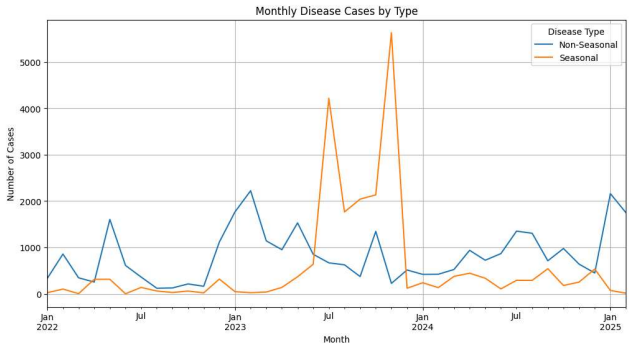
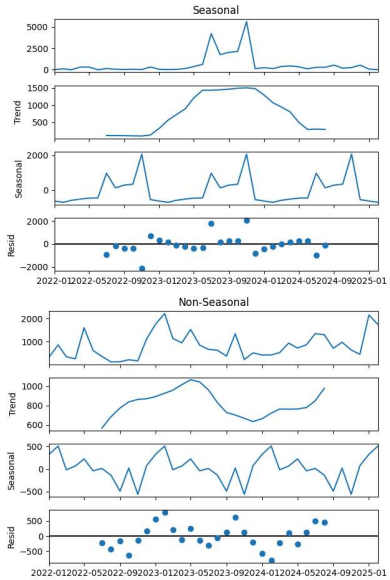
Disease Outbreak Analysis and Forecasting

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.statespace.sarimax import SARIMAX

from prophet import Prophet

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

# Load and preprocess the dataset
df = pd.read_excel("/content/Copy of 22-25_Combined_New(1).xlsx")
df.columns = df.columns.str.strip()
df['Date of outbreak'] = pd.to_datetime(df['Date of outbreak'], errors='coerce')

# Keep records only till 2025
df = df[df['Date of outbreak'].dt.year <= 2025]
df = df.dropna(subset=['Date of outbreak', 'Diseases'])

# Categorize diseases
seasonal = ['Dengue', 'Malaria', 'Influenza']
non_seasonal = ['Chickenpox', 'Food Poisoning']

def categorize(disease):
    for s in seasonal:
        if s.lower() in disease.lower():
            return 'Seasonal'
    for ns in non_seasonal:
        if ns.lower() in disease.lower():
            return 'Non-Seasonal'
    return 'Other'

df['Disease Type'] = df['Diseases'].apply(categorize)
df = df[df['Disease Type'].isin(['Seasonal', 'Non-Seasonal'])]

# Monthly aggregation
df['Month'] = df['Date of outbreak'].dt.to_period('M').dt.to_timestamp()
monthly = df.groupby(['Month', 'Disease Type'])['No of cases'].sum().unstack().fillna(0)

# Time series plot
monthly.plot(figsize=(12, 6), title="Monthly Disease Cases by Type")
plt.ylabel("Number of Cases")
plt.grid(True)
plt.show()

# Augmented Dickey-Fuller test
def adf_test(series, title=''):
    print(f'ADF Test - {title}')
    result = adfuller(series.dropna())
    labels = ['ADF Statistic', 'p-value', 'Lags Used', 'Observations Used']
    for val, label in zip(result, labels):
        print(f'{label}: {val}')
    print("Conclusion: Stationary\n" if result[1] <= 0.05 else "Conclusion: Non-Stationary\n")

adf_test(monthly['Seasonal'], 'Seasonal Diseases')
adf_test(monthly['Non-Seasonal'], 'Non-Seasonal Diseases')

# Seasonal decomposition
seasonal_decomp = seasonal_decompose(monthly['Seasonal'], model='additive', period=12)
seasonal_decomp.plot()
plt.suptitle("")
plt.show()

non_seasonal_decomp = seasonal_decompose(monthly['Non-Seasonal'], model='additive', period=12)
non_seasonal_decomp.plot()
plt.suptitle("")
plt.show()

# SARIMA modeling
sarima_seasonal = SARIMAX(monthly['Seasonal'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
seasonal_result = sarima_seasonal.fit(disp=False)
monthly['SARIMA_Forecast_Seasonal'] = seasonal_result.predict(start=0, end=len(monthly)-1)

monthly[['Seasonal', 'SARIMA_Forecast_Seasonal']].plot(figsize=(12, 6), title="SARIMA Forecast - Seasonal Diseases")
plt.ylabel("Number of Cases")
plt.grid(True)
plt.show()

sarima_non_seasonal = SARIMAX(monthly['Non-Seasonal'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
non_seasonal_result = sarima_non_seasonal.fit(disp=False)
monthly['SARIMA_Forecast_Non_Seasonal'] = non_seasonal_result.predict(start=0, end=len(monthly)-1)

monthly[['Non-Seasonal', 'SARIMA_Forecast_Non_Seasonal']].plot(figsize=(12, 6), title="SARIMA Forecast - Non-Seasonal Diseases")
plt.ylabel("Number of Cases")
plt.grid(True)
plt.show()

# Prophet forecasting
# Seasonal
prophet_seasonal = monthly.reset_index()[['Month', 'Seasonal']].rename(columns={'Month': 'ds', 'Seasonal': 'y'})
model_seasonal = Prophet()
model_seasonal.fit(prophet_seasonal)
future_seasonal = model_seasonal.make_future_dataframe(periods=6, freq='M')
forecast_seasonal = model_seasonal.predict(future_seasonal)
model_seasonal.plot(forecast_seasonal)
plt.title("Prophet Forecast - Seasonal Diseases")
plt.show()

# Non-Seasonal
prophet_non_seasonal = monthly.reset_index()[['Month', 'Non-Seasonal']].rename(columns={'Month': 'ds', 'Non-Seasonal': 'y'})
model_non_seasonal = Prophet()
model_non_seasonal.fit(prophet_non_seasonal)
future_non_seasonal = model_non_seasonal.make_future_dataframe(periods=6, freq='M')
forecast_non_seasonal = model_non_seasonal.predict(future_non_seasonal)
model_non_seasonal.plot(forecast_non_seasonal)
plt.title("Prophet Forecast - Non-Seasonal Diseases")
plt.show()

# Summary statistics
print(f"Seasonal - Mean: {monthly['Seasonal'].mean()}, Max: {monthly['Seasonal'].max()}")
print(f"Non-Seasonal - Mean: {monthly['Non-Seasonal'].mean()}, Max: {monthly['Non-Seasonal'].max()}")

# Random Forest - Outbreak Prediction
df['Outbreak'] = df['No of cases'].apply(lambda x: 1 if x >= 10 else 0)
df['Month'] = pd.to_datetime(df['Month'])

# Feature engineering
model_data = df.groupby('Month').agg({'No of cases': 'sum', 'Outbreak': 'max'}).reset_index()
model_data['Month_num'] = model_data['Month'].dt.month
model_data['Year'] = model_data['Month'].dt.year
model_data['Lag_1'] = model_data['No of cases'].shift(1).fillna(0)

# Prepare data
X = model_data[['Month_num', 'Year', 'Lag_1']]
y = model_data['Outbreak']

# Train model with cross-validation
rf = RandomForestClassifier(n_estimators=100, random_state=42)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(rf, X, y, cv=cv, scoring='accuracy')

print(f"\nCross-validation scores: {scores}")
print(f"Mean Accuracy: {scores.mean():.4f}, Std Dev: {scores.std():.4f}")

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# Evaluation
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plot predictions
plt.figure(figsize=(10, 5))
plt.plot(y_test.values, label='Actual', marker='o')
plt.plot(y_pred, label='Predicted', marker='x')
plt.title("Random Forest Outbreak Predictions")
plt.xlabel("Time Index")
plt.ylabel("Outbreak (0 = No, 1 = Yes)")
plt.legend()
plt.grid(True)
plt.show()

# Confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=[0, 1])
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Outbreak", "Outbreak"]).plot()
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

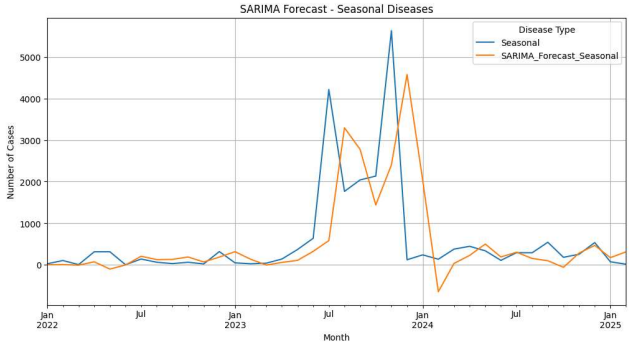
[+]



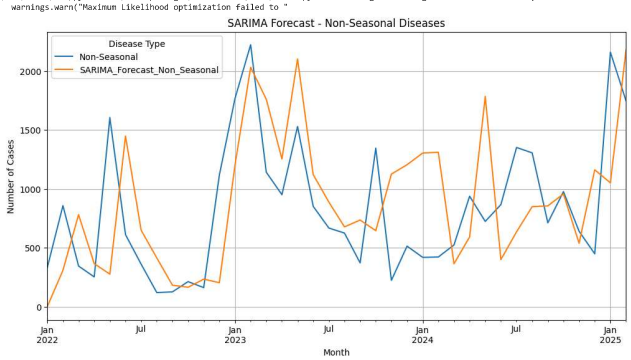

```
ADF Test - Seasonal Diseases
ADF Statistic: -3.990011480892016
p-value: 0.0014637267146866797
Lags Used: 0
Observations Used: 37
Conclusion: Stationary

ADF Test - Non-Seasonal Diseases
ADF Statistic: -3.708526787248083
p-value: 0.0039948033571888557
Lags Used: 0
Observations Used: 37
Conclusion: Stationary
```



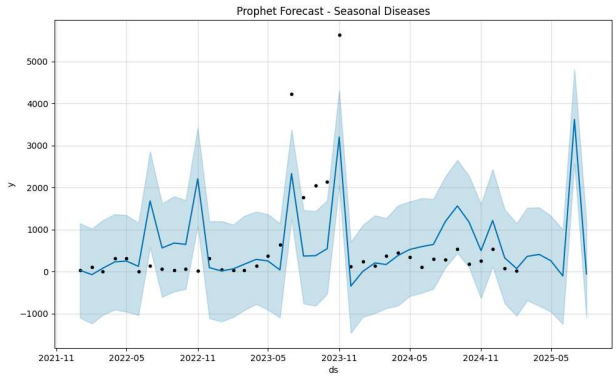

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate starting parameters for seasonal ARMA. All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.11/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```



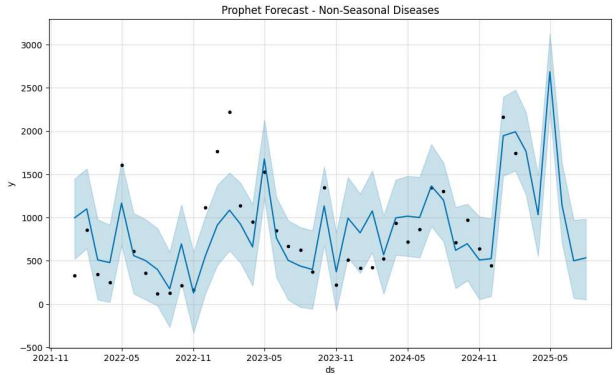

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate starting parameters for seasonal ARMA. All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.11/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```



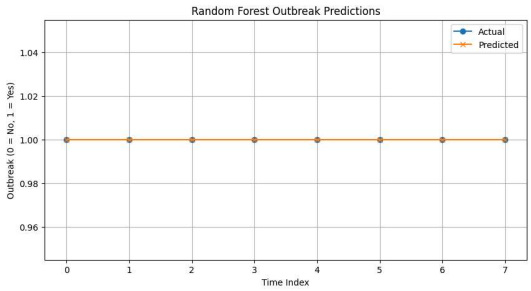

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/f9hjnvjv.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/18mbp_37.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=60931', 'data', 'file=/tmp/tmpw6ucpt6i/f9hjnvjv.json', 'init=/tmp/tmpw6ucpt6i/18mbp_37.json', 'output', 'file=/tmp/tmpw6ucpt6i/prophet_modeltwuhr9ka/prophet_model-202
17:37:55 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
17:37:55 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
  dates = pd.date_range(
```

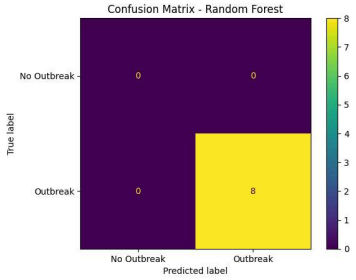



```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/xcg1a8sv.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/8bgoqek6.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=49602', 'data', 'file=/tmp/tmpw6ucpt6i/xcg1a8sv.json', 'init=/tmp/tmpw6ucpt6i/8bgoqek6.json', 'output', 'file=/tmp/tmpw6ucpt6i/prophet_model18az68n7i/prophet_model-202
17:37:56 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
17:37:56 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
  dates = pd.date_range(
```



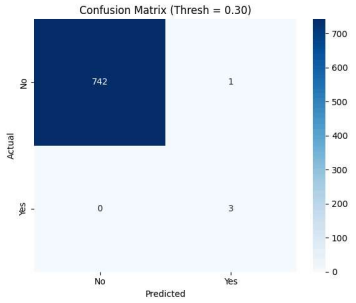

```
Seasonal - Mean: 587.1315789473684, Max: 5633.0
Non-Seasonal - Mean: 829.5526315789474, Max: 2223.0

Cross-validation scores: [1. 1. 1. 1. 1.]
Mean Accuracy: 1.0000, Std Dev: 0.0000

Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         8

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8
```

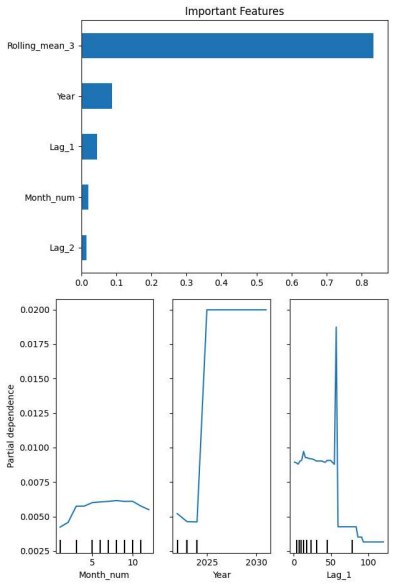




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix, precision_recall_curve, f1_score, auc
from imblearn.over_sampling import SMOTE
from sklearn.calibration import CalibratedClassifierCV
from sklearn.inspection import PartialDependenceDisplay

# 1. LOADING AND PREPARING DATA

# Reading the dataset from Excel
df = pd.read_excel("/content/Copy of 22-25_Combined_New(1).xlsx")
df.columns = df.columns.str.strip()  # Removing any extra spaces in column names

# Converting date column to datetime
df['Date of outbreak'] = pd.to_datetime(df['Date of outbreak'], errors='coerce')

# Creating new features
df['Month_num'] = df['Date of outbreak'].dt.month
df['Year'] = df['Date of outbreak'].dt.year
df['Lag_1'] = df['No of cases'].shift(1).fillna(0)
df['Lag_2'] = df['No of cases'].shift(2).fillna(0)
df['Rolling_mean_3'] = df['No of cases'].rolling(window=3).mean().fillna(0)

# Creating the target label (outbreak or not)
df['Outbreak'] = (df['No of cases'] >= 500).astype(int)

# Picking the required columns
final_df = df[['Month_num', 'Year', 'Lag_1', 'Lag_2', 'Rolling_mean_3', 'Outbreak']].dropna()

# Splitting features and target
X = final_df.drop('Outbreak', axis=1)
y = final_df['Outbreak']

# Checking how many outbreak and non-outbreak entries we have
print("Class distribution before resampling:\n", y.value_counts())

# 2. TRAINING RANDOM FOREST MODEL

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

# Balancing the training data using SMOTE
sm = SMOTE(random_state=42)
X_train_bal, y_train_bal = sm.fit_resample(X_train, y_train)

# Checking new class distribution
print("\nAfter SMOTE:\n", pd.Series(y_train_bal).value_counts())

# Building the model
model = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')

# Doing cross-validation to check performance
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(model, X, y, cv=cv, scoring='accuracy')
print("\nCV Accuracy scores:", cv_scores)
print("Mean accuracy = {:.4f}, Std = {:.4f}".format(cv_scores.mean(), cv_scores.std()))

# Training the model on balanced data
model.fit(X_train_bal, y_train_bal)

# 3. EVALUATING PERFORMANCE

# Getting prediction probabilities
y_probs = model.predict_proba(X_test)[:, 1]

# Calculating ROC-AUC and PR-AUC
roc = roc_auc_score(y_test, y_probs)
prec, rec, thresh = precision_recall_curve(y_test, y_probs)
pr = auc(rec, prec)
print("\nROC-AUC = {:.4f}".format(roc))
print("PR-AUC = {:.4f}".format(pr))

# Finding the threshold with best F1 score
f1s = [f1_score(y_test, y_probs >= t) for t in thresh]
best_idx = np.argmax(f1s)
best_threshold = thresh[best_idx]
print("\nBest threshold = {:.2f} (F1 = {:.4f})".format(best_threshold, f1s[best_idx]))

# Making final predictions based on best threshold
y_pred = (y_probs >= best_threshold).astype(int)

# Showing classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=["No_Outbreak", "Outbreak"]))

# Confusion matrix heatmap
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
plt.title(f"Confusion Matrix (Thresh = {best_threshold:.2f})")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

# 4. FEATURE IMPORTANCE PLOT

# Checking which features were most useful
feat_imp = pd.Series(model.feature_importances_, index=X.columns)
feat_imp.sort_values().plot(kind='barh', title='Important Features')
plt.tight_layout()
plt.show()

# 5. SAVING MODEL AND THRESHOLD

joblib.dump(model, 'outbreak_rf_model.pkl')
joblib.dump(best_threshold, 'best_threshold.pkl')

# 6. PREDICTING ON NEW DATASET

def run_alert_prediction(file_path):
    clf = joblib.load('outbreak_rf_model.pkl')
    threshold = joblib.load('best_threshold.pkl')

    new_df = pd.read_excel(file_path)
    new_df['Date of outbreak'] = pd.to_datetime(new_df['Date of outbreak'], errors='coerce')
    new_df['Month_num'] = new_df['Date of outbreak'].dt.month
    new_df['Year'] = new_df['Date of outbreak'].dt.year
    new_df['Lag_1'] = new_df['No of cases'].shift(1).fillna(0)
    new_df['Lag_2'] = new_df['No of cases'].shift(2).fillna(0)
    new_df['Rolling_mean_3'] = new_df['No of cases'].rolling(3).mean().fillna(0)

    features = new_df[['Month_num', 'Year', 'Lag_1', 'Lag_2', 'Rolling_mean_3']].fillna(0)
    new_df['Outbreak_Prob'] = clf.predict_proba(features)[:, 1]
    new_df['Outbreak_Flag'] = (new_df['Outbreak_Prob'] >= threshold).astype(int)

    flagged = new_df[new_df['Outbreak_Flag'] == 1]
    flagged.to_excel("Predicted_Outbreaks.xlsx", index=False)
    print(f"\n{len(flagged)} outbreaks flagged and saved.")
    return flagged

# 7. PARTIAL DEPENDENCE PLOT (Just Exploring)

PartialDependenceDisplay.from_estimator(
    model, X_train, features=[0, 1, 2], feature_names=X.columns.tolist(), grid_resolution=50
)
plt.tight_layout()
plt.show()

# 8. GENERATING ALERT LIST FROM TEST SET

alert_df = pd.DataFrame({
    'Date': df.loc[X_test.index, 'Date of outbreak'],
    'Predicted Outbreak': y_pred,
    'Outbreak Probability': y_probs
})
alert_df = alert_df[alert_df['Predicted Outbreak'] == 1]
print("\nALERTS GENERATED:\n", alert_df)

# Saving alerts to file
alert_df.to_csv("outbreak_alerts.csv", index=False)
```

```
Class distribution before resampling:
 Outbreak
0     3711
1       16
Name: count, dtype: int64

After SMOTE:
 Outbreak
0    2968
1    2968
Name: count, dtype: int64

CV Accuracy scores: [0.99731903 0.99463807 0.99731544 0.99865772 0.99865772]
Mean accuracy = 0.9973, Std = 0.0015

ROC-AUC = 0.9996
PR-AUC = 0.9028

Best threshold = 0.30 (F1 = 0.8571)

Classification Report:
              precision    recall  f1-score   support

 No_Outbreak       1.00      1.00      1.00       743
    Outbreak       0.75      1.00      0.86         3

    accuracy                           1.00       746
   macro avg       0.88      1.00      0.93       746
weighted avg       1.00      1.00      1.00       746
```






```
ALERTS GENERATED:
           Date  Predicted Outbreak  Outbreak Probability
3788 2024-01-05                   1                  0.79
68   2022-05-03                   1                  0.30
6126 2025-02-06                   1                  0.33
1220 2023-02-18                   1                  0.71
```

```
!pip uninstall fbprophet -y
```

```
WARNING: Skipping fbprophet as it is not installed.
```

```
!pip install prophet
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.11/dist-packages (1.1.6)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.11/dist-packages (from prophet) (1.2.5)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.11/dist-packages (from prophet) (2.0.2)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from prophet) (3.10.0)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.11/dist-packages (from prophet) (2.2.2)
Requirement already satisfied: holidays<1,>=0.25 in /usr/local/lib/python3.11/dist-packages (from prophet) (0.70)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.11/dist-packages (from prophet) (4.67.1)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.11/dist-packages (from prophet) (6.5.2)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from holidays<1,>=0.25->prophet) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->prophet) (3.2.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil->holidays<1,>=0.25->prophet) (1.17.0)
```

```
print("Available columns:", data.columns.tolist())
```

```
Available columns: ['Unique ID', 'State', 'District', 'Diseases ', 'No of cases', 'No of Death', 'Date of outbreak', 'Date of Report', 'Current Status', 'Action Taken']
```

```
import pandas as pd
import matplotlib.pyplot as plt
from prophet import Prophet

# Loading the dataset
data = pd.read_excel("/content/Copy of 22-25_Combined_New(1).xlsx")

# Cleaning column names
data.columns = data.columns.str.strip().str.lower()

# Making sure the 'diseases' column exists
if 'diseases' not in data.columns:
    raise KeyError("Column 'diseases' not found in the dataset.")

# Cleaning disease names
data['diseases_clean'] = data['diseases'].str.strip().str.lower()

# Defining diseases of interest and the outbreak threshold
target_diseases = ['Dengue', 'Malaria', 'Influenza', 'Chickenpox', 'Food Poisoning']
threshold = 500

# Looping through each disease
for disease in target_diseases:
    print(f"\nProcessing: {disease}")
    disease_clean = disease.strip().lower()

    # Filtering data for the specific disease
    disease_data = data[data['diseases_clean'] == disease_clean].copy()

    # Skipping if no data available
    if disease_data.empty:
```

```
        print(f"No data found for {disease}. Skipping.")
        continue

    # Converting date column to datetime format
    disease_data['date of outbreak'] = pd.to_datetime(
        disease_data['date of outbreak'], dayfirst=True, errors='coerce'
    )

    # Removing rows with invalid dates
    disease_data = disease_data.dropna(subset=['date of outbreak'])

    # Renaming columns for Prophet
    prophet_data = disease_data.rename(columns={'date of outbreak': 'ds', 'no of cases': 'y'})

    # Keeping only date and case count
    prophet_data = prophet_data[['ds', 'y']].sort_values('ds')

    # Checking if there are enough data points
    if len(prophet_data) < 2:
        print(f"Not enough data points for {disease}. Skipping.")
        continue

    # Initializing and training the Prophet model
    model = Prophet(daily_seasonality=False, yearly_seasonality=True)
    model.fit(prophet_data)

    # Creating a future dataframe for prediction
    future = model.make_future_dataframe(periods=365)

    # Making the forecast
    forecast = model.predict(future)

    # Creating subplots for combined visualization
    fig, axes = plt.subplots(2, 1, figsize=(10, 10))

    # Plotting the forecast
    model.plot(forecast, ax=axes[0])
    axes[0].set_title(f'{disease} Outbreak Forecast (Next 1 Year)', fontsize=14)
    axes[0].set_xlabel('Date')
    axes[0].set_ylabel('Number of Cases')

    # Extracting and plotting trend component separately
    components = model.plot_components(forecast)
    components_axes = components.get_axes()

    for child in components_axes[0].get_children():
        if isinstance(child, plt.Line2D):
            axes[1].plot(child.get_xdata(), child.get_ydata())

    axes[1].set_title(f'{disease} - Trend Component', fontsize=12)
    axes[1].set_xlabel('Date')

    # Saving the combined plot as image
    plt.tight_layout()
    fig.savefig(f"{disease_clean.replace(' ', '_')}_forecast_combined.png")
    plt.close(fig)

    # Saving forecast data as CSV
    forecast_file = f"{disease_clean.replace(' ', '_')}_forecast.csv"
    forecast.to_csv(forecast_file, index=False)

    # Selecting future predictions beyond the original dataset
    future_predictions = forecast[forecast['ds'] > prophet_data['ds'].max()][['ds', 'yhat', 'yhat_lower', 'yhat_upper']]

    # Showing top 10 future predictions
    print(f"Top 10 Future Predictions for {disease}:")
    print(future_predictions.head(10))

    # Identifying potential outbreak dates based on threshold
    alerts = future_predictions[future_predictions['yhat'] > threshold]

    if not alerts.empty:
        print(f"Outbreak Alert for {disease}! Predicted cases exceed {threshold} on:")
        print(alerts[['ds', 'yhat']])
    else:
        print(f"No predicted outbreaks above {threshold} for {disease}.")

    print(f"Plots and forecast saved for {disease}.")
```

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/0gw25gmq.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/moiess4z.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=22951', 'data', 'file=/tmp/tmpw6ucpt6i/0gw25gmq.json', 'init=/tmp/tmpw6ucpt6i/moiess4z.json', 'output', 'file=/tmp/tmpw6ucpt6i/prophet_modelvey_3q1w/prophet_model-202
17:38:29 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
17:38:29 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing

Processing: Dengue
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/nb2xic5y.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/iuu05qp2.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=89564', 'data', 'file=/tmp/tmpw6ucpt6i/nb2xic5y.json', 'init=/tmp/tmpw6ucpt6i/iuu05qp2.json', 'output', 'file=/tmp/tmpw6ucpt6i/prophet_model9b89zn70/prophet_model-202
17:38:29 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
17:38:29 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Top 10 Future Predictions for Dengue:
            ds       yhat  yhat_lower  yhat_upper
325 2025-02-11  38.502988 -153.778581  221.059211
326 2025-02-12  29.546110 -157.502275  216.649393
327 2025-02-13  35.356284 -163.090314  214.457943
328 2025-02-14  70.040584 -124.289578  261.816680
329 2025-02-15  63.597228 -109.898020  263.011913
330 2025-02-16  59.202481 -125.169027  242.687917
331 2025-02-17  42.296926 -155.794400  248.518458
332 2025-02-18  49.686580 -144.578990  238.489392
333 2025-02-19  38.975302 -136.494868  223.045094
334 2025-02-20  42.870073 -145.642797  232.137321
No predicted outbreaks above 500 for Dengue.
Plots and forecast saved for Dengue.

Processing: Malaria
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/hxnvvavz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/zf658ksn.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=91713', 'data', 'file=/tmp/tmpw6ucpt6i/hxnvvavz.json', 'init=/tmp/tmpw6ucpt6i/zf658ksn.json', 'output', 'file=/tmp/tmpw6ucpt6i/prophet_modelktlm4lyl/prophet_model-202
17:38:30 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
17:38:30 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Top 10 Future Predictions for Malaria:
            ds        yhat  yhat_lower  yhat_upper
140 2025-02-01  304.186780 -144.787404  753.158996
141 2025-02-02   29.209455 -428.853950  494.583652
142 2025-02-03   48.409876 -413.786720  523.279693
143 2025-02-04   41.240855 -359.707568  457.829425
144 2025-02-05    3.546617 -414.110584  480.517839
145 2025-02-06   -4.944002 -441.322867  432.841128
146 2025-02-07  -83.229291 -519.284585  352.973893
147 2025-02-08  127.014380 -303.291648  576.512147
148 2025-02-09 -161.135300 -602.249625  285.623509
149 2025-02-10 -151.230117 -613.920298  302.364170
Outbreak Alert for Malaria! Predicted cases exceed 500 on:
```

```
print(df.columns)
```

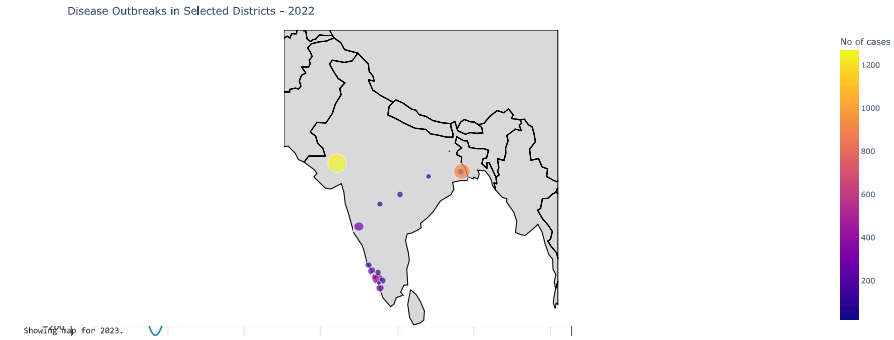
```
Index(['Unique ID', 'State', 'District', 'Diseases', 'No of cases',
       'No of Death', 'Date of outbreak', 'Date of Report', 'Current Status',
       'Action Taken', 'Disease Type', 'Month', 'Outbreak'],
      dtype='object')
```

```
Processing: Chikenpox
DEBUG:cmdstanpy:input tempfile: /tmp/tmpw6ucpt6i/pq7glr2u.json
Not enough data points for influenza. Skipping.
```

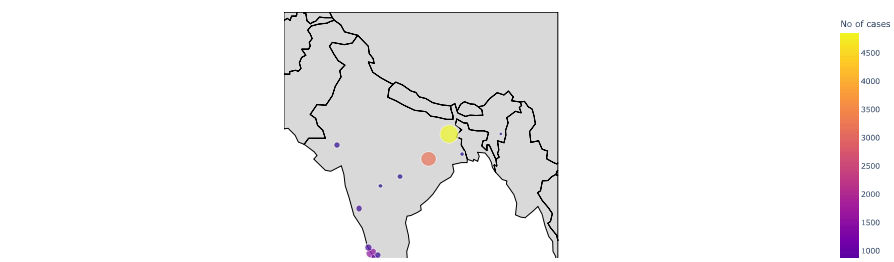
```
import os
import pandas as pd
import plotly.express as px

# Step 1: Loading the dataset
df = pd.read_excel("/content/Copy of 22-25_Combined_New(1).xlsx")

# Step 2: Cleaning column names and date values
df.columns = df.columns.str.strip()
df['Date of outbreak'] = pd.to_datetime(df['Date of outbreak'], errors='coerce')

# Removing rows with missing date, district, or case information
df = df.dropna(subset=['Date of outbreak', 'District', 'No of cases'])

# Step 3: Defining coordinates for selected districts
district_coords = {
    'Kannur': {'Latitude': 11.8745, 'Longitude': 75.3704},
    'Godda': {'Latitude': 24.8286, 'Longitude': 87.2120},
    'Ernakulam': {'Latitude': 9.9816, 'Longitude': 76.2999},
    'Sundargarh': {'Latitude': 22.1167, 'Longitude': 84.0333},
    'Imphal West': {'Latitude': 24.8074, 'Longitude': 93.9326},
    'Malappuram': {'Latitude': 11.0410, 'Longitude': 76.0814},
    'Palakkad': {'Latitude': 10.7867, 'Longitude': 76.6548},
    'Kolhapur': {'Latitude': 16.7050, 'Longitude': 74.2433},
    'Kozhikode': {'Latitude': 11.2588, 'Longitude': 75.7804},
    'Kasaragod': {'Latitude': 12.4984, 'Longitude': 75.0453},
    'Thrissur': {'Latitude': 10.5276, 'Longitude': 76.2144},
    'Wayanad': {'Latitude': 11.6854, 'Longitude': 76.1320},
    'Mahesana': {'Latitude': 23.5979, 'Longitude': 72.3693},
    'Nanded': {'Latitude': 19.1383, 'Longitude': 77.3210},
    'Gadchiroli': {'Latitude': 20.1885, 'Longitude': 80.0030},
    'Howrah': {'Latitude': 22.5958, 'Longitude': 88.2636},
    'North 24 Parganas': {'Latitude': 22.6338, 'Longitude': 88.4220}
}

# Step 4: Filtering the data for selected districts
```

```
selected_districts = list(district_coords.keys())
filtered_df = df[df['District'].isin(selected_districts)].copy()

# Adding latitude and longitude columns based on district names
filtered_df['Latitude'] = filtered_df['District'].map(lambda x: district_coords[x]['Latitude'])
filtered_df['Longitude'] = filtered_df['District'].map(lambda x: district_coords[x]['Longitude'])

# Adding a 'Year' column for grouping
filtered_df['Year'] = filtered_df['Date of outbreak'].dt.year

# Step 5: Grouping by year and district, summing up the number of cases
grouped = filtered_df.groupby(['Year', 'District', 'Latitude', 'Longitude'])['No of cases'].sum().reset_index()

# Step 6: Creating an output directory to save the plots (if needed)
output_dir = "india_district_maps_by_year"
os.makedirs(output_dir, exist_ok=True)

# Step 7: Looping through each year and plotting the maps
years = sorted(grouped['Year'].unique())

for year in years:
    # Skipping map generation for the year 2031
    if year == 2031:
        print("Skipping map generation for the year 2031 as it is data computation error.")
        continue

    data_year = grouped[grouped['Year'] == year]

    # Creating a geo scatter plot using Plotly
    fig = px.scatter_geo(
        data_year,
        lat='Latitude',
        lon='Longitude',
        color='No of cases',
        size='No of cases',
        hover_name='District',
        projection='natural earth',
        title=f'Disease Outbreaks in Selected Districts - {year}'
    )

    # Updating map settings to focus on India
    fig.update_geos(
        scope="asia",
        lataxis_range=[6, 38],
        lonaxis_range=[68, 98],
        showland=True,
        landcolor="rgb(217, 217, 217)",
        showcountries=True,
        countrycolor="Black"
    )

    # Adjusting layout margins
    fig.update_layout(margin={"r": 0, "t": 50, "l": 0, "b": 0})

    # Showing the plot
    print(f"Showing map for {year}. ")
    fig.show()
```

```
Showing map for 2022.
```

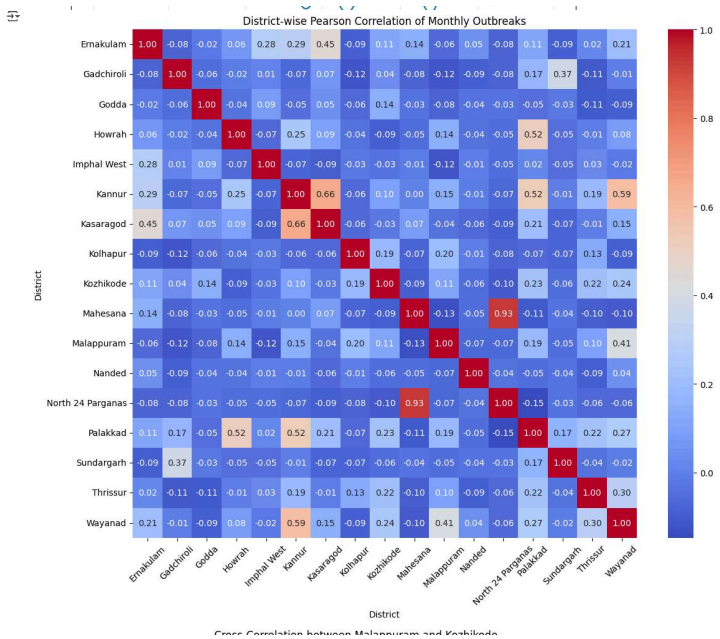



```
Showing map for 2023.
```




```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Loading the dataset
df = pd.read_excel("/content/Copy of 22-25_Combined_New(1).xlsx")

# Cleaning column names and converting date column to datetime format
df.columns = df.columns.str.strip()
df['Date of outbreak'] = pd.to_datetime(df['Date of outbreak'], errors='coerce')

# Removing rows with missing important information
df = df.dropna(subset=['Date of outbreak', 'District', 'No of cases'])

# Filtering only the selected districts
districts_of_interest = [
    'Kannur', 'Godda', 'Ernakulam', 'Sundargarh', 'Imphal West',
    'Malappuram', 'Palakkad', 'Kolhapur', 'Kozhikode', 'Kasaragod',
    'Thrissur', 'Wayanad', 'Mahesana', 'Nanded', 'Gadchiroli',
    'Howrah', 'North 24 Parganas'
]
df = df[df['District'].isin(districts_of_interest)]

# Creating a 'Month' column for grouping data by month
df['Month'] = df['Date of outbreak'].dt.to_period('M').dt.to_timestamp()

# Grouping and summing cases by month and district
monthly_cases = df.groupby(['Month', 'District'])['No of cases'].sum().unstack(fill_value=0)

# Plotting the Pearson correlation matrix between districts
plt.figure(figsize=(12, 10))
sns.heatmap(monthly_cases.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("District-wise Pearson Correlation of Monthly Outbreaks")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Defining a function to calculate cross-correlation for different lags
def cross_correlation(a, b, max_lag=12):
    """
    Calculating correlation between two time series with varying lags.
    Returns a list of lags and corresponding correlation values.
    """
    lags = np.arange(-max_lag, max_lag + 1)
    corrs = [a.shift(lag).corr(b) for lag in lags]
    return lags, corrs

# Selecting two example districts to compare
district_a = 'Malappuram'
district_b = 'Kozhikode'

a = monthly_cases[district_a]
b = monthly_cases[district_b]

# Calculating lag-based correlations
lags, corrs = cross_correlation(a, b, max_lag=12)

# Plotting the cross-correlation between the two districts
plt.figure(figsize=(10, 6))
sns.lineplot(x=lags, y=corrs, marker="o")
plt.title(f"Cross-Correlation between {district_a} and {district_b}")
plt.xlabel("Lag (months)")
plt.ylabel("Correlation")
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

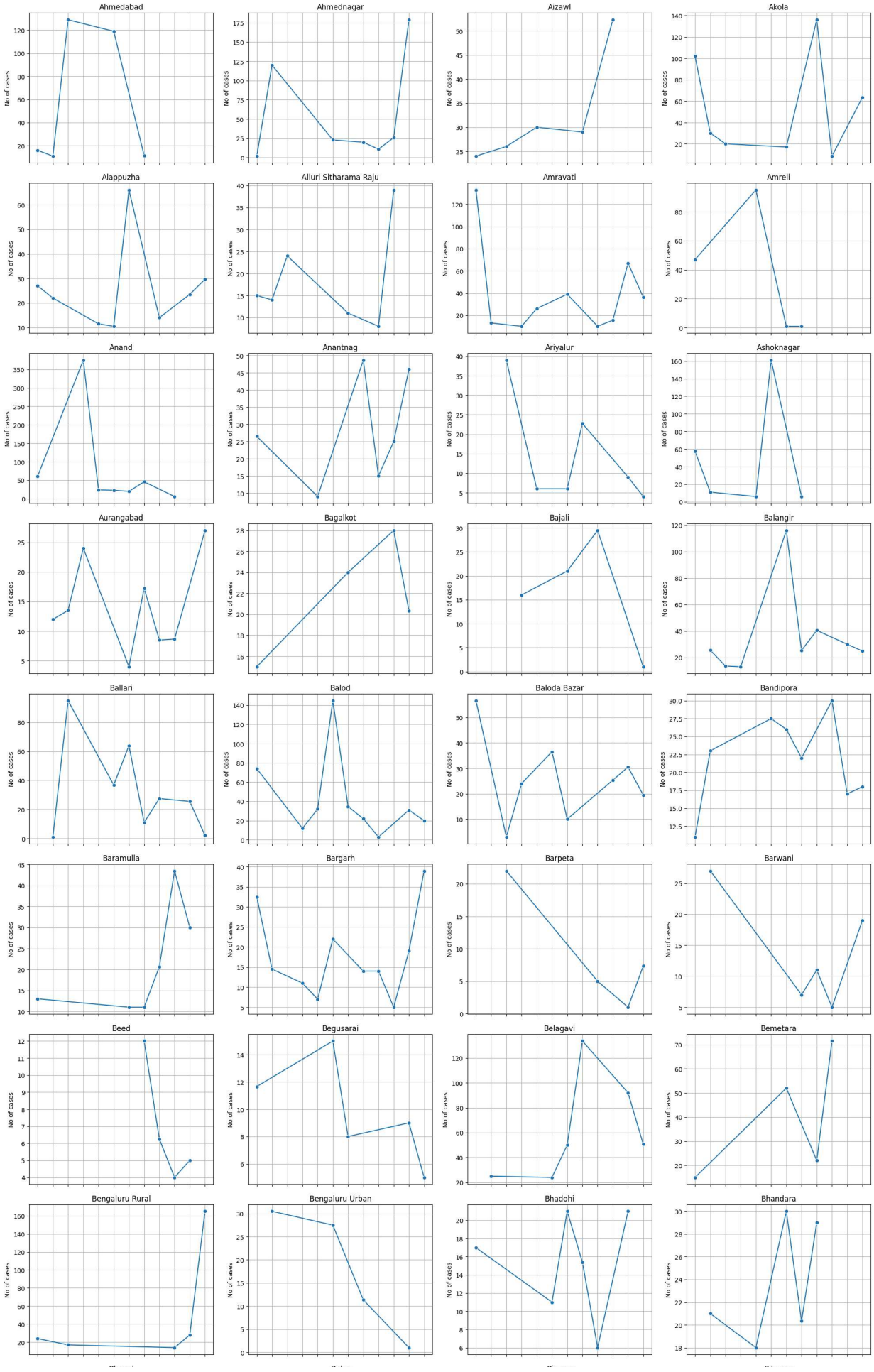


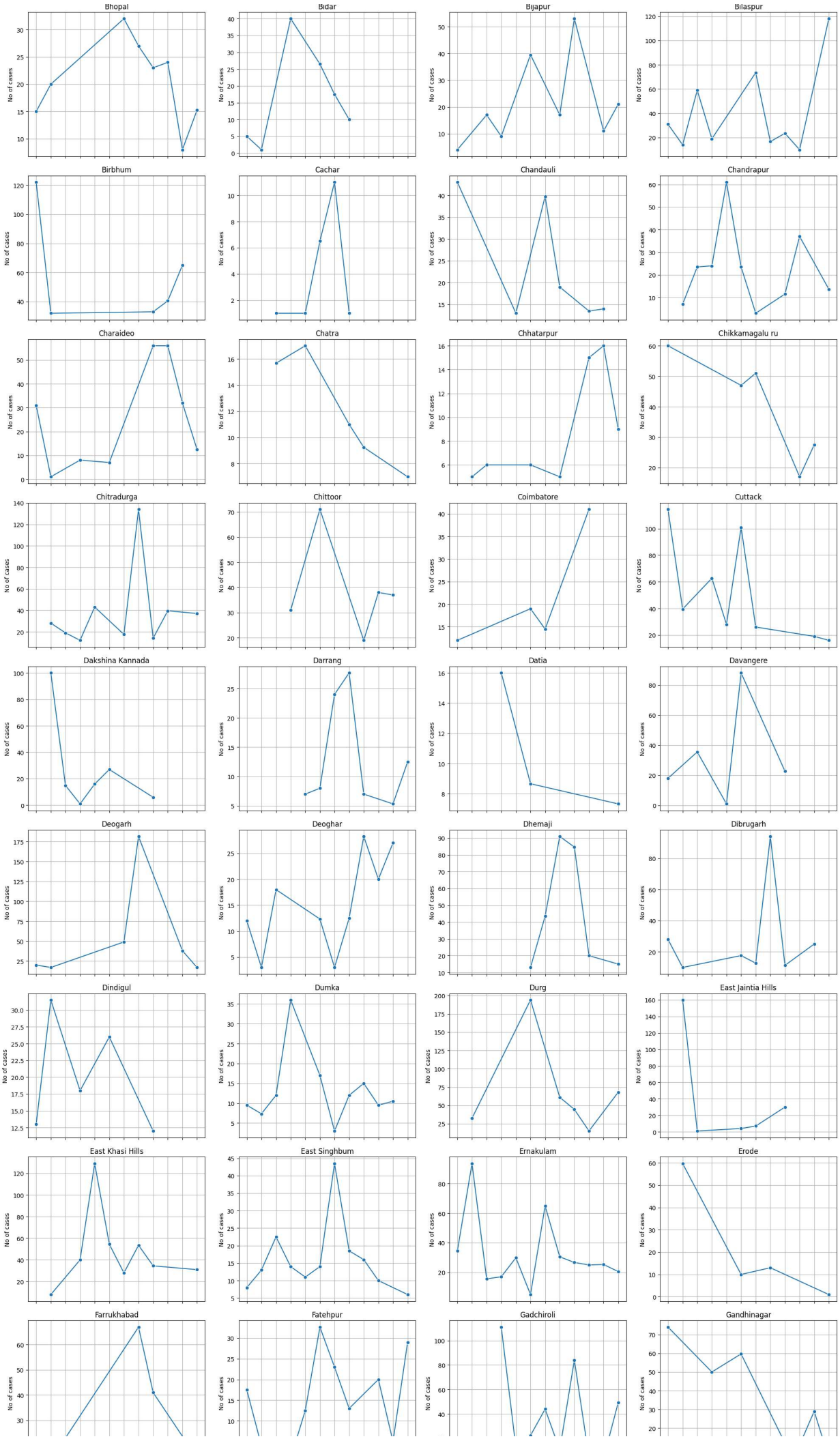
Cross Correlation between Malappuram and Kozhikode

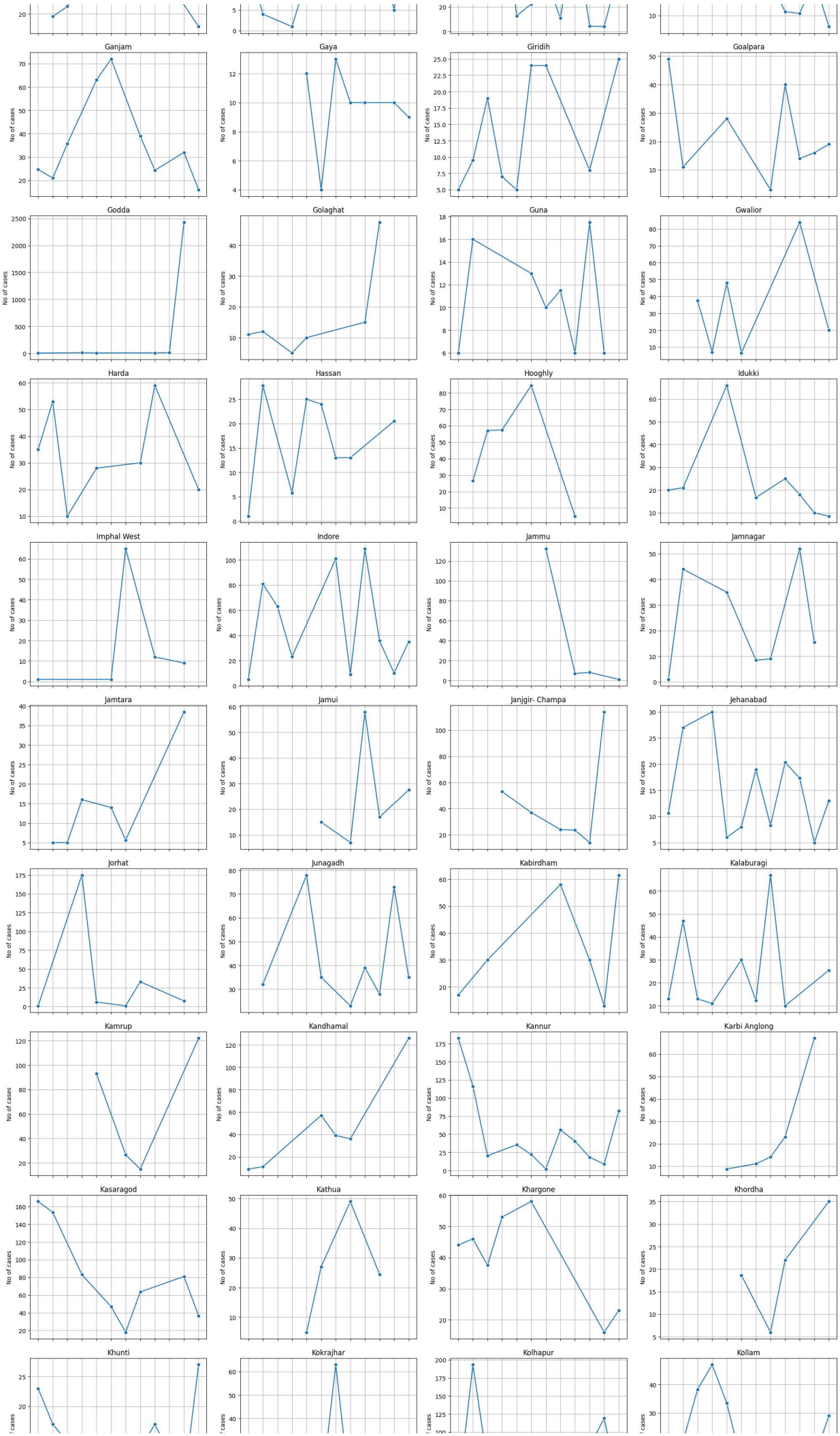
```
#omit for interpretation
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math

# Loading and preparing the dataset
df = pd.read_excel("/content/Copy of 22-25_Combined_New(1).xlsx")
df.columns = df.columns.str.strip()
df['Date of outbreak'] = pd.to_datetime(df['Date of outbreak'], errors='coerce')
df = df.dropna(subset=['Date of outbreak', 'District', 'No of cases'])

# Filtering districts with at least 6 months of data
month_counts = df.groupby('District')['Date of outbreak'].nunique()
valid_districts = month_counts[month_counts >= 6].index.tolist()
df = df[df['District'].isin(valid_districts)]

# Extracting the month from the outbreak date
df['Month'] = df['Date of outbreak'].dt.month

# Calculating average number of cases per month for each district
monthly_avg = df.groupby(['District', 'Month'])['No of cases'].mean().reset_index()

# Getting the list of districts to plot
districts = sorted(monthly_avg['District'].unique())
n_districts = len(districts)

# Defining the layout for subplots
n_cols = 4
n_rows = math.ceil(n_districts / n_cols)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(5*n_cols, 4*n_rows), sharex=True)

# Plotting each district in a separate subplot
axes = axes.flatten()
for i, district in enumerate(districts):
    subset = monthly_avg[monthly_avg['District'] == district]
    sns.lineplot(data=subset, x='Month', y='No of cases', marker='o', ax=axes[i])
    axes[i].set_title(district)
    axes[i].set_xticks(range(1, 13))
    axes[i].set_xticklabels(["Jan", "Feb", "Mar", "Apr", "May", "Jun",
                             "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"], rotation=45)
    axes[i].grid(True)

# Hiding any empty subplot spaces
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

# Setting the overall title and layout
fig.suptitle("Monthly Average Disease Outbreaks per District", fontsize=20)
fig.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```

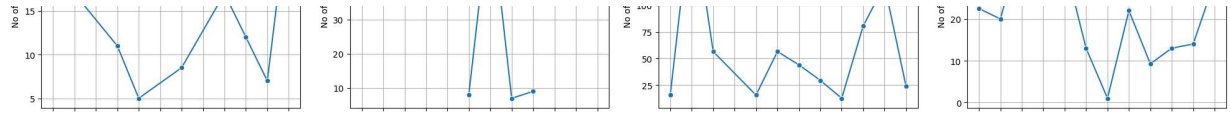
[+]

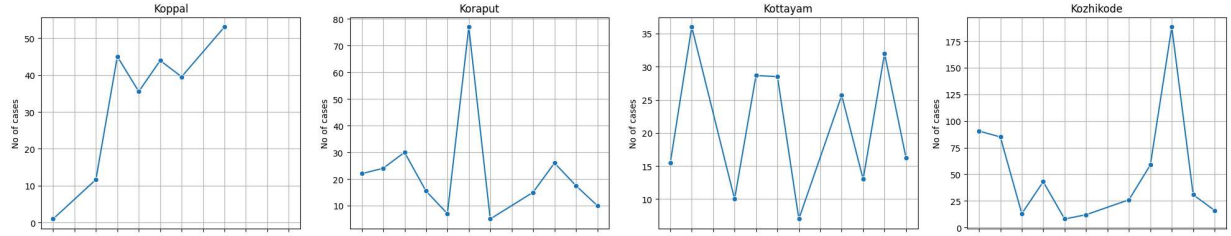
Monthly Average Disease Outbreaks per District

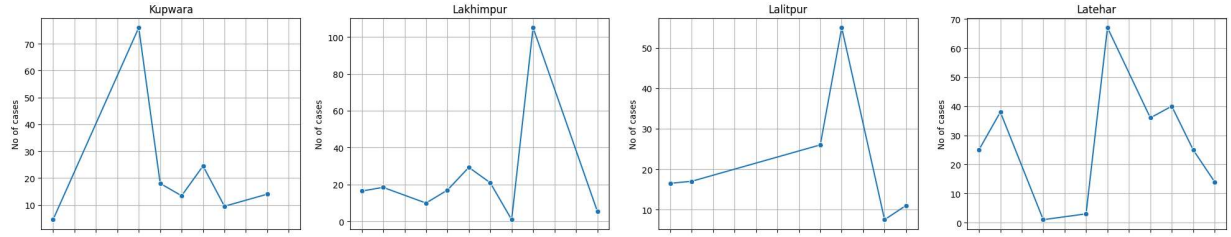


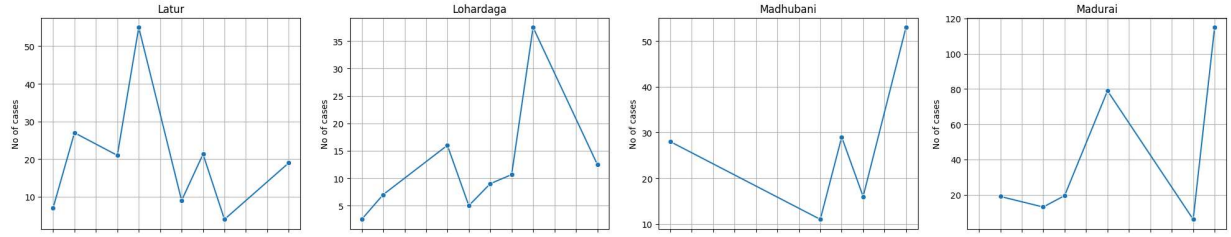

Bhopal
Bidar
Bijapur
Bilaspur
Birbhum
Cachar
Chandauli
Chandrapur
Charaideo
Chatra
Chhatarpur
Chikkamagalu ru
Chitradurga
Chittoor
Coimbatore
Cuttack
Dakshina Kannada
Darrang
Datia
Davangere
Deogarh
Deoghar
Dhemaji
Dibrugarh
Dindigul
Dumka
Durg
East Jaintia Hills
East Khasi Hills
East Singhbum
Ernakulam
Erode
Farrukhabad
Fatehpur
Gadchiroli
Gandhinagar
No of cases

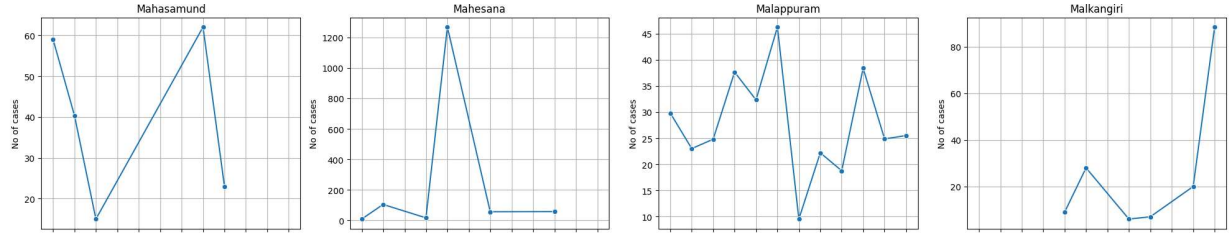

Ganjam
Gaya
Giridih
Goalpara
Godda
Golaghat
Guna
Gwalior
Harda
Hassan
Hooghly
Idukki
Imphal West
Indore
Jammu
Jamnagar
Jamtara
Jamui
Janjgir- Champa
Jehanabad
Jorhat
Junagadh
Kabirdham
Kalaburagi
Kamrup
Kandhamal
Kannur
Karbi Anglong
Kasaragod
Kathua
Khargone
Khordha
Khunti
Kokrajhar
Kolhapur
Kollam
No of cases

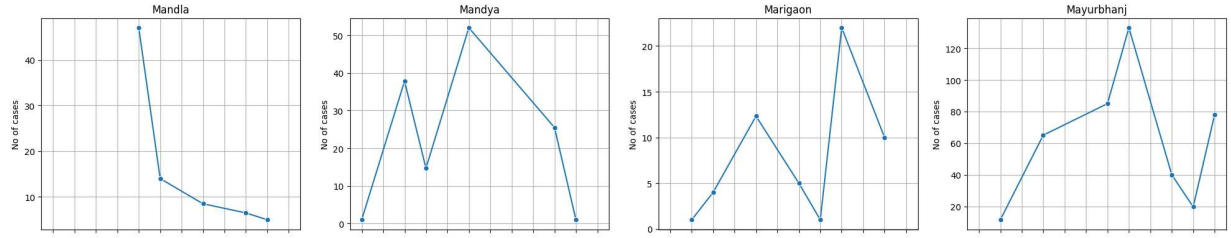


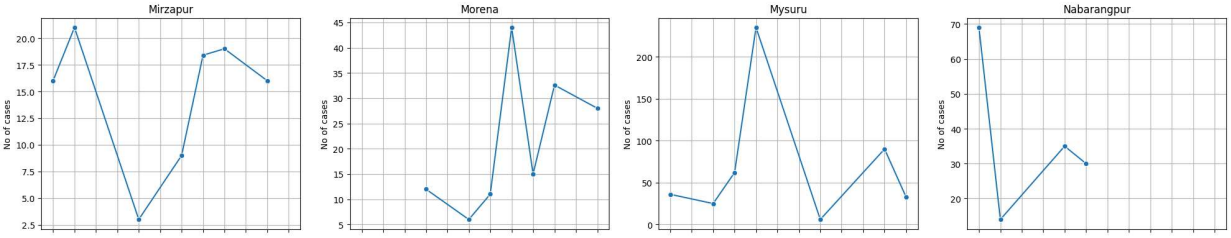

Koppal
Koraput
Kottayam
Kozhikode

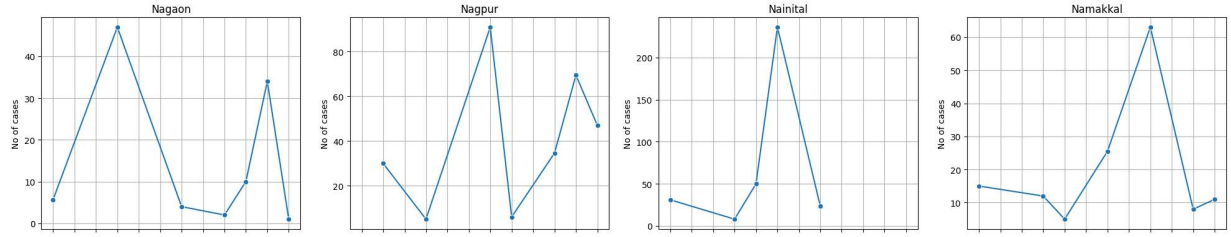

Kupwara
Lakhimpur
Lalitpur
Latehar

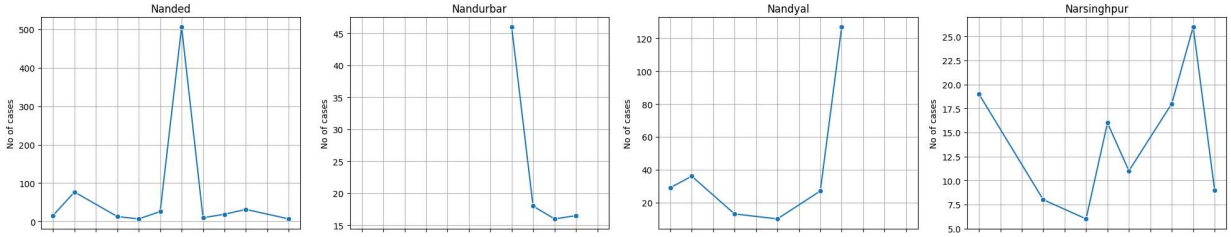

Latur
Lohardaga
Madhubani
Madurai

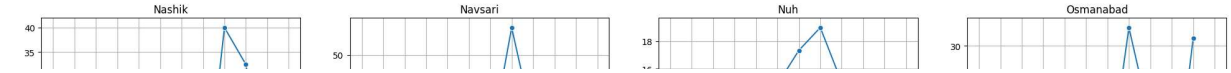

Mahasamund
Mahesana
Malappuram
Malkangiri

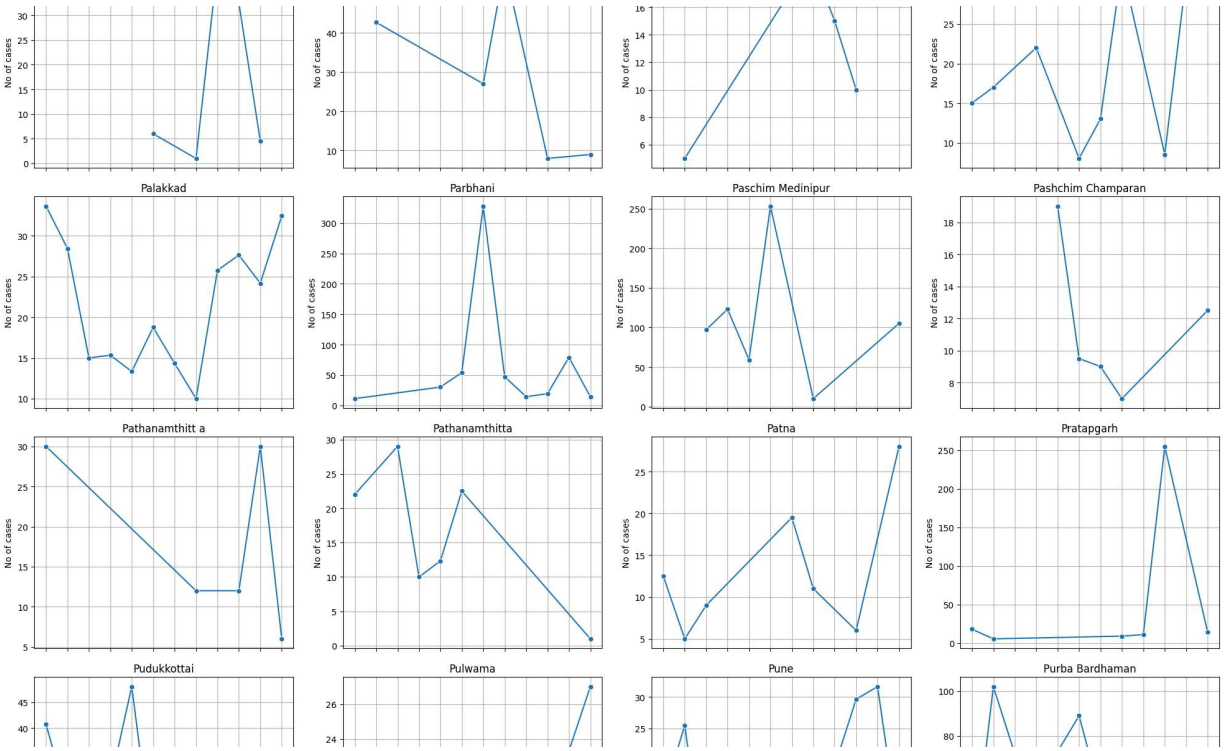

Mandla
Mandya
Marigaon
Mayurbhanj


Mirzapur
Morena
Mysuru
Nabarangpur


Nagaon
Nagpur
Nainital
Namakkal


Nanded
Nandurbar
Nandyal
Narsinghpur


Nashik
Navsari
Nuh
Osmanabad




```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```