Project Report

Problem Statement and Objectives

The objective of this project is to predict house prices based on historical real estate data. The app allows users to explore dataset trends, train regression models, and make predictions interactively.

Data Analysis Approach and Findings

- EDA: Conducted exploratory data analysis, including visualization of target variable (SalePrice), Pearson correlation heatmap and distribution plots to show the relationship with other numerical and categorical features.
- Numerical features with highest correlation to SalePrice:
  - OverallQual
  - GrLivArea
  - GarageCars
  - TotalBsmtSF
  - 1stFlrSF
- Categorical features with highest correlation to SalePrice
  - Neighborhood
  - ExterQual
  - KitchenQual
  - BsmtQual
  - GarageFinish
- Insights:
  - Quality matters more than size; in fact, features which express quality (Overall, Kitchen, Exter) have higher correlation with the price.
  - Neighborhood has a strong impact on price; as expected, the location is important to the price.
  - House style also matters; houses with more than one story tend to have higher prices.
  - Practical amenities (that is, used spaces), like Garage and Basement, tend to increase the price more than just sheer dimensions.
  - Sale condition affects the price; houses sold under normal conditions tend to have higher prices.
  - There are some outliers given by large, poor-quality homes or by special sales.

Machine Learning Methodology and Results

- Models used:
  - Linear Regression
  - Random Forest Regression
  - Gradient Boosting
- Hyperparameter tuning: Optional training models for RFR and GB using GridSearchCV.
- Pipeline: ColumnTransformer to apply the preprocessing; StandardScaler for numerical features and OneHotEncoder for categorical features; SimpleInputer for missing values. This allows the application to perform both the preprocessing and the training in a single operation.
- Evaluation metrics:
  - RMSE

- R^2

Technical Implementation Details

- Project Structure:

```
HousePricePrediction/
├── app.py                      # Main Flask application
├── dataset_manager.py          # Dataset handling, preprocessing, and EDA
├── training_manager.py         # Model training and management
├── eda.py                      # File for EDA operations
├── models/                     # Storage for trained models and metadata
│   ├── models.json             # JSON file tracking trained models
│   └── *.pkl                   # Serialized trained model files
├── static/                     # Static assets for the frontend
│   ├── css/                    # CSS stylesheets
│   │   └── *.css               # Custom styling (Bootstrap-inspired)
│   ├── js/                     # Frontend JS scripts for dynamic behavior
│   │   └── *.js                # JS files
│   └── eda/                    # Generated plots from EDA
│       └── *.png               # Plot images
├── templates/                  # HTML templates for Flask routes
│   └── *.html                  # Application's pages
├── data/                       # Folder to store the dataset
│   └── dataset.csv             # Dataset file
├── Dockerfile                  # Configuration for building the container
├── docker-compose.yml          # Docker Compose to orchestrate container
├── requirements.txt            # Python dependencies
├── README.md                   # Project information
├── .gitignore                  # Gitignore file
└── report/                     # Folder for documentation
    └── report.pdf              # Final report
```

- Server:
  - Python
  - Flask
  - Libraries: Pandas, Numpy, Matplotlib, Seaborn, Scikit-learn, Joblib, Os
- Client:
  - HTML
  - CSS with Bootstrap style
  - JS for dynamic feature input
- Data Storage:
  - CSV Dataset
  - Models saved as '.pkl'
  - Models' metadata saved in a JSON file
- Containerization: Docker Compose orchestrates Flask app container

Challenges Encountered and Solutions

- Handling missing values and categorical encoding consistently between training and prediction - Solved using scikit-learn pipelines.
- Model management (saving/loading/deleting) with metadata - Implemented JSON file for metadata storing.

- Predicting from partial input - Introduced NaN filling and checkbox option for incomplete input.
- Deployment across PCs with WSL/Docker differences: used Docker Compose for consistent containerization.

Possible Future Improvements

- Extend model selection with additional regression models.
- Add CSV file input when predicting for multiple predictions
- Output CSV file for multiple predictions
- Implement user authentication for multi-user predictions
- Provide richer downloadable reports for predictions