Summer Internship AZ5513

Project Report

Food Commodities Price Prediction and Analysis Across Tamil Nadu

By

Udith V - 2022510015

Surya K S - 2022510022

Abstract:

This project focuses on the real-time prediction and analysis of food commodity prices in Tamil Nadu, leveraging historical data and advanced machine learning models. The system utilizes historical data sourced from government-regulated agricultural markets to forecast prices and provide actionable insights for stakeholders. By employing machine learning techniques such as Prophet and Random Forest Regressor (RFR), the model achieves high accuracy in forecasting trends and future prices of agricultural commodities. A Power BI dashboard and a Dash web application have been developed as part of the project to ensure seamless visualization and interaction. The Power BI dashboard enables stakeholders to explore historical trends, compare commodity prices, and monitor forecasted trends in an interactive manner. The Dash application, on the other hand, integrates the forecasting models to provide real-time updates and visualizations of predicted prices. This dual approach ensures that users have access to both historical insights and predictive analytics in a user-friendly format.

Aim:

To develop a robust and user-friendly system for predicting and visualizing food commodity prices in TamilNadu, enabling stakeholders to make informed decisions through real-time analytics and interactive dashboards, while promoting market stability and efficiency.

The primary goals of this project are:

1. **Accurate Price Forecasting:** Use Prophet for time series modeling and RFR for feature-based predictions to achieve robust and reliable price forecasts.
2. **Real-Time Insights:** Ensure real-time updates of predictions through the Dash application to assist stakeholders in making informed decisions.
3. **Interactive Visualization:** Leverage Power BI and Dash to create intuitive dashboards that allow users to track trends, identify anomalies, and extract actionable insights.
4. **Market Stability and Efficiency:** Support farmers, traders, and policymakers in managing price volatility and optimizing market operations.

This system bridges the gap between complex data-driven predictions and practical decision-making by integrating advanced analytics with real-time visualization tools. It is a step forward in ensuring transparency, efficiency, and sustainability in agricultural markets in Tamil Nadu.

Introduction

Agricultural markets play a pivotal role in the economy of Tamil Nadu, impacting the livelihoods of farmers, traders, and consumers. Price volatility in agricultural commodities often leads to economic instability, affecting production planning, market operations, and policymaking. To address these challenges, this project leverages historical data and advanced analytics to forecast food commodity prices and visualize trends in real-time.

The dataset for this project was obtained from the CEDA Agrimarket website, covering four years (2020 to 2023). It consists of 19 columns and approximately 165,000 rows, providing detailed information on various aspects of food commodity prices, including market location, commodity type, date of transaction, and price levels. This extensive dataset serves as the foundation for building predictive models and interactive dashboards.

The core objective of the project is to empower stakeholders with actionable insights by combining the predictive capabilities of machine learning models with user-friendly visualization tools. The use of Prophet for time series analysis and Random Forest Regressor (RFR) for feature-based predictions ensures a comprehensive and accurate approach to forecasting. These models are integrated into a real-time Dash web application, offering users continuous updates on commodity price trends. Additionally, a Power BI dashboard facilitates historical trend analysis and intuitive data exploration, enabling stakeholders to make informed decisions efficiently.

By providing accurate price forecasts and real-time visualizations, this project aims to enhance transparency in agricultural markets, mitigate the effects of price volatility, and support efficient decision-making processes. The ultimate goal is to contribute to market stability, improve resource allocation, and ensure economic sustainability in Tamil Nadu's agricultural sector.

Data Preprocessing and integration:

The data preprocessing and integration process was essential to ensure the quality and usability of the dataset for analysis and model development. The key steps involved include:

1. **Column Removal:** Unnecessary columns, such as `id`, `state_id`, and `market_id`, were removed to focus on relevant data.
2. **Null Value Handling:** Missing values and outliers were handled to maintain data integrity and consistency.
3. **Datetime Splitting:** The datetime column was split into separate components (e.g., date and time) to enable real-time predictions.
4. **Price Range Creation:** A new column, 'price-range', was created through feature engineering to enhance the analysis.
5. **Storing:** Cleaned data for each year (2020, 2021, 2022, 2023) was stored separately for better organization.
6. **Combining:** All cleaned yearly datasets were integrated into a single `.csv` file for further analysis and model development.

Creating Localhost Database:

MySQL Server: Setting up a MySQL server using XAMPP to store predicted commodity values.

Database Tables: 3 tables - forecast_results_prophet, forecast_results_rf, and market_prices.

Predicting Values and Database Updation:

Model Selection: Predicting commodity prices using two models: Random Forest Regressor and Prophet.

Model Training: Training the models with historical data and then making predictions.

Database Updation: Updating the database with the predicted values in real-time.

Steps in Prediction:

1. Group data based on categories like district ,market and commodity. (Example: if predicting Paddy price in Tamil Nadu, categories like Paddy and area Tamil Nadu only should be considered for prediction).
2. Choose one model at first (prophet here).
3. Training the model with data (date).
4. Predicting target feature price value (modal_price) based on data current time.

5. Updating the new value in the prophet database.
6. For randomforest, the data datetime is splitted into year, month, for easier prediction.
7. Predicting target feature y=modal price by splitting current time into segments.
8. Updating the predicted value in RandomForest database.
9. Evaluating RandomForest model performance (Root mean square error, R2 score).

Visualizing in Real-Time Using Dashboard:

The real-time dashboard will visualize the predicted prices for various commodities in Tamilnadu, providing valuable insights and supporting informed decision-making for stakeholders.

Connection to the dashboard:

1. **Connect MySQL (XAMPP) to Power BI Desktop:** Install MySQL on XAMPP Ensure MySQL is running through XAMPP (http://localhost/phpmyadmin/).
2. **Enable Remote Connections (Optional):** Modify my.ini file: Set bind-address = 0.0.0.0 for external connections. Restart MySQL in XAMPP.
3. **Create Database & Tables (Optional):** Use phpMyAdmin to create or select a database.
4. **Install MySQL ODBC Driver:** Download and install MySQL ODBC Connector from MySQL website.
5. **Set up ODBC Data Source:** Open ODBC Data Source Administrator. Add a System DSN with MySQL ODBC 8.0 driver. Enter connection details: localhost, port 3306, MySQL user (default root), and database.
6. **Connect Power BI:** `Open Power BI Desktop → Get Data → ODBC. Select ODBC data source (e.g., MySQL_xampp), enter credentials. Load data to Power BI.`

Scripts:

Python forecast_app.py:

```
import pandas as pd

import plotly.express as px
```

```
import dash
from dash import dcc, html, Input, Output
from prophet import Prophet # Using Prophet for time series forecasting
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
import mysql.connector # To connect and interact with MySQL
import numpy as np
# Load and preprocess the dataset
df = pd.read_csv('final_data.csv')
df['date'] = pd.to_datetime(df['date']) # Convert date to datetime format
# Remove outliers based on modal_price (e.g., beyond 3 standard deviations)
def remove_outliers(data, column):
mean = data[column].mean()
std_dev = data[column].std()
return data[(data[column] >= mean - 3 * std_dev) & (data[column] <= mean + 3 *
std_dev)]
df = remove_outliers(df, 'modal_price')
# Create additional time-based features
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day
df['day_of_week'] = df['date'].dt.dayofweek
df['year'] = df['date'].dt.year
# Create the Dash app
```

```
app = dash.Dash(__name__)
# Connect to MySQL database
conn = mysql.connector.connect(
host='localhost', # Update with your MySQL host
user='root', # Update with your MySQL username
password='', # Update with your MySQL password
database='commodity_data' # Database name
)
cursor = conn.cursor()
# Create tables to store forecast results for Prophet and Random Forest
create_table_prophet = """
CREATE TABLE IF NOT EXISTS forecast_results_prophet (
id INT AUTO_INCREMENT PRIMARY KEY,
forecast_date DATE,
forecast_price FLOAT,
state_name VARCHAR(255),
district_name VARCHAR(255),
market_name VARCHAR(255),
commodity_name VARCHAR(255)
);
"""
create_table_rf = """
CREATE TABLE IF NOT EXISTS forecast_results_rf (
id INT AUTO_INCREMENT PRIMARY KEY,
```

```
forecast_date DATE,
forecast_price FLOAT,
state_name VARCHAR(255),
district_name VARCHAR(255),
market_name VARCHAR(255),
commodity_name VARCHAR(255)
);
"""
cursor.execute(create_table_prophet)
cursor.execute(create_table_rf)
conn.commit()
# Layout for the dashboard
app.layout = html.Div([
html.H1('Commodity Price Forecast Dashboard', style={'textAlign': 'center',
'padding': '20px'}),
html.Div([
html.Div([
html.Label('Select State:'),
dcc.Dropdown(
id='state-dropdown',
options=[{'label': state, 'value': state} for state in df['state_name'].unique()],
value=df['state_name'].unique()[0]
),
], style={'width': '24%', 'display': 'inline-block', 'padding': '10px'}),
```

```
html.Div([
html.Label('Select District:'),
dcc.Dropdown(id='district-dropdown'),
], style={'width': '24%', 'display': 'inline-block', 'padding': '10px'}),
html.Div([
html.Label('Select Market:'),
dcc.Dropdown(id='market-dropdown'),
], style={'width': '24%', 'display': 'inline-block', 'padding': '10px'}),
html.Div([
html.Label('Select Commodity:'),
dcc.Dropdown(id='commodity-dropdown'),
], style={'width': '24%', 'display': 'inline-block', 'padding': '10px'}),
]),
html.Div([
html.Div([
html.Label('Enter Forecast Period (Days):'),
dcc.Input(
id='forecast-period-input',
type='number',
value=30,
min=1,
step=1,
placeholder='Enter the number of days to forecast'
),
```

```
], style={'width': '48%', 'display': 'inline-block', 'padding': '10px'}),
html.Div([
html.Label('Select Model:'),
dcc.RadioItems(
id='model-selection',
options=[
{'label': 'Prophet', 'value': 'prophet'},
{'label': 'Random Forest', 'value': 'random_forest'}
],
value='prophet',
inline=True
),
], style={'width': '48%', 'display': 'inline-block', 'padding': '10px'}),
]),
dcc.Graph(id='historical-price-graph', style={'margin-top': '30px'}),
dcc.Graph(id='price-forecast-graph', style={'margin-top': '30px'}),
html.Div(id='forecast-metrics', style={'padding': '20px'}),
dcc.Graph(id='heatmap', style={'margin-top': '30px'})
])
# Update district dropdown based on selected state
@app.callback(
Output('district-dropdown', 'options'),
Input('state-dropdown', 'value')
)
```

```
def update_district_dropdown(selected_state):
districts = df[df['state_name'] == selected_state]['district_name'].unique()
return [{'label': district, 'value': district} for district in districts]
# Update market dropdown based on selected district
@app.callback(
Output('market-dropdown', 'options'),
Input('district-dropdown', 'value')
)
def update_market_dropdown(selected_district):
markets = df[df['district_name'] == selected_district]['market_name'].unique()
return [{'label': market, 'value': market} for market in markets]
# Update commodity dropdown based on selected market
@app.callback(
Output('commodity-dropdown', 'options'),
Input('market-dropdown', 'value')
)
def update_commodity_dropdown(selected_market):
commodities = df[df['market_name'] ==
selected_market]['commodity_name'].unique()
return [{'label': commodity, 'value': commodity} for commodity in commodities]
# Plot historical price trends
@app.callback(
Output('historical-price-graph', 'figure'),
[Input('state-dropdown', 'value'),
```

```
Input('district-dropdown', 'value'),

Input('market-dropdown', 'value'),

Input('commodity-dropdown', 'value')]

)

def plot_historical_prices(selected_state, selected_district, selected_market,
selected_commodity):

if None in [selected_state, selected_district, selected_market,
selected_commodity]:

return px.line(title='Select all filters to view historical prices')

# Filter data

filtered_df = df[(df['state_name'] == selected_state) &

(df['district_name'] == selected_district) &

(df['market_name'] == selected_market) &

(df['commodity_name'] == selected_commodity)]

fig = px.line(

filtered_df, x='date', y='modal_price',

title=f'Historical Prices for {selected_commodity} in {selected_market},
{selected_district}, {selected_state}',

labels={'date': 'Date', 'modal_price': 'Modal Price'}

)

return fig

from sklearn.metrics import mean_squared_error, r2_score

import math

@app.callback(

[Output('price-forecast-graph', 'figure'), Output('forecast-metrics', 'children')],
```

```
[Input('state-dropdown', 'value'),

Input('district-dropdown', 'value'),

Input('market-dropdown', 'value'),

Input('commodity-dropdown', 'value'),

Input('forecast-period-input', 'value'),

Input('model-selection', 'value')]

)

def update_graph(selected_state, selected_district, selected_market,
selected_commodity, forecast_period, selected_model):

if None in [selected_state, selected_district, selected_market,
selected_commodity, forecast_period]:

return px.line(title='Select all filters and enter a forecast period to view
forecast'), ''

# Filter the data

filtered_df = df[(df['state_name'] == selected_state) &

(df['district_name'] == selected_district) &

(df['market_name'] == selected_market) &

(df['commodity_name'] == selected_commodity)]

if selected_model == 'prophet':

# Prepare the data for Prophet

prophet_df = filtered_df[['date', 'modal_price']].rename(columns={'date': 'ds',
'modal_price': 'y'})

if len(prophet_df) < 30: # Check if there is enough data to train the model

return px.line(title='Not enough data to create a forecast'), ''

# Fit the Prophet model
```

```
model = Prophet(seasonality_mode='multiplicative', yearly_seasonality=True)

model.add_seasonality(name='monthly', period=30.5, fourier_order=3)

model.fit(prophet_df)

# Make future predictions for the user-defined forecast period

future = model.make_future_dataframe(periods=forecast_period,
include_history=False)

forecast = model.predict(future)

# Store forecast data in the Prophet MySQL table

forecast_data = forecast[['ds', 'yhat']].rename(columns={'ds': 'forecast_date',
'yhat': 'forecast_price'})

for index, row in forecast_data.iterrows():

cursor.execute("""

INSERT INTO forecast_results_prophet (forecast_date, forecast_price,
state_name, district_name, market_name, commodity_name)

VALUES (%s, %s, %s, %s, %s, %s)

""", (row['forecast_date'], row['forecast_price'], selected_state,
selected_district, selected_market, selected_commodity))

conn.commit()

# Plot forecast

fig = px.line(

forecast, x='ds', y='yhat',

title=f'Price Forecast for {selected_commodity} in {selected_market},
{selected_district}, {selected_state}',

labels={'ds': 'Date', 'yhat': 'Forecast Price'}

)

return fig, ''
```

```
elif selected_model == 'random_forest':

# Prepare the data for Random Forest

rf_df = filtered_df.copy()

rf_df['days_from_start'] = (rf_df['date'] - rf_df['date'].min()).dt.days

X = rf_df[['days_from_start', 'month', 'day', 'day_of_week', 'year']]

y = rf_df['modal_price']

if len(X) < 30: # Ensure enough data for training

return px.line(title='Not enough data to create a forecast'), ''

# Train-test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train the Random Forest Regressor

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)

# Generate predictions for the forecast period

last_date = rf_df['date'].max()

future_days = pd.date_range(start=last_date + pd.Timedelta(days=1),
periods=forecast_period)

future_features = pd.DataFrame({

'days_from_start': (future_days - rf_df['date'].min()).days,

'month': future_days.month,

'day': future_days.day,

'day_of_week': future_days.dayofweek,

'year': future_days.year
```

```
})

forecast_prices = rf_model.predict(future_features)

forecast_df = pd.DataFrame({'forecast_date': future_days, 'forecast_price':
forecast_prices})

# Store forecast data in the Random Forest MySQL table

for index, row in forecast_df.iterrows():

cursor.execute("""

INSERT INTO forecast_results_rf (forecast_date, forecast_price, state_name,
district_name, market_name, commodity_name)

VALUES (%s, %s, %s, %s, %s, %s)

""", (row['forecast_date'], row['forecast_price'], selected_state,
selected_district, selected_market, selected_commodity))

conn.commit()

# Calculate evaluation metrics

y_pred = rf_model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = math.sqrt(mse)

r2 = r2_score(y_test, y_pred)

# Plot forecast

fig = px.line(

forecast_df, x='forecast_date', y='forecast_price',

title=f'Price Forecast for {selected_commodity} in {selected_market},
{selected_district}, {selected_state}',

labels={'forecast_date': 'Date', 'forecast_price': 'Forecast Price'}
```

```
)

# Display evaluation metrics

metrics = html.Div([

html.H4('Forecast Evaluation Metrics:'),

html.P(f"Mean Absolute Error (MAE): {mae:.2f}"),

html.P(f"Mean Squared Error (MSE): {mse:.2f}"),

html.P(f"Root Mean Squared Error (RMSE): {rmse:.2f}"),

html.P(f"R-squared (R²): {r2:.2f}")

])

return fig, metrics

# Heatmap visualization of average prices by year and month

@app.callback(

Output('heatmap', 'figure'),

[Input('state-dropdown', 'value'),

Input('district-dropdown', 'value'),

Input('market-dropdown', 'value'),

Input('commodity-dropdown', 'value')]

)

def update_heatmap(selected_state, selected_district, selected_market,
selected_commodity):

if None in [selected_state, selected_district, selected_market,
selected_commodity]:

return px.imshow([[0]], labels={'x': 'Month', 'y': 'Year', 'color': 'Price'})

filtered_df = df[(df['state_name'] == selected_state) &
```

(df['district_name'] == selected_district) &

(df['market_name'] == selected_market) &

(df['commodity_name'] == selected_commodity)]

heatmap_data = filtered_df.groupby(['year',
'month'])['modal_price'].mean().unstack()

fig = px.imshow(

heatmap_data,

labels={'x': 'Month', 'y': 'Year', 'color': 'Average Price'},

title=f'Average Monthly Prices for {selected_commodity} in {selected_market},
{selected_district}, {selected_state}'

)

return fig

Run the Dash app

if __name__ == '__main__':

app.run_server(debug=True)

Preprocessing.ipynb:

import pandas as pd

import os

def preprocess_csv(file_path, output_path):

Load the data

df = pd.read_csv(file_path)

Remove unnecessary columns (adjust based on your data)

unwanted_columns = ['id', 'state_id', 'census_state_id', 'census_state_name',

'census_district_id', 'census_district_name', 'district_id', 'grade']

```
df = df.drop(columns=[col for col in unwanted_columns if col in df.columns], errors='ignore')

# Drop rows with missing essential values (e.g., `grade`, `min_price`, etc.)

df = df.dropna(subset=['min_price', 'max_price', 'modal_price', 'date'])

# Convert `date` to a datetime object

df['date'] = pd.to_datetime(df['date'], errors='coerce')

df = df.dropna(subset=['date']) # Drop rows with invalid `date` values

# Feature engineering (optional):

# Add `price_range` column if both `min_price` and `max_price` are available

if 'min_price' in df.columns and 'max_price' in df.columns:

df['price_range'] = df['max_price'] - df['min_price']

# Save the cleaned data to a new CSV

os.makedirs(output_path, exist_ok=True) # Create output directory if it doesn't exist

cleaned_file_path = os.path.join(output_path, os.path.basename(file_path).replace(".csv", "_cleaned.csv"))

df.to_csv(cleaned_file_path, index=False)

print(f"Cleaned file saved at: {cleaned_file_path}")

# List of input CSV file paths (replace with actual file paths)

input_files = [

"data (1).csv", # Replace with the path to your 2020 data file

"data (2).csv", # Replace with the path to your 2021 data file

"data (3).csv", # Replace with the path to your 2022 data file

"data (4).csv", # Replace with the path to your 2023 data file

]
```

```
# Output directory for cleaned files

output_directory = "cleaned_data"

# Preprocess each file

for file_path in input_files:

preprocess_csv(file_path, output_directory)
```

combined.ipynb:

```
import pandas as pd

# List of CSV file paths

csv_files = ["cleaned_data\data_2020.csv", "cleaned_data\data_2021.csv",
"cleaned_data\data_2022.csv", "cleaned_data\data_2023.csv"]

# Read and combine all CSV files

combined_df = pd.concat([pd.read_csv(file) for file in csv_files],
ignore_index=True)

# Save the combined DataFrame to a new CSV file

combined_df.to_csv("final_data.csv", index=False)

print("CSV files combined successfully into 'final_data.csv'")
```

MySQL Connection.ipynb:

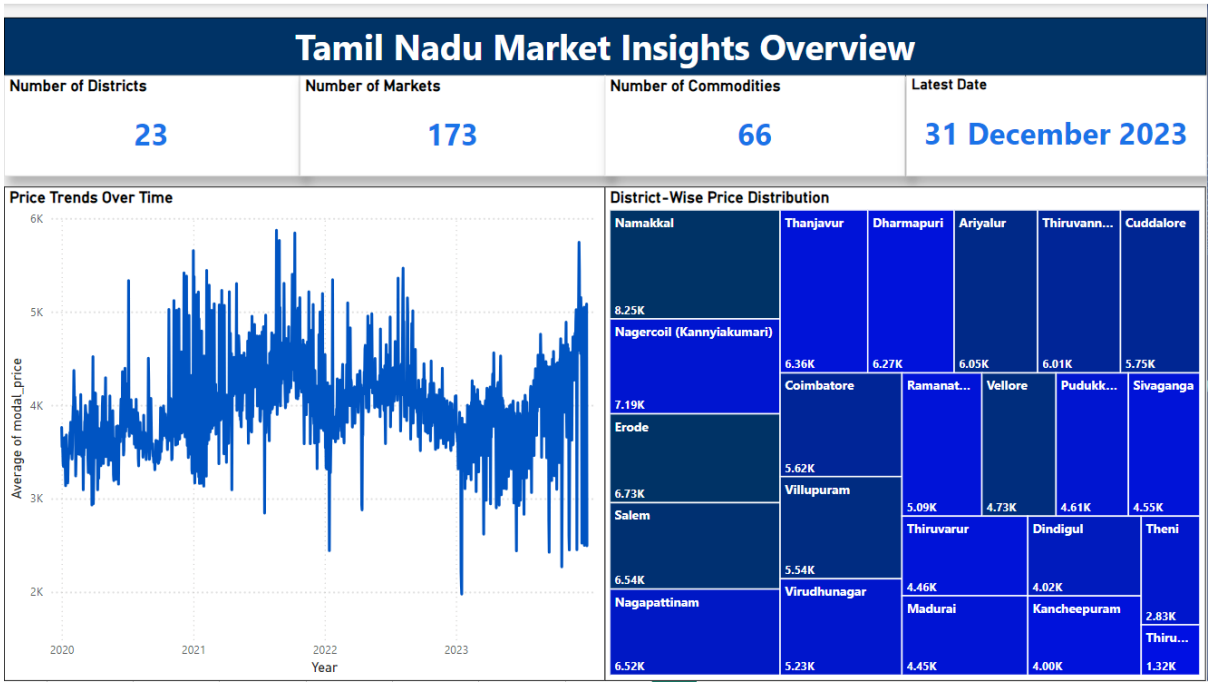
```
import mysql.connector

import pandas as pd

# MySQL connection setup

db_config = {

'host': 'localhost',

'user': 'root', # Replace with your MySQL username

'password': '', # Replace with your MySQL password
```

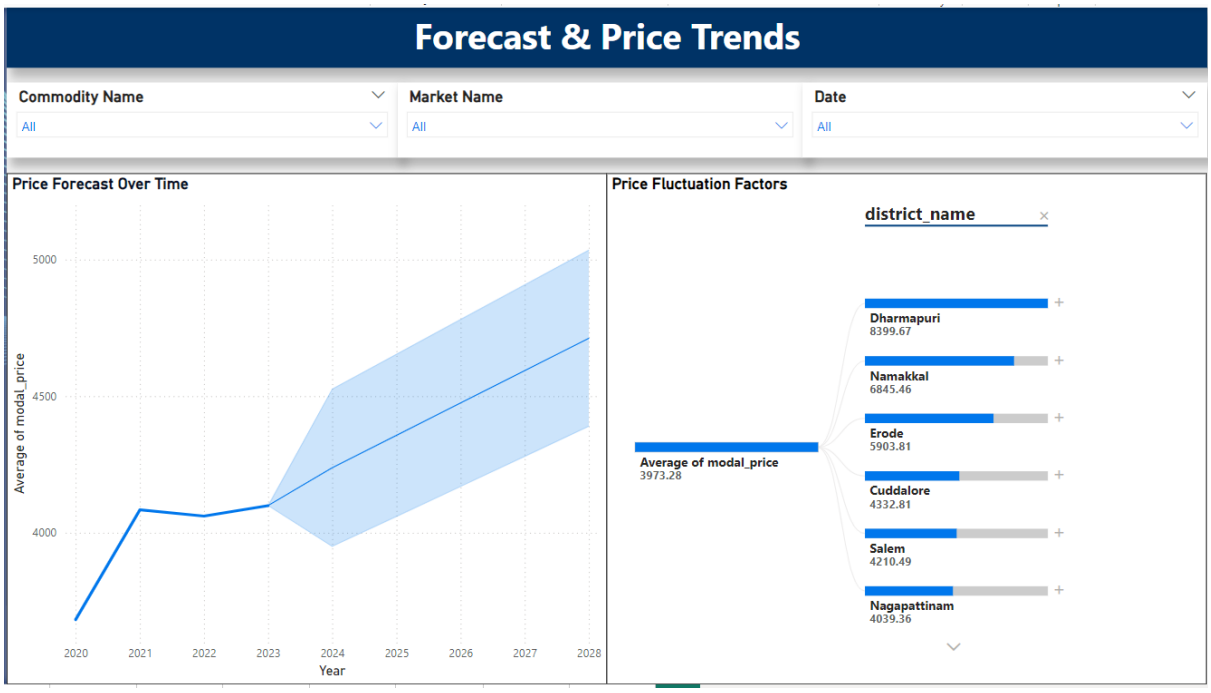
```
'database': 'commodity_data'
}
# Connect to the MySQL database
conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()
# Step 1: Create the market_prices table
create_table_query = """
CREATE TABLE IF NOT EXISTS market_prices (
id INT AUTO_INCREMENT PRIMARY KEY,
state_name VARCHAR(255),
district_name VARCHAR(255),
market_id INT,
market_name VARCHAR(255),
commodity_id INT,
commodity_name VARCHAR(255),
variety VARCHAR(255),
min_price FLOAT,
max_price FLOAT,
modal_price FLOAT,
date DATE,
price_range FLOAT
);
"""
cursor.execute(create_table_query)
```

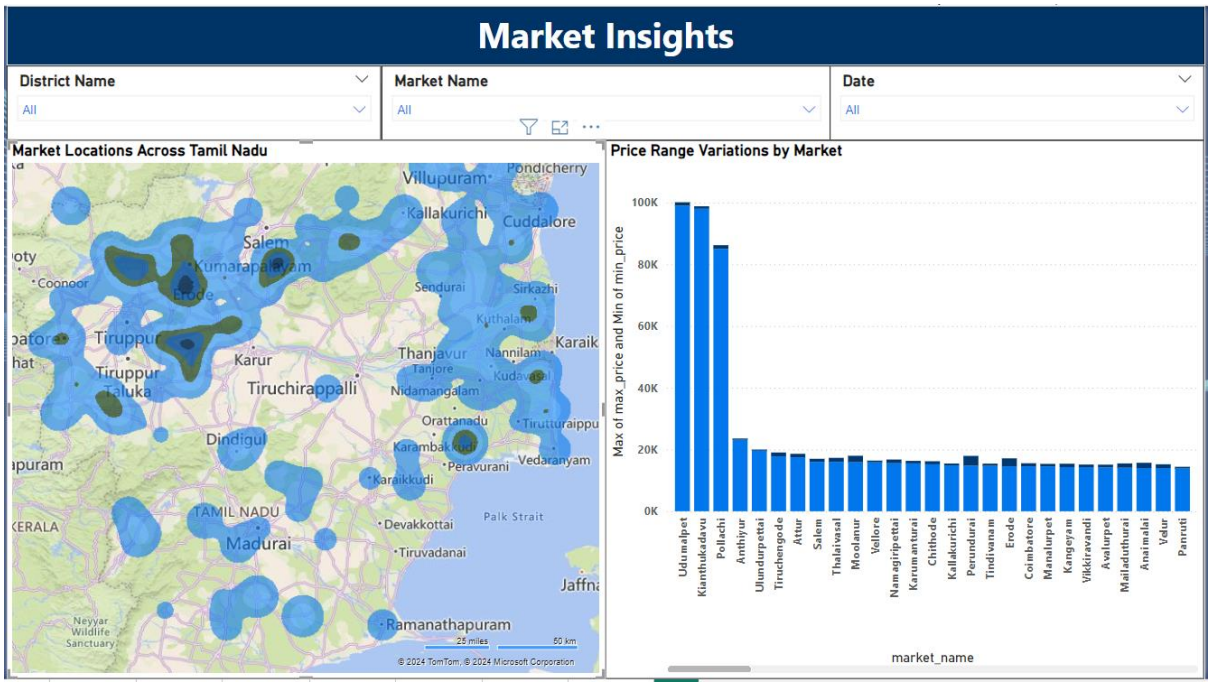
```
print("Table created successfully!")

# Step 2: Load CSV data and insert into the table in batches

csv_file = "final_data.csv" # Replace with the path to your CSV file

batch_size = 1000 # Adjust batch size as needed

# Load the data into a DataFrame

data = pd.read_csv(csv_file)

# Replace NaN values with NULL for SQL compatibility

data = data.where(pd.notnull(data), None)

# Insert data into the table in batches

insert_query = """

INSERT INTO market_prices (

state_name, district_name, market_id, market_name, commodity_id,

commodity_name, variety, min_price, max_price, modal_price, date,
price_range

) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

"""

# Divide the data into batches

for i in range(0, len(data), batch_size):

batch_data = data.iloc[i:i + batch_size]

# Convert DataFrame rows to tuples

records = list(batch_data.itertuples(index=False, name=None))

cursor.executemany(insert_query, records)

conn.commit()

print(f"Inserted batch {i // batch_size + 1} successfully!")
```

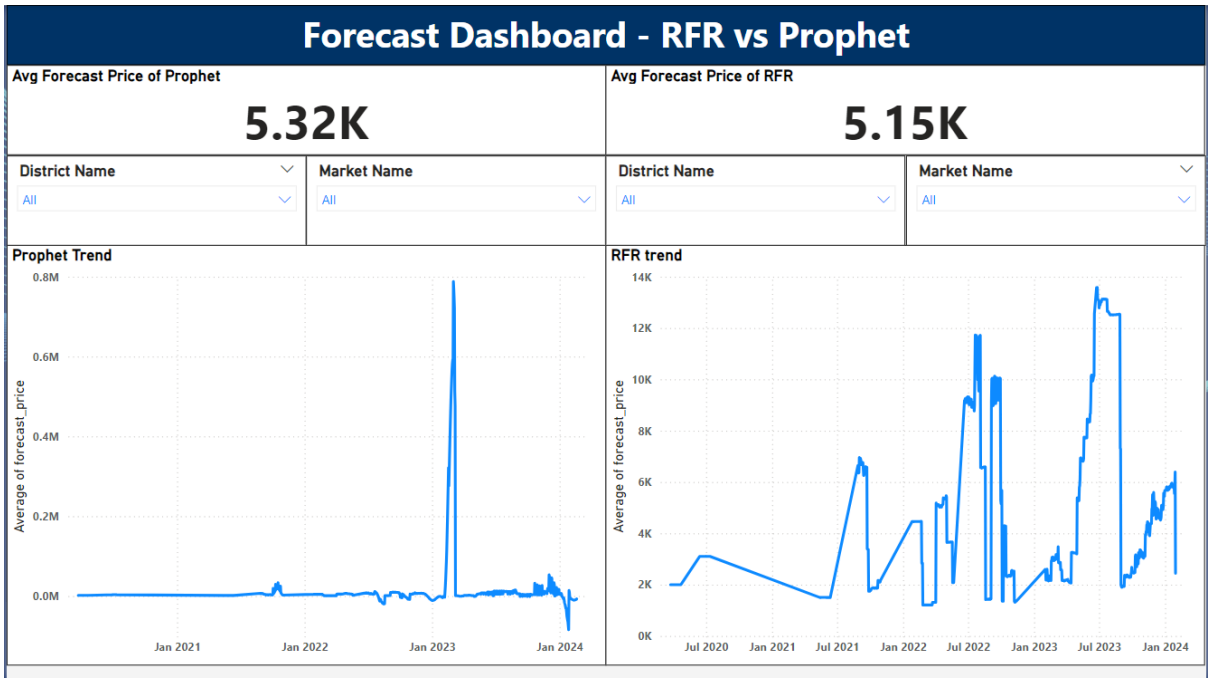
```
# Close the cursor and connection

cursor.close()

conn.close()

print("Data insertion completed and connection closed.")
```

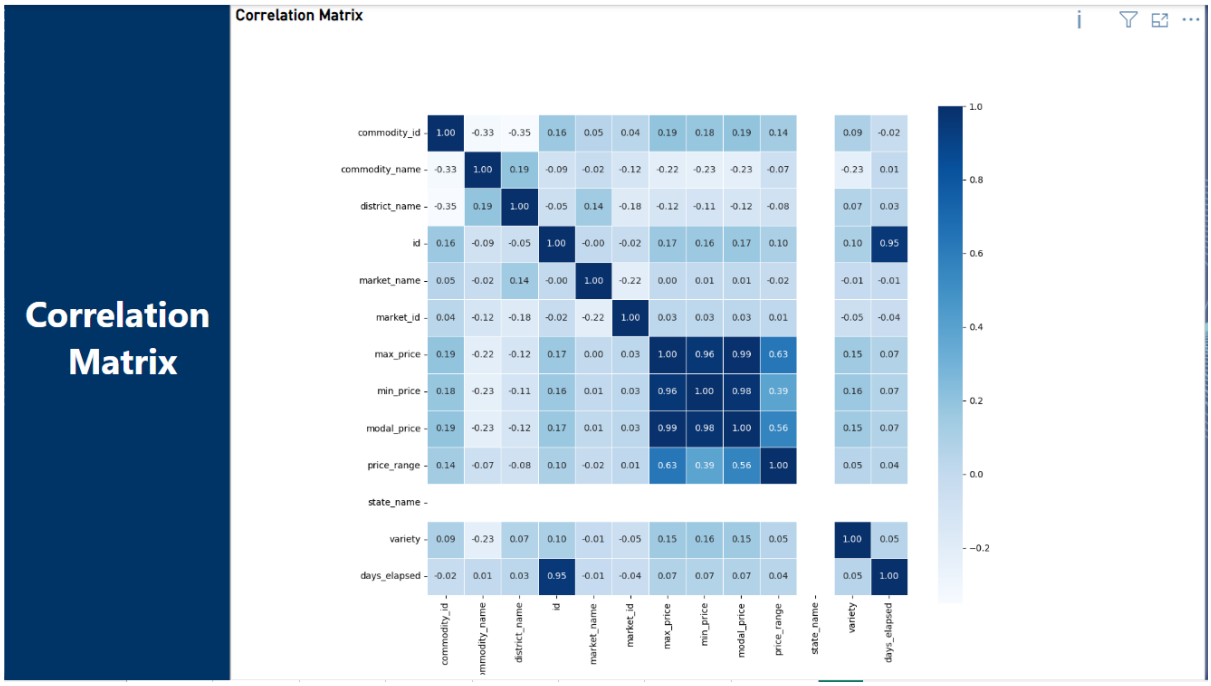
Power BI Dashboard Screenshots:



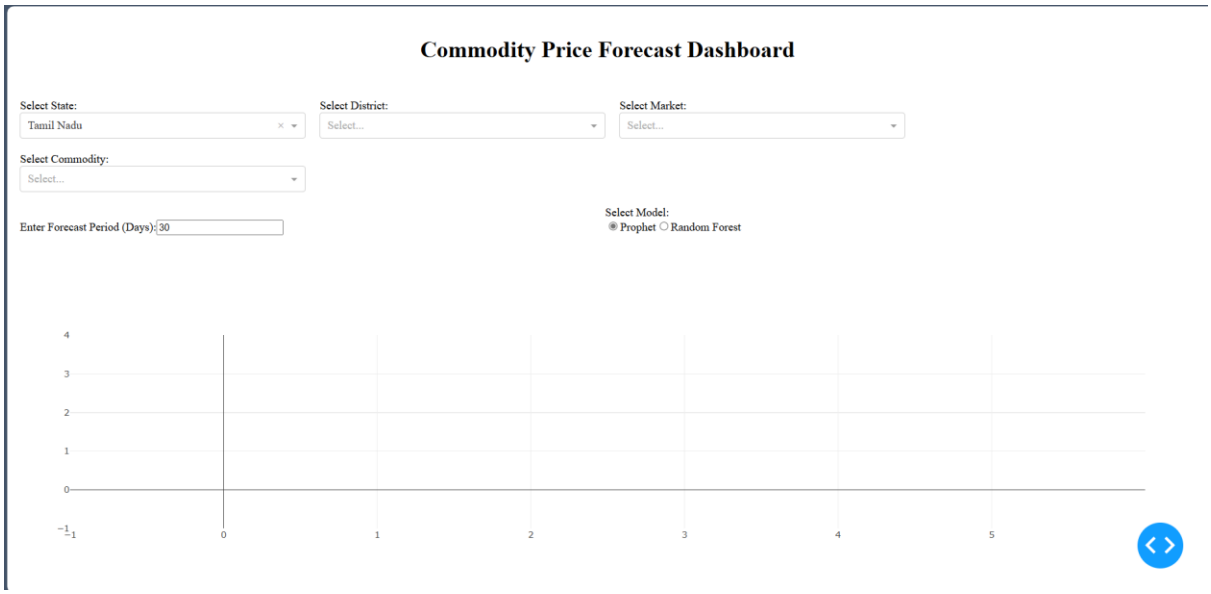


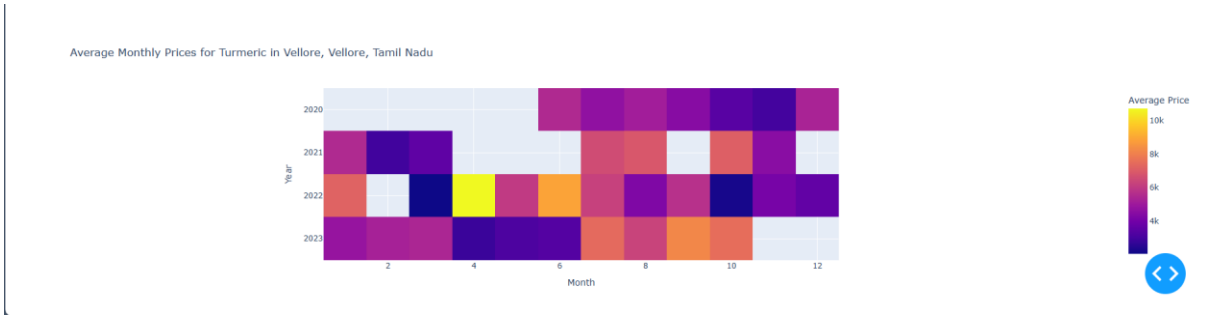

Market Insights
District Name
All
Market Name
All
Date
All
Market Locations Across Tamil Nadu
Villupuram
Pondicherry
Kallakurichi
Cuddalore
Salem
Kumarapalayam
Coonoor
Erode
Sendurai
Sirkazhi
Tiruppur
Karur
Thanjavur
Tanjore
Nannilam
Karaik
Tiruppur Taluka
Tiruchirappalli
Nidamangalam
Kudavasal
Orattanadu
Tirutturaippu
Dindigul
Karambakkudi
Vedaranyam
Peravurani
Karaikkudi
TAMIL NADU
Palk Strait
Devakkottai
Madurai
Tiruvadanai
Ramanathapuram
Neyyar Wildlife Sanctuary
25 miles
50 km
© 2024 TomTom, © 2024 Microsoft Corporation
Price Range Variations by Market
Max of max_price and Min of min_price
100K
80K
60K
40K
20K
0K
Udumalpet
Kianthukadavu
Pollachi
Anthiyur
Ulundurpettai
Tiruchengode
Attur
Salem
Thalaivasal
Moolanur
Vellore
Namagiripettai
Karumanturai
Chithode
Kallakurichi
Perundurai
Tindivanam
Erode
Coimbatore
Manalurpet
Kangeyam
Vikkiravandi
Avalurpet
Mailaduthurai
Anaimalai
Velur
Panruti
market_name


Forecast Dashboard - RFR vs Prophet
Avg Forecast Price of Prophet
5.32K
Avg Forecast Price of RFR
5.15K
District Name
All
Market Name
All
District Name
All
Market Name
All
Prophet Trend
Average of forecast_price
0.8M
0.6M
0.4M
0.2M
0.0M
Jan 2021
Jan 2022
Jan 2023
Jan 2024
RFR trend
Average of forecast_price
14K
12K
10K
8K
6K
4K
2K
0K
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 2024


Correlation Matrix
Correlation Matrix
commodity_id
commodity_name
district_name
id
market_name
market_id
max_price
min_price
modal_price
price_range
state_name
variety
days_elapsed

Dash App:


Commodity Price Forecast Dashboard
Select State:
Tamil Nadu
Select District:
Select...
Select Market:
Select...
Select Commodity:
Select...
Enter Forecast Period (Days):
30
Select Model:
Prophet
Random Forest



Forecast Evaluation Metrics:

Mean Absolute Error (MAE): 2095.04

Mean Squared Error (MSE): 7224849.40

Root Mean Squared Error (RMSE): 2687.91

R-squared (R^2): -0.30

Conclusion:

This project successfully demonstrates the application of advanced machine learning techniques and interactive visualization tools to predict and analyze food commodity prices in Tamil Nadu. By leveraging historical data and employing models like Prophet and Random Forest Regressor (RFR), the system achieves accurate and reliable price forecasts. The integration of real-time analytics through the Dash web application and interactive visualizations using Power BI ensures that stakeholders have access to actionable insights for informed decision-making.

The data preprocessing and integration steps, including column removal, null value handling, and feature engineering, were instrumental in enhancing the quality of the dataset and ensuring robust model performance. By combining historical trends with predictive analytics, the project addresses critical challenges in agricultural markets, such as price volatility and resource planning.

The developed system has significant implications for farmers, traders, and policymakers, enabling them to monitor market trends, optimize operations, and enhance economic stability. This real-time forecasting and visualization platform serves as a powerful tool for ensuring transparency, improving decision-making, and fostering sustainable agricultural practices in Tamil Nadu.

Future work can focus on incorporating additional external factors, such as weather conditions and global market trends, to further improve the accuracy and scope of the forecasts. Moreover, expanding the system to other regions and commodities can enhance its impact and applicability on a larger scale.