**Project Overview: Predictive Modeling Pipeline**

In this project, we developed a predictive model that uses various machine learning techniques to make predictions on real estate transaction data. The process involves several stages, from data preprocessing to model evaluation. Below is a breakdown of the steps followed, methodologies used, and the final model's performance.

**1. Data Preprocessing**

**1.1 Data Loading**

* Data was loaded from two primary sources: the **transactions dataset** and the **rents dataset**.
* Both datasets were in CSV format, and the primary goal was to merge them for feature engineering.

**1.2 Data Cleaning and Transformation**

* **Handling Missing Data**:
  + We checked for missing values in both datasets and decided to drop or impute values where necessary.
  + For instance, if a column had missing data, we either removed the rows containing missing values or applied a forward fill method to propagate the last valid observation.
* **Feature Selection**:
  + We selected features relevant to predicting property size, such as transaction\_number, transaction\_datetime, property\_usage\_en, and others.
  + Categorical columns such as transaction\_type\_en and is\_freehold\_text were identified and converted into appropriate data types for model compatibility.

**1.3 Handling Datetime Features**

* **Datetime Columns**:
  + Columns like transaction\_datetime, contract\_start\_date, and contract\_end\_date were converted to datetime objects using pd.to\_datetime().
  + To handle these datetime columns for machine learning, we extracted relevant features like year, month, day, or converted the datetime values to Unix timestamps (seconds since epoch).

**1.4 Encoding Categorical Features**

* **Label Encoding**:
  + We used **LabelEncoder** to convert categorical variables into numerical representations (e.g., 'Mortgage' → 1, 'Rent' → 0).
  + Ensured the encoding was consistent across training and test datasets.

**1.5 Feature Scaling**

* **Normalization and Scaling**:
  + Numerical features (e.g., property\_size\_sqm, building\_age, etc.) were scaled for better model performance.
  + For this, we used **StandardScaler** to normalize the data, ensuring all features are on a comparable scale.

**2. Feature Selection Techniques**

**2.1 Recursive Feature Elimination (RFE)**

* We used **RFE** to eliminate less important features and select a subset of the most relevant features.
* A **Linear Regression** model was used as the base estimator for RFE, and we aimed to select the top 10 features.
* Features like property\_size\_sqm were deemed highly relevant, while others with lower importance were dropped.

**2.2 Univariate Feature Selection**

* We performed univariate statistical tests (e.g., **Chi-squared tests**) to evaluate the relationship between each feature and the target variable (property\_size\_sqm).
* Features that showed high correlation with the target were retained, and others were discarded.

**3. Model Building and Training**

**3.1 Base Model Development**

* We implemented multiple machine learning models to serve as base learners in the ensemble:
  + **Linear Regression**
  + **Random Forest Regressor**
  + **XGBoost**
  + **Support Vector Regression (SVR)**

**3.2 Hyperparameter Tuning**

* We used **GridSearchCV** and **Bayesian Optimization** to fine-tune the hyperparameters of these models.
  + For example, for the RandomForestRegressor, parameters like n\_estimators, max\_depth, and min\_samples\_split were optimized to improve model performance.

**3.3 Ensemble Learning**

* We combined predictions from multiple base models using a **meta-model** approach, where the output of base models was fed into a second-level learner (meta-learner).
* This meta-model was trained to combine base models' predictions and make the final prediction.

**3.4 Final Model**

* After training all base models, we utilized a **Neural Network** as the meta-learner. The neural network learned how to best combine the outputs of the base models into the final prediction.

**4. Model Evaluation**

**4.1 Evaluation Metrics**

* We used the following metrics to evaluate model performance:
  + **RMSE (Root Mean Squared Error)**: Measures the average magnitude of the errors in the predictions, with larger errors penalized more heavily.
  + **R2 Score**: Measures the proportion of variance explained by the model. A higher R2 score indicates better model performance.
  + **MAE (Mean Absolute Error)**: Measures the average of the absolute differences between predicted and actual values. MAE is less sensitive to outliers than RMSE.

**4.2 Performance Results**

* The models were trained and evaluated on a training/test split (80/20).
* We compared the performance of different models and noted which performed best on the test data.
* The **meta-learner (Neural Network)** showed improvements in performance, particularly when combining predictions from diverse base models.

**5. Hyperparameter Optimization**

We implemented **Bayesian Optimization** to fine-tune the hyperparameters of the models. This optimization method is more efficient than grid search as it searches the hyperparameter space probabilistically, focusing on regions with better performance.

**6. Model Deployment and Usage**

**6.1 Saving the Trained Model**

* The final trained model (meta-model + base models) was saved using **joblib** to allow for future use or deployment.

**6.2 Making Predictions**

* Once the model is trained, we can use it to make predictions on new data:
  + **Input**: A dataset similar to the one used for training (with the same preprocessing steps applied).
  + **Output**: Predicted property size (property\_size\_sqm).

**7. Final Remarks**

* The project follows a **modular** approach, where each component (data loading, preprocessing, model building, and evaluation) is clearly separated and reusable.
* **Version Control**: All the code and configurations were tracked using **Git**, and a comprehensive README.md file was provided for easy setup and execution.
* **Testing**: Unit tests were written to ensure key components of the pipeline were working correctly, including feature preprocessing and model evaluation.

**8. Files in the Git Repository**

* config.py: Stores configurable parameters (e.g., file paths, hyperparameters).
* main.py: Orchestrates the pipeline, including data preprocessing, model training, and evaluation.
* evaluation.py: Contains model evaluation metrics such as RMSE, R2 Score, and MAE.
* test.py: Contains unit tests for key components.
* README.md: Provides instructions on how to set up, train, and use the model.
* requirements.txt: Lists all required Python libraries.