```
import pandas as pd

# Load the dataset
data = pd.read_csv('/content/BD-RTFX-mkt-2007-2025 - Sheet1.csv')


# Step 1: Inspect the dataset
print("\nInitial Dataset Overview:\n")
print(data.info())
```

```
Initial Dataset Overview:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16709 entries, 0 to 16708
Data columns (total 13 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   price_date                          16709 non-null  object 
 1   adm1_name                           16709 non-null  int64  
 2   adm2_name                           16709 non-null  int64  
 3   mkt_name                            16709 non-null  int64  
 4   lat                                 16709 non-null  float64
 5   lon                                 16709 non-null  float64
 6   year                                16709 non-null  int64  
 7   month                               16709 non-null  int64  
 8   o_exchange_rate_unofficial          16709 non-null  float64
 9   h_exchange_rate_unofficial          16709 non-null  float64
 10  l_exchange_rate_unofficial          16709 non-null  float64
 11  c_exchange_rate_unofficial          16709 non-null  float64
 12  inflation_exchange_rate_unofficial  16709 non-null  float64
dtypes: float64(7), int64(5), object(1)
memory usage: 1.7+ MB
None
```

```
# Step 2: Identify columns with missing values
missing_values = data.isnull().sum()
missing_percent = (missing_values / len(data)) * 100
missing_report = pd.DataFrame({
    'Column': data.columns,
    'Missing Values': missing_values,
    'Missing Percentage': missing_percent
}).sort_values(by='Missing Percentage', ascending=False)
print("\nMissing Values Report:\n")
print(missing_report)

# Step 3: Check unique values in each column
unique_values = data.nunique()
unique_report = pd.DataFrame({
    'Column': data.columns,
    'Unique Values': unique_values
}).sort_values(by='Unique Values', ascending=True)
print("\nUnique Values Report:\n")
print(unique_report)
```

```
adm2_name                                                       adm2_name
mkt_name                                                         mkt_name
lat                                                                   lat
lon                                                                   lon
year                                                                 year
month                                                               month
o_exchange_rate_unofficial                     o_exchange_rate_unofficial
h_exchange_rate_unofficial                     h_exchange_rate_unofficial
l_exchange_rate_unofficial                     l_exchange_rate_unofficial
c_exchange_rate_unofficial                     c_exchange_rate_unofficial
inflation_exchange_rate_unofficial  inflation_exchange_rate_unofficial

                                    Missing Values  Missing Percentage
price_date                                       0                 0.0
adm1_name                                        0                 0.0
adm2_name                                        0                 0.0
mkt_name                                         0                 0.0
lat                                              0                 0.0
```

```
c_exchange_rate_unofficial                       0                 0.0
inflation_exchange_rate_unofficial               0                 0.0

Unique Values Report:

                                                                Column  \
adm1_name                                                    adm1_name   
month                                                            month   
year                                                              year   
adm2_name                                                    adm2_name   
mkt_name                                                      mkt_name   
lat                                                                lat   
lon                                                                lon   
c_exchange_rate_unofficial                  c_exchange_rate_unofficial   
inflation_exchange_rate_unofficial  inflation_exchange_rate_unofficial   
o_exchange_rate_unofficial                  o_exchange_rate_unofficial   
l_exchange_rate_unofficial                  l_exchange_rate_unofficial   
h_exchange_rate_unofficial                  h_exchange_rate_unofficial   
price_date                                                  price_date   

                                    Unique Values
adm1_name                                       9
month                                          12
year                                           19
adm2_name                                      64
mkt_name                                       77
lat                                            77
lon                                            77
c_exchange_rate_unofficial                    159
inflation_exchange_rate_unofficial            171
o_exchange_rate_unofficial                    178
l_exchange_rate_unofficial                    178
h_exchange_rate_unofficial                    184
price_date                                    217
```

```
# Analyze unique values for each column
unique_values = data.nunique()

# Print unique values for review
print("\nUnique Values in Each Column:\n")
print(unique_values)

# Decide on redundant columns (e.g., geo_id might overlap with adm1_name and adm2_name)
# For this example, we will drop geo_id if adm1_name and adm2_name provide the same context
if 'geo_id' in data.columns:
    data_cleaned = data.drop(columns=['geo_id'], errors='ignore')
    print("\nDropped 'geo_id' column due to redundancy.")
```

```
Unique Values in Each Column:

price_date                            217
adm1_name                               9
adm2_name                              64
mkt_name                               77
lat                                    77
lon                                    77
year                                   19
month                                  12
o_exchange_rate_unofficial            178
h_exchange_rate_unofficial            184
l_exchange_rate_unofficial            178
c_exchange_rate_unofficial            159
inflation_exchange_rate_unofficial    171
dtype: int64
```

```
unique_values
```

	0
price_date	217
adm1_name	9
adm2_name	64
mkt_name	77
lat	77
lon	77
year	19
month	12
o_exchange_rate_unofficial	178
h_exchange_rate_unofficial	184
l_exchange_rate_unofficial	178
c_exchange_rate_unofficial	159
inflation_exchange_rate_unofficial	171

dtype: int64

```
adm2_name = data['adm2_name'].unique()
```

```
len(adm2_name)
```

```
64
```

```
# Inspect the dataset
data_cleaned = data
print(data_cleaned.info())
print(data_cleaned.nunique())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16709 entries, 0 to 16708
Data columns (total 13 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   price_date                          16709 non-null  object 
 1   adm1_name                           16709 non-null  int64  
 2   adm2_name                           16709 non-null  int64  
 3   mkt_name                            16709 non-null  int64  
 4   lat                                 16709 non-null  float64
 5   lon                                 16709 non-null  float64
 6   year                                16709 non-null  int64  
 7   month                               16709 non-null  int64  
 8   o_exchange_rate_unofficial          16709 non-null  float64
 9   h_exchange_rate_unofficial          16709 non-null  float64
 10  l_exchange_rate_unofficial          16709 non-null  float64
 11  c_exchange_rate_unofficial          16709 non-null  float64
 12  inflation_exchange_rate_unofficial  16709 non-null  float64
dtypes: float64(7), int64(5), object(1)
memory usage: 1.7+ MB
None
price_date                            217
adm1_name                               9
adm2_name                              64
mkt_name                               77
lat                                    77
lon                                    77
year                                   19
month                                  12
o_exchange_rate_unofficial            178
h_exchange_rate_unofficial            184
l_exchange_rate_unofficial            178
c_exchange_rate_unofficial            159
inflation_exchange_rate_unofficial    171
dtype: int64
```

```
# Check missing values
print(data_cleaned.isnull().sum())
```

```
price_date                         0
adm1_name                          0
adm2_name                          0
mkt_name                           0
lat                                0
lon                                0
year                               0
month                              0
o_exchange_rate_unofficial         0
h_exchange_rate_unofficial         0
l_exchange_rate_unofficial         0
c_exchange_rate_unofficial         0
inflation_exchange_rate_unofficial    0
dtype: int64
```

```
data_cleaned.head(4)
```

	price_date	adm1_name	adm2_name	mkt_name	lat	lon	year	month	o_exchange_rate_unofficial	h_exchange_rate_unofficial
0	2007-01-01	3	0	0	22.665347	89.792432	2007	1	69.91	69.78
1	2007-02-01	3	0	0	22.665347	89.792432	2007	2	69.47	69.59
2	2007-03-01	3	0	0	22.665347	89.792432	2007	3	68.44	68.95
3	2007-04-01	3	0	0	22.665347	89.792432	2007	4	68.87	69.14

Next steps: Generate code with data_cleaned View recommended plots New interactive sheet

```
data_cleaned['price_date'] = pd.to_datetime(data_cleaned['price_date'], errors='coerce')
```

```
data_cleaned.head(4)
```

	price_date	adm1_name	adm2_name	mkt_name	lat	lon	year	month	o_exchange_rate_unofficial	h_exchange_rate_unofficial
0	2007-01-01	3	0	0	22.665347	89.792432	2007	1	69.91	69.78
1	2007-02-01	3	0	0	22.665347	89.792432	2007	2	69.47	69.59
2	2007-03-01	3	0	0	22.665347	89.792432	2007	3	68.44	68.95
3	2007-04-01	3	0	0	22.665347	89.792432	2007	4	68.87	69.14

Next steps: Generate code with data_cleaned View recommended plots New interactive sheet

```
# # Check for duplicates
# duplicates = data_cleaned.duplicated(subset=['price_date', 'adm1_name'])
# print(f"Number of duplicate rows: {duplicates.sum()}")
# ----------Number of duplicate rows: 14756
```

```
# # Check for duplicates
# duplicates = data_cleaned.duplicated(subset=['price_date', 'adm2_name'])
# print(f"Number of duplicate rows: {duplicates.sum()}")
# /Number of duplicate rows: 2821
# # Remove duplicates if any
# # data_cleaned = data_cleaned.drop_duplicates(subset=['price_date', 'adm2_name'])
```

```
Number of duplicate rows: 2821
```

```
# Check for duplicates using the full composite key
duplicates = data_cleaned.duplicated(subset=['price_date', 'adm1_name', 'adm2_name'])
print(f"Number of duplicate rows: {duplicates.sum()}")

# # Remove duplicates if any
# data_cleaned = data_cleaned.drop_duplicates(subset=['price_date', 'adm1_name', 'adm2_name'])
# print(f"Data shape after removing duplicates: {data_cleaned.shape}")
```

```
Number of duplicate rows: 2821
```

```
# Identify duplicate rows based on the composite key
duplicates = data_cleaned[data_cleaned.duplicated(subset=['price_date', 'adm1_name', 'adm2_name'], keep=False)]

# Display duplicate rows
print(f"Number of duplicate rows: {len(duplicates)}")
print(duplicates)
```

```
Number of duplicate rows: 4774
       price_date  adm1_name  adm2_name  mkt_name        lat        lon  year  \
651    2007-01-01          0          3         3  22.701944  90.371111  2007
652    2007-02-01          0          3         3  22.701944  90.371111  2007
653    2007-03-01          0          3         3  22.701944  90.371111  2007
654    2007-04-01          0          3         3  22.701944  90.371111  2007
655    2007-05-01          0          3         3  22.701944  90.371111  2007
...           ...        ...        ...       ...        ...        ...   ...
16487  2024-09-01          1         11        76  21.242928  92.140437  2024
16488  2024-10-01          1         11        76  21.242928  92.140437  2024
16489  2024-11-01          1         11        76  21.242928  92.140437  2024
16490  2024-12-01          1         11        76  21.242928  92.140437  2024
16491  2025-01-01          1         11        76  21.242928  92.140437  2025

       month  o_exchange_rate_unofficial  h_exchange_rate_unofficial  \
651        1                       69.91                       69.78
652        2                       69.47                       69.59
653        3                       68.44                       68.95
654        4                       68.87                       69.14
655        5                       68.78                       69.09
...      ...                         ...                         ...
16487      9                      119.66                      120.21
16488     10                      119.07                      120.00
16489     11                      121.06                      121.61
16490     12                      118.93                      119.77
16491      1                      120.06                      121.37

       l_exchange_rate_unofficial  c_exchange_rate_unofficial  \
651                         69.53                       69.72
652                         69.01                       69.01
653                         68.23                       68.95
654                         68.60                       68.94
655                         68.58                       69.09
...                           ...                         ...
16487                      118.87                      118.87
16488                      118.55                      120.00
16489                      119.32                      119.32
16490                      118.09                      119.52
16491                      119.40                      121.37

       inflation_exchange_rate_unofficial
651                                 10.33
652                                 10.33
653                                 10.33
654                                 10.33
655                                 10.33
...                                   ...
16487                                8.10
16488                                8.60
16489                                7.61
16490                                8.50
16491                               10.33

[4774 rows x 13 columns]
```

```
# Sort data by composite key to ensure chronological order
data_cleaned = data_cleaned.sort_values(by=['adm1_name', 'adm2_name', 'price_date'])
```

```
data_cleaned.tail(2)
```

	price_date	adm1_name	adm2_name	mkt_name	lat	lon	year	month	o_exchange_rate_unofficial	h_exchange_rate_unoffic
15406	2025-01-01	8	61	71	24.896667	91.871667	2025	1	120.06	121
15623	2025-01-01	8	61	72	24.890531	91.871936	2025	1	120.06	121

```
# Check for duplicates based on composite key
duplicates = data_cleaned.duplicated(subset=['price_date', 'adm1_name', 'adm2_name'], keep='first')

print(f"Number of rows with duplicate composite keys: {duplicates.sum()}")
# Number of rows with duplicate composite keys: 2821
# # Display the duplicate rows
# duplicate_rows = data_cleaned[duplicates]
# print(duplicate_rows)
```

```
Number of rows with duplicate composite keys: 2821
```

```
data_cleaned = data_cleaned.drop_duplicates(subset=['price_date', 'adm1_name', 'adm2_name'], keep='first')
print(f"Data shape after removing duplicates: {data_cleaned.shape}")
# Data shape after removing duplicates: (13888, 13)
```

```
Data shape after removing duplicates: (13888, 13)
```

```
duplicates_check = data_cleaned.duplicated(subset=['price_date', 'adm1_name', 'adm2_name']).sum()
print(f"Remaining duplicates: {duplicates_check}")  # Should print 0
```

```
Remaining duplicates: 0
```

Division level forcasting

```
# Aggregate data by division and date
division_data = data_cleaned.groupby(['adm1_name', 'price_date']).mean().reset_index()

# Check the structure of the division-level dataset
print(f"Division-Level Data Shape: {division_data.shape}")
print(division_data.head())
```

```
Division-Level Data Shape: (1953, 13)
   adm1_name price_date  adm2_name  mkt_name       lat        lon    year  \
0          0 2007-01-01       16.4      19.6  22.56618  90.261829  2007.0   
1          0 2007-02-01       16.4      19.6  22.56618  90.261829  2007.0   
2          0 2007-03-01       16.4      19.6  22.56618  90.261829  2007.0   
3          0 2007-04-01       16.4      19.6  22.56618  90.261829  2007.0   
4          0 2007-05-01       16.4      19.6  22.56618  90.261829  2007.0   

   month  o_exchange_rate_unofficial  h_exchange_rate_unofficial  \
0    1.0                       69.91                       69.78   
1    2.0                       69.47                       69.59   
2    3.0                       68.44                       68.95   
3    4.0                       68.87                       69.14   
4    5.0                       68.78                       69.09   

   l_exchange_rate_unofficial  c_exchange_rate_unofficial  \
0                       69.53                       69.72   
1                       69.01                       69.01   
2                       68.23                       68.95   
3                       68.60                       68.94   
4                       68.58                       69.09   

   inflation_exchange_rate_unofficial  
0                               10.33  
1                               10.33  
2                               10.33  
3                               10.33  
4                               10.33  
```

District

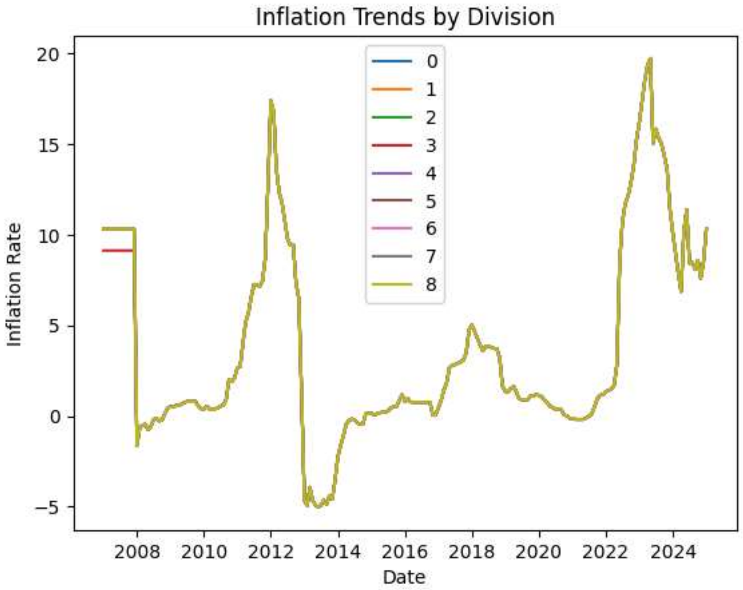
```
# Sort the district-level data by district and date
district_data = data_cleaned.sort_values(by=['adm2_name', 'price_date'])

# Check the structure of the district-level dataset
print(f"District-Level Data Shape: {district_data.shape}")
print(district_data.head())
```

```
District-Level Data Shape: (13888, 13)
  price_date  adm1_name  adm2_name  mkt_name        lat        lon  year  \
0 2007-01-01          3          0         0  22.665347  89.792432  2007   
1 2007-02-01          3          0         0  22.665347  89.792432  2007   
2 2007-03-01          3          0         0  22.665347  89.792432  2007   
3 2007-04-01          3          0         0  22.665347  89.792432  2007   
4 2007-05-01          3          0         0  22.665347  89.792432  2007   

   month  o_exchange_rate_unofficial  h_exchange_rate_unofficial  \
0      1                       69.91                       69.78   
1      2                       69.47                       69.59   
2      3                       68.44                       68.95   
3      4                       68.87                       69.14   
4      5                       68.78                       69.09   

   l_exchange_rate_unofficial  c_exchange_rate_unofficial  \
0                       69.53                       69.72   
1                       69.01                       69.01   
2                       68.23                       68.95   
3                       68.60                       68.94   
4                       68.58                       69.09   

   inflation_exchange_rate_unofficial  
0                               -1.63  
1                               -1.63  
2                               -1.63  
3                               -1.63  
4                               -1.63  
```

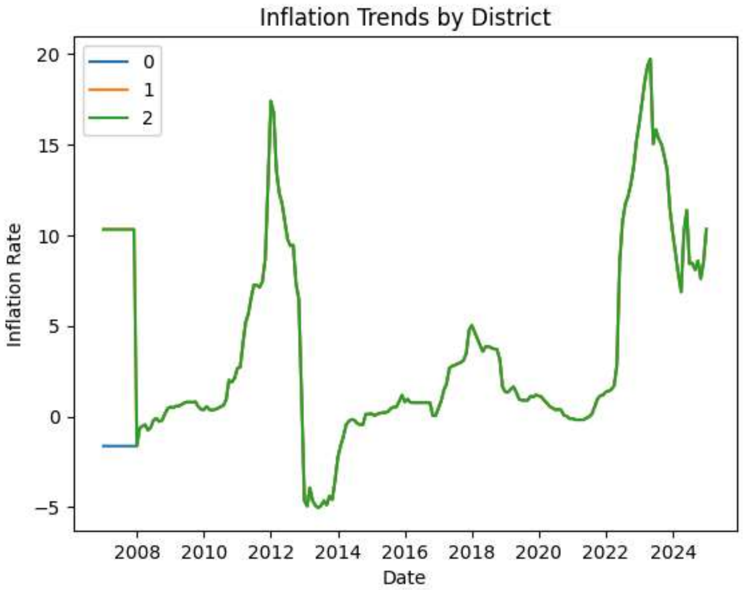
```
# Save the datasets to CSV files
division_data.to_csv("division_level_data.csv", index=False)
district_data.to_csv("district_level_data.csv", index=False)
```

Plot inflation trends by division

```
import matplotlib.pyplot as plt

# Plot inflation trends by division
for division in division_data['adm1_name'].unique():
    division_df = division_data[division_data['adm1_name'] == division]
    plt.plot(division_df['price_date'], division_df['inflation_exchange_rate_unofficial'], label=division)

plt.legend()
plt.title('Inflation Trends by Division')
plt.xlabel('Date')
plt.ylabel('Inflation Rate')
plt.show()
```



```
# Plot inflation trends for a few districts
for district in district_data['adm2_name'].unique()[:3]:  # Visualize 5 districts
    district_df = district_data[district_data['adm2_name'] == district]
    plt.plot(district_df['price_date'], district_df['inflation_exchange_rate_unofficial'], label=district)

plt.legend()
plt.title('Inflation Trends by District')
plt.xlabel('Date')
plt.ylabel('Inflation Rate')
plt.show()
```



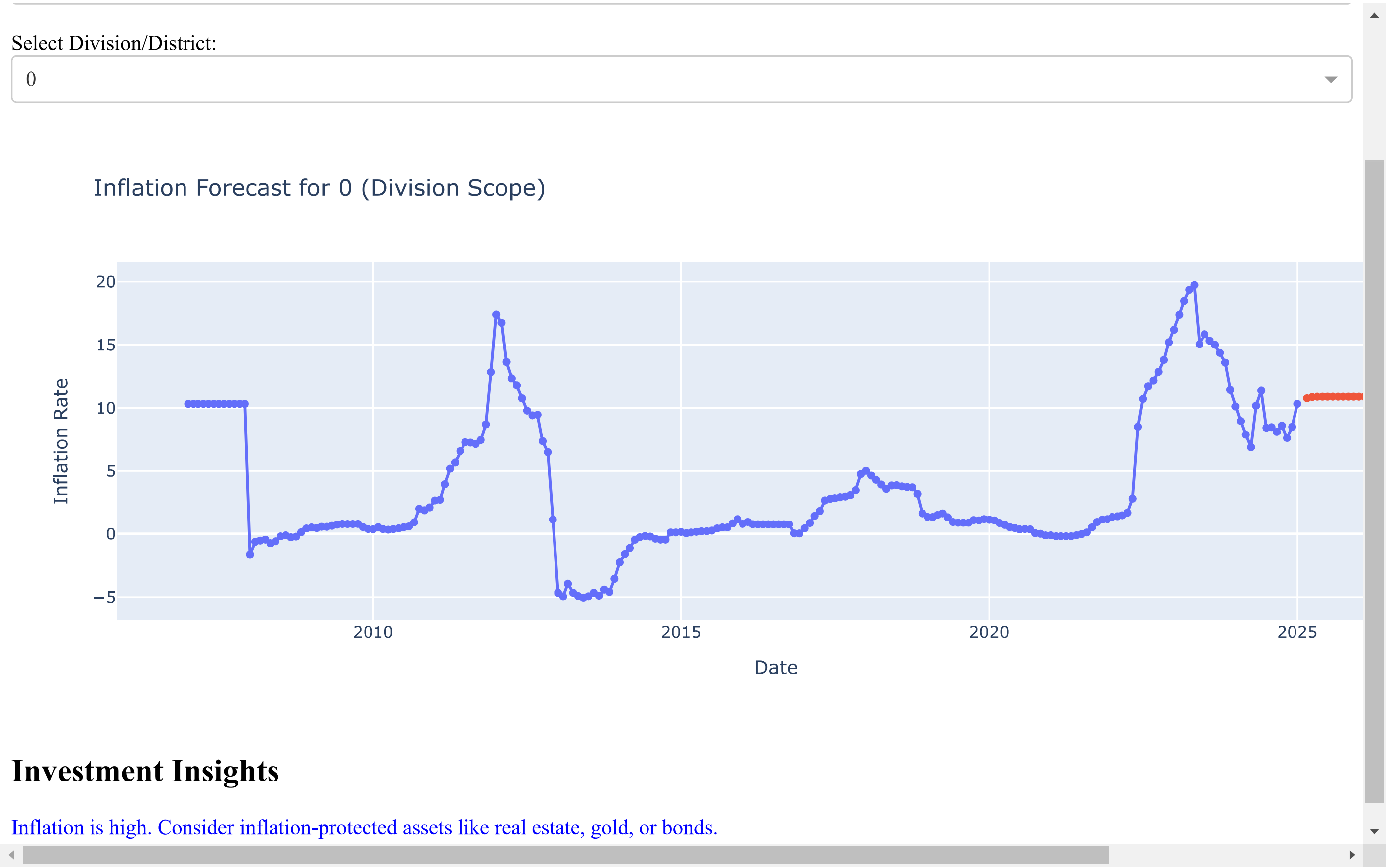
```
# !pip install jupyter-dash dash pandas plotly
```

```
# from jupyter_dash import JupyterDash
# from dash import dcc, html
# from dash.dependencies import Input, Output
# import pandas as pd
# import plotly.graph_objects as go
# from statsmodels.tsa.arima.model import ARIMA

# # Load prepared datasets
# division_data = pd.read_csv("division_level_data.csv")  # Aggregated by division
# district_data = pd.read_csv("district_level_data.csv")  # Raw district-level data
# # Convert price_date to datetime
# division_data['price_date'] = pd.to_datetime(division_data['price_date'])
# district_data['price_date'] = pd.to_datetime(district_data['price_date'])

# # Initialize the Dash app
# app = JupyterDash(__name__)

# # Layout
# app.layout = html.Div([
#     html.H1("Inflation Forecasting Dashboard"),

#     # Dropdown to select forecasting scope
#     html.Div([
#         html.Label("Select Forecasting Scope:"),
#         dcc.Dropdown(
#             id='forecast-scope',
#             options=[
#                 {'label': 'Division-Level Forecasting', 'value': 'division'},
#                 {'label': 'District-Level Forecasting', 'value': 'district'}
#             ],
#             value='division',  # Default selection
#             clearable=False
#         )
#     ], style={'marginBottom': '20px'}),

#     # Dropdown to select specific division or district
#     html.Div([
#         html.Label("Select Division/District:"),
#         dcc.Dropdown(id='region-selector', clearable=False)
#     ], style={'marginBottom': '20px'}),

#     # Graph for forecasting
#     dcc.Graph(id='forecast-graph')
# ])

# # Callbacks
# @app.callback(
#     Output('region-selector', 'options'),
#     Output('region-selector', 'value'),
#     Input('forecast-scope', 'value')
# )
# def update_region_selector(scope):
#     if scope == 'division':
#         regions = division_data['adm1_name'].unique()
#     else:
#         regions = district_data['adm2_name'].unique()

#     options = [{'label': region, 'value': region} for region in regions]
#     return options, regions[0]  # Default to the first region


# @app.callback(
#     Output('forecast-graph', 'figure'),
#     Input('forecast-scope', 'value'),
#     Input('region-selector', 'value')
# )
# def update_forecast(scope, region):
#     if scope == 'division':
#         data = division_data[division_data['adm1_name'] == region]
#     else:
#         data = district_data[district_data['adm2_name'] == region]

#     # Train ARIMA model
#     model = ARIMA(data['inflation_exchange_rate_unofficial'], order=(1, 1, 1))
#     model_fit = model.fit()

#     # Forecast future values
#     future_steps = 12
#     forecast = model_fit.forecast(steps=future_steps)

#     # Generate future dates
#     last_date = data['price_date'].iloc[-1]
#     future_dates = pd.date_range(start=last_date, periods=future_steps + 1, freq='M')[1:]

#     # Combine dates and forecast values
#     forecast_df = pd.DataFrame({'Date': future_dates, 'Forecast': forecast})
#     # Create figure
#     fig = go.Figure()
#     fig.add_trace(go.Scatter(
#         x=data['price_date'],
#         y=data['inflation_exchange_rate_unofficial'],
#         mode='lines+markers',
#         name='Historical Data'
#     ))
#     fig.add_trace(go.Scatter(
#         x=forecast_df['Date'],
#         y=forecast_df['Forecast'],
#         mode='lines+markers',
#         name='Forecast'
#     ))
#     fig.update_layout(
#         title=f"Inflation Forecast for {region} ({scope.capitalize()} Scope)",
#         xaxis_title="Date",
#         yaxis_title="Inflation Rate"
#     )
#     return fig

# # Run the app
# app.run_server(mode='inline', debug=True)
```

Start coding or generate with AI.

```
from jupyter_dash import JupyterDash
from dash import dcc, html
from dash.dependencies import Input, Output
import pandas as pd
import plotly.graph_objects as go
from statsmodels.tsa.arima.model import ARIMA

# Load prepared datasets
division_data = pd.read_csv("division_level_data.csv")  # Aggregated by division
district_data = pd.read_csv("district_level_data.csv")  # Raw district-level data

# Convert price_date to datetime
division_data['price_date'] = pd.to_datetime(division_data['price_date'])
district_data['price_date'] = pd.to_datetime(district_data['price_date'])

# Function to generate investment advice
def generate_investment_advice(inflation_rate):
    if inflation_rate > 10:
        return "Inflation is high. Consider inflation-protected assets like real estate, gold, or bonds."
    elif inflation_rate > 5:
        return "Inflation is moderate. Diversify with a mix of stocks, real estate, and commodities."
    else:
        return "Inflation is low. It's a good time to invest in growth-oriented sectors like technology or start-ups."

# Initialize the Dash app
app = JupyterDash(__name__)

# Layout
app.layout = html.Div([
    html.H1("Inflation Forecasting Dashboard"),

    # Dropdown to select forecasting scope
    html.Div([
        html.Label("Select Forecasting Scope:"),
        dcc.Dropdown(
            id='forecast-scope',
            options=[
                {'label': 'Division-Level Forecasting', 'value': 'division'},
                {'label': 'District-Level Forecasting', 'value': 'district'}
            ],
            value='division',  # Default selection
            clearable=False
        )
    ], style={'marginBottom': '20px'}),

    # Dropdown to select specific division or district
    html.Div([
        html.Label("Select Division/District:"),
        dcc.Dropdown(id='region-selector', clearable=False)
        ], style={'marginBottom': '20px'}),

        # Graph for forecasting
        dcc.Graph(id='forecast-graph'),

        # Investment advice section
        html.Div([
            html.H2("Investment Insights"),
            html.Div(id='investment-advice', style={'fontSize': '16px', 'color': 'blue'})
        ], style={'marginBottom': '20px'})
    ])

    # Callbacks
    @app.callback(
        Output('region-selector', 'options'),
        Output('region-selector', 'value'),
        Input('forecast-scope', 'value')
    )
    def update_region_selector(scope):
        if scope == 'division':
            regions = division_data['adm1_name'].unique()
        else:
            regions = district_data['adm2_name'].unique()

        options = [{'label': region, 'value': region} for region in regions]
        return options, regions[0]  # Default to the first region


    @app.callback(
        Output('forecast-graph', 'figure'),
        Output('investment-advice', 'children'),
        Input('forecast-scope', 'value'),
        Input('region-selector', 'value')
    )
    def update_forecast(scope, region):
        if scope == 'division':
            data = division_data[division_data['adm1_name'] == region]
        else:
            data = district_data[district_data['adm2_name'] == region]

        # Train ARIMA model
        model = ARIMA(data['inflation_exchange_rate_unofficial'], order=(1, 1, 1))
        model_fit = model.fit()

        # Forecast future values
        future_steps = 12
        forecast = model_fit.forecast(steps=future_steps)

        # Generate future dates
        last_date = data['price_date'].iloc[-1]
        future_dates = pd.date_range(start=last_date, periods=future_steps + 1, freq='M')[1:]

        # Combine dates and forecast values
        forecast_df = pd.DataFrame({'Date': future_dates, 'Forecast': forecast})

        # Create figure
        fig = go.Figure()
        fig.add_trace(go.Scatter(
            x=data['price_date'],
            y=data['inflation_exchange_rate_unofficial'],
            mode='lines+markers',
            name='Historical Data'
        ))
        fig.add_trace(go.Scatter(
            x=forecast_df['Date'],
            y=forecast_df['Forecast'],
            mode='lines+markers',
            name='Forecast'
        ))
        fig.update_layout(
            title=f"Inflation Forecast for {region} ({scope.capitalize()} Scope)",
            xaxis_title="Date",
            yaxis_title="Inflation Rate"
        )

        # Generate investment advice
        latest_rate = forecast.iloc[-1]  # Use the last forecasted value for advice
        advice = generate_investment_advice(latest_rate)
        advice = generate_investment_advice(latest_rate)

        return fig, advice

    # Run the app
    # app.run_server(mode='inline', debug=True) # if not work change the port number
    app.run_server(mode='inline', debug=True, port=8051)
```

```
/usr/local/lib/python3.11/dist-packages/dash/dash.py:579: UserWarning:

JupyterDash is deprecated, use Dash instead.
See https://dash.plotly.com/dash-in-jupyter for more details.
```



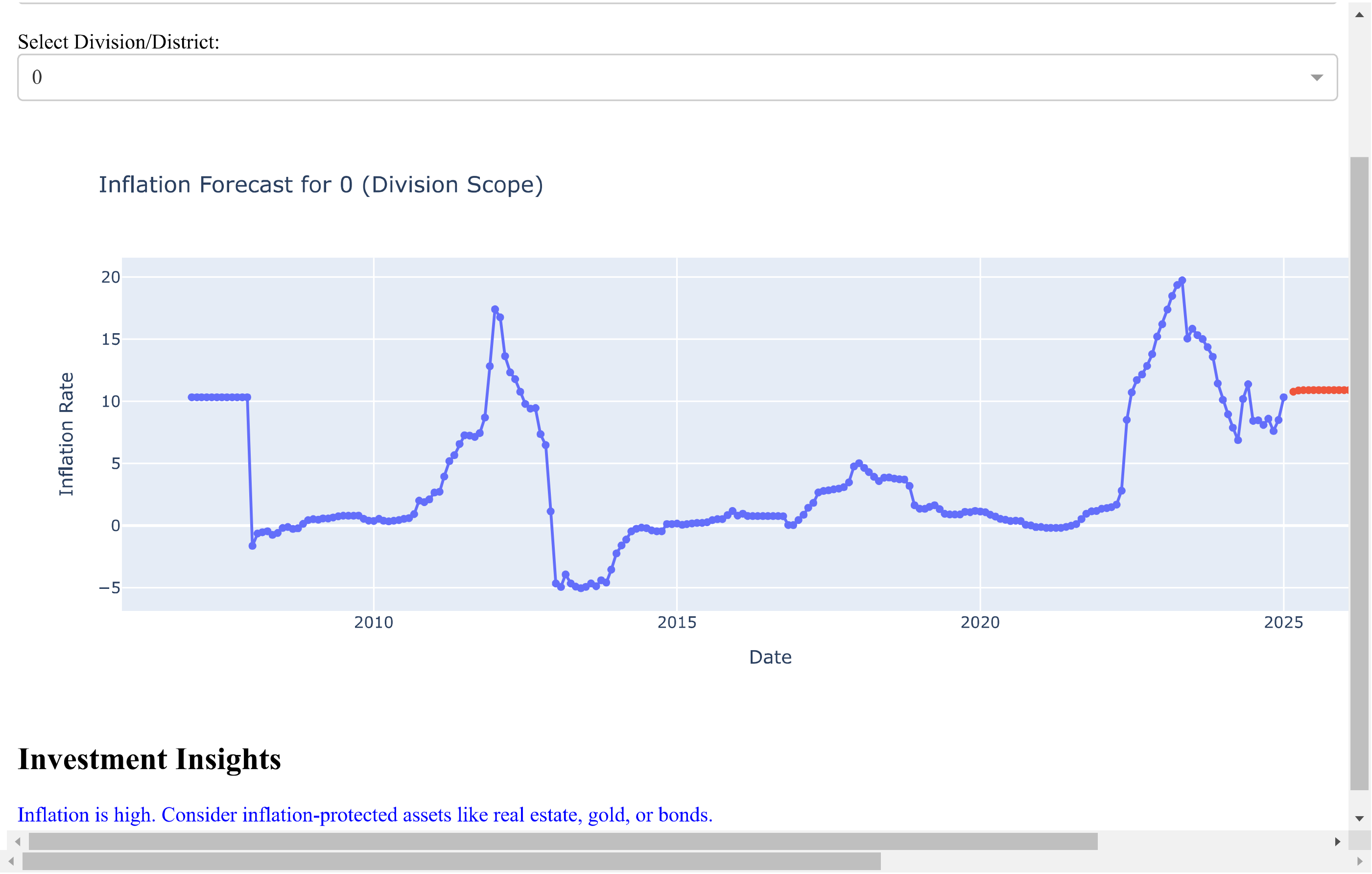
Select Division/District:

0

Inflation Forecast for 0 (Division Scope)

Investment Insights

Inflation is high. Consider inflation-protected assets like real estate, gold, or bonds.



Select Division/District:

0

Inflation Forecast for 0 (Division Scope)

Investment Insights

Inflation is high. Consider inflation-protected assets like real estate, gold, or bonds.

```
from jupyter_dash import JupyterDash
from dash import dcc, html
from dash.dependencies import Input, Output
import pandas as pd
import plotly.graph_objects as go
from statsmodels.tsa.arima.model import ARIMA

# Load prepared datasets
division_data = pd.read_csv("division_level_data.csv")  # Aggregated by division
district_data = pd.read_csv("district_level_data.csv")  # Raw district-level data

# Convert price_date to datetime
division_data['price_date'] = pd.to_datetime(division_data['price_date'])
district_data['price_date'] = pd.to_datetime(district_data['price_date'])

# Function to generate investment advice
def generate_investment_advice(inflation_rate):
    if inflation_rate > 10:
        return "Inflation is high. Consider inflation-protected assets like real estate, gold, or bonds."
    elif inflation_rate > 5:
        return "Inflation is moderate. Diversify with a mix of stocks, real estate, and commodities."
    else:
        return "Inflation is low. It's a good time to invest in growth-oriented sectors like technology or start-ups."

# Initialize the Dash app
app = JupyterDash(__name__)

# Layout
app.layout = html.Div([
    html.H1("Inflation Forecasting Dashboard"),

    # Dropdown to select forecasting scope
    html.Div([
        html.Label("Select Forecasting Scope:"),
        dcc.Dropdown(
            id='forecast-scope',
            options=[
                {'label': 'Division-Level Forecasting', 'value': 'division'},
                {'label': 'District-Level Forecasting', 'value': 'district'}
            ],
            value='division',  # Default selection
            clearable=False
        )
    ], style={'marginBottom': '20px'}),

    # Dropdown to select specific division or district
    html.Div([
        html.Label("Select Division/District:"),
        dcc.Dropdown(id='region-selector', clearable=False)
    ], style={'marginBottom': '20px'}),

    # Slider to select specific year or date
    html.Div([
        html.Label("Select Year:"),
        dcc.Slider(id='year-slider', min=2000, max=2025, step=1, value=2025,
                   marks={i: str(i) for i in range(2000, 2026, 5)}),
    ], style={'marginBottom': '20px'}),

    # Graph for forecasting
    dcc.Graph(id='forecast-graph'),

    # Investment advice section
    html.Div([
        html.H2("Investment Insights"),
        html.Div(id='investment-advice', style={'fontSize': '16px', 'color': 'blue'})
    ], style={'marginBottom': '20px'})
])

# Callbacks
@app.callback(
    Output('region-selector', 'options'),
    Output('region-selector', 'value'),
    Input('forecast-scope', 'value')
)
def update_region_selector(scope):
    if scope == 'division':
        regions = division_data['adm1_name'].unique()
        else:
            regions = district_data['adm2_name'].unique()

        options = [{'label': region, 'value': region} for region in regions]
        return options, regions[0]  # Default to the first region


    @app.callback(
        Output('forecast-graph', 'figure'),
        Output('investment-advice', 'children'),
        Input('forecast-scope', 'value'),
        Input('region-selector', 'value'),
        Input('year-slider', 'value')
    )
    def update_forecast(scope, region, selected_year):
        if scope == 'division':
            data = division_data[division_data['adm1_name'] == region]
        else:
            data = district_data[district_data['adm2_name'] == region]

        # Filter data based on the selected year
        data = data[data['price_date'].dt.year <= selected_year]

        # Train ARIMA model
        model = ARIMA(data['inflation_exchange_rate_unofficial'], order=(1, 1, 1))
        model_fit = model.fit()

        # Forecast future values
        future_steps = 12
        forecast = model_fit.forecast(steps=future_steps)

        # Generate future dates
        last_date = data['price_date'].iloc[-1]
        future_dates = pd.date_range(start=last_date, periods=future_steps + 1, freq='M')[1:]

        # Combine dates and forecast values
        forecast_df = pd.DataFrame({'Date': future_dates, 'Forecast': forecast})

        # Color-coded historical data
        colors = ['green' if rate <= 5 else 'orange' if rate <= 10 else 'red'
                  for rate in data['inflation_exchange_rate_unofficial']]

        # Create figure
        fig = go.Figure()
        fig.add_trace(go.Scatter(
            x=data['price_date'],
            y=data['inflation_exchange_rate_unofficial'],
            mode='lines+markers',
            marker=dict(color=colors),
            name='Historical Data'
        ))
        fig.add_trace(go.Scatter(
            x=forecast_df['Date'],
            y=forecast_df['Forecast'],
            mode='lines+markers',
            marker=dict(color='blue'),
            name='Forecast'
        ))
        fig.update_layout(
            title=f"Inflation Forecast for {region} ({scope.capitalize()} Scope) - Up to {selected_year}",
            xaxis_title="Date",
            yaxis_title="Inflation Rate"
        )

        # Determine the most relevant inflation rate for advice
        if not data.empty:
            relevant_rate = data['inflation_exchange_rate_unofficial'].iloc[-1]
        else:
            relevant_rate = forecast.iloc[-1]
```