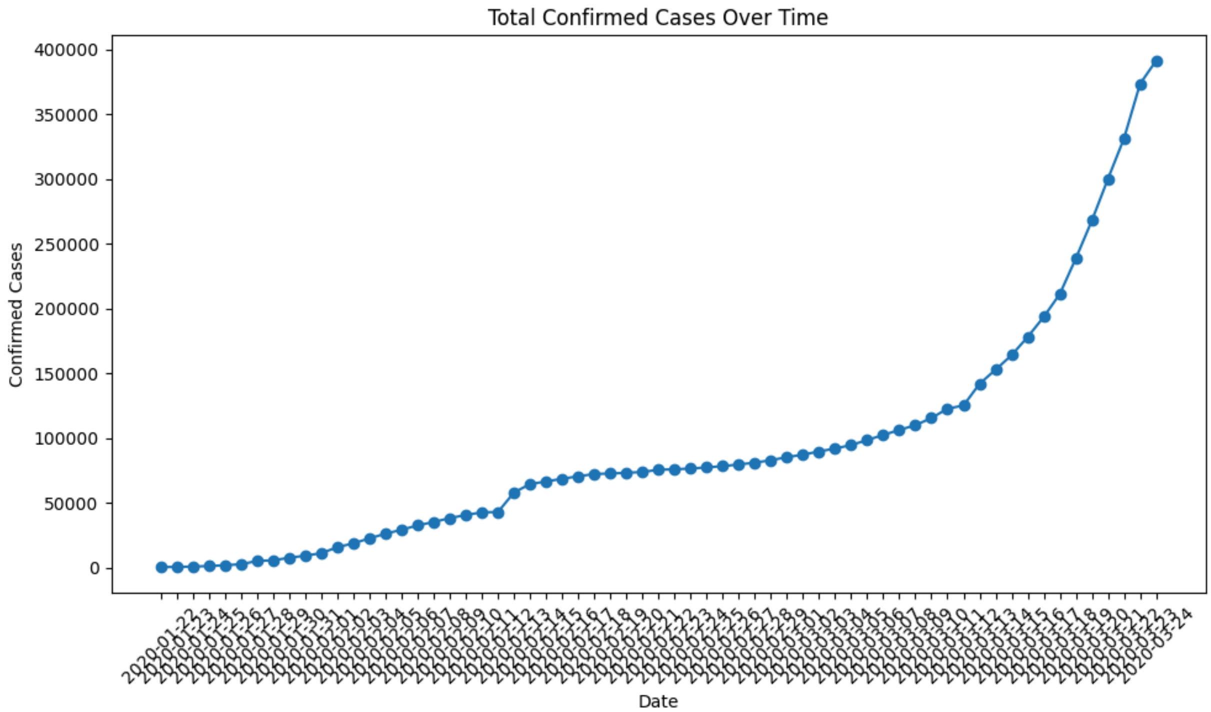
Data Exploration

In [ ]:

```
First we plot cases over time to see what the relationship looks like
```

In [19]:

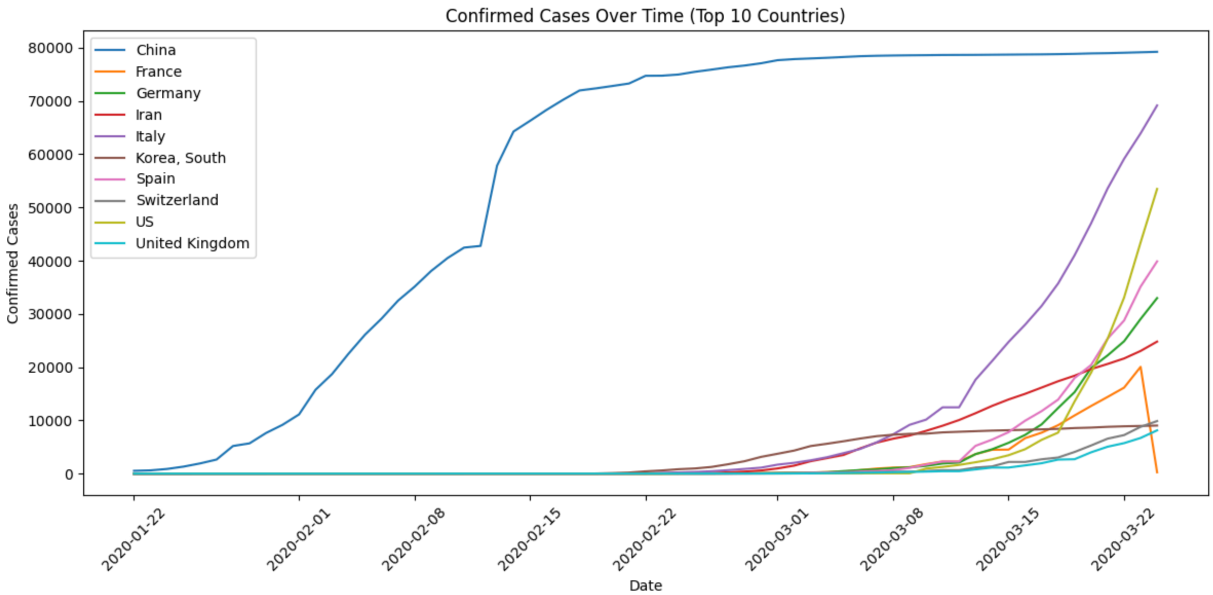
```python
# Group by date, summing (or averaging) confirmed_cases
df_agg = df.groupby('Date', as_index=False)['ConfirmedCases'].sum()

# Sort the aggregated data by date (best practice for time series)
df_agg.sort_values('Date', inplace=True)

# Plot
plt.figure(figsize=(10, 6))
plt.plot(df_agg['Date'], df_agg['ConfirmedCases'], marker='o')
plt.xlabel('Date')
plt.ylabel('Confirmed Cases')
plt.title('Total Confirmed Cases Over Time')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Then want to brake it down by country, but there are too many countries, so picked the top 10 to see those patterns

In [27]:
```
# Convert 'Date' to a proper datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Aggregate ConfirmedCases by (Date, Country/Region)
df_agg = df.groupby(['Date', 'Country/Region'], as_index=False)['ConfirmedCa

# Find the top 10 countries by total confirmed cases (across all dates)
country_totals = df_agg.groupby('Country/Region')['ConfirmedCases'].sum()
top_10_countries = country_totals.nlargest(10).index  # get the country name

# Filter df_agg to only those 10 countries
df_top10 = df_agg[df_agg['Country/Region'].isin(top_10_countries)]

# Pivot so each country becomes its own column, indexed by Date
df_pivot = df_top10.pivot_table(
    index='Date',
    columns='Country/Region',
    values='ConfirmedCases',
    aggfunc='sum'
)

# Sort by Date to ensure chronological order (good practice for time series)
df_pivot.sort_index(inplace=True)

# Plot each country as its own line
plt.figure(figsize=(12, 6))
for country in df_pivot.columns:
    plt.plot(df_pivot.index, df_pivot[country], label=country)

plt.xlabel('Date')
```

```python
plt.ylabel('Confirmed Cases')
plt.title('Confirmed Cases Over Time (Top 10 Countries)')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels if needed
plt.tight_layout()       # Adjust spacing
plt.show()
```



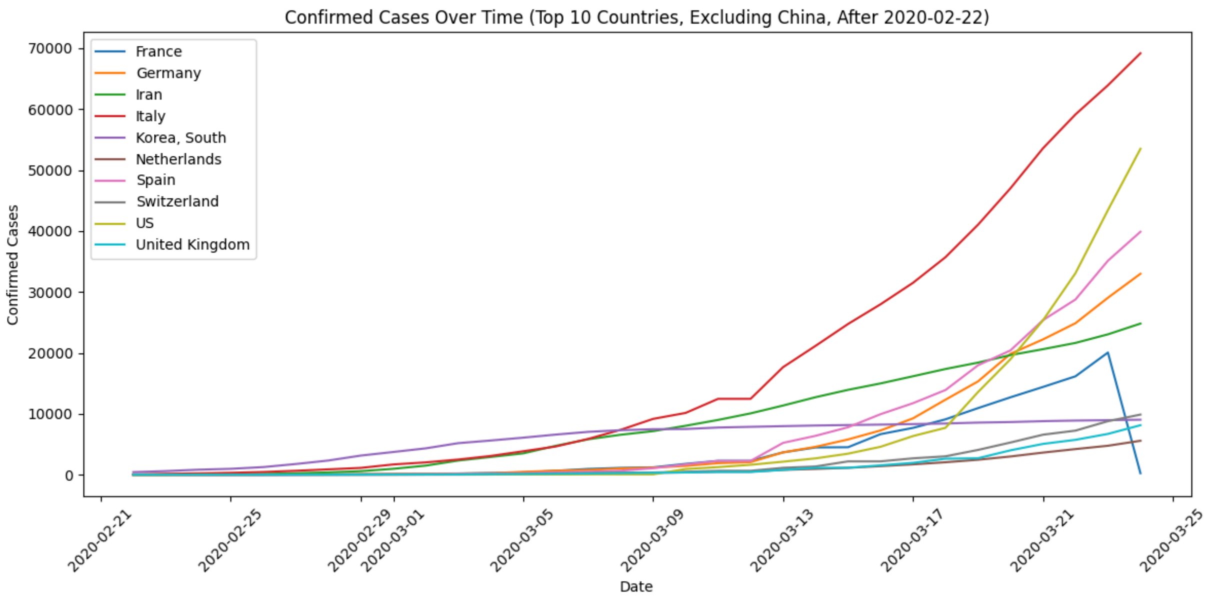
As you can see China is a major outlier for our data because it is the origin location. Remove it to clean our data because although it is the origin country, it will skew the overall countries data.

Also, most countries only have cases after Mar 02/2022, so decided to use dates from 2020-02-22 onwards.

In [36]:

```python
df_new = df[df['Country/Region'] != 'China'].copy()

# Keep only rows on or after "2020-02-22"
df_new = df_new[df_new['Date'] >= '2020-02-22']

# Aggregate (Date, Country/Region) so if there are multiple rows per date-co
df_agg = df_new.groupby(["Date", "Country/Region"], as_index=False)["Confirm

# Identify the top 10 countries by total confirmed cases
country_totals = df_agg.groupby("Country/Region")["ConfirmedCases"].sum()
top_10_countries = country_totals.nlargest(10).index

# Filter to those 10 countries
df_top10 = df_agg[df_agg["Country/Region"].isin(top_10_countries)]

# Pivot: rows = Date, columns = Country, values = ConfirmedCases
df_pivot = df_top10.pivot_table(
    index="Date",
    columns="Country/Region",
    values="ConfirmedCases",
    aggfunc="sum"
).sort_index()
```

```
# Plot each country as its own line
plt.figure(figsize=(12, 6))
for country in df_pivot.columns:
    plt.plot(df_pivot.index, df_pivot[country], label=country)

plt.xlabel("Date")
plt.ylabel("Confirmed Cases")
plt.title("Confirmed Cases Over Time (Top 10 Countries, Excluding China, Aft
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

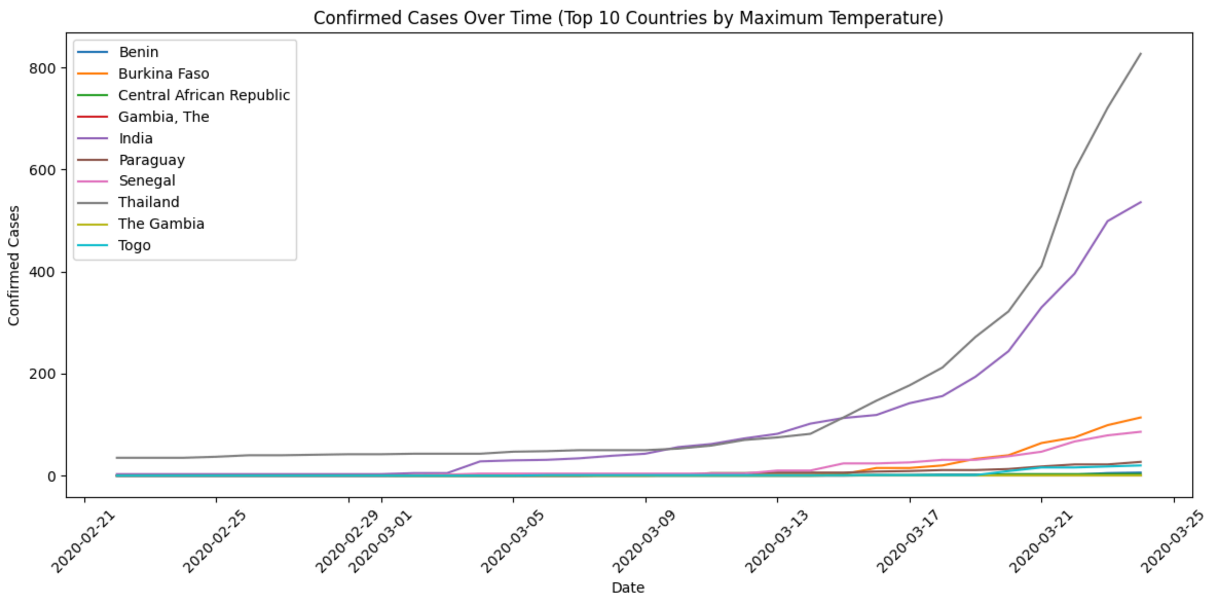


Brainstorming to see if there are any patterns amoung hottest countries (Max Temp).

Growth of infection rates is ALWAYS exponential regardless of which used.

In [37]:
```
# Identify the top 10 countries by maximum temperature
temp_by_country = df_new.groupby("Country/Region")["max"].max()
top_10_temp_countries = temp_by_country.nlargest(10).index

# Filter df_new to only those top 10 countries
df_temp_top10 = df_new[df_new["Country/Region"].isin(top_10_temp_countries)]

# Aggregate (Date, Country/Region) so if multiple rows exist for that combir
df_agg = df_temp_top10.groupby(["Date", "Country/Region"], as_index=False)["

# Pivot: rows = Date, columns = Country, values = ConfirmedCases
df_pivot = df_agg.pivot_table(
    index="Date",
    columns="Country/Region",
    values="ConfirmedCases",
    aggfunc="sum"
).sort_index()

# Plot each country (with highest max temp) as its own line
plt.figure(figsize=(12, 6))
```

```
for country in df_pivot.columns:
    plt.plot(df_pivot.index, df_pivot[country], label=country)

plt.xlabel("Date")
plt.ylabel("Confirmed Cases")
plt.title("Confirmed Cases Over Time (Top 10 Countries by Maximum Temperatur
plt.legend()
plt.xticks(rotation=45)  # Rotate labels if needed
plt.tight_layout()
plt.show()
```



One key issue with COVID infections is that there is an incubation period.
Studies showed between 2-14 days between infection and symptoms.
Used average incubation time of 7 days to create a "lag feature" for all datapoints Next tried New Cases instead of total Confirmed Cases. This makes it "per day" rather than totals. Because of the exponential growth feature, applied a log function to try to make it more linear.

In [57]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Exclude China and sort
df_no_china = df[df["Country/Region"] != "China"].copy()
df_no_china.sort_values(by=["Country/Region", "date"], inplace=True)

# Create 7-day lag
# Include "temp" plus the other weather columns
weather_cols = ["temp", "max", "min", "fog", "prcp", "wdsp", "stp"]
for col in weather_cols:
    df_no_china[f"{col}_lag_7"] = (
        df_no_china.groupby("Country/Region")[col]
                   .shift(7)
    )

# Filter to after 2020-02-22
```

```python
# This ensures each row can reference up to 7 days earlier
df_new = df_no_china[df_no_china["date"] >= "2020-02-22"].copy()

# Aggregate (Date, Country/Region)
# Summing confirmed cases, averaging the new lag columns.
agg_dict = {
    "ConfirmedCases": "sum",
    "temp_lag_7": "mean",
    "max_lag_7": "mean",
    "min_lag_7": "mean",
    "fog_lag_7": "mean",   # fog is boolean, so mean = fraction of days that
    "prcp_lag_7": "mean",
    "wdsp_lag_7": "mean",
    "stp_lag_7": "mean"
}
df_agg = df_new.groupby(["Date", "Country/Region"], as_index=False).agg(agg_

# Create a "newcases" column which is:
# Today's cases minus yesterday's, grouped by country.
# Note: We do this on df_no_china if we want the lag to be correct,
#       then filter to 2020-02-22 if desired for final usage.
df_no_china["newcases"] = (
    df_no_china.groupby("Country/Region")["ConfirmedCases"]
               .diff()
               .fillna(0)
)

# If you only want newcases in df_new (after date filter), assign it there a
df_new["newcases"] = (
    df_new.groupby("Country/Region")["ConfirmedCases"]
          .diff()
          .fillna(0)
)
df_new["logcases"] = np.log1p(df_new["newcases"])
df_new.drop(
    columns=[
        "Province/State",
        "Lat",
        "Long",
        "day_from_jan_first",
        "slp",
        "dewp",
        "rh",
        "ah"
        "id"
    ],
    inplace=True
)

# Display the first 20 rows of df_new
print(df_new.head(20))
```

	Id	Country/Region	Date	ConfirmedCases	Fatalities	temp	min
31	32	Afghanistan	2020-02-22	0.0	0.0	37.9	26.4
32	33	Afghanistan	2020-02-23	0.0	0.0	39.3	28.2
33	34	Afghanistan	2020-02-24	1.0	0.0	40.0	32.4
34	35	Afghanistan	2020-02-25	1.0	0.0	40.2	32.9
35	36	Afghanistan	2020-02-26	1.0	0.0	46.7	35.2
36	37	Afghanistan	2020-02-27	1.0	0.0	39.3	33.4
37	38	Afghanistan	2020-02-28	1.0	0.0	36.5	29.5
38	39	Afghanistan	2020-02-29	1.0	0.0	36.5	29.5
39	40	Afghanistan	2020-03-01	1.0	0.0	58.6	45.7
40	41	Afghanistan	2020-03-02	1.0	0.0	36.3	32.2
41	42	Afghanistan	2020-03-03	1.0	0.0	36.3	32.2
42	43	Afghanistan	2020-03-04	1.0	0.0	35.4	28.4
43	44	Afghanistan	2020-03-05	1.0	0.0	31.9	27.3
44	45	Afghanistan	2020-03-06	1.0	0.0	34.1	25.2
45	46	Afghanistan	2020-03-07	1.0	0.0	25.0	18.1
46	47	Afghanistan	2020-03-08	4.0	0.0	28.6	16.5
47	48	Afghanistan	2020-03-09	4.0	0.0	31.1	23.0
48	49	Afghanistan	2020-03-10	5.0	0.0	28.9	24.3
49	50	Afghanistan	2020-03-11	7.0	0.0	26.7	21.7
50	51	Afghanistan	2020-03-12	7.0	0.0	30.7	16.9

	max	stp	wdsp	...	new_cases	newcases	temp_lag_7	max_lag_7
31	52.0	781.1	5.6	...	0.0	0.0	26.3	35.8
32	50.2	779.4	6.4	...	0.0	0.0	24.2	33.1
33	48.0	778.4	4.0	...	1.0	1.0	32.9	41.4
34	47.5	778.1	5.8	...	0.0	0.0	32.8	46.2
35	55.9	775.4	6.2	...	0.0	0.0	23.9	33.1
36	48.0	772.6	5.8	...	0.0	0.0	26.0	38.3
37	44.8	773.4	2.8	...	0.0	0.0	34.5	44.4
38	44.8	773.4	2.8	...	0.0	0.0	37.9	52.0
39	73.4	999.9	1.4	...	0.0	0.0	39.3	50.2
40	48.7	771.7	5.8	...	0.0	0.0	40.0	48.0
41	48.7	771.7	5.8	...	0.0	0.0	40.2	47.5
42	47.7	771.1	3.4	...	0.0	0.0	46.7	55.9
43	38.5	772.6	5.1	...	0.0	0.0	39.3	48.0
44	45.0	773.5	4.6	...	0.0	0.0	36.5	44.8
45	34.7	774.9	6.6	...	0.0	0.0	36.5	44.8
46	42.1	774.7	3.6	...	3.0	3.0	58.6	73.4
47	41.9	774.0	4.4	...	0.0	0.0	36.3	48.7
48	39.0	773.3	5.3	...	1.0	1.0	36.3	48.7
49	32.9	772.3	3.9	...	2.0	2.0	35.4	47.7
50	47.7	775.4	2.9	...	0.0	0.0	31.9	38.5

	min_lag_7	fog_lag_7	prcp_lag_7	wdsp_lag_7	stp_lag_7	logcases
31	19.6	0.0	0.00	5.6	780.7	0.000000
32	17.4	0.0	0.00	4.7	783.2	0.000000
33	28.2	0.0	0.00	5.9	782.4	0.693147
34	24.6	1.0	0.04	6.5	779.3	0.000000
35	16.2	1.0	0.04	5.0	778.6	0.000000
36	12.9	0.0	0.00	7.2	778.5	0.000000
37	22.6	1.0	0.00	6.4	778.5	0.000000
38	26.4	1.0	0.00	5.6	781.1	0.000000
39	28.2	1.0	0.00	6.4	779.4	0.000000
40	32.4	1.0	0.47	4.0	778.4	0.000000
41	32.9	1.0	0.00	5.8	778.1	0.000000

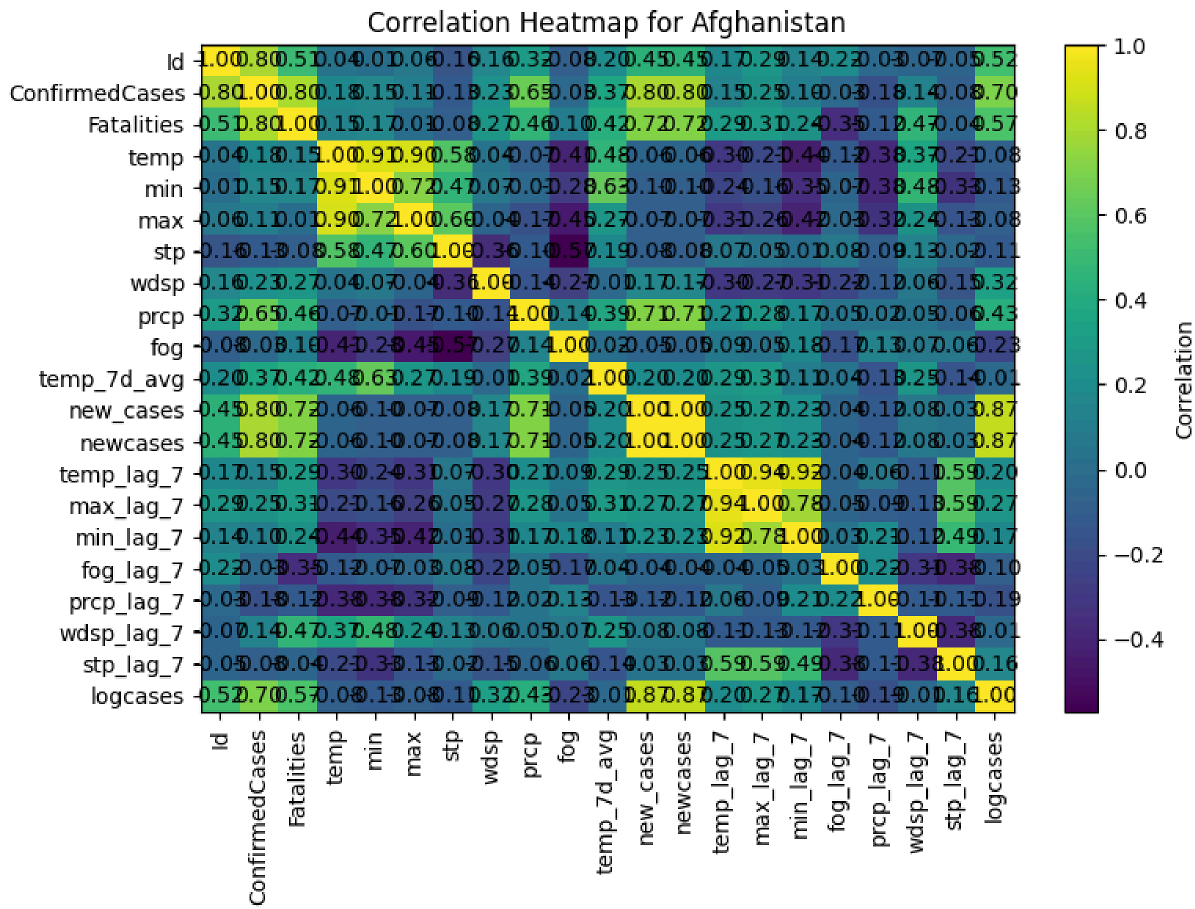
42	35.2	1.0	0.00	6.2	775.4	0.000000
43	33.4	1.0	1.57	5.8	772.6	0.000000
44	29.5	1.0	0.47	2.8	773.4	0.000000
45	29.5	1.0	0.47	2.8	773.4	0.000000
46	45.7	0.0	0.00	1.4	999.9	1.386294
47	32.2	1.0	0.00	5.8	771.7	0.000000
48	32.2	1.0	0.00	5.8	771.7	0.693147
49	28.4	1.0	0.35	3.4	771.1	1.098612
50	27.3	1.0	1.57	5.1	772.6	0.000000

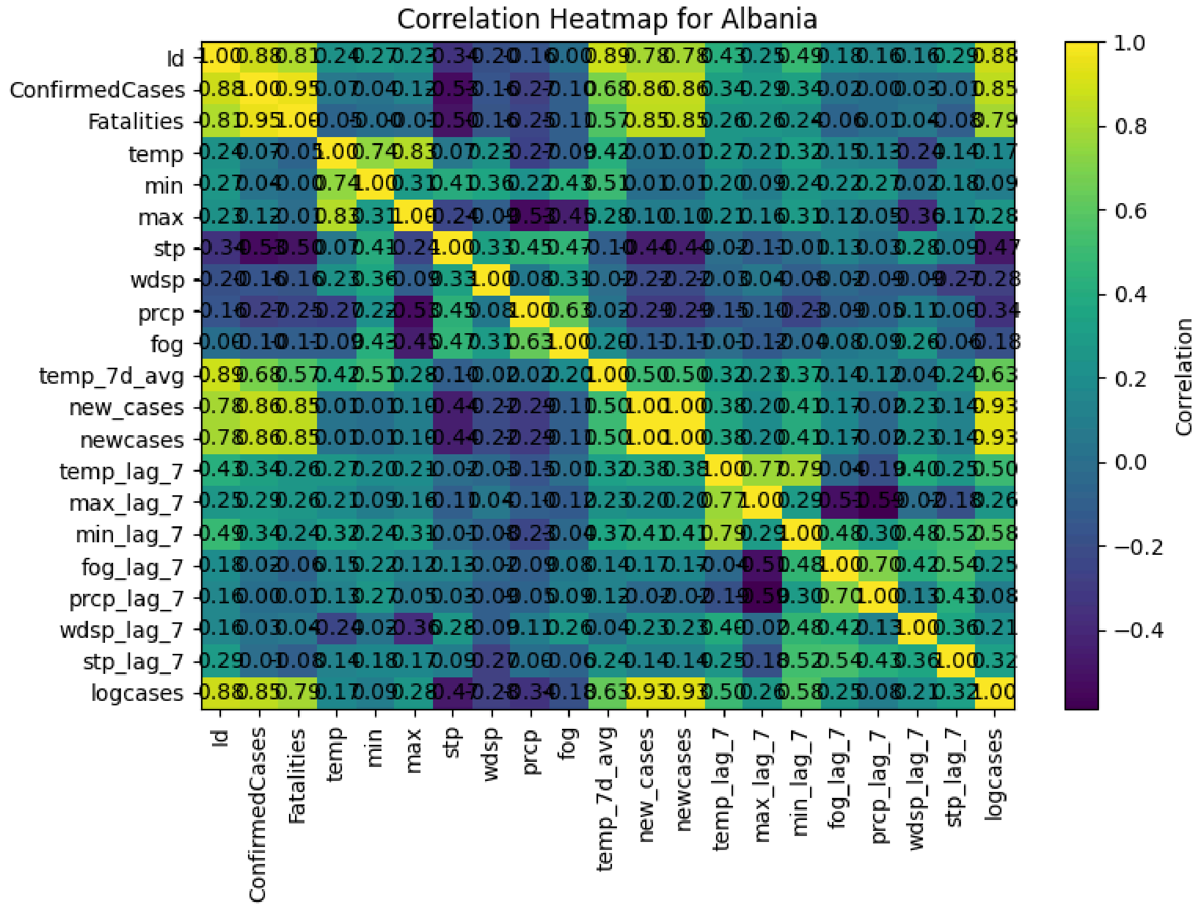
[20 rows x 24 columns]

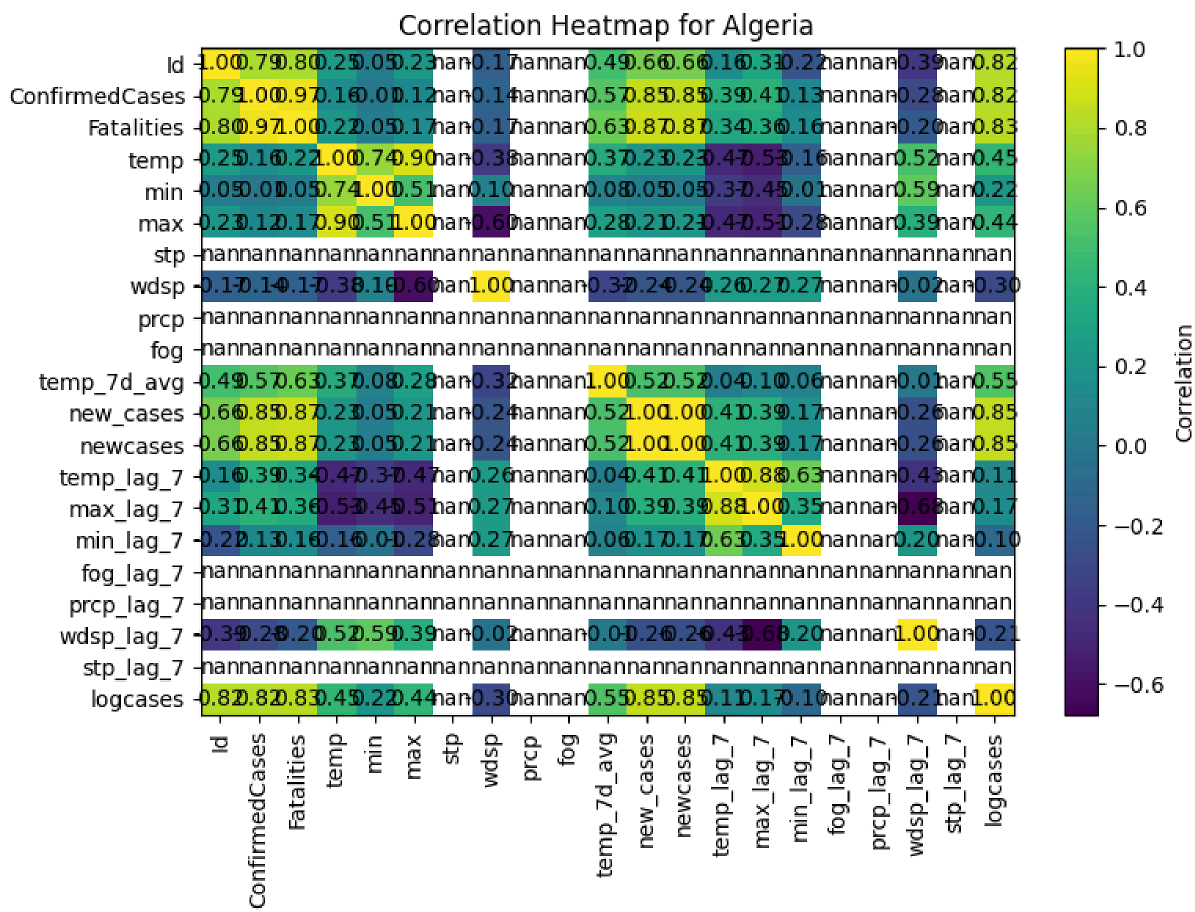
Created some heat maps for different countries to try to see correlation of all variables

In [61]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

Get the first 5 distinct countries
countries = df_new["Country/Region"].unique()[:5]

for country in countries:
#Subset to rows for this country; select only numeric columns
    df_country_numeric = df_new[df_new["Country/Region"] == country].select_

    # If there's not enough data or few numeric columns, you may skip or har
    if df_country_numeric.shape[1] < 2:
        print(f"Skipping {country}: not enough numeric columns to form a cor
        continue

# Compute the correlation matrix
    corr_matrix = df_country_numeric.corr()

# Create a new figure for this country
    plt.figure(figsize=(8, 6))

    # Display the correlation matrix as an image
    plt.imshow(corr_matrix, interpolation='nearest', cmap='viridis', aspect=
    plt.colorbar(label="Correlation")

# Label the x-axis and y-axis ticks with the column names
    columns = corr_matrix.columns
    plt.xticks(range(len(columns)), columns, rotation=90)
    plt.yticks(range(len(columns)), columns)

# Annotate each cell with the correlation value
    for i in range(corr_matrix.shape[0]):
        for j in range(corr_matrix.shape[1]):
            val = corr_matrix.iat[i, j]
            plt.text(j, i, f"{val:.2f}", ha="center", va="center", color="bl

# Title and layout
    plt.title(f"Correlation Heatmap for {country}")
    plt.tight_layout()
```

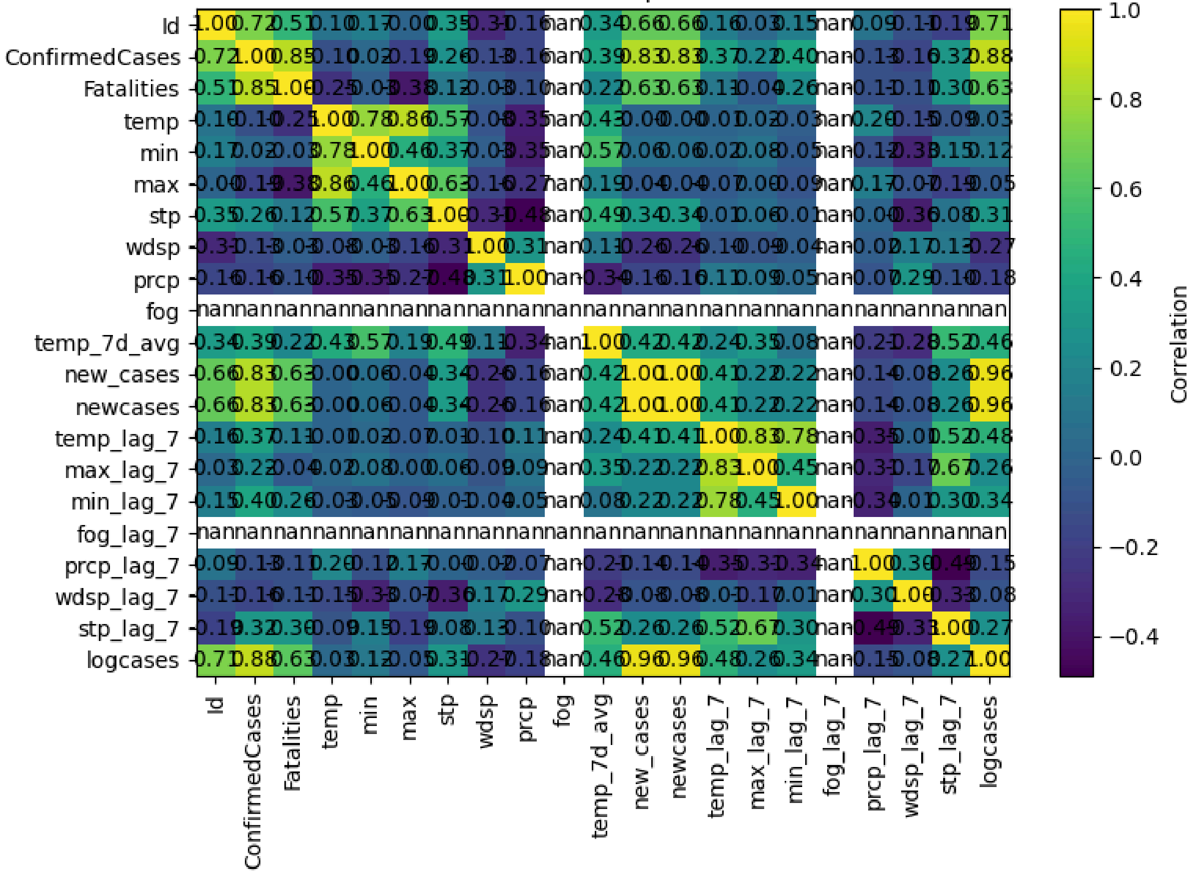
```python
# Display this heatmap
    plt.show()
```



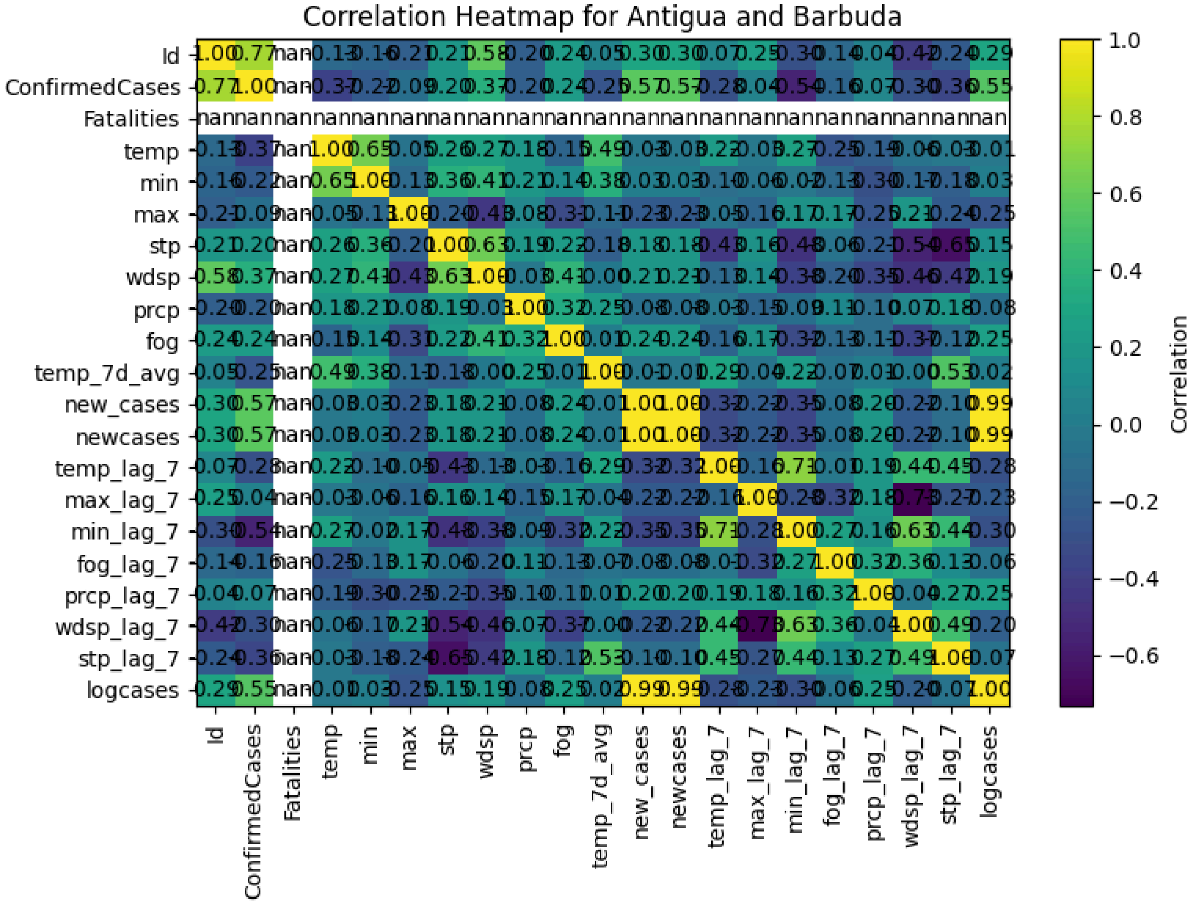

Correlation Heatmap for Albania
Id
ConfirmedCases
Fatalities
temp
min
max
stp
wdsp
prcp
fog
temp_7d_avg
new_cases
newcases
temp_lag_7
max_lag_7
min_lag_7
fog_lag_7
prcp_lag_7
wdsp_lag_7
stp_lag_7
logcases
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
Correlation


Correlation Heatmap for Algeria
Id
ConfirmedCases
Fatalities
temp
min
max
stp
wdsp
prcp
fog
temp_7d_avg
new_cases
newcases
temp_lag_7
max_lag_7
min_lag_7
fog_lag_7
prcp_lag_7
wdsp_lag_7
stp_lag_7
logcases
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
Correlation

Correlation Heatmap for Andorra

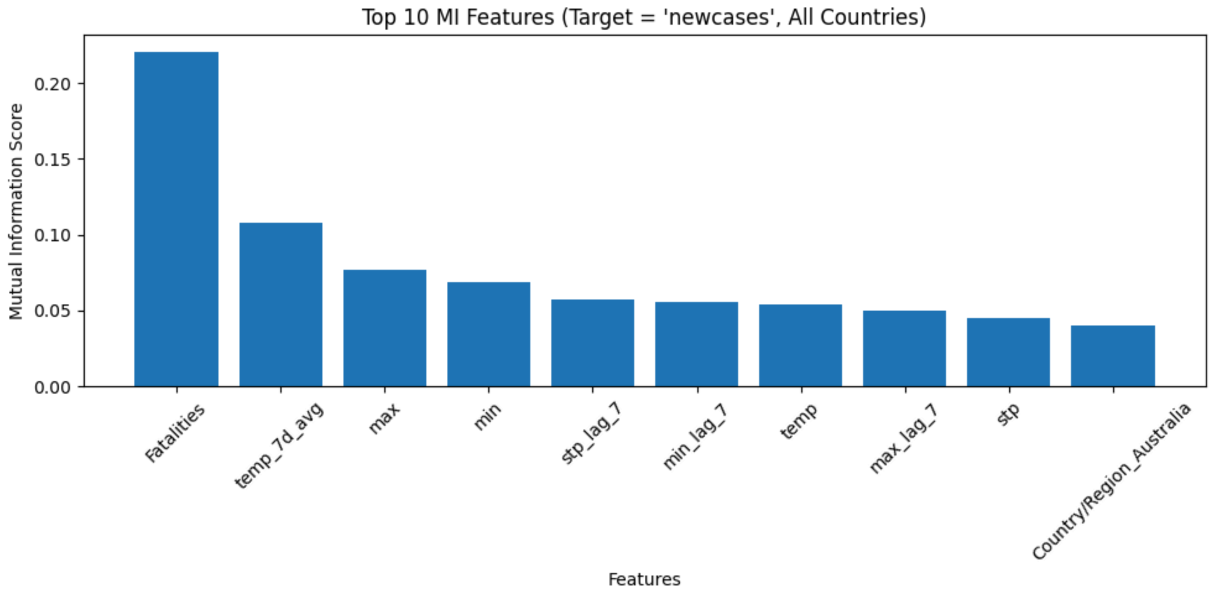


Correlation Heatmap for Antigua and Barbuda



In [ ]:
```
Mutual information features for new cases
```

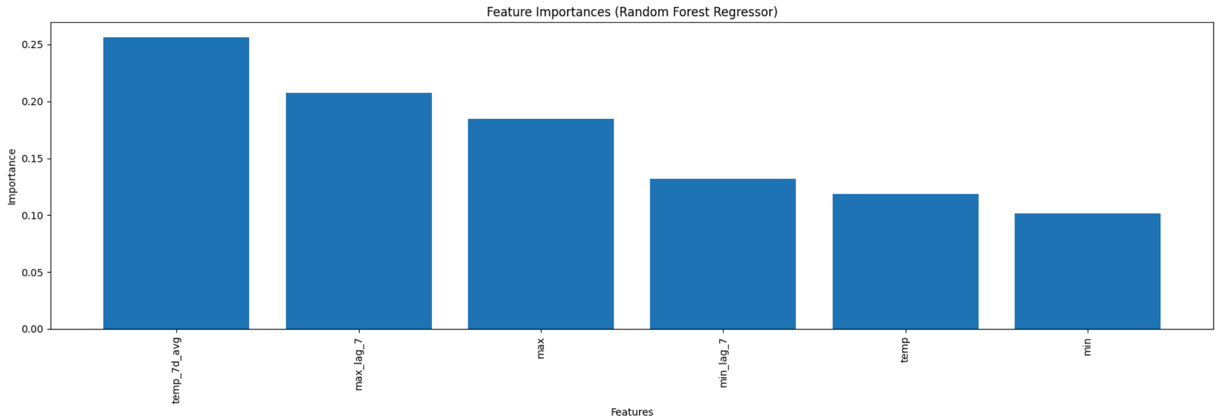
In [74]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import mutual_info_regression

# Use the entire dataset (all countries, even China)
# Assume df_new already exists and includes all countries.
df_all = df_new.copy()

# Set the target and remove excluded columns from features
y_mi = df_all["newcases"]
# Drop "id", "newcases", and "ConfirmedCases" from the features.
X = df_all.drop(columns=["Id", "new_cases", "logcases", "ConfirmedCases","ne

# Encode any categorical variables

X_encoded = pd.get_dummies(X, drop_first=True)

# Ensure all features are numeric and handle inf
# Convert all columns to numeric, coercing errors to NaN.
X_encoded = X_encoded.apply(pd.to_numeric, errors='coerce')
# Replace any infinite values with NaN, then fill NaN with 0.
X_encoded = X_encoded.replace([np.inf, -np.inf], np.nan).fillna(0)

# Calculate mutual information
mi_scores = mutual_info_regression(X_encoded, y_mi)

# Define a function to plot the top-N features
def plot_mutual_information(mi_scores, feature_names, top_n=10):
    sorted_indices = (-mi_scores).argsort()[:top_n]
    top_mi_scores = mi_scores[sorted_indices]
    top_feature_names = feature_names[sorted_indices]

    plt.figure(figsize=(10, 5))
    plt.bar(range(len(top_mi_scores)), top_mi_scores)
    plt.xticks(range(len(top_mi_scores)), top_feature_names, rotation=45)
    plt.xlabel("Features")
    plt.ylabel("Mutual Information Score")
    plt.title(f"Top {top_n} MI Features (Target = 'newcases', All Countries)
    plt.tight_layout()
    plt.show()

# Plot the top 10 features
plot_mutual_information(mi_scores, X_encoded.columns, top_n=10)
```



Do a Random Forest to determine key features in the data

In [73]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Create dummy variables for a categorical feature (e.g., 'Period')
period_dummies = pd.get_dummies(df_new['Date'], prefix='Date', drop_first=Tr

# Concatenate the dummy variables to df_new to form df_importance
df_importance = pd.concat([df_new, period_dummies], axis=1)

# Define the features and target
# Here, we select a set of features. Adjust the list below to match your dat
features = df_importance[['temp_7d_avg', 'max', 'min', 'temp',
                          'max_lag_7', 'min_lag_7']]
# The target variable is 'newcases'
target = df_importance['newcases']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,
                                                    test_size=0.2, random_st

# Create and fit the Random Forest Regressor
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)

# Extract feature importances
importances = rf.feature_importances_
feature_importances = pd.DataFrame({'Feature': features.columns, 'Importance
feature_importances = feature_importances.sort_values(by='Importance', ascer

# Plot the feature importances
plt.figure(figsize=(17, 6))
plt.bar(feature_importances['Feature'], feature_importances['Importance'])
plt.xticks(range(len(feature_importances['Feature'])),
```

```
            feature_importances['Feature'], rotation=90)
plt.title('Feature Importances (Random Forest Regressor)')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.tight_layout()
plt.show()
```



Feature Engineering

This snippet code encodes the cetagorical features.

- In order to keep end of each month close to the first day of next month, I moved Day to a 2D space.
- The same thing was tested for the week feature but it didn't turned out to work well.