Code:

```
import plotly.graph_objects as go
import plotly.express as px
import pandas as pd
import numpy as np
from plotly.subplots import make_subplots
import dash
from dash import dcc, html, callback, Input, Output
from dash.dependencies import State, ALL
import dash_bootstrap_components as dbc
import json
import os
from sklearn.linear_model import LinearRegression

# Load the datasets
item_df = pd.read_csv('itemIndex.csv')
state_df = pd.read_csv('stateIndex.csv')

# Convert date columns to datetime
item_df['date'] = pd.to_datetime(item_df['year'].astype(str) + '-' +
item_df['month'].astype(str) + '-01')
state_df['date'] = pd.to_datetime(state_df['year'].astype(str) + '-' +
state_df['month'].astype(str) + '-01')

# Get unique items and states for dropdown options
items = sorted(item_df['description'].unique())
states = sorted(state_df['state'].unique())

# Create the Dash app with a nicer theme
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.COSMO],
suppress_callback_exceptions=True)
app.title = "GoIStats Data Visualization Dashboard"

# Define the layout
app.layout = dbc.Container([
    # Header section with logo and title
    dbc.Row([
        dbc.Col([
            html.Div([
                html.Img(src="https://www.mospi.gov.in/sites/all/themes/mospi/images/emble
m-dark.png",
                         height="60px", className="me-2"),
                html.H1("Innovate with GoIStats: Data-Driven Insights for Viksit Bharat",
                        className="text-primary mb-0")
            ], style={"display": "flex", "alignItems": "center", "justifyContent":
"center"}),
            html.P("Interactive Price Index Dashboard for Economic Analysis",
                   className="text-center text-muted")
        ], width=12)
    ], className="mb-4 mt-3"),

    # Landing page content that will be shown initially
    dbc.Row([
```

```
        dbc.Col([
            dbc.Card([
                dbc.CardHeader([
                    html.H4("Welcome to the GoIStats Dashboard", className="card-title"),
                    html.P("Explore price trends across India through interactive
visualizations",
                           className="card-subtitle text-muted")
                ]),
                dbc.CardBody([
                    html.P("Hover over the map to see state-wise inflation trends"),
                    # This is where the interactive map and graph will go
                    html.Div(id="landing-page-content", className="mt-3"),
                    dbc.Button("Start Analysis", id="start-analysis-btn", color="primary",
className="mt-3")
                ])
            ], className="shadow")
        ], width=12)
    ], id="landing-content", className="mb-4"),

    # Main dashboard controls
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader([
                    html.H5("Analysis Controls", className="mb-0"),
                    html.Small("Select analysis type and parameters below")
                ]),
                dbc.CardBody([
                    html.H5("Select View Type:"),
                    dcc.Tabs(id='view-type-tabs', value='item-view', children=[
                        dcc.Tab(label='Item-wise Analysis', value='item-view'),
                        dcc.Tab(label='State-wise Analysis', value='state-view'),
                        dcc.Tab(label='Comparative Analysis', value='compare-view'),
                        dcc.Tab(label='Forecasting', value='forecast-view'),
                    ]),
                    html.Div(id='view-controls')
                ])
            ], className="shadow"),
        ], width=12),
    ], className="mb-4"),

    # Visualization section
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader([
                    html.H5("Data Visualization", className="mb-0"),
                    html.Small("Interactive charts based on your selections")
                ]),
                dbc.CardBody([
                    dcc.Loading(
                        id="loading-main-graph",
                        type="circle",
                        children=html.Div(id='main-graph-container')
```

```
                    )
                ])
            ], className="shadow"),
        ], width=12),
    ], className="mb-4"),

    # Insights panel
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader([
                    html.H5("Data Insights", className="mb-0"),
                    html.Small("Automated analysis of trends and patterns")
                ]),
                dbc.CardBody([
                    html.Div(id='insights-panel')
                ])
            ], className="shadow"),
        ], width=12),
    ], className="mb-4"),

    # Store components to hold state
    dcc.Store(id='item-view-data', data={}),
    dcc.Store(id='state-view-data', data={}),
    dcc.Store(id='compare-view-data', data={}),
    dcc.Store(id='forecast-view-data', data={}),
    dcc.Store(id='selected-state', data='ALL India'),

    dbc.Modal([
        dbc.ModalHeader("About This Dashboard"),
        dbc.ModalBody([
            html.P("This dashboard was created for the 'Innovate with GoIStats' hackathon
organized by the Ministry of Statistics and Programme Implementation (MoSPI)."),
            html.P("It provides interactive visualizations of price indices for various
food items across different states in India."),
            html.P("The data is sourced from official government statistics and covers the
period from 2014 to 2025."),
            html.P("Use this tool to analyze trends, compare regions, and gain insights
for policy-making towards building a 'Viksit Bharat'.")
        ]),
        dbc.ModalFooter(
            dbc.Button("Close", id="close-about", className="ml-auto")
        ),
    ], id="about-modal"),

    html.Footer([
        html.Div([
            html.Span("© 2025 Innovate with GoIStats Hackathon | "),
            html.A("Ministry of Statistics & Programme Implementation",
href="https://mospi.gov.in/", target="_blank"),
            html.Span(" | "),
            html.Button("About", id="open-about-btn", className="btn btn-link btn-sm p-0")
        ], className="text-center")
    ], className="text-muted mt-4 pt-3 border-top")
```

```
], fluid=True, className="px-4")

# Callbacks for dynamic controls based on view type
@app.callback(
    Output('view-controls', 'children'),
    Input('view-type-tabs', 'value')
)
def update_view_controls(view_type):
    if view_type == 'item-view':
        return [
            html.H5("Select Item:", className="mt-3"),
            dcc.Dropdown(
                id='item-dropdown',
                options=[{'label': item, 'value': item} for item in items],
                value=items[0]
            ),
            html.H5("Select Date Range:", className="mt-3"),
            dcc.RangeSlider(
                id='item-year-slider',
                min=item_df['year'].min(),
                max=item_df['year'].max(),
                value=[2018, 2023],
                marks={str(year): str(year) for year in range(item_df['year'].min(),
item_df['year'].max()+1, 2)},
                step=1
            ),
            html.H5("Select Visualization Type:", className="mt-3"),
            dcc.RadioItems(
                id='item-viz-type',
                options=[
                    {'label': 'Line Chart', 'value': 'line'},
                    {'label': 'Bar Chart', 'value': 'bar'},
                    {'label': 'Heatmap', 'value': 'heatmap'},
                    {'label': 'Seasonal Decomposition', 'value': 'seasonal'}
                ],
                value='line',
                inline=True
            )
        ]

    elif view_type == 'state-view':
        return [
            html.H5("Select State:", className="mt-3"),
            dcc.Dropdown(
                id='state-dropdown',
                options=[{'label': state, 'value': state} for state in states],
                value='ALL India'
            ),
            html.H5("Select Region Type:", className="mt-3"),
            dcc.RadioItems(
                id='region-type',
                options=[
                    {'label': 'Rural', 'value': 'rural'},
```

```
                    {'label': 'Urban', 'value': 'urban'},
                    {'label': 'Combined', 'value': 'combined'}
                ],
                value='combined',
                inline=True
            ),
            html.H5("Select Date Range:", className="mt-3"),
            dcc.RangeSlider(
                id='state-year-slider',
                min=state_df['year'].min(),
                max=state_df['year'].max(),
                value=[state_df['year'].min(), state_df['year'].max()],
                marks={str(year): str(year) for year in range(state_df['year'].min(),
state_df['year'].max()+1)},
                step=1
            ),
            html.H5("Select Visualization Type:", className="mt-3"),
            dcc.RadioItems(
                id='state-viz-type',
                options=[
                    {'label': 'Line Chart', 'value': 'line'},
                    {'label': 'Choropleth Map', 'value': 'map'},
                    {'label': 'Bar Chart', 'value': 'bar'},
                ],
                value='line',
                inline=True
            )
        ]

    elif view_type == 'compare-view':
        return [
            dbc.Tabs([
                dbc.Tab(label="Compare Items", tab_id="compare-items", children=[
                    html.H5("Select Items to Compare:", className="mt-3"),
                    dcc.Dropdown(
                        id='compare-items-dropdown',
                        options=[{'label': item, 'value': item} for item in items],
                        value=[items[0], items[1]] if len(items) > 1 else [items[0]],
                        multi=True
                    )
                ]),
                dbc.Tab(label="Compare States", tab_id="compare-states", children=[
                    html.H5("Select States to Compare:", className="mt-3"),
                    dcc.Dropdown(
                        id='compare-states-dropdown',
                        options=[{'label': state, 'value': state} for state in states],
                        value=['ALL India', states[1]] if len(states) > 1 else ['ALL
India'],
                        multi=True
                    ),
                    html.H5("Select Region Type:", className="mt-3"),
                    dcc.RadioItems(
                        id='compare-region-type',
                        options=[
```

```
                            {'label': 'Rural', 'value': 'rural'},
                            {'label': 'Urban', 'value': 'urban'},
                            {'label': 'Combined', 'value': 'combined'}
                        ],
                        value='combined',
                        inline=True
                    )
                ])
            ], id="compare-tabs", active_tab="compare-items"),
            html.H5("Select Date Range:", className="mt-3"),
            dcc.RangeSlider(
                id='compare-year-slider',
                min=item_df['year'].min(),
                max=item_df['year'].max(),
                value=[2018, 2023],
                marks={str(year): str(year) for year in range(item_df['year'].min(),
item_df['year'].max()+1, 2)},
                step=1
            ),
            html.H5("Select Visualization Type:", className="mt-3"),
            dcc.RadioItems(
                id='compare-viz-type',
                options=[
                    {'label': 'Line Chart', 'value': 'line'},
                    {'label': 'Bar Chart', 'value': 'bar'},
                    {'label': 'Radar Chart', 'value': 'radar'}
                ],
                value='line',
                inline=True
            )
        ]

    elif view_type == 'forecast-view':
        return [
            html.H5("Select Item for Forecasting:", className="mt-3"),
            dcc.Dropdown(
                id='forecast-item-dropdown',
                options=[{'label': item, 'value': item} for item in items],
                value=items[0]
            ),
            html.H5("Historical Data Years:", className="mt-3"),
            dcc.RangeSlider(
                id='historical-year-slider',
                min=item_df['year'].min(),
                max=2024,  # Use actual data until 2024
                value=[2020, 2024],
                marks={str(year): str(year) for year in range(item_df['year'].min(), 2025,
2)},
                step=1
            ),
            html.H5("Forecast Period (Months):", className="mt-3"),
            dcc.Slider(
                id='forecast-period-slider',
                min=3,
```

```
                max=24,
                value=12,
                marks={3: '3m', 6: '6m', 12: '1yr', 18: '18m', 24: '2yrs'},
                step=3
            ),
            html.H5("Forecasting Method:", className="mt-3"),
            dcc.RadioItems(
                id='forecast-method',
                options=[
                    {'label': 'Linear Regression', 'value': 'linear'},
                    {'label': 'Moving Average', 'value': 'ma'},
                    {'label': 'Exponential Smoothing', 'value': 'exp'}
                ],
                value='linear',
                inline=True
            )
        ]

# Store item view selections
@app.callback(
    Output('item-view-data', 'data'),
    [Input('item-dropdown', 'value'),
     Input('item-year-slider', 'value'),
     Input('item-viz-type', 'value')],
    [State('item-view-data', 'data')]
)
def store_item_selections(item, years, viz_type, data):
    ctx = dash.callback_context
    if not ctx.triggered:
        return data

    data = data or {}
    data.update({
        'item': item,
        'years': years,
        'viz_type': viz_type
    })
    return data

# Store state view selections
@app.callback(
    Output('state-view-data', 'data'),
    [Input('state-dropdown', 'value'),
     Input('region-type', 'value'),
     Input('state-year-slider', 'value'),
     Input('state-viz-type', 'value')],
    [State('state-view-data', 'data')]
)
def store_state_selections(state, region_type, years, viz_type, data):
    ctx = dash.callback_context
    if not ctx.triggered:
        return data

    data = data or {}
```

```
    data.update({
        'state': state,
        'region_type': region_type,
        'years': years,
        'viz_type': viz_type
    })
    return data

# Store compare view selections
@app.callback(
    Output('compare-view-data', 'data'),
    [Input('compare-items-dropdown', 'value'),
     Input('compare-states-dropdown', 'value'),
     Input('compare-region-type', 'value'),
     Input('compare-year-slider', 'value'),
     Input('compare-tabs', 'active_tab'),
     Input('compare-viz-type', 'value')],
    [State('compare-view-data', 'data')]
)
def store_compare_selections(items, states, region_type, years, active_tab, viz_type,
data):
    ctx = dash.callback_context
    if not ctx.triggered:
        return data

    data = data or {}
    data.update({
        'items': items,
        'states': states,
        'region_type': region_type,
        'years': years,
        'active_tab': active_tab,
        'viz_type': viz_type
    })
    return data

# Store forecast view selections
@app.callback(
    Output('forecast-view-data', 'data'),
    [Input('forecast-item-dropdown', 'value'),
     Input('historical-year-slider', 'value'),
     Input('forecast-period-slider', 'value'),
     Input('forecast-method', 'value')],
    [State('forecast-view-data', 'data')]
)
def store_forecast_selections(item, hist_years, period, method, data):
    ctx = dash.callback_context
    if not ctx.triggered:
        return data

    data = data or {}
    data.update({
        'item': item,
        'hist_years': hist_years,
```

```
        'period': period,
        'method': method
    })
    return data

# Main callback to update visualization and insights
@app.callback(
    Output('main-graph-container', 'children'),
    Output('insights-panel', 'children'),
    [Input('view-type-tabs', 'value'),
     Input('item-view-data', 'data'),
     Input('state-view-data', 'data'),
     Input('compare-view-data', 'data'),
     Input('forecast-view-data', 'data')]
)
def update_visualization(view_type, item_data, state_data, compare_data, forecast_data):
    ctx = dash.callback_context
    if not ctx.triggered:
        # Default view
        fig = px.line(title="Please select parameters to visualize data")
        return dcc.Graph(figure=fig), "Select parameters to see insights"

    # Item View
    if view_type == 'item-view' and item_data:
        item = item_data.get('item')
        item_years = item_data.get('years', [2018, 2023])
        item_viz_type = item_data.get('viz_type', 'line')

        # Proceed with visualization logic for item view
        filtered_df = item_df[(item_df['description'] == item) &
                              (item_df['year'] >= item_years[0]) &
                              (item_df['year'] <= item_years[1])]

        # Rest of your item view visualization code...
        if item_viz_type == 'line':
            fig = px.line(
                filtered_df,
                x='date',
                y='combined_index',
                title=f'Price Index Trend for {item} ({item_years[0]}-{item_years[1]})'
            )
            fig.update_layout(xaxis_title='Date', yaxis_title='Price Index')

        elif item_viz_type == 'bar':
            fig = px.bar(
                filtered_df,
                x='date',
                y='combined_index',
                title=f'Price Index Trend for {item} ({item_years[0]}-{item_years[1]})'
            )
            fig.update_layout(xaxis_title='Date', yaxis_title='Price Index')

        elif item_viz_type == 'heatmap':
            pivot_df = filtered_df.pivot_table(
```

```
                index='year',
                columns='month',
                values='combined_index'
            )
            fig = px.imshow(
                pivot_df,
                labels=dict(x="Month", y="Year", color="Price Index"),
                x=[f'Month {m}' for m in range(1, 13)],
                y=pivot_df.index,
                title=f'Monthly Price Index Heatmap for {item} ({item_years[0]}-
{item_years[1]})'
            )

        elif item_viz_type == 'seasonal':
            # Monthly trend visualization
            if len(filtered_df) > 12:  # Need enough data for seasonal analysis
                # Calculate average by month across years
                monthly_avg =
filtered_df.groupby('month')['combined_index'].mean().reset_index()

                fig = go.Figure()

                # Add monthly average line
                fig.add_trace(go.Scatter(
                    x=monthly_avg['month'],
                    y=monthly_avg['combined_index'],
                    mode='lines+markers',
                    name='Monthly Average',
                    line=dict(color='blue', width=3)
                ))

                # Add yearly lines for comparison
                for year in sorted(filtered_df['year'].unique()):
                    year_data = filtered_df[filtered_df['year'] == year]
                    if len(year_data) > 6:  # Only include years with sufficient data
                        fig.add_trace(go.Scatter(
                            x=year_data['month'],
                            y=year_data['combined_index'],
                            mode='lines',
                            name=f'Year {year}',
                            line=dict(width=1, dash='dot'),
                            opacity=0.5
                        ))

                fig.update_layout(
                    title=f'Seasonal Pattern for {item} ({item_years[0]}-
{item_years[1]})',
                    xaxis=dict(
                        title='Month',
                        tickmode='array',
                        tickvals=list(range(1, 13)),
                        ticktext=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                                  'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
                    ),
```

```
                    yaxis=dict(title='Price Index')
                )
            else:
                fig = px.line(title=f"Insufficient data for seasonal analysis of {item}")

        # Calculate insights
        start_val = filtered_df.iloc[0]['combined_index'] if not filtered_df.empty else 0
        end_val = filtered_df.iloc[-1]['combined_index'] if not filtered_df.empty else 0
        total_change = end_val - start_val
        percent_change = (total_change / start_val * 100) if start_val > 0 else 0

        monthly_changes = filtered_df['combined_index'].diff()
        avg_monthly_change = monthly_changes.mean() if not filtered_df.empty else 0
        max_increase = monthly_changes.max() if not filtered_df.empty else 0
        max_decrease = monthly_changes.min() if not filtered_df.empty else 0

        # Find months with highest and lowest values
        if not filtered_df.empty:
            max_month = filtered_df.loc[filtered_df['combined_index'].idxmax()]
            min_month = filtered_df.loc[filtered_df['combined_index'].idxmin()]
            max_month_str = f"{max_month['year']}-{max_month['month']}"
            min_month_str = f"{min_month['year']}-{min_month['month']}"
        else:
            max_month_str = "N/A"
            min_month_str = "N/A"

        insights = [
            html.H4(f"Insights for {item}"),
            html.Ul([
                html.Li(f"Total change: {total_change:.2f} index points
({percent_change:.2f}%)"),
                html.Li(f"Average monthly change: {avg_monthly_change:.2f} index points"),
                html.Li(f"Highest value: {end_val:.2f} (Month: {max_month_str})"),
                html.Li(f"Lowest value: {start_val:.2f} (Month: {min_month_str})"),
                html.Li(f"Largest monthly increase: {max_increase:.2f} index points"),
                html.Li(f"Largest monthly decrease: {max_decrease:.2f} index points")
            ])
        ]

        return dcc.Graph(figure=fig), insights

    # State View
    elif view_type == 'state-view' and state_data:
        # Extract state view selections
        state = state_data.get('state')
        region_type = state_data.get('region_type', 'combined')
        state_years = state_data.get('years', [state_df['year'].min(),
state_df['year'].max()])
        state_viz_type = state_data.get('viz_type', 'line')

        # Your state view visualization code...
        filtered_df = state_df[
            (state_df['state'] == state) &
            (state_df['year'] >= state_years[0]) &
```

```
            (state_df['year'] <= state_years[1])
        ]

        if state_viz_type == 'line':
            fig = px.line(
                filtered_df,
                x='date',
                y=region_type,
                title=f'Price Index for Cereals in {state} ({state_years[0]}-
{state_years[1]}) - {region_type.capitalize()} Areas'
            )
            fig.update_layout(xaxis_title='Date', yaxis_title='Price Index')

        elif state_viz_type == 'bar':
            # Create quarterly aggregates for better visualization
            filtered_df['quarter'] = filtered_df['date'].dt.to_period('Q').astype(str)
            quarterly_df =
filtered_df.groupby('quarter')[region_type].mean().reset_index()

            fig = px.bar(
                quarterly_df,
                x='quarter',
                y=region_type,
                title=f'Quarterly Price Index for Cereals in {state} ({state_years[0]}-
{state_years[1]}) - {region_type.capitalize()}'
            )
            fig.update_layout(xaxis_title='Quarter', yaxis_title='Price Index')

        elif state_viz_type == 'map':
            # For this view, we need data from all states for the most recent date
            latest_year = state_years[1]
            latest_month = state_df[(state_df['year'] == latest_year)]['month'].max()

            map_df = state_df[
                (state_df['year'] == latest_year) &
                (state_df['month'] == latest_month)
            ]

            fig = px.choropleth(
                map_df,
                locations='state',
                color=region_type,
                locationmode='country names',
                scope='asia',
                center={'lat': 22, 'lon': 82},
                title=f'Price Index Across India ({latest_year}-{latest_month}) -
{region_type.capitalize()}',
                color_continuous_scale=px.colors.sequential.Plasma
            )

        # Calculate insights
        start_val = filtered_df.iloc[0][region_type] if not filtered_df.empty else 0
        end_val = filtered_df.iloc[-1][region_type] if not filtered_df.empty else 0
        total_change = end_val - start_val
```

```
        percent_change = (total_change / start_val * 100) if start_val > 0 else 0

        monthly_changes = filtered_df[region_type].diff()
        avg_monthly_change = monthly_changes.mean() if not filtered_df.empty else 0

        if not filtered_df.empty and state != 'ALL India':
            national_df = state_df[
                (state_df['state'] == 'ALL India') &
                (state_df['year'] >= state_years[0]) &
                (state_df['year'] <= state_years[1])
            ]
            nat_end_val = national_df.iloc[-1][region_type] if not national_df.empty else
0
            diff_from_national = end_val - nat_end_val
            diff_percent = (diff_from_national / nat_end_val * 100) if nat_end_val > 0
else 0

            national_insight = html.Li(f"Difference from national average:
{diff_from_national:.2f} index points ({diff_percent:.2f}%)")
        else:
            national_insight = None

        insights = [
            html.H4(f"Insights for {state} ({region_type.capitalize()})"),
            html.Ul([
                html.Li(f"Total change: {total_change:.2f} index points
({percent_change:.2f}%)"),
                html.Li(f"Average monthly change: {avg_monthly_change:.2f} index points"),
                html.Li(f"Starting index: {start_val:.2f}"),
                html.Li(f"Current index: {end_val:.2f}")
            ] + ([national_insight] if national_insight else []))
        ]

        return dcc.Graph(figure=fig), insights

    # Comparative View
    elif view_type == 'compare-view' and compare_data:
        # Extract compare view selections
        compare_items = compare_data.get('items', [items[0]])
        compare_states = compare_data.get('states', ['ALL India'])
        compare_region = compare_data.get('region_type', 'combined')
        compare_years = compare_data.get('years', [2018, 2023])
        compare_tab = compare_data.get('active_tab', 'compare-items')
        compare_viz_type = compare_data.get('viz_type', 'line')

        if compare_tab == 'compare-items':
            filtered_df = item_df[
                (item_df['description'].isin(compare_items)) &
                (item_df['year'] >= compare_years[0]) &
                (item_df['year'] <= compare_years[1])
            ]

            if compare_viz_type == 'line':
                fig = px.line(
```

```
                    filtered_df,
                    x='date',
                    y='combined_index',
                    color='description',
                    title=f'Comparative Analysis of Items ({compare_years[0]}-
{compare_years[1]})'
                )
                fig.update_layout(xaxis_title='Date', yaxis_title='Price Index')

            elif compare_viz_type == 'bar':
                fig = px.bar(
                    filtered_df,
                    x='date',
                    y='combined_index',
                    color='description',
                    title=f'Comparative Analysis of Items ({compare_years[0]}-
{compare_years[1]})'
                )
                fig.update_layout(xaxis_title='Date', yaxis_title='Price Index')

            elif compare_viz_type == 'radar':
                radar_df =
filtered_df.groupby('description')['combined_index'].mean().reset_index()
                fig = px.line_polar(
                    radar_df,
                    r='combined_index',
                    theta='description',
                    line_close=True,
                    title=f'Radar Chart of Items ({compare_years[0]}-{compare_years[1]})'
                )

            # Calculate insights
            insights_list = []
            for item in compare_items:
                item_df_filtered = filtered_df[filtered_df['description'] == item]
                if not item_df_filtered.empty:
                    start_val = item_df_filtered.iloc[0]['combined_index']
                    end_val = item_df_filtered.iloc[-1]['combined_index']
                    total_change = end_val - start_val
                    percent_change = (total_change / start_val * 100) if start_val > 0
else 0

                    insights_list.append(html.Li([
                        f"{item}: ",
                        html.Span(f"{percent_change:.2f}%",
                                  style={'color': 'red' if percent_change > 0 else 'green'})
                    ]))

            insights = [
                html.H4(f"Comparative Insights"),
                html.H5("Price Index Change (%)"),
                html.Ul(insights_list)
            ]
```

```
        else:  # compare-states
            filtered_df = state_df[
                (state_df['state'].isin(compare_states)) &
                (state_df['year'] >= compare_years[0]) &
                (state_df['year'] <= compare_years[1])
            ]

            if compare_viz_type == 'line':
                fig = px.line(
                    filtered_df,
                    x='date',
                    y=compare_region,
                    color='state',
                    title=f'Comparative Analysis of States ({compare_years[0]}-
{compare_years[1]}) - {compare_region.capitalize()} Areas'
                )
                fig.update_layout(xaxis_title='Date', yaxis_title='Price Index')

            elif compare_viz_type == 'bar':
                fig = px.bar(
                    filtered_df,
                    x='date',
                    y=compare_region,
                    color='state',
                    title=f'Comparative Analysis of States ({compare_years[0]}-
{compare_years[1]}) - {compare_region.capitalize()} Areas'
                )
                fig.update_layout(xaxis_title='Date', yaxis_title='Price Index')

            elif compare_viz_type == 'radar':
                radar_df =
filtered_df.groupby('state')[compare_region].mean().reset_index()
                fig = px.line_polar(
                    radar_df,
                    r=compare_region,
                    theta='state',
                    line_close=True,
                    title=f'Radar Chart of States ({compare_years[0]}-{compare_years[1]})
- {compare_region.capitalize()} Areas'
                )

            # Calculate insights
            insights_list = []
            for state in compare_states:
                state_df_filtered = filtered_df[filtered_df['state'] == state]
                if not state_df_filtered.empty:
                    start_val = state_df_filtered.iloc[0][compare_region]
                    end_val = state_df_filtered.iloc[-1][compare_region]
                    total_change = end_val - start_val
                    percent_change = (total_change / start_val * 100) if start_val > 0
else 0

                    insights_list.append(html.Li([
                        f"{state}: ",
```

```
                        html.Span(f"{percent_change:.2f}%",
                                  style={'color': 'red' if percent_change > 0 else 'green'})
                    ]))

            insights = [
                html.H4(f"Comparative Insights"),
                html.H5("Price Index Change (%)"),
                html.Ul(insights_list)
            ]

        return dcc.Graph(figure=fig), insights

    # Forecast View
    elif view_type == 'forecast-view' and forecast_data:
        # Extract forecast view selections
        forecast_item = forecast_data.get('item')
        historical_years = forecast_data.get('hist_years', [2020, 2024])
        forecast_period = forecast_data.get('period', 12)
        forecast_method = forecast_data.get('method', 'linear')

        filtered_df = item_df[
            (item_df['description'] == forecast_item) &
            (item_df['year'] >= historical_years[0]) &
            (item_df['year'] <= historical_years[1])
        ].sort_values('date')

        if filtered_df.empty:
            fig = px.line(title="No data available for the selected parameters")
            return dcc.Graph(figure=fig), "No data available for forecasting"

        # Prepare data for forecasting
        filtered_df = filtered_df.reset_index(drop=True)
        X = np.array(range(len(filtered_df))).reshape(-1, 1)
        y = filtered_df['combined_index'].values

        if forecast_method == 'linear':
            # Fit linear regression model
            model = LinearRegression()
            model.fit(X, y)

            # Predict historical values
            historical_pred = model.predict(X)

            # Forecast future values
            future_X = np.array(range(len(filtered_df), len(filtered_df) +
forecast_period)).reshape(-1, 1)
            forecast_values = model.predict(future_X)

            # Create future dates for forecast
            last_date = filtered_df['date'].iloc[-1]
            forecast_dates = pd.date_range(start=last_date, periods=forecast_period+1,
freq='MS')[1:]

            # Create figure
```

```
            fig = go.Figure()

            # Add historical data
            fig.add_trace(go.Scatter(
                x=filtered_df['date'],
                y=filtered_df['combined_index'],
                mode='lines+markers',
                name='Historical Data',
                line=dict(color='blue')
            ))

            # Add model fit for historical data
            fig.add_trace(go.Scatter(
                x=filtered_df['date'],
                y=historical_pred,
                mode='lines',
                name='Model Fit',
                line=dict(color='green', dash='dash')
            ))

            # Add forecast
            fig.add_trace(go.Scatter(
                x=forecast_dates,
                y=forecast_values,
                mode='lines',
                name='Forecast',
                line=dict(color='red')
            ))

            # Add confidence intervals (simple estimate)
            residuals = y - historical_pred
            residual_std = np.std(residuals)

            fig.add_trace(go.Scatter(
                x=list(forecast_dates) + list(reversed(forecast_dates)),
                y=list(forecast_values + 1.96 * residual_std) +
list(reversed(forecast_values - 1.96 * residual_std)),
                fill='toself',
                fillcolor='rgba(255,0,0,0.2)',
                line=dict(color='rgba(255,0,0,0)'),
                name='95% Confidence Interval'
            ))

            fig.update_layout(
                title=f'Price Index Forecast for {forecast_item}',
                xaxis_title='Date',
                yaxis_title='Price Index',
                legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right",
x=1)
            )

            # Calculate insights
            last_historical = filtered_df['combined_index'].iloc[-1]
            forecast_end = forecast_values[-1]
```

```
            forecast_change = forecast_end - last_historical
            forecast_percent = (forecast_change / last_historical * 100) if
last_historical > 0 else 0

            slope = model.coef_[0]
            monthly_growth_rate = slope
            annual_growth_rate = slope * 12

            r_squared = model.score(X, y)

            insights = [
                html.H4(f"Forecast Insights for {forecast_item}"),
                html.Ul([
                    html.Li(f"Current index value: {last_historical:.2f}"),
                    html.Li(f"Forecasted value ({forecast_period} months from now):
{forecast_end:.2f}"),
                    html.Li(f"Expected change: {forecast_change:.2f} index points
({forecast_percent:.2f}%)"),
                    html.Li(f"Monthly growth rate: {monthly_growth_rate:.4f} points per
month"),
                    html.Li(f"Yearly growth rate: {annual_growth_rate:.2f} points per
year"),
                    html.Li(f"Model fit quality (R²): {r_squared:.4f}")
                ]),
                html.Div([
                    html.P("Interpretation:", className="font-weight-bold"),
                    html.P([
                        "Based on historical trends, we forecast a ",
                        html.Span(
                            f"{'rise' if forecast_change > 0 else 'fall'} of
{abs(forecast_percent):.2f}%",
                            style={'fontWeight': 'bold', 'color': 'red' if forecast_change
> 0 else 'green'}
                        ),
                        f" in the price index for {forecast_item} over the next
{forecast_period} months."
                    ]),
                    html.P(f"The model explains {r_squared*100:.2f}% of the historical
price variation."),
                ])
            ]

        elif forecast_method == 'ma':
            # Moving average forecast
            window_size = forecast_period // 2  # Example window size
            moving_avg = filtered_df['combined_index'].rolling(window=window_size).mean()

            # Forecast future values
            forecast_values = np.tile(moving_avg.iloc[-1], forecast_period)

            # Create future dates for forecast
            last_date = filtered_df['date'].iloc[-1]
            forecast_dates = pd.date_range(start=last_date, periods=forecast_period+1,
freq='MS')[1:]
```

```
            # Create figure
            fig = go.Figure()

            # Add historical data
            fig.add_trace(go.Scatter(
                x=filtered_df['date'],
                y=filtered_df['combined_index'],
                mode='lines+markers',
                name='Historical Data',
                line=dict(color='blue')
            ))

            # Add moving average
            fig.add_trace(go.Scatter(
                x=filtered_df['date'],
                y=moving_avg,
                mode='lines',
                name='Moving Average',
                line=dict(color='green', dash='dash')
            ))

            # Add forecast
            fig.add_trace(go.Scatter(
                x=forecast_dates,
                y=forecast_values,
                mode='lines',
                name='Forecast',
                line=dict(color='red')
            ))

            fig.update_layout(
                title=f'Price Index Forecast for {forecast_item}',
                xaxis_title='Date',
                yaxis_title='Price Index',
                legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right",
x=1)
            )

            # Calculate insights
            last_historical = filtered_df['combined_index'].iloc[-1]
            forecast_end = forecast_values[-1]
            forecast_change = forecast_end - last_historical
            forecast_percent = (forecast_change / last_historical * 100) if
last_historical > 0 else 0

            insights = [
                html.H4(f"Forecast Insights for {forecast_item}"),
                html.Ul([
                    html.Li(f"Current index value: {last_historical:.2f}"),
                    html.Li(f"Forecasted value ({forecast_period} months from now):
{forecast_end:.2f}"),
                    html.Li(f"Expected change: {forecast_change:.2f} index points
({forecast_percent:.2f}%)"),
```

```
                    html.Li(f"Method: Moving Average")
                ]),
                html.Div([
                    html.P("Interpretation:", className="font-weight-bold"),
                    html.P([
                        "Based on historical trends, we forecast a ",
                        html.Span(
                            f"{'rise' if forecast_change > 0 else 'fall'} of
{abs(forecast_percent):.2f}%",
                            style={'fontWeight': 'bold', 'color': 'red' if forecast_change
> 0 else 'green'}
                        ),
                        f" in the price index for {forecast_item} over the next
{forecast_period} months."
                    ])
                ])
            ]

        elif forecast_method == 'exp':
            # Exponential smoothing forecast
            alpha = 0.2  # Example smoothing factor
            exp_smooth = filtered_df['combined_index'].ewm(alpha=alpha).mean()

            # Forecast future values
            forecast_values = np.tile(exp_smooth.iloc[-1], forecast_period)

            # Create future dates for forecast
            last_date = filtered_df['date'].iloc[-1]
            forecast_dates = pd.date_range(start=last_date, periods=forecast_period+1,
freq='MS')[1:]

            # Create figure
            fig = go.Figure()

            # Add historical data
            fig.add_trace(go.Scatter(
                x=filtered_df['date'],
                y=filtered_df['combined_index'],
                mode='lines+markers',
                name='Historical Data',
                line=dict(color='blue')
            ))

            # Add exponential smoothing
            fig.add_trace(go.Scatter(
                x=filtered_df['date'],
                y=exp_smooth,
                mode='lines',
                name='Exponential Smoothing',
                line=dict(color='green', dash='dash')
            ))

            # Add forecast
            fig.add_trace(go.Scatter(
```

```
                x=forecast_dates,
                y=forecast_values,
                mode='lines',
                name='Forecast',
                line=dict(color='red')
            ))

            fig.update_layout(
                title=f'Price Index Forecast for {forecast_item}',
                xaxis_title='Date',
                yaxis_title='Price Index',
                legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right",
x=1)
            )

            # Calculate insights
            last_historical = filtered_df['combined_index'].iloc[-1]
            forecast_end = forecast_values[-1]
            forecast_change = forecast_end - last_historical
            forecast_percent = (forecast_change / last_historical * 100) if
last_historical > 0 else 0

            insights = [
                html.H4(f"Forecast Insights for {forecast_item}"),
                html.Ul([
                    html.Li(f"Current index value: {last_historical:.2f}"),
                    html.Li(f"Forecasted value ({forecast_period} months from now):
{forecast_end:.2f}"),
                    html.Li(f"Expected change: {forecast_change:.2f} index points
({forecast_percent:.2f}%)"),
                    html.Li(f"Method: Exponential Smoothing")
                ]),
                html.Div([
                    html.P("Interpretation:", className="font-weight-bold"),
                    html.P([
                        "Based on historical trends, we forecast a ",
                        html.Span(
                            f"{'rise' if forecast_change > 0 else 'fall'} of
{abs(forecast_percent):.2f}%",
                            style={'fontWeight': 'bold', 'color': 'red' if forecast_change
> 0 else 'green'}
                        ),
                        f" in the price index for {forecast_item} over the next
{forecast_period} months."
                    ])
                ])
            ]

        return dcc.Graph(figure=fig), insights

    else:
        # Default fallback
        fig = px.line(title="Please select parameters to visualize data")
        return dcc.Graph(figure=fig), "Select parameters to see insights"
```

```python
# Add this new callback for the landing page content
@app.callback(
    Output('landing-page-content', 'children'),
    [Input('selected-state', 'data')]
)
def update_landing_content(selected_state):
    # Get the most recent data for all states for the map
    latest_year = state_df['year'].max()
    latest_month = state_df[state_df['year'] == latest_year]['month'].max()

    map_df = state_df[
        (state_df['year'] == latest_year) &
        (state_df['month'] == latest_month)
    ]

    # Create the map figure
    fig_map = px.choropleth(
        map_df,
        locations='state',
        color='combined',
        locationmode='country names',
        scope='asia',
        center={'lat': 22, 'lon': 82},
        title=f'Consumer Price Index for Cereals - {latest_year}',
        color_continuous_scale=px.colors.sequential.Plasma,
        labels={'combined': 'CPI'},
        hover_name='state',
        custom_data=['state', 'combined', 'rural', 'urban']
    )

    fig_map.update_layout(
        geo=dict(
            showcoastlines=True,
            projection_type='mercator',
            lataxis=dict(range=[5, 38]),
            lonaxis=dict(range=[65, 100]),
            showland=True,
            landcolor='lightgray'
        ),
        autosize=True,
        margin=dict(l=0, r=0, b=0, t=40),
        height=450
    )

    # Add hover template
    fig_map.update_traces(
        hovertemplate="<b>%{customdata[0]}</b><br>CPI: %{customdata[1]:.1f}<br>Rural:
%{customdata[2]:.1f}<br>Urban: %{customdata[3]:.1f}"
    )

    # Get historical data for the selected state for the graph
    state_historical = state_df[state_df['state'] == selected_state].sort_values('date')
```

```
    # Create the time series graph
    fig_graph = px.line(
        state_historical,
        x='date',
        y=['combined', 'rural', 'urban'],
        title=f'Price Index Trend - {selected_state}',
        labels={'value': 'CPI', 'date': 'Date', 'variable': 'Region Type'},
        color_discrete_map={
            'combined': '#19A7CE',
            'rural': '#8BC34A',
            'urban': '#FF7043'
        }
    )

    fig_graph.update_layout(
        legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1),
        margin=dict(l=0, r=0, b=0, t=40),
        height=450,
        hovermode="x unified"
    )

    # Return a row with the map and graph side by side
    return dbc.Row([
        dbc.Col([
            html.Div([
                dcc.Graph(
                    id='india-map',
                    figure=fig_map,
                    config={'displayModeBar': False}
                )
            ])
        ], width=6),
        dbc.Col([
            html.Div([
                dcc.Graph(
                    id='state-trend-graph',
                    figure=fig_graph,
                    config={'displayModeBar': False}
                )
            ])
        ], width=6)
    ])

# Add callback to update selected state when hovering on the map
@app.callback(
    Output('selected-state', 'data'),
    [Input('india-map', 'hoverData')]
)
def update_selected_state(hoverData):
    if hoverData:
        state_name = hoverData['points'][0]['customdata'][0]
        return state_name
    return 'ALL India'
```

```python
# Add a start analysis button callback to toggle the landing page visibility
@app.callback(
    [Output('landing-content', 'style'),
     Output('start-analysis-btn', 'children')],
    [Input('start-analysis-btn', 'n_clicks')]
)
def toggle_landing_page(n_clicks):
    if n_clicks and n_clicks % 2 == 1:
        return {'display': 'none'}, "Show Overview Map"
    return {'display': 'block'}, "Start Analysis"

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```