California Housing Price Prediction Model

Overview

This project builds and evaluates multiple machine learning models to predict housing prices in California using the **California Housing Dataset**. The dataset contains various housing-related features, and our goal is to create models that can accurately predict the **median house value**.

Dataset Description

The dataset is obtained from the **Scikit-learn datasets module** and includes features such as:

- **MedInc**: Median income in block group
- **HouseAge**: Median house age in block group
- **AveRooms**: Average number of rooms per household
- **AveBedrms**: Average number of bedrooms per household
- **Population**: Block group population
- **AveOccup**: Average household occupancy
- **Latitude**: Latitude of block group
- **Longitude**: Longitude of block group
- **Target Variable**: `MedHouseVal` (Median house value, scaled to USD for easier interpretation)

Data Preprocessing

1. **Handling Missing Values**: The dataset is cleaned by removing rows with missing values.
2. **Feature Scaling**: Numerical features are standardized using `StandardScaler` to improve model performance.
3. **Train-Test Split**: The dataset is split into **80% training data** and **20% test data** to evaluate model performance.

Exploratory Data Analysis (EDA)

- **Feature Correlation Heatmap**: A heatmap is generated to visualize the correlation between features.
- **Histogram Analysis**: Distribution of each feature is plotted to understand data distribution.

Machine Learning Models Implemented

1. Linear Regression

- A simple model that assumes a linear relationship between independent variables and the target variable.
- Implemented using `LinearRegression()` from `sklearn.linear_model`.
- Performance Metrics: **MAE, RMSE, R^2 Score, MAPE, Median Absolute Error**.

2. Random Forest Regressor

- A robust ensemble method using multiple decision trees to improve predictive accuracy.
- Hyperparameters (`n_estimators`, `max_depth`, `min_samples_split`) are tuned using `GridSearchCV`.
- Implemented using `RandomForestRegressor()` from `sklearn.ensemble`.
- Identified as the **best performing model** based on evaluation metrics.

3. Decision Tree Regressor

- A simple decision tree model that splits data into subsets for predictions.
- Implemented using `DecisionTreeRegressor()` from `sklearn.tree`.

4. Neural Network (MLP Regressor)

- A multi-layer perceptron (MLP) model with two hidden layers (64, 32 units).
- Optimized using `max_iter=1000` to ensure convergence.
- Implemented using `MLPRegressor()` from `sklearn.neural_network`.

Model Evaluation

Each model is evaluated using the following metrics:

- **Mean Absolute Error (MAE)**: Measures average absolute differences between actual and predicted values.
- **Root Mean Squared Error (RMSE)**: Measures standard deviation of residuals.
- **R^2 Score**: Measures how well the model explains the variance in the target variable.
- **Mean Absolute Percentage Error (MAPE)**: Percentage-based error measurement.
- **Median Absolute Error**: Robust metric for handling outliers.

Evaluation Results

Model	MAE	RMSE	R^2 Score	MAPE	Median AE
Linear Regression	51941.1	72043.3	0.5482	0.3712	36409.77
Random Forest	31768.5	45178.0	0.8213	0.2178	22537.94
Decision Tree	42230.9	58836.0	0.6932	0.2941	30912.86
Neural Network	38974.6	55632.1	0.7284	0.2693	28965.44

Note: Random Forest achieved the best performance across most evaluation metrics.

Key Takeaways

- **Random Forest is the most reliable model**, achieving the highest accuracy and lowest error rates.
- **Linear Regression provides a simple, interpretable model**, but lacks predictive power compared to ensemble methods.
- **Neural Networks perform well**, but require fine-tuning to achieve optimal results.
- **Feature scaling is crucial** for ensuring consistent model performance.

Future Enhancements

- **Feature Engineering**: Add more location-based or demographic features.
- **Hyperparameter Tuning**: Further optimize parameters for Neural Networks and Decision Trees.
- **Deploy Model as an API**: Integrate Flask or FastAPI for real-time predictions.
- **Try Additional Models**: Explore Gradient Boosting (XGBoost, LightGBM) for improved accuracy.

Conclusion

This project demonstrates how various machine learning models can be used for **predicting house prices**. By leveraging feature engineering, model tuning, and evaluation metrics, we identified **Random Forest** as the best model for this task. Future improvements can be made by incorporating additional features and deploying the model for real-world use.