**Detailed Report on Housing Price Prediction Workflow**

**1. Introduction**

The objective of this project is to predict housing prices using a regression model. We use a Random Forest Regressor within an end-to-end pipeline that preprocesses data, handles missing values, encodes categorical variables, and applies the model. The project leverages Python libraries such as Pandas, NumPy, and scikit-learn. The process follows best practices for data preprocessing and model evaluation.

**2. Data Preprocessing**

**2.1. Data Loading**

* **Training Data:**  
  The training data is loaded from a CSV file (train.csv). The target variable is defined as SalePrice.
* **Test Data:**  
  A separate test dataset (test.csv) is loaded; note that this dataset does not contain the target variable.

**2.2. Separating Features and Target**

* **Features (X):**  
  All columns except SalePrice are used as features.
* **Target (y):**  
  The SalePrice column is extracted as the target variable.

**2.3. Handling Missing Values**

* **Numeric Variables:**  
  For numeric features such as LotFrontage, missing values are imputed using the median value. This is crucial as it preserves the central tendency without being affected by outliers.

python

Copy

X["LotFrontage"] = X["LotFrontage"].fillna(X["LotFrontage"].median())

test["LotFrontage"] = test["LotFrontage"].fillna(X["LotFrontage"].median())

* **Categorical Variables:**  
  For categorical features, any missing value is replaced with the string "None". This avoids dropping rows and retains all available information.

python

Copy

cat\_cols\_tmp = X.select\_dtypes(include=['object']).columns

for col in cat\_cols\_tmp:

X[col] = X[col].fillna("None")

if col in test.columns:

test[col] = test[col].fillna("None")

**2.4. Exploratory Data Analysis (EDA)**

* **Missing Values Summary:**  
  A quick count of missing values is performed to understand the data quality.

python

Copy

missing\_counts = X.isna().sum().sort\_values(ascending=False)

print("Missing value counts (top 10):\n", missing\_counts.head(10))

* **Data Types:**  
  The dataset is examined for numeric and categorical variables using Pandas’ select\_dtypes method. This helps in designing the preprocessing pipeline.

**3. Data Splitting**

Before final model training, the data is split into training and validation sets to perform local evaluation:

* **Train/Validation Split:**  
  The data is split into 80% training and 20% validation. A fixed random state ensures reproducibility.

python

Copy

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**4. Preprocessing Pipeline**

A robust pipeline is built to ensure that the same preprocessing steps are applied to both training and test data.

**4.1. Preprocessing for Numeric Features**

* **Numeric Transformer:**  
  A SimpleImputer is used with a median strategy to fill missing numeric values.

python

Copy

numeric\_transformer = SimpleImputer(strategy='median')

**4.2. Preprocessing for Categorical Features**

* **Categorical Transformer:**  
  A pipeline is created that first imputes missing values with a constant value ("None") and then encodes categorical features using OneHotEncoder with the parameter handle\_unknown='ignore' to safely process any new categories encountered in test data.

python

Copy

categorical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='constant', fill\_value='None')),

('encoder', OneHotEncoder(handle\_unknown='ignore'))

])

**4.3. ColumnTransformer**

* **Combining Preprocessing Steps:**  
  Both numeric and categorical transformations are combined using a ColumnTransformer. This applies the appropriate transformation to each column subset.

python

Copy

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, num\_cols),

('cat', categorical\_transformer, cat\_cols),

]

)

**5. Modeling Techniques**

**5.1. Model Selection**

* **Random Forest Regressor:**  
  The model of choice is a Random Forest Regressor, which is well-suited for regression tasks with tabular data. It is set up with 100 trees and a fixed random state to ensure reproducibility.

python

Copy

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

**5.2. Pipeline Integration**

* **End-to-End Pipeline:**  
  The preprocessing steps and the model are chained together into a single pipeline. This ensures that during cross-validation and testing, all transformations are applied consistently.

python

Copy

pipeline = Pipeline(steps=[

('preprocessor', preprocessor),

('model', model)

])

**5.3. Hyperparameter Tuning (Optional)**

* **Grid Search Example:**  
  An example grid search is provided (currently commented out) to tune hyperparameters such as the number of trees (n\_estimators), tree depth (max\_depth), and the minimum number of samples required to split an internal node (min\_samples\_split). GridSearchCV uses 5-fold cross-validation to identify the best hyperparameters.

python

Copy

param\_grid = {

'model\_\_n\_estimators': [50, 100, 200],

'model\_\_max\_depth': [None, 10, 20],

'model\_\_min\_samples\_split': [2, 5, 10]

}

grid\_search = GridSearchCV(

pipeline,

param\_grid,

scoring='neg\_root\_mean\_squared\_error',

cv=5,

verbose=1

)

grid\_search.fit(X\_train, y\_train)

print("Best Params:", grid\_search.best\_params\_)

best\_model = grid\_search.best\_estimator\_

**6. Model Evaluation and Results**

**6.1. Validation Performance**

* **Fit and Predict:**  
  The pipeline is first fitted on the training subset (X\_train, y\_train) and then used to make predictions on the validation set.

python

Copy

pipeline.fit(X\_train, y\_train)

y\_pred\_val = pipeline.predict(X\_val)

* **Metric:**  
  The model performance is evaluated using the Root Mean Squared Error (RMSE). RMSE provides an intuitive measure of the average prediction error.

python

Copy

val\_mse = mean\_squared\_error(y\_val, y\_pred\_val)

val\_rmse = np.sqrt(val\_mse)

print("Validation RMSE:", val\_rmse)

**6.2. Final Model Training**

* **Retraining:**  
  Once the validation performance is satisfactory, the pipeline is retrained on the full dataset (all training data) to maximize the use of available information.

python

Copy

pipeline.fit(X, y)

**6.3. Test Set Predictions**

* **Making Predictions:**  
  The final model is then used to predict the target variable on the test set.

python

Copy

test\_preds = pipeline.predict(test)

* **Saving Output:**  
  Predictions are saved into a CSV file for submission or further analysis.

python

Copy

output = pd.DataFrame({'Id': test['Id'], 'SalePrice': test\_preds})

output.to\_csv('submission.csv', index=False)

print("Final predictions saved to submissionRandomforest.csv")

**7. Summary and Conclusions**

* **Data Preprocessing:**  
  Missing values are handled using median imputation for numeric variables and constant imputation for categorical ones. The data types are also separated to apply the appropriate preprocessing techniques.
* **EDA:**  
  A quick review of missing values and data types informs the preprocessing pipeline design.
* **Modeling:**  
  A Random Forest Regressor is used within a robust pipeline that ensures consistency in data transformation and model training. An optional grid search is provided for hyperparameter tuning.
* **Results:**  
  The model is evaluated using RMSE on a validation set. After evaluation, the pipeline is retrained on the full dataset, and predictions are made on the test set. These predictions are then saved for further use.