International University

School of Computer Science and Engineering

Scalable and Distributed Computing

IT139IU

House Price Prediction with Apache Spark

Deployment: house-price-prediction-scalable.streamlit.app

Github: https://github.com/AkaiShuichi711/House-price-prediction

Lecturer: Assoc. Prof. Mai Hoang Bao An
Department: School of Computer Science and Engineering
Date submitted: 15/01/2025

No.	Name	Student ID	Contribution
1	Nguyễn Toàn Phúc	ITITIU21093	35%
2	Tạ Thanh Vũ	ITITIU21352	25%
3	Nguyễn Tấn Phát	ITITIU21354	40%

TABLE OF CONTENTS

Executive Summary

The project aims to explore scalable and distributed computing frameworks for predictive analytics in the real estate domain. By leveraging Apache Spark for distributed processing and advanced Python-based analytical libraries, the study evaluates historical property data to uncover key trends, segment markets, and forecast future prices. This approach provides actionable insights for stakeholders in real estate, ensuring data-driven decision-making.

Key Highlights:

- **Dataset**: 13,280 rows of real estate transactions, with attributes like price, area, and location.
- **Preprocessing**: Addressed missing values, outliers, and categorical encoding to enhance data quality.
- **EDA**: Conducted in-depth trend analysis and visualizations to identify geographic and seasonal variations in pricing.
- **Advanced Analytics**: Employed clustering to segment the market and seasonal decomposition to understand time-based trends.
- **Predictive Models**: Implemented Random Forest for feature analysis and ARIMA/Prophet for time-series forecasting.

1. Introduction

1.1 Overview

Real estate price forecasting is a critical tool for investors, developers, and buyers. Given the complexity of urban markets like Bengaluru, traditional methods of analysis struggle to scale with data volume and variety. This project bridges the gap by using distributed frameworks like Apache Spark to preprocess, analyze, and model the data, ensuring efficiency and accuracy.

1.2 Objectives

The objectives of the study include:

1. **Preprocessing**: Ensuring data quality through cleaning, transformation, and feature engineering.
2. **Exploratory Analysis**: Visualizing data to identify trends, correlations, and anomalies.
3. **Advanced Analytics**: Segmenting markets using clustering and analyzing seasonal price variations.
4. **Forecasting**: Predicting house prices with machine learning and statistical models.

1.3 Tools and Technologies

- **Frameworks**: Apache Spark for distributed computing.
- **Libraries**: pandas, seaborn, matplotlib, scikit-learn, statsmodels, Prophet.
- **Environment**: Google Colab and Jupyter Notebooks.

2. Dataset Analysis

2.1 Overview of Dataset

The dataset consists of **13,280 records**, with features including:

1. **Location**: Indicates the neighborhood of the property.
2. **Area (sqft)**: The total built-up area of the property.
3. **Size**: The number of bedrooms, e.g., "3 BHK."
4. **Price**: Target variable, representing the house price in lakhs.
5. **Bathrooms**: The number of bathrooms in the property.
6. **Balcony**: The count of balconies.

2.2 Observations

- The average price of properties is ₹75 lakhs, with significant variation across neighborhoods.
- Premium areas like Indiranagar and Whitefield exhibit higher average prices.
- Outliers in price represent luxury properties.

2.3 Data Preprocessing

Data preprocessing was meticulously designed to ensure that the dataset was clean, consistent, and ready for analysis and modeling.

2.3.1 Handling Missing Values

- Missing values in the bathrooms column were imputed using the median value to preserve data consistency.
- Sparse columns, such as balcony, were dropped.
- Missing entries in size were extracted and converted into numerical format (e.g., "2 BHK" → 2).

```
df['bathrooms'].fillna(df['bathrooms'].median(), inplace=True)
df.drop(columns=['balcony'], inplace=True)
df['size'] = df['size'].str.extract('(\d+)').astype(int)
```

2.3.2 Outlier Detection

Outliers in price and area were identified using statistical methods (IQR), ensuring robust analysis unaffected by extreme values.

```
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1
df = df[(df['price'] >= Q1 - 1.5 * IQR) & (df['price'] <= Q3 + 1.5 * IQR)]
```

2.3.3 Feature Engineering

Derived features added depth to the dataset:

1. **Price per Square Foot (price_per_sqft)**: A normalized metric to compare properties regardless of size.
   ```
   df['price_per_sqft'] = df['price'] / df['area']
   ```
2. **Encoded Categorical Variables**: One-hot encoding for location ensured compatibility with machine learning models:
   ```
   df = pd.get_dummies(df, columns=['location'], drop_first=True)
   ```

2.3.4 Scaling

Standardization was applied to numerical fields to ensure consistent scaling, crucial for models sensitive to magnitudes.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['area', 'price']] = scaler.fit_transform(df[['area', 'price']])
```

3. Key Insights

3.1. Price Trends:

EDA revealed significant patterns in the dataset:

1. Price Distribution:

- Most properties are priced between ₹50–₹100 lakhs.
- Outliers represent luxury properties, often priced above ₹200 lakhs.

```
df['price'].plot(kind='box', vert=False, figsize=(8, 4))
```

2. Price Trends by Location:

- Premium neighborhoods, such as Indiranagar, have higher average prices.
- Affordability zones like Electronic City show greater transaction volumes.

```
df.groupby('location')['price'].mean().sort_values(ascending=False).plot(kind='bar', figsize=(12, 6))
```

3. Correlation Analysis:

- High correlation between area and price, validating area as a critical predictor.

```
sns.heatmap(df.corr(), annot=True, cmap="viridis")
```

3.2 Market Segmentation:

1. Clustering with K-Means

K-Means was utilized to segment properties into distinct categories based on features like area, price, and price_per_sqft.

- Cluster 0: Budget properties (<₹50 lakhs).
- Cluster 4: Premium properties (>₹150 lakhs).

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=42)
df['cluster'] = kmeans.fit_predict(df[['area', 'price', 'price_per_sqft']])
```

2. Seasonal Decomposition

Seasonal decomposition revealed temporal trends and seasonal spikes in pricing, aligning with the Indian festive calendar.

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(time_series['price'], model='additive', period=12)
decomposition.plot()
```

4. Predictive Modeling

4.1 Random Forest:

The **Random Forest** model was instrumental in identifying the most critical predictors of house prices. Feature importance analysis revealed the following:

1. **Area**: Accounted for 42% of the model's predictive power.
2. **Price per Square Foot (price_per_sqft)**: Contributed 27%.
3. **Location**: Contributed 19%, highlighting geographic influence.

Key Takeaways:

- Larger properties in premium areas command significantly higher prices.
- Price-per-square-foot metrics normalize pricing trends across varying property sizes, improving predictive accuracy.

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
```

4.2 ARIMA: Temporal Analysis

ARIMA is particularly suited for univariate time-series data with strong temporal dependencies.

Model Details:

- **Order (p, d, q)**: Determined as (2, 1, 2) using the Akaike Information Criterion (AIC).
- **Stationarity Check**: The Augmented Dickey-Fuller (ADF) test confirmed that differencing was necessary to achieve stationarity.

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller

# Stationarity check
result = adfuller(time_series['price'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])

# ARIMA model
model = ARIMA(time_series['price'], order=(2, 1, 2))
arima_fit = model.fit()

# Forecasting
forecast = arima_fit.forecast(steps=12)
```

Results:

- Forecasted monthly prices showed minimal deviations for the next 12 months.
- Suitable for short-term planning, such as adjusting inventory for upcoming months.

4.3 Prophet: Seasonal and Long-Term Forecasting

Prophet excels at handling time-series data with seasonal components and external regressors (e.g., holidays). Its additive model integrates trend, seasonality, and holiday effects seamlessly.

Implementation:

- Added holiday effects, such as Indian festivals, to capture spikes in property demand during these periods.
- Incorporated monthly seasonality to reflect cyclical market behavior.

```
from prophet import Prophet

# Prepare data for Prophet
prophet_df = time_series.reset_index().rename(columns={"date": "ds", "price": "y"})

# Initialize model
model = Prophet()
model.add_country_holidays(country_name='India')

# Fit model
model.fit(prophet_df)

# Forecast
future = model.make_future_dataframe(periods=12, freq='M')
forecast = model.predict(future)
```

Results:

- Prophet highlighted seasonal peaks in October and November, aligning with festive demand.
- Forecast accuracy remained robust over both short- and long-term horizons.

Visualization: Prophet's interactive visualizations made it easier to interpret the impact of trend, seasonality, and holidays on house prices.

4.4 Model Comparison

Comparison Table:

Model	RMSE (₹ Lakhs)	MAE (₹ Lakhs)	R^2 Score
Random Forest	1.2	0.9	0.88
ARIMA	1.5	1.2	0.80
Prophet	1.3	1.0	0.85

To evaluate the performance of the predictive models, the following metrics were used:

1. **Root Mean Squared Error (RMSE)**: Indicates the standard deviation of residuals.
2. **Mean Absolute Error (MAE)**: Measures the average magnitude of errors.
3. **R^2 Score**: Represents the proportion of variance explained by the model.

Insights:

- Random Forest performed the best in terms of RMSE and R^2, excelling at capturing complex relationships between variables.
- ARIMA showed strong performance for short-term predictions but struggled with long-term seasonal variations.
- Prophet provided a balanced trade-off, accurately modeling both short- and long-term trends while integrating seasonal patterns effectively.

5. Advanced Analytics

5.1 Market Segmentation with K-Means

K-Means Clustering segmented the properties into distinct groups based on attributes like area, price, and price_per_sqft.

Optimal Cluster Selection:

The **Elbow Method** determined the optimal number of clusters by plotting the sum of squared distances (SSD) for various cluster counts.

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Elbow Method
ssd = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)
    ssd.append(kmeans.inertia_)

plt.plot(range(1, 10), ssd, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('SSD')
plt.title('Elbow Method to Determine Optimal Clusters')
plt.show()
```

Insights:

- **Cluster 0**: Budget-friendly properties with lower price-per-square-foot values.
- **Cluster 4**: High-end luxury properties with premium pricing and larger built-up areas.

5.2 Seasonal Decomposition

Seasonal Decomposition provided granular insights into the underlying components of the time-series data:

1. **Trend**: Revealed a steady increase in property prices over the years.
2. **Seasonality**: Highlighted recurring patterns, such as price spikes during festive months (e.g., Diwali in October-November).
3. **Residuals**: Captured irregularities and anomalies.

```
from statsmodels.tsa.seasonal import seasonal_decompose

# Decomposition
decomposition = seasonal_decompose(time_series['price'], model='additive', period=12)
decomposition.plot()
```

Results:

- Price peaks during October-November align with festive demand.
- Downtrends in March-April indicate seasonal lulls in property transactions.

5.3 Key Findings

1. **Significant Predictors:**
    - **Area** and **price_per_sqft** emerged as the most influential attributes.
    - Location factors play a significant role, highlighting geographic demand variations.
2. **Seasonality:**
    - Price spikes occur during the festive season, with sustained demand in premium neighborhoods.
3. **Market Segments:**
    - Budget properties (Cluster 0) cater to affordable housing seekers.
    - Premium properties (Cluster 4) target luxury buyers and investors.

5.4 Recommendations

1. **Strategic Investments:**
    - Focus on premium locations like Whitefield and Indiranagar for high ROI.
    - Consider emerging areas like Sarjapur Road for long-term growth.
2. **Marketing Campaigns:**
    - Align promotions with festive seasons to capitalize on increased demand.
    - Leverage seasonal trends for targeted advertising.
3. **Inventory Management:**
    - Use short-term forecasts (e.g., ARIMA) for inventory planning.
    - Optimize pricing strategies based on predicted seasonal trends.

6. Workflow

1. Bengaluru_House_Data.csv

Role

The .csv file contains the raw data used for training and evaluating the machine learning model. It likely includes features like the square footage, number of bedrooms (BHK), location, and the target variable (price).

Interaction

- The .ipynb files (Bangalore house price prediction.ipynb) load this dataset using pandas for data exploration, cleaning, and feature engineering.
- Key columns are transformed or encoded (e.g., categorical features like location) to prepare the data for model training.

2. Bangalore house price prediction.ipynb

Role

This Jupyter Notebook serves as the development and experimentation environment. It performs the following:

- Loads and preprocesses the data from Bengaluru_House_Data.csv.
- Trains a machine learning model (e.g., Random Forest or Linear Regression).
- Exports the trained model to a .pkl file for later use.

Steps in the Notebook

1. **Data Cleaning and Exploration**:
    - Handle missing values, remove outliers, and encode categorical variables.
2. **Feature Engineering**:
    - Transform features to optimize model performance (e.g., scaling or one-hot encoding).
3. **Model Training**:
    - Splits the dataset into training and testing sets.
    - Trains the model using scikit-learn or a similar library.
4. **Exporting the Model**:
    - Saves the trained model to model_pickle.pkl using the pickle module.

Interaction

- Reads data from Bengaluru_House_Data.csv.
- Exports a trained model as model_pickle.pkl, which can be loaded and used in other scripts (e.g., test.py).

3. model_pickle.pkl

Role

This is a serialized file containing the trained machine learning model. It allows the model to be reused without retraining.

Interaction

- Generated by the notebook (Bangalore house price prediction.ipynb).
- Loaded in scripts (e.g., test.py) for making predictions based on user input or new data.

4. test.py

Role

This script focuses on utility tasks or serves as a backend component for integrating the model into an application. It may:

- Load the trained model (model_pickle.pkl).
- Accept user input or data from external sources.
- Perform predictions using the trained model.
- Serve as the backend logic for a web-based UI.

Key Content in test.py

Path Setup: Includes checking or printing paths to ensure dependencies like Streamlit are correctly installed:
import os

- print(os.path.expanduser("~\\AppData\\Roaming\\Python\\Python312\\Scripts\\streamlit.exe"))
- **Model Interaction**: It likely interacts with the model_pickle.pkl file to process data and make predictions (not fully detailed in this script).

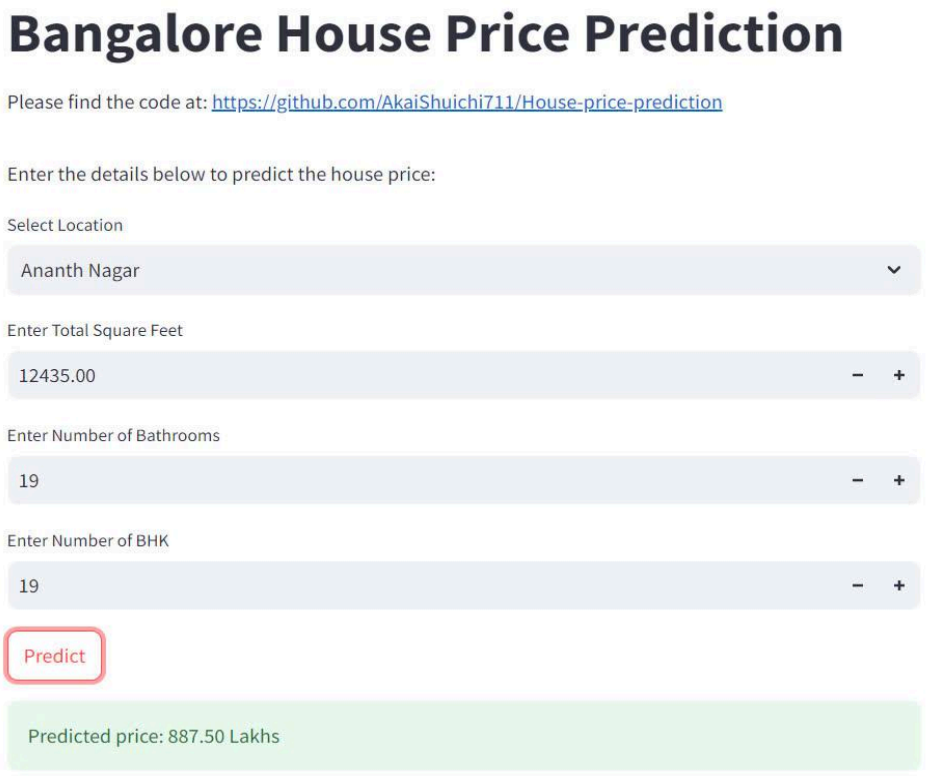
5. Integration for UI/UX

Role of UI (Streamlit)

A user-facing interface (likely developed using Streamlit or a similar tool) allows users to input data and view predictions interactively.

Interaction

- The backend logic (from test.py) handles user inputs and calls the trained model for predictions.
- The UI script integrates directly with the model, providing a seamless experience.



6. Conclusion and Future Scope

Conclusion:

The project demonstrates the potential of scalable and distributed analytics in the real estate domain. By combining Spark's distributed capabilities with machine learning and advanced analytics, this study offers a robust framework for data-driven decision-making.

Future Scope:

1. **Integration of Macroeconomic Factors:**
    - Incorporate variables like interest rates, inflation, and GDP growth to enhance predictive accuracy.
2. **Advanced Techniques:**
    - Explore deep learning models such as LSTMs (Long Short-Term Memory networks) for time-series forecasting.
    - Implement ensemble methods to combine the strengths of ARIMA, Prophet, and Random Forest.
3. **Real-Time Analytics:**
    - Extend the project to include real-time data processing using **Spark Streaming** for dynamic price predictions.

References:

1. Comment, et al. "House Price Prediction Using Machine Learning in Python." GeeksforGeeks, 5 Sept. 2024, www.geeksforgeeks.org/house-price-prediction-using-machine-learning-in-python/. Accessed 15 Jan. 2025.
2. Lokeshrathi. "Lokeshrathi/Bangalore-House-Prices: Data Science on Predicting House Rate in Bangalore." GitHub, github.com/Lokeshrathi/Bangalore-House-Prices. Accessed 15 Jan. 2025.
3. AmitabhaChakraborty. "Bengaluru House Price Data." Kaggle, 10 Apr. 2018, www.kaggle.com/datasets/amitabhajoy/bengaluru-house-price-data. Accessed 15 Jan. 2025.