COMPREHENSIVE MACHINE LEARNING PIPELINE FOR HOUSE RENT PREDICTION

This document outlines a complete machine learning pipeline for predicting house rents, detailing each stage from initial data exploration to model deployment.

PIPELINE SUMMARY

1. EDA (Exploratory Data Analysis)
2. Preprocessing
3. Encoding
4. Modeling
5. Evaluation
6. Hyperparameter Tuning
7. Model Saving
8. Deployment

✅ 1. UNDERSTAND DATASET

- **Shape:** 4746 rows × 12 columns
- **Target:** `Rent` (continuous numeric → regression problem)
- **Numeric columns:** `BHK`, `Rent`, `Size`, `Bathroom`
- **Categorical columns:** `Posted On`, `Floor`, `Area Type`, `Area Locality`, `City`, `Furnishing Status`, `Tenant Preferred`, `Point of Contact`

✅ 2. EDA (EXPLORATORY DATA ANALYSIS)

- **Check missing values:**
  `python df.isnull().sum()`
- **Describe numeric & categorical:**
  `python df.describe(include='all')`
- **Distribution plots:**
  - Histograms for numeric columns (`BHK`, `Rent`, `Size`, `Bathroom`)

- Countplots for categorical columns (`City` , `Area Type` , `Furnishing Status`)

- **Correlation heatmap for numeric columns:** `python sns.heatmap(df.corr(), annot=True)`
- **Outlier detection with boxplots:** (Rent and Size have high outliers)

✅ 3. HANDLE OUTLIERS & SKEWNESS

- **Detect outliers using IQR method:** `python Q1 = df['Rent'].quantile(0.25) Q3 = df['Rent'].quantile(0.75) IQR = Q3 - Q1 df = df[(df['Rent'] >= Q1 - 1.5*IQR) & (df['Rent'] <= Q3 + 1.5*IQR)]`
- **Check skewness:** `python df['Rent'].skew()`
- **Apply log transform for skewed columns:** `python df['Rent'] = np.log1p(df['Rent']) df['Size'] = np.log1p(df['Size'])`

✅ 4. FEATURE ENGINEERING

- **Drop unnecessary columns:** `python df.drop(['Posted On', 'Area Locality'], axis=1, inplace=True)`

- **Convert `Floor` column to `Current_Floor` and `Total_Floors` :** ```python
def extract_floor_info(floor):
try:
parts = floor.split(' out of ')
current = 0 if parts[0].strip().lower() == 'ground' else int(parts[0])
total = int(parts[1])
return pd.Series([current, total])
except:
return pd.Series([None, None])

df[['Current_Floor', 'Total_Floors']] = df['Floor'].apply(extract_floor_info)
df.drop('Floor', axis=1, inplace=True)
```

---
```

✅ 5. ENCODING CATEGORICAL VARIABLES

- **One-Hot Encoding:**
  `python df = pd.get_dummies(df, columns=['Area Type', 'City', 'Furnishing Status', 'Tenant Preferred', 'Point of Contact'], drop_first=True)`

✅ 6. PREPARE DATA FOR MODELING

- **Separate features & target:**
  `python X = df.drop('Rent', axis=1) y = df['Rent']`
- **Train-Test Split:**
  `python from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`
- **Scale data for linear models:**
  `python from sklearn.preprocessing import StandardScaler scaler = StandardScaler() X_train_scaled = scaler.fit_transform(X_train) X_test_scaled = scaler.transform(X_test)`

✅ 7. TRAIN MULTIPLE MODELS

Models:

- Linear Regression
- Lasso, Ridge
- Decision Tree
- Random Forest
- Gradient Boosting
- XGBoost

Example:

```
from sklearn.linear_model import LinearRegression, Lasso,
Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error
```

✅ 8. EVALUATE MODELS

Use:

- **R^2 Score**
- **RMSE**
- **Custom Accuracy (within ±10%)**

```
def accuracy_within_tolerance(y_true, y_pred,
tolerance=0.10):
    correct = abs(y_true - y_pred) <= (tolerance *
y_true)
    return correct.mean()
```

Compare models:

```
results = {
    'Linear Regression': (r2_score, rmse, accuracy),
    ...
}
```

Visualize performance:

```
results_df.sort_values(by='R² Score')['R²
Score'].plot(kind='bar')
```

✅ 9. BEST MODELS

- **XGBoost** or **Gradient Boosting** usually performs best on tabular data.
- **Random Forest** is strong but slower than XGBoost.

✅ 10. HYPERPARAMETER TUNING

WHY?

Hyperparameter tuning improves model performance by finding the optimal set of parameters.

TECHNIQUES

1. **GridSearchCV** → Tests all combinations of parameters (exhaustive search)
2. **RandomizedSearchCV** → Tests a random subset of combinations (faster)

EXAMPLE: RANDOM FOREST

```
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestRegressor(random_state=42)
random_search = RandomizedSearchCV(rf,
param_distributions=param_grid, n_iter=10, cv=5,
scoring='r2', random_state=42)
random_search.fit(X_train, y_train)

best_rf = random_search.best_estimator_
print("Best Parameters:", random_search.best_params_)
```

EXAMPLE: XGBOOST

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1],
    'colsample_bytree': [0.8, 1]
}

xgb = XGBRegressor(random_state=42)
random_search = RandomizedSearchCV(xgb,
param_distributions=param_grid, n_iter=10, cv=5,
scoring='r2', random_state=42)
random_search.fit(X_train, y_train)

best_xgb = random_search.best_estimator_
print("Best Parameters:", random_search.best_params_)
```

✅ 11. SAVE & LOAD MODEL

Use **joblib** for serialization:

```
import joblib

# Save the model
joblib.dump(best_xgb, 'best_xgb_model.pkl')

# Load the model
loaded_model = joblib.load('best_xgb_model.pkl')
```

✅ 12. CROSS-VALIDATION

Perform **k-fold cross-validation** to check model stability:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(best_xgb, X_train, y_train,
cv=5, scoring='r2')
print("Cross-Validation R² Scores:", scores)
print("Mean CV Score:", scores.mean())
```

✅ 13. FEATURE IMPORTANCE

Check which features influence predictions:

```
import matplotlib.pyplot as plt

importances = best_xgb.feature_importances_
features = X.columns

plt.figure(figsize=(10,6))
plt.barh(features, importances)
plt.title('Feature Importance')
plt.show()
```

✅ 14. RESIDUAL ANALYSIS

Evaluate errors visually:

```
y_pred = best_xgb.predict(X_test)

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Rent")
plt.ylabel("Predicted Rent")
plt.title("Actual vs Predicted")
plt.show()
```

✅ 15. DEPLOY MODEL (OPTIONAL)

- **Flask / FastAPI** → Create an API
- **Streamlit / Gradio** → Interactive UI for predictions
- **Cloud Deployment** → AWS, Azure, GCP
