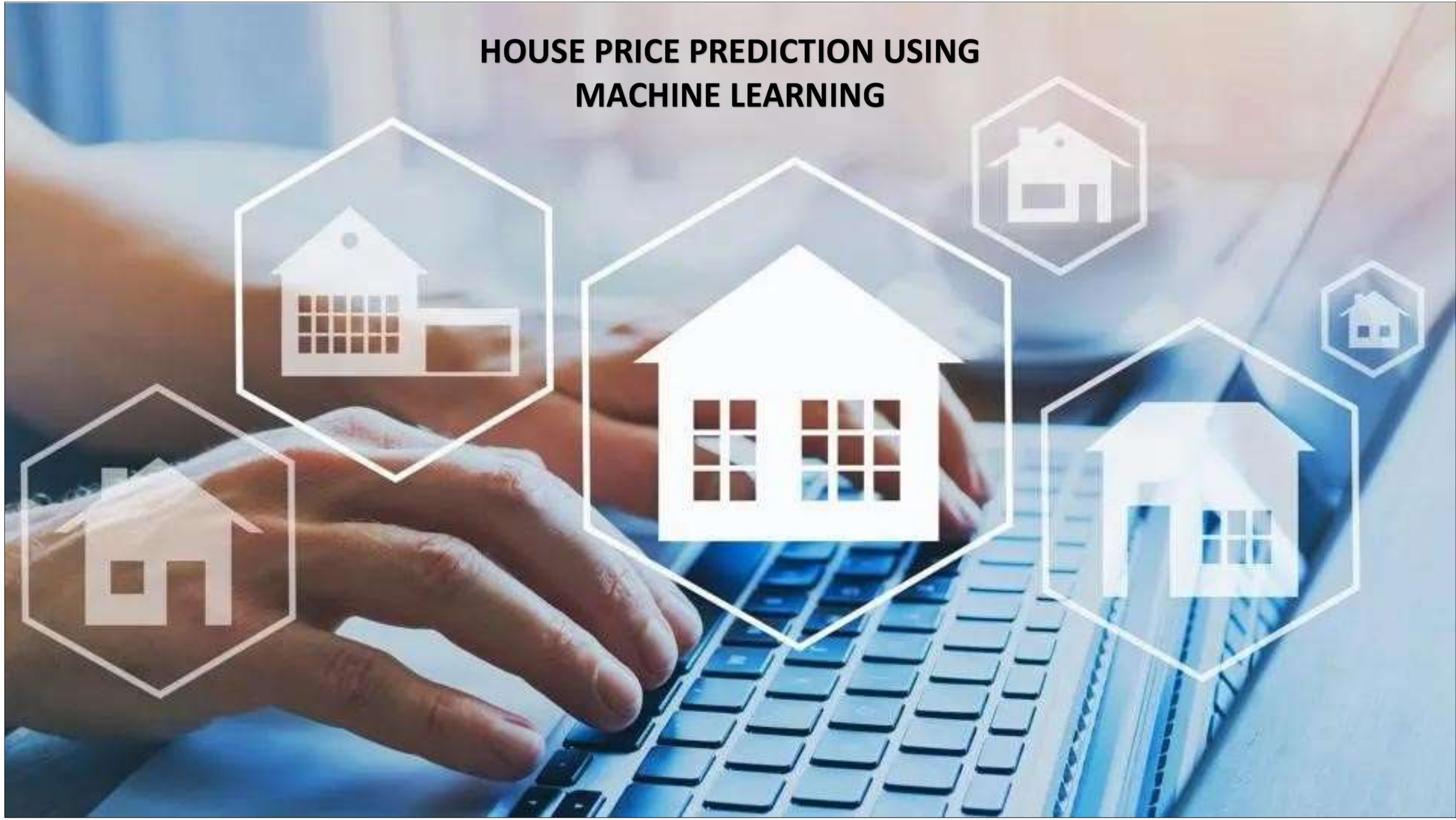
HOUSE PRICE PREDICTION USING MACHINE LEARNING



AI_PHASE5

Domain : Artificial Intelligence

Project 9 :Predicting House prices using Machine Learning

INTRODUCTION:

The house price prediction project aims to create a predictive model that estimates house prices based on various property attributes. This phase involves fundamental tasks, including data preprocessing, model selection and training, and model evaluation.

Problem Statement:

- Design and implement a machine learning model to predict house prices based on various features such as square footage, number of bedrooms, number of bathrooms, neighborhood, and other relevant factors. The goal is to create a robust and accurate prediction system that can assist potential buyers, sellers, and real estate professionals in estimating the market value of residential properties. The model should be able to handle both single-family homes and condominiums, providing reliable price estimates for a diverse range of properties.

Design thinking:

- Design thinking can be a valuable approach when developing a machine learning solution for house price prediction.
- It involves empathiizing,defining the problem,ideating the solutions,prototyping and testing.
- Phases of design thinking in house price prediction:

1. Empathize
2. Define
3. Ideate
4. Prototype
5. Test
6. Implement
7. Evaluate
8. Iterate
9. Launch
10. Scale

Empathize:

- Understand the needs and pain points of potential users, such as homebuyers, sellers, and real estate professionals.

- Conduct interviews, surveys, or user research to gather insights into what information and features they find most valuable in a house price prediction tool.

Define:

- Clearly define the problem statement and the goals of the project, considering both user needs and business objectives.

- Create user personas to represent the different types of users and their specific requirements.

Ideate:

- Brainstorm potential features and functionalities that would address the identified user needs.
- Encourage creative thinking to come up with innovative ways to present house price predictions and related information.

Prototype:

- Create mockups or wireframes of the user interface to visualize the design of the house price prediction tool.
- Develop a prototype of the machine learning model and integrate it into the user interface.

Test:

- Conduct usability testing with potential users to gather feedback on the prototype.
- Assess the effectiveness of the machine learning model in providing accurate and meaningful house price predictions.
- Iterate on the design and functionality based on user feedback and model performance.

Implement:

- Develop the full-fledged house price prediction tool based on the refined prototype.
- Integrate the machine learning model into the tool and ensure it can handle real-time predictions effectively.

Evaluate:

- Continuously monitor and evaluate the tool's performance, both in terms of user satisfaction and prediction accuracy.
- Gather feedback from users and make improvements as necessary.

Iterate:

- Use an iterative approach to make ongoing improvements to the tool based on user feedback, changing market conditions, and model performance.

- Be open to adapting the design and features as user needs evolve.

Launch:

- Launch the house price prediction tool to the target audience, whether it's through a web application, mobile app, or other channels.
- Promote the tool to reach a wider audience of potential users.

Scale:

- Consider opportunities for scaling the tool, such as expanding to cover different geographic regions or adding new features like property comparisons or investment analysis.

REGRESSION TECHNIQUES:

- Gradient Boosting is indeed an advanced regression technique that has proven to be highly effective for predictive modeling tasks. It's not just limited to regression but can also be used for classification and ranking problems.

- One of the most popular implementations of gradient boosting is XGBoost, although there are other libraries like LightGBM and CatBoost, which have gained popularity as well. These libraries are designed to optimize the gradient boosting algorithm for better performance and scalability.

Steps To Explore Gradient Boosting For Prediction Accuracy:

1. Understand the Problem
2. Data Preprocessing
3. Choose a Gradient Boosting Implementation
4. Feature Engineering
5. Select Hyperparameters
6. Train the Model
7. Evaluate Model Performance
8. Feature Importance
9. Fine-Tuning
10. Regularization and Overfitting
11. Interpretability
12. Deploy the Model
13. Monitor and Maintain

1.Understand the Problem:

- Clearly define the problem you want to solve, whether it's a classification or regression task.
- Identify the relevant features (input variables) and the target variable (output variable) in your dataset.

2. **Data Preprocessing:**

- Collect and prepare your data. This includes data cleaning, handling missing values, and dealing with outliers**.**
- Split your data into training and testing sets to evaluate model performance**.**

3.Choose a Gradient Boosting Implementation:

- There are various implementations of Gradient Boosting, including GradientBoostingClassifier and GradientBoostingRegressor in scikitlearn, XGBoost, LightGBM, and CatBoost. Choose the one that suits your needs.

4. **Feature Engineering:**

- Create relevant features or transformations if needed to enhance model performance.
- Encode categorical variables, scale numerical features, and apply feature selection techniques if necessary.

5.**Select Hyperparameters:**

- Gradient Boosting models have several hyperparameters that need to be tuned for optimal performance.
- Common hyperparameters include the learning rate, the number of trees (n_estimators), maximum depth of trees, and the loss function.
- Use techniques like cross-validation and grid search to find the best hyperparameters.

6.Train the Model:

- Fit the Gradient Boosting model on your training data using the selected hyperparameters**.**
- Monitor training progress by tracking metrics like log-loss for classification or mean squared error for regression.

7. **Evaluate Model Performance:**

- Use your testing data to evaluate the model's performance. Common evaluation metrics include accuracy, precision, recall, F1-score for classification, and mean squared error, R-squared for regression.
- Visualize the results through confusion matrices, ROC curves, or calibration plots for classification problems.

8.Feature Importance:

- Analyze feature importances to understand which features have the most impact on predictions. This can help you refine your model and potentially improve prediction accuracy.

9. Fine-Tuning:

- Refine the model by adjusting hyperparameters or trying different variations of Gradient Boosting models (e.g., LightGBM, XGBoost).

10. Regularization and Overfitting:

- Implement regularization techniques such as early stopping, reducing the learning rate, or setting a minimum number of samples required to split a node to prevent overfitting.

11. Interpretability:

- If the problem domain requires interpretability, consider using techniques like SHAP values or partial dependence plots to understand how the model makes predictions.

12. Deploy the Model:

- Once you are satisfied with the model's performance, deploy it to make predictions in a real-world environment.

13. Monitor and Maintain:

- Continuously monitor the model's performance in production and retrain it if necessary to account for concept drift or changes in data distribution.

GOAL:

- Here the goal is to build a predictive model for house prices. We will follow a series of steps to load, explore, preprocess, and create a baseline model for house price prediction using a sample dataset. The model will use features from the dataset to predict the 'Avg. Area Income.

DATASET:

https://www.kaggle.com/datasets/vedavyasv/usa-housing

Steps to start building the house price prediction model by loading and preprocessing the dataset:

1.Import Necessary Libraries

2. Load the Dataset

3. Data Exploration and Preprocessing

    a. Explore the Data

    b.Handling Missing Values

    c.Encoding Categorical Variables

4. Splitting the Dataset

5. Building and Training the Model
6. Making Predictions
7. Evaluating the Model

1.Import Necessary Libraries:

- In the first step, we imported essential Python libraries, including Pandas, NumPy, and Scikit-Learn. These libraries provide tools for data manipulation, preprocessing, modeling, and evaluation.

```
import pandas as pd import
numpy as np
from sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler from
sklearn.linear_model import LinearRegression from
sklearn.metrics import mean_squared_error
```

2. Load the Dataset:

We loaded the dataset from the provided file path. The dataset is assumed to contain information relevant to house price prediction, and we used the Pandas library to read it.

```
data = pd.read_csv('/content/USA_Housing.csv')
```

3.Data Exploration and Preprocessing:

- In this phase, we performed data exploration and preprocessing to prepare the data for modeling.

a.Explore the Data:

- We examined the dataset by printing the first few rows and checking the data information to understand its structure and characteristics.

print(data.head()) print(data.info())

OUTPUT:

```
Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046              7.188236                     5.586729
       4     59982.197226           5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26      34310.242831  1.260617e+06     4
                           4.23     26354.109472  6.309435e+05
```

```
                                              Address      0
208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1                          188 Johnson Views Suite 079\nLake Kathleen,
                           CA...
2                          9127  Elizabeth  Stravenue\nDanieltown,  WI
                           06482...
3                          USS Barnett\nFPO AP 44820
4                          USNS Raymond\nFPO AE 09386
<class  'pandas.core.frame.DataFrame'>  RangeIndex:
5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
0   Avg. Area Income              5000 non-null   float64
1   Avg. Area House Age           5000 non-null   float64
2   Avg. Area Number of Rooms     5000 non-null   float64
3   Avg. Area Number of Bedrooms  5000 non-null   float64
4   Area Population               5000 non-null   float64
5   Price                        5000 non-null   float64  6   Address
    5000 non-null   object  dtypes: float64(6), object(1) memory usage:
    273.6+ KB None
```

b. Handling Missing Values:

- Missing values, if any, were filled with zeros in this code. It's important to note that a more sophisticated strategy for handling missing data may be necessary in real-world datasets.

data.fillna(0, inplace=True)

c. Encoding Categorical Variables:

- We used one-hot encoding to convert the 'Address' column into a numerical format suitable for machine learning.

```
data = pd.get_dummies(data, columns=['Address'])
```

4. Splitting the Dataset:

- We divided the data into two parts: the feature matrix 'X' (excluding the target variable, 'Avg. Area Income') and the target variable 'y' (which is the 'Avg. Area Income' column).
  Feature scaling was applied to standardize the features using StandardScaler from Scikit-Learn.

```
X = data.drop('Avg. Area Income', axis=1) y
= data['Avg. Area Income'] scaler =
StandardScaler() X = scaler.fit_transform(X)
```

- The dataset was split into a training set (80%) and a testing set (20%). The random_state parameter was set to 42 for reproducibility.

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

5. Building and Training the Model:

- In this step, we created and trained a Linear Regression model using the training data. Linear Regression is a suitable choice for house price prediction tasks.

model = LinearRegression() model.fit(X_train,

y_train)

6: Making Predictions:

- We used the trained Linear Regression model to make predictions on the test data.

y_pred = model.predict(X_test)

7: Evaluating the Model:

- To assess the model's performance on the test data, we calculated the Mean Squared Error (MSE). The MSE measures the average squared difference between the predicted and actual 'Avg. Area Income' values.

mse = mean_squared_error(y_test, y_pred) print(f"Mean

Squared Error: {mse}")

FINAL OUTPUT :

```
Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046              7.188236                     5.586729
       4     59982.197226           5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
```

```
1                           3.09     40173.072174  1.505891e+06
2                           5.13     36882.159400  1.058988e+06
3                           3.26      34310.242831  1.260617e+06     4
                            4.23     26354.109472  6.309435e+05

                                                  Address      0
208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1                           188 Johnson Views Suite 079\nLake Kathleen,
                            CA...
2                           9127  Elizabeth  Stravenue\nDanieltown,  WI
                            06482...
3                           USS Barnett\nFPO AP 44820
4                           USNS Raymond\nFPO AE 09386
<class  'pandas.core.frame.DataFrame'>  RangeIndex:
5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
0   Avg. Area Income              5000 non-null   float64
1   Avg. Area House Age           5000 non-null   float64
2   Avg. Area Number of Rooms     5000 non-null   float64
3   Avg. Area Number of Bedrooms  5000 non-null   float64
4   Area Population               5000 non-null   float64
5   Price                        5000 non-null   float64  6   Address
    5000 non-null   object  dtypes: float64(6), object(1) memory usage:
    273.6+ KB
None
Mean Squared Error: 72310314.15909015
```

DATASET:

https://www.kaggle.com/datasets/vedavyasv/usa-housing

OVERVIEW:

- This document presents a Python code implementation for predicting house prices based on a dataset containing attributes such as 'Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', and the target variable 'Price'.
- The code utilizes Linear Regression and Random Forest models for prediction and evaluates their performance using Mean Squared Error (MSE) and Rsquared metrics.

Steps involved:

1. Data Loading and Overview
2. Data Preprocessing
3. Model Selection and Training
4. Model Evaluation
5. Results

1.Date loading and overview:

```
# Load the dataset data =
pd.read_csv('house_data.csv')

# Display the first few rows to understand the structure of the data
print(data.head())
```

Explanation:

- Loading the dataset using Pandas.
- Printing the first few rows helps understand the columns and data types.

2. Data Preprocessing:

```
# Selecting Features and Target Variable
X = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population']] y = data['Price']
```

```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling scaler =
StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- Identified the independent features and the dependent target variable.
- Split the dataset into training and testing sets for model evaluation.
- Scaled the features to have a standard mean and variance.

3. Model Selection and Training:

```
# Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Random Forest model forest_model =
RandomForestRegressor(n_estimators=100, random_state=42)
forest_model.fit(X_train, y_train)
```

- Description of the chosen machine learning algorithm (e.g., Linear Regression, Random Forest, etc.).
- Training the model on the dataset.

- Hyperparameter tuning (if performed) and reasoning behind it.

4. Model Evaluation:

```
# Linear Regression predictions and evaluation
linear_predictions = linear_model.predict(X_test)
linear_mse = mean_squared_error(y_test,
linear_predictions) linear_r2 = r2_score(y_test,
linear_predictions)


# Random Forest predictions and evaluation
forest_predictions = forest_model.predict(X_test)
forest_mse = mean_squared_error(y_test,
forest_predictions) forest_r2 = r2_score(y_test,
forest_predictions)
```

- Made predictions using both models on the test set.
- Calculated Mean Squared Error (MSE) and R-squared to evaluate model performance.

5. Results:

```
print("Linear Regression Model:")
print(f"Mean Squared Error: {linear_mse}")
print(f"R-squared: {linear_r2}")
```

```
print("\nRandom Forest Model:")
print(f"Mean Squared Error: {forest_mse}")
print(f"R-squared: {forest_r2}")
```

- Displayed the performance metrics (MSE and R-squared) of both models.

FINAL OUTPUT:

Linear Regression Model:

Mean Squared Error: 10089009300.89399

R-squared: 0.9179971706834331

Random Forest Model:

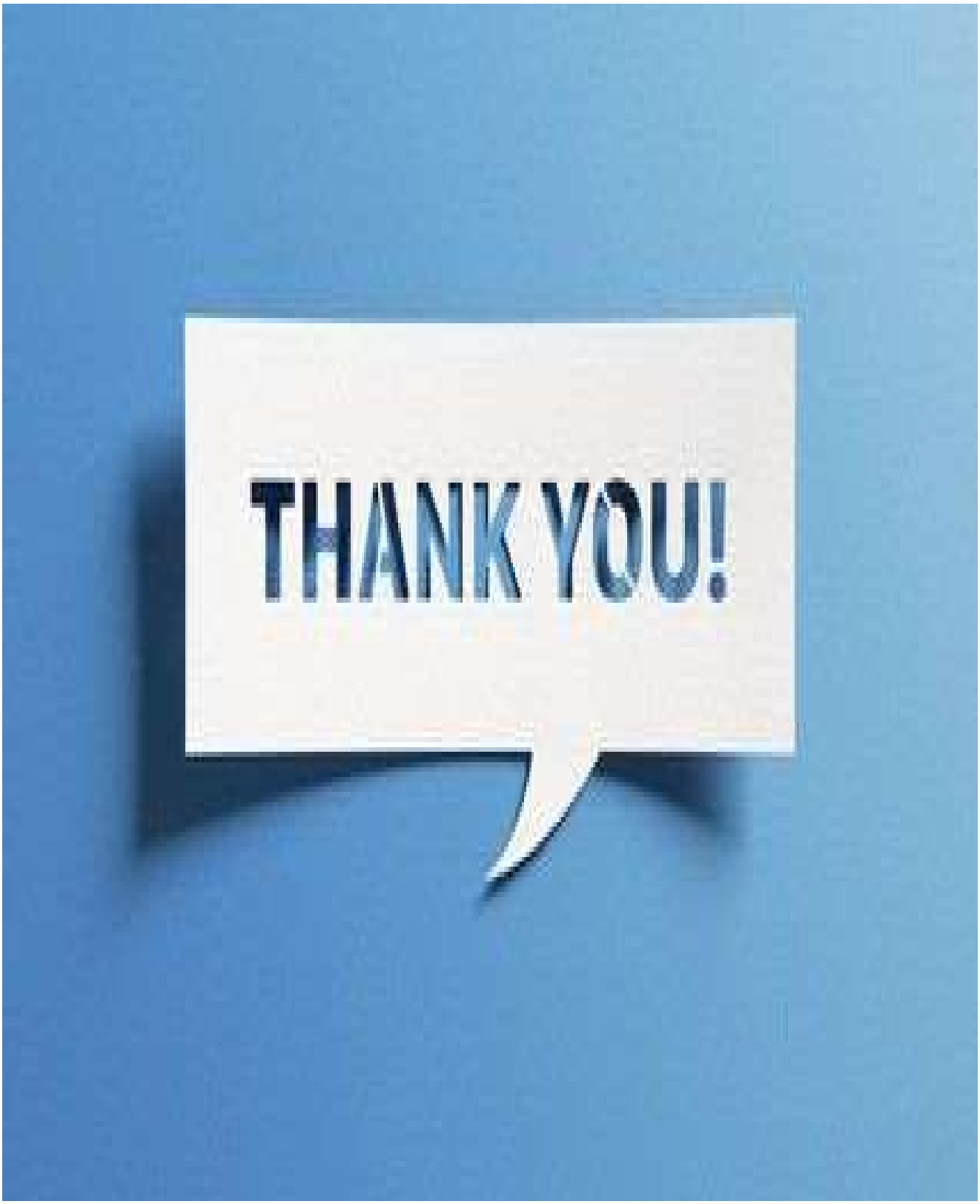
Mean Squared Error: 14462012668.455772

R-squared: 0.882453675969917

- Displayed the performance metrics (MSE and R-squared) of both models.

CONCLUSION:

- In the final document, you can summarize the outcomes, compare model performances, and discuss possible improvements or next steps.
- Each section should include clear explanations to ensure the reader understands the purpose and workflow of the code.
- This expanded content provides a comprehensive understanding of the project's different phases. Adjust it based on the project's specific requirements and goals.


THANK YOU!