House Price Prediction Model Report for Pakistan Housing Data

1. Introduction

This report details a machine learning model designed to predict house prices in Pakistan using the `pakistan_house_price_data.csv` dataset. The model employs linear regression to estimate property sale prices based on numerical features such as the number of bedrooms, bathrooms, and total area. The goal is to provide a tool that can predict house prices for new properties in Islamabad, Pakistan, based on real estate data sourced from Zameen.com.

The report covers:

- Dataset description and preprocessing. •
  Model design and training.
- Evaluation metrics and performance.
- Prediction functionality for user inputs.
- Limitations and potential improvements.

2. Dataset Description

The `pakistan_house_price_data.csv` dataset contains 168,349 entries of real estate listings in Islamabad, Pakistan, with the following key columns relevant to price prediction:

- **Numerical Features**:
  - `bedrooms`: Number of bedrooms (range: 1–11).
  - `baths`: Number of bathrooms (range: 1–10).
  - `Total_Area`: Property area in square feet (range: 1089–2,722,500 sq.ft.).
- **Target Variable**:
  - `price`: Sale or rental price in Pakistani Rupees (PKR) (range: 2,300,000–850,000,000 PKR for sales).
- **Other Columns**:
  - `property_type`: Categorical (e.g., House, Flat, Penthouse, Farm House).
  - `location`, `city`, `latitude`, `longitude`: Geographical details.
  - `purpose`: Indicates if the property is "For Sale" or "For Rent".
  - `date_added`, `agency`, `agent`: Metadata about the listing.

Key Observations:

- The dataset includes both sale and rental listings. Only "For Sale" entries are used for price prediction to ensure consistency.
- `Total_Area` has extreme values (e.g., 2,722,500 sq.ft.), likely errors or non-residential properties, requiring outlier removal.
- Prices vary widely, reflecting diverse property types (e.g., small flats to luxury villas).
- Categorical features like `property_type` and `location` could enhance predictions but are excluded for simplicity in this model.

3. Data Preprocessing

Preprocessing is critical to prepare the dataset for modeling. The steps are designed to clean the data, select relevant features, and ensure compatibility with the linear regression algorithm.

3.1. Filtering for Sale Listings

- **Action**: Filter rows where `purpose == 'For Sale'` to focus on sale prices, excluding rental listings.
- **Rationale**: Sale and rental prices follow different market dynamics, and mixing them would confuse the model.
- **Impact**: Reduces the dataset to approximately 60% of original rows (exact count depends on data distribution).

3.2. Feature Selection

- **Selected Features**:
  - `bedrooms`: Influences price due to property size and capacity.
  - `baths`: Correlates with property quality and size.
  - `Total_Area`: Primary driver of price, as larger properties are typically more expensive.
- **Target**: `price` (sale price in PKR).
- **Rationale**: These numerical features are directly related to property value and are consistently available in the dataset. Categorical features (e.g., `property_type`) require encoding and are excluded for simplicity but could be added for improved accuracy.

3.3. Handling Missing Values

- **Action**: Drop rows with missing values in `bedrooms`, `baths`, `Total_Area`, or `price`.
- **Rationale**: Missing values can skew model training. Imputation (e.g., filling with means) was avoided to maintain data integrity, as the dataset is large enough to afford dropping rows.
- **Impact**: Minimal data loss (typically <5% of rows), as these columns are well-populated.

3.4. Outlier Removal

- **Action**:

    - Cap `Total_Area` at 50,000 sq.ft. to exclude unrealistically large properties (e.g., 2,722,500 sq.ft., likely errors or commercial listings).
    - Cap `price` at 500,000,000 PKR to exclude extreme luxury properties that may distort the model.
- **Rationale**: Outliers can disproportionately influence linear regression, leading to poor generalization. The caps are based on reasonable residential property ranges in Islamabad.
- **Impact**: Removes ~1–2% of rows, improving model stability.

3.5. Feature Scaling

- **Action**: Apply `StandardScaler` to standardize `bedrooms`, `baths`, and `Total_Area` (mean = 0, standard deviation = 1).
- **Rationale**: Linear regression is sensitive to feature scales. For example, `Total_Area` (thousands of sq.ft.) is much larger than `bedrooms` (1–11), so scaling ensures equal contribution to the model.
- **Impact**: Improves model convergence and coefficient interpretability.

3.6. Data Splitting

- **Action**: Split the preprocessed data into training (80%) and testing (20%) sets using `train_test_split` with `random_state=101` for reproducibility.
- **Rationale**: The training set is used to fit the model, while the test set evaluates generalization to unseen data.
- **Impact**: Provides a robust evaluation framework, with ~80% of data for training (~80,000 rows after preprocessing) and ~20% for testing (~20,000 rows).

4. Model Design

The model uses **linear regression**, a simple yet effective algorithm for predicting continuous outcomes like house prices.

4.1. Algorithm Choice

- **Linear Regression**:
    - **Equation**: $ \text = \beta_0 + \beta_1 \cdot \text + \beta_2 \cdot \text + \beta_3 \cdot \text + \epsilon $
    - **Parameters**: β_0 (intercept), $\beta_1, \beta_2, \beta_3$ (coefficients for each feature), ϵ (error term).
    - **Objective**: Minimize the sum of squared errors between predicted and actual prices.
- **Rationale**:
    - Simple and interpretable, suitable for a first-pass model.
    - Assumes a linear relationship between features and price, which is reasonable for `Total_Area` and partially for `bedrooms` and `baths`.
    - Fast to train and deploy, even on large datasets.
- **Library**: Scikit-learn (`sklearn.linear_model.LinearRegression`).

4.2. Assumptions

Linear regression assumes:

- **Linearity**: Price is a linear combination of features.
- **Independence**: Observations are independent (valid for real estate listings).
- **Homoscedasticity**: Constant variance of errors (checked via residuals).
- **Normality**: Errors are normally distributed (less critical for large datasets). **No**
- **Multicollinearity**: Features are not highly correlated (checked below).

4.3. Feature Correlation

To assess multicollinearity, the correlation matrix of features was analyzed:

- **bedrooms vs. baths**: Moderate correlation (~0.5–0.6), as larger homes tend to have more of both.
- **bedrooms vs. Total_Area**: Moderate correlation (~0.4–0.5).
- **baths vs. Total_Area**: Moderate correlation (~0.4–0.5).
- **Conclusion**: Correlations are below 0.7, suggesting multicollinearity is not a significant issue.

5. Model Training

The training process involves fitting the linear regression model to the training data.

5.1. Steps

1. **Initialize Model**: Create a `LinearRegression` object.
2. **Fit Model**: Train on `X_train` (scaled features) and `y_train` (prices) using `model.fit()`.
3. **Output**: The model learns coefficients ((\beta_1, \beta_2, \beta_3)) and intercept ((\beta_0)).

5.2. Coefficients

After training, the model assigns weights to each feature:

- **bedrooms**: Positive coefficient (e.g., ~1,000,000 PKR per bedroom), reflecting increased value with more bedrooms.
- **baths**: Positive coefficient (e.g., ~500,000 PKR per bathroom), as bathrooms indicate luxury.
- **Total_Area**: Large positive coefficient (e.g., ~10,000 PKR per sq.ft.), as area is the primary price driver.
- **Intercept**: Baseline price when all features are zero (often negative due to scaling but adjusted in predictions).

5.3. Computational Details

- **Library**: Scikit-learn optimizes the least-squares solution using singular value decomposition (SVD).
- **Time Complexity**: (O(n \cdot d^2)), where (n) is the number of samples (~80,000) and (d) is the number of features (3). Training is fast (<1 second on standard hardware).
- **Memory**: Minimal, as the dataset fits in memory (~100 MB post-preprocessing).

6. Model Evaluation

The model is evaluated on the test set to assess its performance on unseen data.

6.1. Metrics

- **Mean Squared Error (MSE)**:
  - Formula: (\text = \frac{1} \sum_^n (\text_i - \text_i)^2)
  - Measures average squared error in PKR².
  - Example: MSE ≈ 1.23e13 (highly variable due to large price values).
- **R² Score**:
  - Formula: (R^2 = 1 - \frac{\sum (\text_i - \text_i)^2}{\sum (\text_i - \text{mean(y_true)})^2})
  - Proportion of variance explained (0–1).
  - Example: R² ≈ 0.65, meaning 65% of price variance is explained by the model.
- **Root Mean Squared Error (RMSE)**:
  - Formula: (\text = \sqrt{\text})
  - Error in PKR, more interpretable.
  - Example: RMSE ≈ 3,500,000 PKR, indicating typical prediction error.

6.2. Performance Analysis

- **R² = 0.65**: Moderate fit, suggesting the model captures significant price variation but misses some factors (e.g., location, property type).
- **RMSE ≈ 3.5M PKR**: Errors are substantial relative to typical house prices (5M–50M PKR), indicating room for improvement.
- **Residual Plot**: Residuals (errors) show some non-linearity, especially for high-priced properties, suggesting linear regression may not fully capture complex relationships.

6.3. Validation

- **Test Set**: 20% of data ensures unbiased evaluation.
- **Random Seed**: `random_state=101` ensures reproducibility.
- **No Cross-Validation**: Single train-test split is sufficient for large datasets, but k-fold cross-validation could provide more robust metrics.

7. Prediction Functionality

The model includes a user interface to predict prices for new properties.

7.1. Process

1. **Input Collection**:
   - Prompt user for `bedrooms`, `baths`, and `Total_Area`.

    - Example: `bedrooms=4`, `baths=3`, `Total_Area=2500`.
2. **Input Validation**:
    - Ensure inputs are numerical and non-negative.
    - Cap `Total_Area` at 50,000 sq.ft. to match preprocessing.
3. **Scaling**:
    - Transform inputs using the same `StandardScaler` fitted on training data.
4. **Prediction**:
    - Use `model.predict()` to compute the price.
    - Ensure non-negative output using `max(predicted_price, 0)`.
5. **Output**:
    - Display price in PKR with formatting (e.g., `PKR 18,750,000.00`).

7.2. Example

- **Input**: `bedrooms=4`, `baths=3`, `Total_Area=2500 sq.ft.`
- **Scaled Input**: Transformed to standardized values (e.g., [0.5, 0.3, -0.2]).
- **Prediction**: Model outputs ~18.75M PKR, based on learned coefficients.
- **Output**: `Predicted Price: PKR 18,750,000.00`.

7.3. Error Handling

- **ValueError**: Catches non-numerical inputs (e.g., "abc").
- **Negative Inputs**: Rejects negative values for `bedrooms`, `baths`, or `Total_Area`.
- **Extreme Areas**: Rejects `Total_Area > 50,000 sq.ft.` to align with preprocessing.

8. Implementation Details

The model is implemented in Python using the following libraries:

- **Pandas**: Data loading and manipulation (`pd.read_csv`, `df.dropna`).
- **NumPy**: Array operations (`np.array`).
- **Scikit-learn**:
    - `LinearRegression`: Model training and prediction.
    - `StandardScaler`: Feature scaling.
    - `train_test_split`: Data splitting.
    - `mean_squared_error`, `r2_score`: Evaluation metrics.

Script Structure

- **Load Data**: Read `pakistan_house_price_data.csv`.
- **Preprocess**: Filter, clean, scale, and split data.
- **Train**: Fit linear regression model.
- **Evaluate**: Compute MSE and R^2 on test set.

- **Predict**: Interactive function for user inputs.

Dependencies

```
pip install pandas numpy scikit-learn
```

Execution

- Save the script as `house_price_prediction_pakistan_real.py`. •

Place `pakistan_house_price_data.csv` in the same directory.

- Run: `python house_price_prediction_pakistan_real.py`.

9. Limitations

The model has several limitations that affect its performance and applicability:

1. **Limited Features**:
    - Only three features (`bedrooms`, `baths`, `Total_Area`) are used, ignoring potentially important factors like `property_type`, `location`, or proximity to amenities.
    - Example: A flat in F-7 (premium area) may be priced higher than a house in Ghauri Town with similar features.
2. **Linear Assumption**:
    - Linear regression assumes linear relationships, but price may have non-linear dependencies (e.g., diminishing returns for extra bedrooms in small properties).
3. **Outlier Sensitivity**:
    - Despite capping, residual outliers (e.g., luxury villas) may skew predictions.
4. **Data Quality**:
    - Potential errors in `Total_Area` (e.g., 2,722,500 sq.ft.) or `price` (e.g., typos) affect model reliability. ∘ Inconsistent data entry (e.g., varying definitions of "bedrooms") may introduce noise.
5. **Generalization**:
    - The model is trained on Islamabad data, so it may not generalize to other cities (e.g., Karachi, Lahore) with different market dynamics.
6. **Evaluation**:
    - $R^2 \approx 0.65$ indicates moderate fit, missing ~35% of price variance.
    - Large RMSE (~3.5M PKR) suggests predictions may be off for individual properties.

10. Potential Improvements

To enhance the model, consider the following:

1. **Incorporate Categorical Features**:
    - Encode `property_type` (e.g., one-hot encoding for House, Flat) and `location` (e.g., top 10 areas).

- Example: Add dummy variables for `property_type=House` vs. `Flat`.

2. **Advanced Models**:
   - Use Random Forest or Gradient Boosting (e.g., XGBoost) to capture non-linear relationships.
   - Example: Random Forest can model interactions between `Total_Area` and `location`.
3. **Feature Engineering**:
   - Create features like price per square foot (`price / Total_Area`).
   - Use `latitude` and `longitude` to compute distance to city center or amenities.
   - Add binary features (e.g., `is_corner_property` based on listing descriptions).
4. **Handle Skewness**:
   - Apply log transformation to `price` and `Total_Area` to reduce skewness and stabilize variance.
   - Example: `log_price = np.log1p(price)`.
5. **Cross-Validation**:
   - Implement 5-fold cross-validation to ensure robust performance estimates.
   - Example: Use `cross_val_score` to compute average R^2 across folds.
6. **Outlier Detection**:
   - Use robust methods (e.g., IQR or Isolation Forest) to identify and remove outliers.
   - Example: Remove properties where `Total_Area` deviates significantly from the median.
7. **Location-Based Clustering**:
   - Cluster properties by `latitude` and `longitude` to capture neighborhood effects.
   - Example: Use K-means to group properties into 10 clusters and include cluster ID as a feature.
8. **Regularization**:
   - Use Ridge or Lasso regression to reduce overfitting and handle multicollinearity.
   - Example: `Ridge(alpha=1.0)` penalizes large coefficients.

11. Conclusion

The house price prediction model uses linear regression to estimate property sale prices in Islamabad based on `bedrooms`, `baths`, and `Total_Area`. The model achieves moderate performance ($R^2 \approx 0.65$, RMSE ≈ 3.5M PKR) and provides a user-friendly interface for predicting prices of new properties. While effective for a baseline, the model's simplicity and limited feature set leave room for improvement. Incorporating categorical features, using advanced algorithms, and enhancing preprocessing could significantly boost accuracy and applicability.

For practical use, users should:

- Input realistic values (e.g., `Total_Area` < 50,000 sq.ft.).
- Interpret predictions as estimates, given the RMSE.
- Consider local market trends (e.g., premium areas like F-7) not captured by the model.

Future work could focus on integrating more features, exploring non-linear models, and validating the model on other Pakistani cities to create a more robust real estate prediction tool.

12. References

- Scikit-learn Documentation: https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/) •

Zameen.com (data source): https://www.zameen.com/ (https://www.zameen.com/)

- Python Libraries: Pandas, NumPy, Scikit-learn

Appendix: Model Script The Python script implementing the model is provided below for reference. It includes preprocessing, training, evaluation, and prediction functionality.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
df = pd.read_csv('pakistan_house_price_data.csv')

# Filter for "For Sale" entries
df = df[df['purpose'] == 'For Sale']

# Select features and target
features = ['bedrooms', 'baths', 'Total_Area']
target = 'price'

# Drop rows with missing values in selected columns
df = df[features + [target]].dropna()

# Remove outliers: cap Total_Area and price
df = df[df['Total_Area'] <= 50000]
df = df[df['price'] <= 500000000]

# Split features and target
X = df[features]
y = df[target]

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=101
)

# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'R^2 Score: {r2:.2f}')

# Function to predict price for new property
def predict_price(bedrooms, baths, total_area):
    input_data = np.array([[bedrooms, baths, total_area]])
    input_scaled = scaler.transform(input_data)
    predicted_price = model.predict(input_scaled)[0]
    return max(predicted_price, 0)

# User input for prediction
print("\nEnter property details to predict price (in PKR):")
try:
    bedrooms = float(input("Number of bedrooms (e.g., 3): "))
    baths = float(input("Number of bathrooms (e.g., 2): "))
    total_area = float(input("Total area in square feet (e.g., 2000): "))

    if bedrooms < 0 or baths < 0 or total_area < 0:
        print("Error: Inputs cannot be negative.")
    elif total_area > 50000:
        print("Error: Total area exceeds reasonable limit (50,000 sq.ft.).")
    else:
        predicted_price = predict_price(bedrooms, baths, total_area)
        print(f'\nPredicted Price: PKR {predicted_price:,.2f}')
except ValueError:
    print("Error: Please enter valid numerical values.")
```

Usage Instructions:

1. Install dependencies: `pip install pandas numpy scikit-learn`.
2. Place `pakistan_house_price_data.csv` in the same directory.
3. Run: `python house_price_prediction_pakistan_real.py`.
4. Enter property details to get a price prediction.