# Real Estate Price RF Regressor

*Table of Contents:*

# Executive Summary

This project presents a comprehensive machine learning solution for predicting real estate transaction prices in Dubai, leveraging a robust dataset of over 31,000 property records that contain a varied price range from the low thousands to the high ten millions. The dataset includes detailed features such as property size, transaction size, location coordinates, property type, project, number of bedrooms, parking details, and proximity to amenities. Furthermore, additional datasets containing updated & accurate coordinates of real-world locations were implemented to boost model performance.

# Motivation

***Market Dynamics:***

The UAE real estate market is distinguished by its quick development, regular regulatory adjustments, and notable regional and property-type demand swings. These dynamic changes are too rapid for traditional appraisal techniques, which frequently rely on manual evaluations or static historical averages. Agile, data-driven pricing solutions that can swiftly adjust to changing market conditions are becoming more and more necessary as new neighborhoods are created and existing regions are transformed. This model can continuously learn from the most recent transaction data by utilizing machine learning. This allows it to identify subtle, neighborhood-specific trends and provide current price predictions that accurately reflect the state of the market.

***Data limitations:***

One of the persistent challenges in UAE property valuation is the presence of a large number of neighborhoods, many of which have relatively few recorded transactions. Because traditional statistical models frequently need a significant quantity of data for each category to yield correct findings, this sparsity makes it challenging for them to estimate prices accurately.

***Regulatory Need:***

Accurate and unbiased property valuations are essential for a range of regulatory and financial activities, including mortgage approvals, taxation, and investment risk

assessment. Subjective or manual evaluations may generate biases and inconsistencies, eroding process confidence. This solution ensures that all stakeholders, banks, investors, and homeowners have access to fair and objective price estimates while supporting regulatory compliance by automating the valuation procedure with a visible and auditable machine learning pipeline. Regulatory control is facilitated and trust is further strengthened by the provision of traceable and reproducible valuations.

# Technical Requirements

***Data Pipeline:***

1. **TargetEncoder**: Encodes neighborhood categories using mean house prices, reset after each CV fold to prevent leakage.
2. **StandardScaler**: Normalizes numerical features (bedrooms, parking, latitude, longitude, transaction & property size).
3. **RandomForestRegressor**: 500 trees(n_estimators = 500) with min_samples_leaf = 1, min_samples_split = 10, max_depth=30 to balance accuracy/compute.

***Infrastructure:***

- Minimum: Modern 6-Core CPU, 8GB RAM, Recommended: 8-Core 16-Thread 16GB for processing ~30K Property records.
- Python 3.8+ with scikit-learn = 1.7.0, category_encoders = 2.6.

# Main Constructs

## Data Preparation and Cleaning

- **Data Gathering:** The dataset Dubai Residential Formatted.csv was provided directly by my clients. In contrast, the datasets metro_stations.csv and mall_data.csv were independently created to supplement features to enrich the model performance.
- **Data Integrity:** Rigorous preprocessing steps were implemented to, including the removal of redundant & inaccurate features, standardization of data types, along with handling of null values, etc.
- **Outlier Treatment:** IQR-based filtering was used to remove the most extreme & possibly inaccurate samples with a very lenient filter. Once filtered, winsorization was performed to maintain the highly varied data while reducing their influence on the model. This was done to preserve the integrity of the data and ensure the model is robust to outliers while allowing it to perform sufficiently well on most price ranges.

## Feature Engineering

- **Spatial Clustering:** Latitude and longitude data were used to create location-based clusters via K-means, capturing local market effects and enhancing spatial awareness in the model. Therefore, capturing subtle location-based patterns.
- **Categorical Encoding:** High-cardinality categorical features (such as area, property type, project, etc.) were encoded using target encoding, optimizing information retention without inflating dimensionality.
- **Feature Transformation:** The parking feature was transformed from a mix of identifiers and ambiguous values into a clear numeric feature representing the number of parking spots per property, with unclear entries removed for data integrity. Nearest Metro & Mall features were updated accurately through a function utilizing the euclidean distance between locations and cross-referenced with metro & mall data.

# Modeling and Evaluation

- **Data Pipeline:** A data pipeline was constructed using StandardScaler and TargetEncoder along with the model to prevent data leakage from the test set. This was crucial because any leakage could distort the TargetEncoder's estimates of the average target values, potentially leading to overfitting.
- **Model Selection:** RandomForestRegressor was tuned using GridSearchCV to maximize predictive performance.
    - **Handles Non-Linearity:** It can capture complex, non-linear relationships in the data that other models might underperform at.
    - **Reduces Overfitting:** By combining multiple decision trees and averaging predictions it reduces the risk of overfitting compared to a single decision tree.
    - **Robust to Outliers:** Random Forest is less sensitive to outliers as it aggregates the predictions from multiple trees which is essential for a real estate dataset such as this that contains a wide variety of samples.
- **Feature Importance:** Model-driven feature importance analysis guided the removal of non-contributory features such as ‘Master Project’, ‘Nearest Landmark’, simplifying the model and improving interpretability and further optimizing features with significant importance to the model.
- **Performance Metrics:**
    - **$R^2$ Score:** Achieved approximately **0.95**, indicating that the model’s features explains **95%** of the variance in property prices—an excellent result for real estate prediction.
    - **MAE**: The mean absolute error is **138,915 AED**, which is only **6.6%** of the mean property price and **8.8%** of the standard deviation of prices in the test set, demonstrating strong predictive accuracy.
    - **RMSE**: Achieved an average prediction error of about **241,000 AED**. This is a reasonable error given the typical property price, indicating the model performs well relative to the data’s scale and variability. This RMSE represents about **11.7%** of the mean price and **15.5%** of the standard deviation.

- Robustness: The model's average error is low compared to both the mean and spread of prices, confirming its practical utility for stakeholders.

## Real-World Applications

This solution enables a range of practical applications for real estate professionals, investors, and urban planners, including:

- Automated property valuation and appraisal based on commonly-retrieved without requiring the medium of a professional.
- Identification of under- or over-valued properties.
- Support for pricing strategies and investment decisions

# Implementation

Function to retrieve updated & accurate nearest places of interest features based on euclidean distance between property coordinates & nearest place of interest coordinates

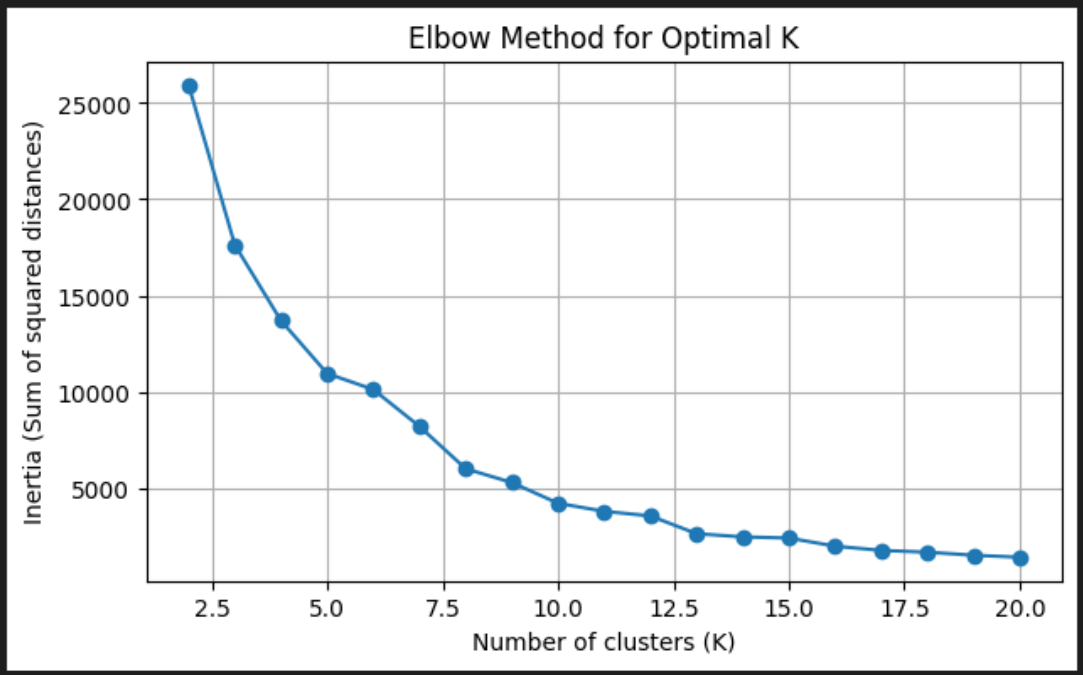
```
# Dataset contains outdated & incorrect nearest metro stations & malls
# Find nearest stations based on real station & mall locations to the euclidean distance to each location
def find_nearest(property_lat, property_lon, nearby_file):
    property_point = (property_lat, property_lon)

    nearby_file["dist"] = nearby_file.apply(lambda row: distance.euclidean((row["latitude"], row["longitude"]), property_point), axis=1)

    # Find the row with the minimum distance
    nearest_place_row = nearby_file.loc[nearby_file["dist"].idxmin()]

    return nearest_place_row["name"]

# Apply function to Nearest Metro & Mall Features
data['Nearest Metro'] = data.apply(
    lambda row: find_nearest(row['Latitude_Project'], row['Longitude_Project'], metro_data), axis=1
)

data['Nearest Mall'] = data.apply(
    lambda row: find_nearest(row['Latitude_Project'], row['Longitude_Project'], mall_data), axis=1
)

print("Updated 'Nearest Metro' & 'Nearest Mall' columns based on geographical distance")
print(data[['Latitude_Project', 'Longitude_Project', 'Nearest Metro', 'Nearest Mall']].head())
```

```
Updated 'Nearest Metro' & 'Nearest Mall' columns based on geographical distance
   Latitude_Project  Longitude_Project                      Nearest Metro  \
0          25.11072           55.38869                Creek Metro Station
1          24.99426           55.16346               Danube Metro Station
2          25.07052           55.14381  Jumeirah Lakes Towers Metro Station
3          25.11000           55.20418   Mall of the Emirates Metro Station
4          25.11000           55.20418   Mall of the Emirates Metro Station

           Nearest Mall
0     Dubai Outlet Mall
1      Ibn Battuta Mall
2     Dubai Marina Mall
3  Mall of the Emirates
4  Mall of the Emirates
```

```
iqr_multiplier = 1.5

# Calculate Q1 and Q3 for numerical columns
```

```
# Selecting optimal no. of clusters for new feature location clusters
k_values = range(2, 21)

inertias = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(coords_scaled)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(7, 4))
plt.plot(k_values, inertias, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of clusters (K)')
plt.ylabel('Inertia (Sum of squared distances)')
plt.grid()
plt.show()
```

```
[120]  ✓ 0.6s
```




```
kmeans = KMeans(n_clusters=12, random_state=42)  #Elbow
data_cleaned_iqr['Location_Cluster'] = kmeans.fit_predict(coords_scaled)
```

Feature-Engineering ‘Location Clusters’ to provide local neighbourhood patterns in data which is crucial for a dataset as varied as this

Real Estate Price RF Pipeline Hyperparameter Tuning

```
X = data_cleaned_iqr
y = data_cleaned_iqr['Amount_winsorized']

X_train, X_test, y_train, y_test = train_test_split(X[selected_features], y, test_size=0.3, random_state=1)
categorical_cols_selected = X[selected_features].select_dtypes(include=['object']).columns

pipeline = Pipeline([
    ('encoder', ce.TargetEncoder(cols=categorical_cols_selected)),
    ('scaler', StandardScaler()),
    ('regressor', RandomForestRegressor(random_state=42))
])

param_grid = {
    'regressor__n_estimators': [100, 200, 300],
    'regressor__max_depth': [10, 20, 30],
    'regressor__min_samples_split': [2, 5, 10],
    'regressor__min_samples_leaf': [1, 2, 4],
    'regressor__max_features': ['sqrt'],
    'regressor__bootstrap': [True, False]
}


grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', verbose=10, n_jobs=10)
grid_search.fit(X_train, y_train)

print("\nBest hyperparameters found:")
print(grid_search.best_params_)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

print("\nBest Model Evaluation on Test Set (Original Scale):")

mae_tuned = metrics.mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae_tuned)

mse_tuned = metrics.mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse_tuned)

rmse_tuned = np.sqrt(mse_tuned)
print("Root Mean Squared Error:", rmse_tuned)

r2_tuned = metrics.r2_score(y_test, y_pred)
print("R-squared:", r2_tuned)

# ---- Hyperparameter Tuning ----
```

⊗ 5m 26.7s

```
Fitting 3 folds for each of 162 candidates, totalling 486 fits
```

Best-Performing Model

```
# Feature Importances of the model
encoder = ce.TargetEncoder(cols=categorical_cols_selected)
X_train_encoded = encoder.fit_transform(X_train, y_train)
X_test_encoded = encoder.transform(X_test)

model = RandomForestRegressor(random_state=42, bootstrap = False, max_depth= 30, max_features= 'sqrt',
                              min_samples_leaf= 1, min_samples_split= 10, n_estimators= 500)
model.fit(X_train_encoded, y_train)

importance = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': model.feature_importances_
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(importance['Feature'], importance['Importance'])
plt.title('Feature Importance (RandomForestRegressor)')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.gca().invert_yaxis()
plt.show()
```

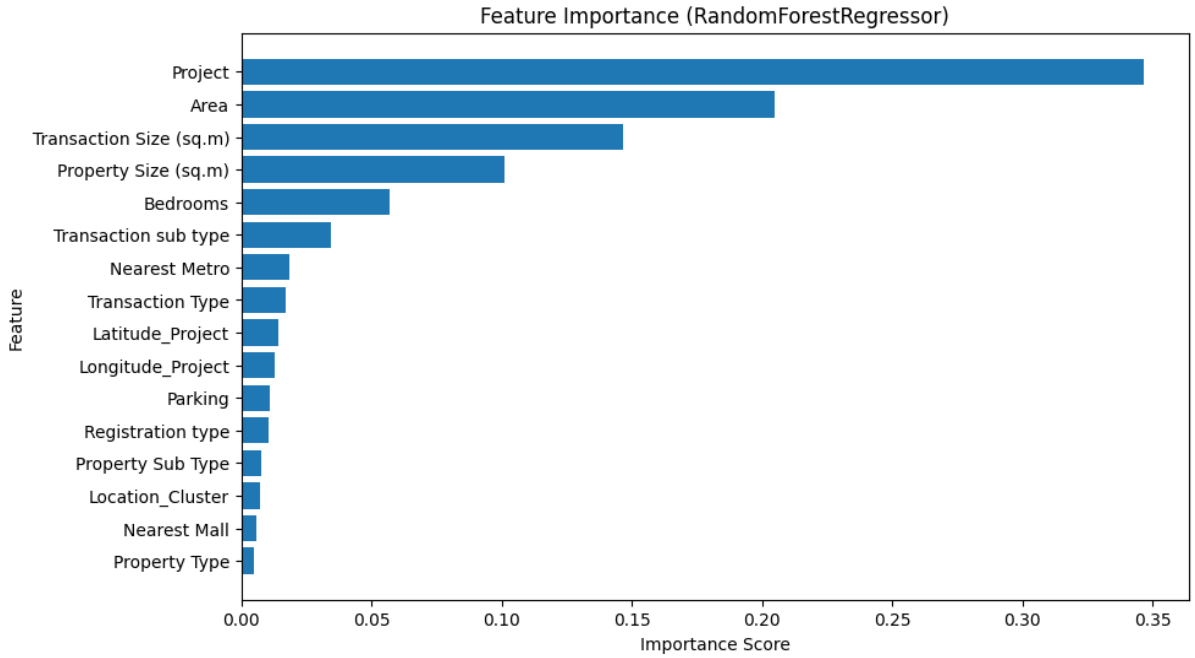
Baseline Linear Regression Model for Comparison

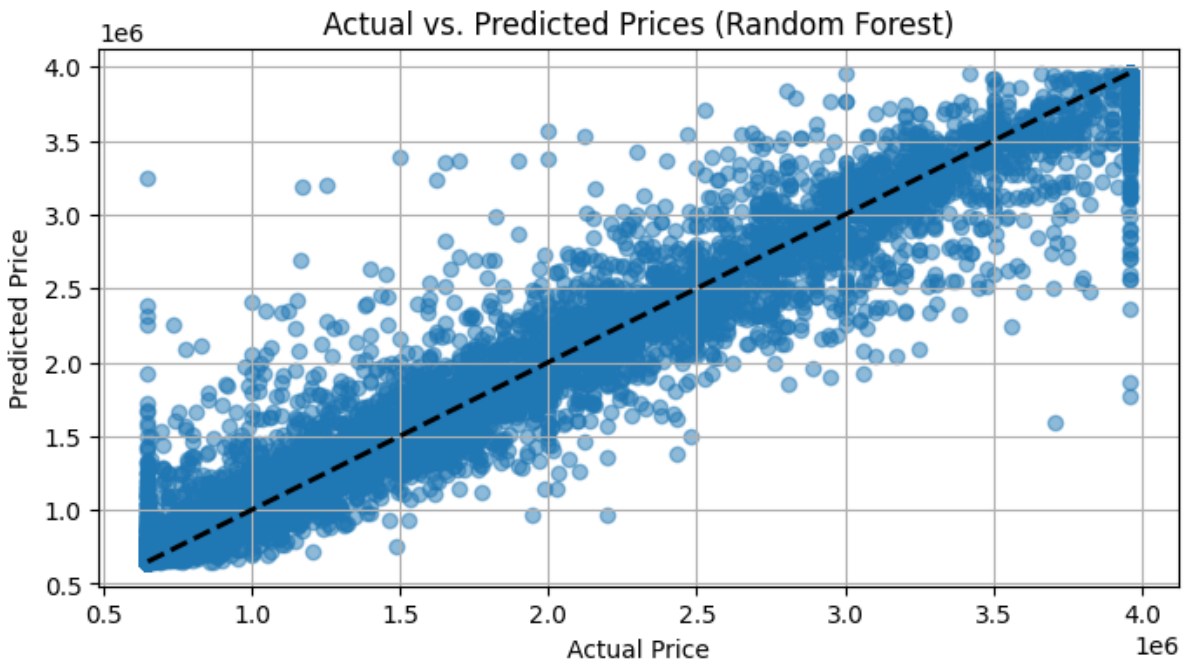
```
pipeline = Pipeline([
    ('encoder', ce.TargetEncoder(cols=categorical_cols_selected)),
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression())
])

pipeline.fit(X_train, y_train)

lr_y_pred = pipeline.predict(X_test)

mae_tuned = metrics.mean_absolute_error(y_test, lr_y_pred)
print("Mean Absolute Error:", mae_tuned)

mse_tuned = metrics.mean_squared_error(y_test, lr_y_pred)
print("Mean Squared Error:", mse_tuned)

rmse_tuned = np.sqrt(mse_tuned)
print("Root Mean Squared Error:", rmse_tuned)

r2_tuned = metrics.r2_score(y_test, lr_y_pred)
print("R-squared:", r2_tuned)

# ----Linear Regression Comparison----
```

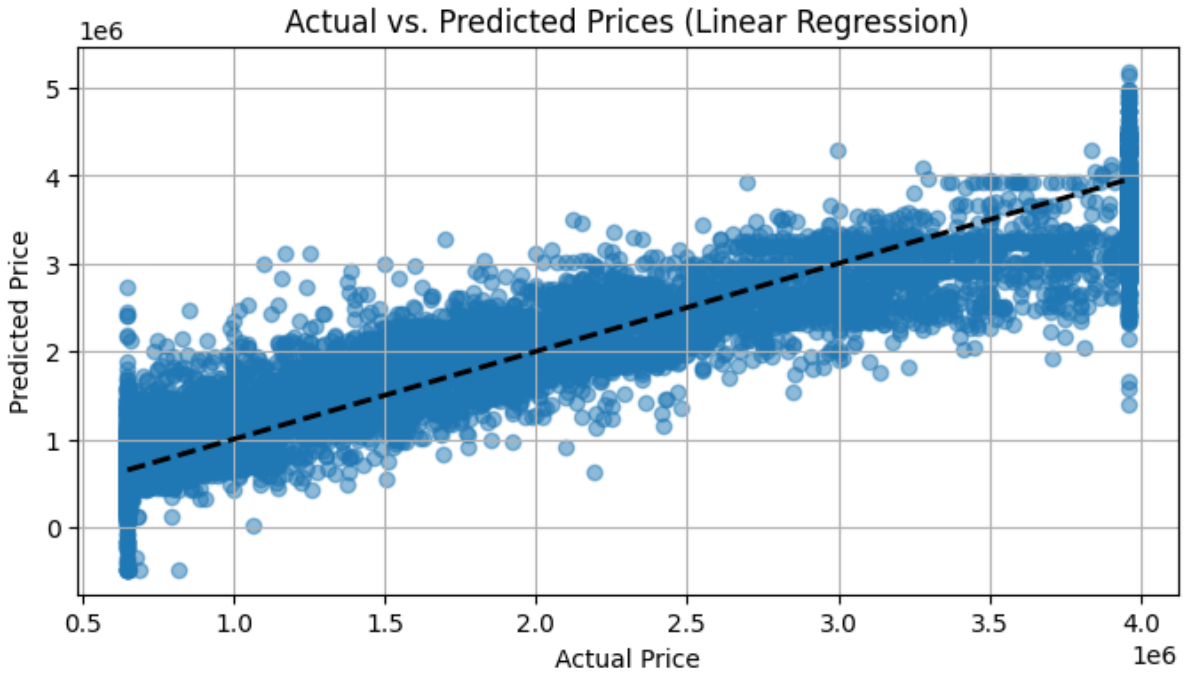
✓ 0.2s

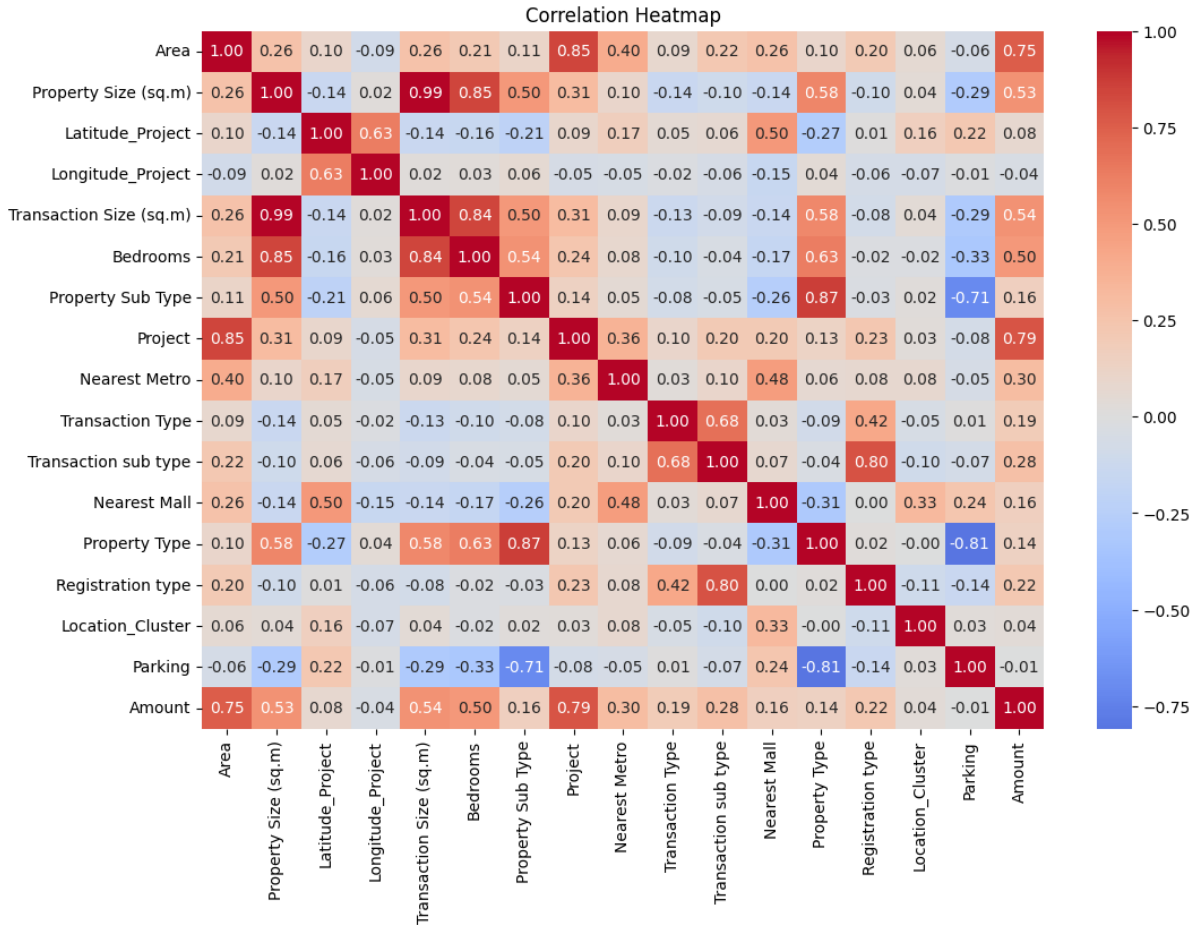
```
Mean Absolute Error: 327544.5286621375
Mean Squared Error: 182198654032.5064
Root Mean Squared Error: 426847.3427731587
R-squared: 0.8429451092774104
```

# Evaluation


Feature Importance (RandomForestRegressor)
Project
Area
Transaction Size (sq.m)
Property Size (sq.m)
Bedrooms
Transaction sub type
Nearest Metro
Transaction Type
Latitude_Project
Longitude_Project
Parking
Registration type
Property Sub Type
Location_Cluster
Nearest Mall
Property Type
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Importance Score


Actual vs. Predicted Prices (Random Forest)
1e6
Predicted Price
Actual Price
1e6
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Actual vs. Predicted Prices (Linear Regression)
1e6
Predicted Price
Actual Price
1e6
0
1
2
3
4
5
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0




# Limitations/Difficulties and Constraints

These drawbacks point to potential areas for improvement in the model, like adding real-time market signals to increase robustness during volatile times, enhancing encoding techniques for sparse categories, or incorporating other data sources.

**Data Sensitivity:**

- Sparse Data for Ultra-Luxury Properties:

  For properties valued at more than AED 20 million, the model's predicted accuracy diminishes. This is mostly because there aren't many of these transactions in the training data, which limits the model's capacity to identify distinctive pricing trends and market dynamics particular to this market niche. Therefore, compared to mid-range or mass-market homes, price estimates for high-end properties may be less accurate and show more volatility.

- Neighbourhood Encoding in Low-Volume Areas:

  For Target Encoding to calculate stable average pricing, there must be a sufficient amount of transactions per neighborhood. Less accurate forecasts may result from noisy or unrepresentative encoded values in neighborhoods with less than 50 past

transactions. In freshly created or less active locations, where transaction data is few and price patterns are not well established, this constraint is especially relevant.

- Challenges from Data Inconsistency and Missing Values:

  The data quality with the original dataset had a significant impact on the model's performance. Extensive data cleaning & preprocessing were necessary for items that were inconsistent and inaccurate, such as missing values, redundant features, and unclear or improperly formatted data (Parking's improperly formatted data, Inaccurate/Outdated nearest Places of Interest, etc). These problems added uncertainty to the model's training phase in addition to making feature engineering more difficult.

**Model Constraints:**

- Prediction latency:

  It can take up to 2 minutes for the current model while taking up to 20 minutes to perform GridSearchCV analysis on the model, which is tuned for accuracy and consistency, to provide a price estimate from raw input data, particularly when processing big batches or during retraining cycles. For platforms that need immediate valuations or real-time applications, this latency could be a hindrance.

- Inability to Capture Unit-Specific Features:

  Features like the number of bedrooms, area, and neighborhood are the main focus of the model, which is made to function with organized tabular data. It ignores unit-specific factors like recent renovations, interior design caliber, distinctive views, or building amenities that can have a big impact on cost. For these elements to be effectively assessed, manual inspection or additional unstructured data (such as textual descriptions or photographs) are frequently needed.

- Sensitivity During Market Shocks:

  During periods of rapid market change such as economic downturns, regulatory shifts, or sudden demand surges, the model's error rate can increase. This is because machine learning models trained on historical data may lag in adapting to abrupt shifts, leading to temporary misalignment between predicted and actual market values. Frequent re-training of the model can help hamper the effect of these changes.

# Conclusion(Discussion & Remarks)

Through meticulous data preparation, advanced feature engineering, and rigorous model validation, this project delivers a high-performing, interpretable, and practical machine learning model for real estate price prediction in Dubai. The methodology ensures reliability, scalability, and actionable insights, providing a strong foundation for deployment in real-world business and urban analytics contexts.

***Discussion:***

The architecture of the model tackles a number of persistent issues in the UAE real estate market. The Random Forest algorithm's ensemble method reduces overfitting and improves generalization across a wide range of property kinds and price ranges, while its use of Target Encoding allows it to handle the nation's various and frequently sparsely inhabited neighborhoods effectively. Because of its versatility and reproducibility, the model may be integrated into digital real estate platforms to facilitate automated investment decision-making, portfolio analysis, and appraisal.

However, the model does have certain drawbacks. Ultra-luxury houses and neighborhoods with little transaction history result in worse forecasting accuracy, which reflects the industry's larger problem of data sparsity in niche markets. Furthermore, because of processing latency, the existing approach does not provide real-time forecasts for every use case and does not capture unit-specific qualitative data that frequently affect final sale prices, such as refurbishment quality or unique vistas.

***Remarks:***

In a market that is constantly changing, this model is a major advancement in automating and standardizing property appraisals. Adoption may speed up transaction workflows, lessen human bias, and give a variety of stakeholders - from ordinary consumers to institutional investors transparent, data-driven information. The use of hybrid models that combine automated and expert-driven valuations for unique or high-value properties, the integration of macroeconomic indicators for increased resilience during market shocks, and image recognition for property photos are some potential future improvements as the UAE real estate ecosystem develops.

In the end, automated models such as this Random Forest model provide a solid, scalable basis for first-pass valuation and market analysis in the UAE's rapidly changing real estate market, even though they won't completely replace human appraisers, particularly for unusual or high-stakes transactions.