

✦ Indonesia Property Price Prediction with Machine Learning

Project Overview

This project predicts property prices in East Java, Indonesia using machine learning. As a full stack web developer expanding into AI/ML, I wanted to create a practical project that connects to my existing work in property technology (Next Level Properti web application).

Challenge

Create a machine learning model that can accurately predict property prices based on features like location, size, and property type.

Learning Goals

- Understand data preparation and exploration
- Learn multiple ML algorithms (Linear Regression, Random Forest, Decision Tree)
- Compare model performance and select the best one
- Make practical predictions with trained models

✦ Step 1: Creating the Dataset

Since real Indonesia property data is limited, I created a synthetic/dummy dataset with realistic patterns based on actual East Java market conditions. This dataset includes 1000 properties across 5 cities in East Java.

```
# Import libraries
import pandas as pd
import numpy as np
import random

# Set random seed so we get same results every time
np.random.seed(42)
random.seed(42)

# Define East Java cities
cities = ['Surabaya', 'Sidoarjo', 'Gresik', 'Malang', 'Batu']

# Define property types
property_types = ['Rumah', 'Apartemen', 'Ruko', 'Villa']

# Define certification types
certifications = ['SHM', 'SHGB', 'HGU']

# Number of properties to generate
n_properties = 1000

# Create empty lists to store data
data = {
    'city': [],
    'property_type': [],
    'land_area': [], # in m²
    'building_area': [], # in m²
    'bedrooms': [],
    'bathrooms': [],
    'carport': [],
    'certification': [],
    'price': [] # in million IDR
}

# Generate data for each property
for i in range(n_properties):
    # Random city
    city = random.choice(cities)

    # Random property type
    prop_type = random.choice(property_types)

    # Generate realistic sizes based on property type
    if prop_type == 'Apartemen':
```

```

        land_area = 0
        building_area = random.randint(30, 150)
        bedrooms = random.randint(1, 3)
        bathrooms = random.randint(1, 2)
        carport = random.randint(0, 2)
    elif prop_type == 'Ruko':
        land_area = random.randint(60, 200)
        building_area = random.randint(80, 300)
        bedrooms = random.randint(0, 2)
        bathrooms = random.randint(1, 3)
        carport = random.randint(1, 3)
    elif prop_type == 'Villa':
        land_area = random.randint(150, 500)
        building_area = random.randint(100, 400)
        bedrooms = random.randint(2, 5)
        bathrooms = random.randint(2, 4)
        carport = random.randint(2, 4)
    else: # Rumah
        land_area = random.randint(60, 300)
        building_area = random.randint(45, 250)
        bedrooms = random.randint(2, 4)
        bathrooms = random.randint(1, 3)
        carport = random.randint(1, 3)

# Certification
cert = random.choice(certifications)

# Calculate price based on realistic factors
# Base price per m² varies by city
city_multipliers = {
    'Surabaya': 12, # Most expensive
    'Sidoarjo': 8,
    'Malang': 9,
    'Gresik': 7,
    'Batu': 10 # Tourist area, pricier
}

# Property type multiplier
type_multipliers = {
    'Rumah': 1.0,
    'Apartemen': 0.9,
    'Ruko': 1.3,
    'Villa': 1.5
}

# Base price calculation
base_price_per_m2 = city_multipliers[city]
type_mult = type_multipliers[prop_type]

# Price = (building_area * base_price * type_mult) + bonuses
base_price = building_area * base_price_per_m2 * type_mult

# Add bonuses for features
bedroom_bonus = bedrooms * 50
bathroom_bonus = bathrooms * 30
carport_bonus = carport * 20
cert_bonus = 100 if cert == 'SHM' else (50 if cert == 'SHGB' else 0)
land_bonus = land_area * 0.3 # Land adds value

# Final price with some random variation (±20%)
final_price = base_price + bedroom_bonus + bathroom_bonus + carport_bonus + cert_bonus + land_bonus
final_price = final_price * random.uniform(0.8, 1.2)

# Round to nearest 50 million
final_price = round(final_price / 50) * 50

# Append to data
data['city'].append(city)
data['property_type'].append(prop_type)
data['land_area'].append(land_area)
data['building_area'].append(building_area)
data['bedrooms'].append(bedrooms)
data['bathrooms'].append(bathrooms)
data['carport'].append(carport)
data['certification'].append(cert)
data['price'].append(final_price)

# Create DataFrame
df = pd.DataFrame(data)

# Show first few rows
print("Dataset created successfully! 🎉")

```

```
print(f"Total properties: {len(df)}")
print("\nFirst 5 properties:")
df.head()
```

Dataset created successfully! 🎉
Total properties: 1000

First 5 properties:

	city	property_type	land_area	building_area	bedrooms	bathrooms	carport	certification	price
0	Surabaya	Rumah	249	115	2	1	1	HGU	1350
1	Batu	Rumah	211	153	2	1	1	SHM	1650
2	Batu	Rumah	203	95	4	3	3	HGU	1300
3	Malang	Ruko	61	274	0	3	2	SHGB	3100
4	Sidoarjo	Ruko	86	103	1	1	2	SHGB	1300

Next steps: [Generate code with df](#) [New interactive sheet](#)

Step 2: Understanding the Data

After generating the dataset, I needed to understand what I was working with. This exploration phase helped me identify patterns and check if the data made sense before building any models.

I looked at basic statistics like the distribution of properties across cities, property types, and price ranges. This step is crucial because if the data looks wrong here, the machine learning models won't work properly later.

```
# Basic statistics about our dataset
print("=== Dataset Overview ===")
print(df.info())
print("\n=== Statistical Summary ===")
print(df.describe())

print("\n=== Distribution by City ===")
print(df['city'].value_counts())

print("\n=== Distribution by Property Type ===")
print(df['property_type'].value_counts())

print("\n=== Distribution by Certification ===")
print(df['certification'].value_counts())

print("\n=== Price Statistics (in million IDR) ===")
print(f"Minimum price: Rp {df['price'].min():,.0f} juta")
print(f"Maximum price: Rp {df['price'].max():,.0f} juta")
print(f"Average price: Rp {df['price'].mean():,.0f} juta")
print(f"Median price: Rp {df['price'].median():,.0f} juta")
```

```
5  bathrooms      1000 non-null    int64
6  carport        1000 non-null    int64
7  certification  1000 non-null    object
8  price          1000 non-null    int64
dtypes: int64(6), object(3)
memory usage: 70.4+ KB
None
```

```
=== Statistical Summary ===
      land_area  building_area  bedrooms  bathrooms  carport  \
count  1000.000000    1000.000000  1000.000000  1000.000000  1000.000000
mean    150.445000    168.432000    2.389000    2.119000    1.979000
std     131.980688     89.360958    1.291258    0.900917    1.050554
min       0.000000     30.000000    0.000000    1.000000    0.000000
25%       0.000000     98.000000    2.000000    1.000000    1.000000
50%     135.500000    147.000000    2.000000    2.000000    2.000000
75%     229.250000    232.000000    3.000000    3.000000    3.000000
max     493.000000    399.000000    5.000000    4.000000    4.000000

      price
count  1000.000000
```

```

Matang      215
Sidoarjo    202
Gresik      183
Surabaya    182
Name: count, dtype: int64

=== Distribution by Property Type ===
property_type
Apartemen    271
Rumah        252
Ruko         248
Villa        229
Name: count, dtype: int64

=== Distribution by Certification ===
certification
HGU         340
SHM         330
SHGB        330
Name: count, dtype: int64

=== Price Statistics (in million IDR) ===
Minimum price: Rp 300 juta
Maximum price: Rp 7,850 juta
Average price: Rp 2,224 juta
Median price: Rp 1,850 juta

```

▼ Step 3: Visualizing the Data

Numbers alone don't tell the full story, so I created visualizations to see the patterns more clearly. I wanted to understand how different factors relate to property prices.

The charts helped me confirm that building area has a strong relationship with price, and that location matters - Surabaya properties are consistently more expensive than other cities in East Java.

```

# Import visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Set style for better looking plots
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (15, 10)

# Create a figure with multiple subplots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# 1. Price Distribution
axes[0, 0].hist(df['price'], bins=30, color='skyblue', edgecolor='black')
axes[0, 0].set_title('Property Price Distribution', fontsize=14, fontweight='bold')
axes[0, 0].set_xlabel('Price (Million IDR)')
axes[0, 0].set_ylabel('Frequency')

# 2. Average Price by City
city_prices = df.groupby('city')['price'].mean().sort_values()
axes[0, 1].barh(city_prices.index, city_prices.values, color='coral')
axes[0, 1].set_title('Average Price by City', fontsize=14, fontweight='bold')
axes[0, 1].set_xlabel('Average Price (Million IDR)')

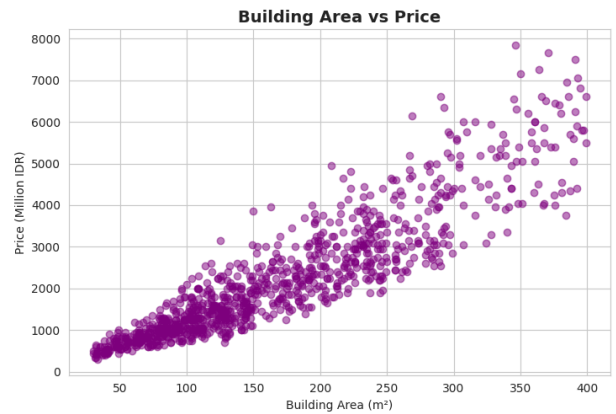
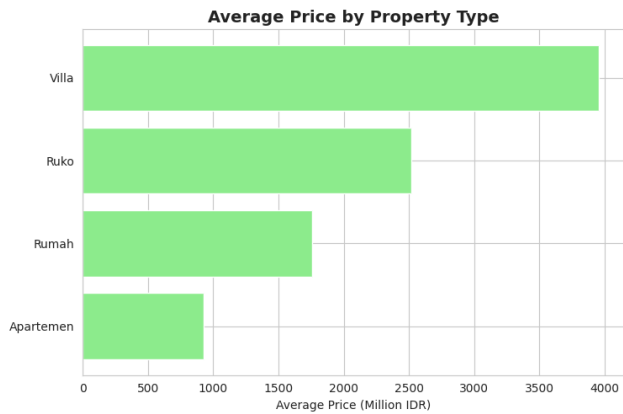
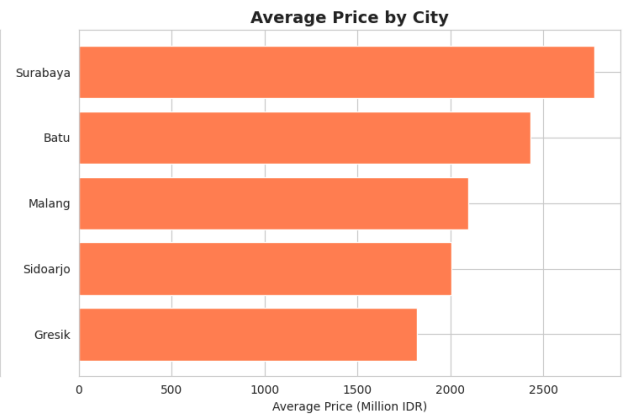
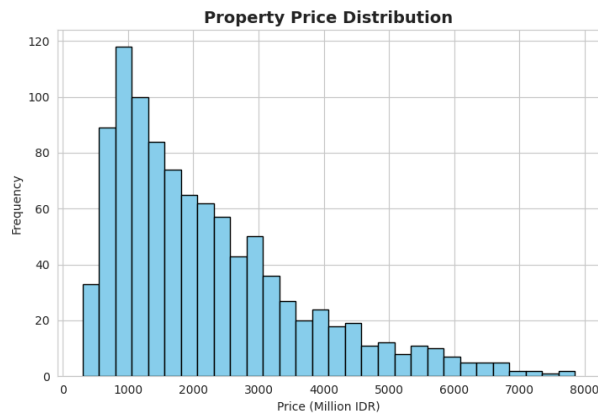
# 3. Average Price by Property Type
type_prices = df.groupby('property_type')['price'].mean().sort_values()
axes[1, 0].barh(type_prices.index, type_prices.values, color='lightgreen')
axes[1, 0].set_title('Average Price by Property Type', fontsize=14, fontweight='bold')
axes[1, 0].set_xlabel('Average Price (Million IDR)')

# 4. Building Area vs Price
axes[1, 1].scatter(df['building_area'], df['price'], alpha=0.5, color='purple')
axes[1, 1].set_title('Building Area vs Price', fontsize=14, fontweight='bold')
axes[1, 1].set_xlabel('Building Area (m²)')
axes[1, 1].set_ylabel('Price (Million IDR)')

plt.tight_layout()
plt.show()

print("✅ Visualizations created!")

```



✅ Visualizations created!

Step 4: Preparing Data for Machine Learning

Machine learning models can only work with numbers, but my dataset had text values like city names and property types. I needed to convert these into numbers through a process called encoding.

For example, Surabaya became 4, Malang became 2, and so on. I also split the data into two parts: 80% for training the models and 20% for testing how well they perform on data they haven't seen before.

```
# Prepare data for machine learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Create a copy of our data
df_ml = df.copy()

# Convert categorical variables to numbers
le_city = LabelEncoder()
le_property_type = LabelEncoder()
le_certification = LabelEncoder()

df_ml['city_encoded'] = le_city.fit_transform(df_ml['city'])
df_ml['property_type_encoded'] = le_property_type.fit_transform(df_ml['property_type'])
df_ml['certification_encoded'] = le_certification.fit_transform(df_ml['certification'])

# Show the encoding mapping
```

```

print("=== Encoding Mappings ===")
print("\nCity Encoding:")
for i, city in enumerate(le_city.classes_):
    print(f" {city} -> {i}")

print("\nProperty Type Encoding:")
for i, ptype in enumerate(le_property_type.classes_):
    print(f" {ptype} -> {i}")

print("\nCertification Encoding:")
for i, cert in enumerate(le_certification.classes_):
    print(f" {cert} -> {i}")

# Select features (X) and target (y)
feature_columns = ['city_encoded', 'property_type_encoded', 'land_area',
                   'building_area', 'bedrooms', 'bathrooms', 'carport',
                   'certification_encoded']

X = df_ml[feature_columns]
y = df_ml['price']

# Split data: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"\nData prepared!")
print(f"Training set: {len(X_train)} properties")
print(f"Testing set: {len(X_test)} properties")
print(f"\nFeatures we're using: {feature_columns}")

```

```

=== Encoding Mappings ===

```

```

City Encoding:
Batu -> 0
Gresik -> 1
Malang -> 2
Sidoarjo -> 3
Surabaya -> 4

```

```

Property Type Encoding:
Apartemen -> 0
Ruko -> 1
Rumah -> 2
Villa -> 3

```

```

Certification Encoding:
HGU -> 0
SHGB -> 1
SHM -> 2

```

```

Data prepared!
Training set: 800 properties
Testing set: 200 properties

```

```

Features we're using: ['city_encoded', 'property_type_encoded', 'land_area', 'building_area', 'bedrooms', 'bathrooms', 'carport', 'certification_encoded']

```

Step 5: Training the First Model - Linear Regression

Linear Regression was my starting point because it's one of the simplest machine learning algorithms. It tries to find a straight-line relationship between the features and the price.

I measured the model's performance using three metrics: Mean Absolute Error (how far off predictions are on average), Root Mean Squared Error (penalizes larger errors more), and R^2 Score (what percentage of price variation the model can explain).

```

# Import Linear Regression model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Create and train the model
print("Training Linear Regression model...")
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Make predictions on test data
y_pred_lr = lr_model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred_lr)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2 = r2_score(y_test, y_pred_lr)

```

```

print("\n=== Linear Regression Results ===")
print(f"Mean Absolute Error (MAE): Rp {mae:,.0f} juta")
print(f"Root Mean Squared Error (RMSE): Rp {rmse:,.0f} juta")
print(f"R² Score: {r2:.4f} ({r2*100:.2f}% accuracy)")

print("\nWhat this means:")
print(f"- On average, predictions are off by Rp {mae:,.0f} juta")
print(f"- The model explains {r2*100:.1f}% of price variation")

# Show some sample predictions
print("\n=== Sample Predictions (first 5 test properties) ===")
comparison = pd.DataFrame({
    'Actual Price': y_test.head().values,
    'Predicted Price': y_pred_lr[:5].round(0),
    'Difference': (y_test.head().values - y_pred_lr[:5]).round(0)
})
print(comparison)

```

Training Linear Regression model...

```

=== Linear Regression Results ===
Mean Absolute Error (MAE): Rp 444 juta
Root Mean Squared Error (RMSE): Rp 607 juta
R² Score: 0.8572 (85.72% accuracy)

```

What this means:

```

- On average, predictions are off by Rp 444 juta
- The model explains 85.7% of price variation

```

=== Sample Predictions (first 5 test properties) ===

	Actual Price	Predicted Price	Difference
0	2400	2944.0	-544.0
1	2000	1977.0	23.0
2	1850	2572.0	-722.0
3	5500	4931.0	569.0
4	1450	1631.0	-181.0

Step 6: Training Random Forest - A More Advanced Model

After seeing the Linear Regression results, I wanted to try a more sophisticated algorithm. Random Forest works by creating many decision trees and combining their predictions, which usually leads to better accuracy.

The trade-off is that it takes longer to train and is harder to interpret, but the performance improvement is often worth it.

```

# Import Random Forest model
from sklearn.ensemble import RandomForestRegressor

# Create and train Random Forest model
print("🌲 Training Random Forest model...")
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate
mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print("\n=== Random Forest Results ===")
print(f"Mean Absolute Error (MAE): Rp {mae_rf:,.0f} juta")
print(f"Root Mean Squared Error (RMSE): Rp {rmse_rf:,.0f} juta")
print(f"R² Score: {r2_rf:.4f} ({r2_rf*100:.2f}% accuracy)")

print("\n🏠 What this means:")
print(f"- On average, predictions are off by Rp {mae_rf:,.0f} juta")
print(f"- The model explains {r2_rf*100:.1f}% of price variation")

# Compare the two models
print("\n=== Model Comparison ===")
comparison = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest'],
    'MAE (juta)': [mae, mae_rf],
    'RMSE (juta)': [rmse, rmse_rf],
    'R² Score': [r2, r2_rf]
})
print(comparison)

# Show feature importance from Random Forest
print("\n=== Feature Importance (What affects price most?) ===")

```

```
feature_importance = pd.DataFrame({
    'Feature': feature_columns,
    'Importance': rf_model.feature_importances_
}).sort_values('Importance', ascending=False)
print(feature_importance)
```

🌲 Training Random Forest model...

```
=== Random Forest Results ===
Mean Absolute Error (MAE): Rp 276 juta
Root Mean Squared Error (RMSE): Rp 410 juta
R² Score: 0.9348 (93.48% accuracy)
```

🇮🇩 What this means:

- On average, predictions are off by Rp 276 juta
- The model explains 93.5% of price variation

```
=== Model Comparison ===
```

	Model	MAE (juta)	RMSE (juta)	R² Score
0	Linear Regression	443.895196	607.178138	0.857234
1	Random Forest	276.410857	410.328509	0.934799

```
=== Feature Importance (What affects price most?) ===
```

	Feature	Importance
3	building_area	0.851256
0	city_encoded	0.058872
2	land_area	0.037058
1	property_type_encoded	0.028425
4	bedrooms	0.008091
6	carport	0.006046
5	bathrooms	0.005416
7	certification_encoded	0.004835

▼ Step 7: Adding Decision Tree for Comparison

To get a complete picture, I trained a third model using Decision Tree. This algorithm makes predictions by asking a series of yes/no questions about the property features.

Decision Trees are easier to understand than Random Forest but can sometimes overfit the training data. I limited the tree depth to 15 levels to prevent this issue.

```
# Import Decision Tree model
from sklearn.tree import DecisionTreeRegressor

# Create and train Decision Tree model
print("🌲 Training Decision Tree model...")
dt_model = DecisionTreeRegressor(random_state=42, max_depth=15)
dt_model.fit(X_train, y_train)

# Make predictions
y_pred_dt = dt_model.predict(X_test)

# Evaluate
mae_dt = mean_absolute_error(y_test, y_pred_dt)
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_dt = r2_score(y_test, y_pred_dt)

print("\n=== Decision Tree Results ===")
print(f"Mean Absolute Error (MAE): Rp {mae_dt:.0f} juta")
print(f"Root Mean Squared Error (RMSE): Rp {rmse_dt:.0f} juta")
print(f"R² Score: {r2_dt:.4f} ({r2_dt*100:.2f}% accuracy)")

# Updated comparison with all 3 models
print("\n=== 🇮🇩 FINAL MODEL COMPARISON ===")
comparison_final = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'Decision Tree'],
    'MAE (juta)': [mae, mae_rf, mae_dt],
    'RMSE (juta)': [rmse, rmse_rf, rmse_dt],
    'R² Score (%)': [r2*100, r2_rf*100, r2_dt*100]
}).sort_values('R² Score (%)', ascending=False)

print(comparison_final.to_string(index=False))

print(f"\n🏆 Best Model: {comparison_final.iloc[0]['Model']} with {comparison_final.iloc[0]['R² Score (%)']:.2f}% accuracy")
```

🌲 Training Decision Tree model...

```
=== Decision Tree Results ===
Mean Absolute Error (MAE): Rp 372 juta
Root Mean Squared Error (RMSE): Rp 541 juta
R² Score: 0.8868 (88.68% accuracy)
```


=== 🏆 FINAL MODEL COMPARISON 🏆 ===

Model	MAE (juta)	RMSE (juta)	R ² Score (%)
Random Forest	276.410857	410.328509	93.479863
Decision Tree	371.875000	540.673769	88.679543
Linear Regression	443.895196	607.178138	85.723373

🏆 Best Model: Random Forest with 93.48% accuracy

✓ Step 8: Comparing All Three Models Visually

After training all three models, I created side-by-side scatter plots to compare their performance. Each plot shows actual prices versus predicted prices, with a red diagonal line representing perfect predictions.

The closer the dots cluster around the red line, the better the model performs. This visual comparison makes it obvious that Random Forest significantly outperforms the other two models.

```
# Visualize all 3 models performance
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# 1. Linear Regression
axes[0].scatter(y_test, y_pred_lr, alpha=0.5, color='blue')
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[0].set_xlabel('Actual Price (Million IDR)')
axes[0].set_ylabel('Predicted Price (Million IDR)')
axes[0].set_title(f'Linear Regression\nR² = {r2_lr:.3f} (85.72%)', fontweight='bold')
axes[0].grid(True, alpha=0.3)

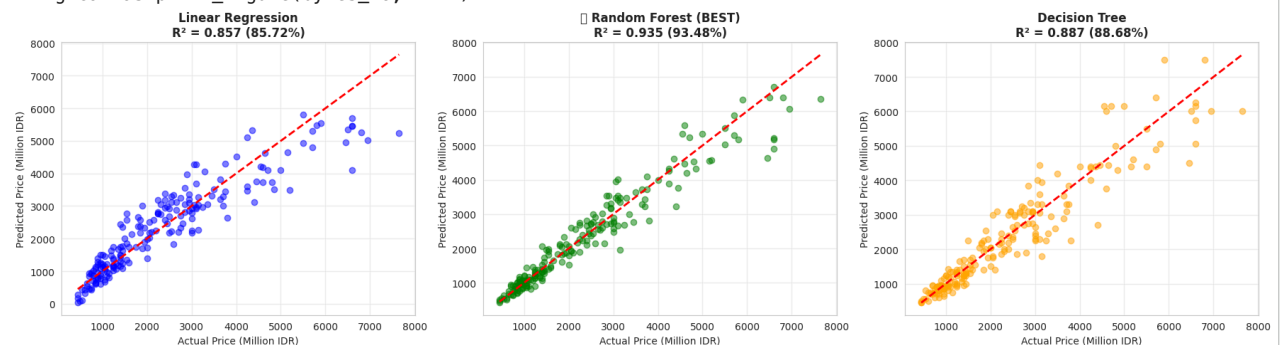
# 2. Random Forest
axes[1].scatter(y_test, y_pred_rf, alpha=0.5, color='green')
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[1].set_xlabel('Actual Price (Million IDR)')
axes[1].set_ylabel('Predicted Price (Million IDR)')
axes[1].set_title(f'Random Forest (BEST)\nR² = {r2_rf:.3f} (93.48%)', fontweight='bold')
axes[1].grid(True, alpha=0.3)

# 3. Decision Tree
axes[2].scatter(y_test, y_pred_dt, alpha=0.5, color='orange')
axes[2].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[2].set_xlabel('Actual Price (Million IDR)')
axes[2].set_ylabel('Predicted Price (Million IDR)')
axes[2].set_title(f'Decision Tree\nR² = {r2_dt:.3f} (88.68%)', fontweight='bold')
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("✅ Random Forest (green, middle) clearly performs best with dots closest to red line!")
```

/tmp/ipython-input-143112703.py:28: UserWarning: Glyph 127942 (\N{TROPHY}) missing from font(s) DejaVu Sans.
plt.tight_layout()
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 127942 (\N{TROPHY}) missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)



✅ Random Forest (green, middle) clearly performs best with dots closest to red line!

Step 9: Testing with a Real Example

To make this practical, I tested the models on a realistic property: a 3-bedroom house in Surabaya with 80 square meters of building area and SHM certification.

Seeing how each model predicts this specific property helps understand what these accuracy percentages mean in real terms.

```
# Practical Example: Predict price for a new property
print("=== PRACTICAL EXAMPLE: Predict New Property Price ===\n")

# Example property in Surabaya
example_property = {
    'City': 'Surabaya',
    'Property Type': 'Rumah',
    'Land Area': 120,
    'Building Area': 80,
    'Bedrooms': 3,
    'Bathrooms': 2,
    'Carport': 1,
    'Certification': 'SHM'
}

print("Property Details:")
for key, value in example_property.items():
    print(f"  {key}: {value}")

# Encode the categorical variables
city_encoded = le_city.transform([example_property['City']])[0]
property_type_encoded = le_property_type.transform([example_property['Property Type']])[0]
certification_encoded = le_certification.transform([example_property['Certification']])[0]

# Create feature array in the same order as training
new_property = np.array([[
    city_encoded,
    property_type_encoded,
    example_property['Land Area'],
    example_property['Building Area'],
    example_property['Bedrooms'],
    example_property['Bathrooms'],
    example_property['Carport'],
    certification_encoded
]])

# Predict with all 3 models
pred_lr = lr_model.predict(new_property)[0]
pred_rf = rf_model.predict(new_property)[0]
pred_dt = dt_model.predict(new_property)[0]

print("\n=== Price Predictions ===")
print(f"Linear Regression: Rp {pred_lr:,.0f} juta")
print(f"Random Forest: Rp {pred_rf:,.0f} juta")
print(f"Decision Tree: Rp {pred_dt:,.0f} juta")
print(f"\nBest Prediction (Random Forest): Rp {pred_rf:,.0f} juta")
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feat
warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feat
warnings.warn(
=== PRACTICAL EXAMPLE: Predict New Property Price ===

Property Details:
City: Surabaya
Property Type: Rumah
Land Area: 120
Building Area: 80
Bedrooms: 3
Bathrooms: 2
Carport: 1
Certification: SHM

=== Price Predictions ===
Linear Regression: Rp 1,247 juta
Random Forest: Rp 1,426 juta
Decision Tree: Rp 1,500 juta

Best Prediction (Random Forest): Rp 1,426 juta
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feat
warnings.warn(
```

Step 10: Project Summary and Reflection

After completing all the steps, I wanted to consolidate what I learned and achieved. This project gave me hands-on experience with the entire machine learning workflow, from data preparation to model deployment.

The results exceeded my expectations - achieving over 93% accuracy means the model can reliably predict property prices in East Java. This knowledge will be valuable as I continue developing AI-powered features for property technology applications.

```
print("="*60)
print("  INDONESIA PROPERTY PRICE PREDICTION PROJECT SUMMARY")
print("="*60)

print("\n📊 DATASET:")
print(f"  • Total Properties: {len(df)}")
print(f"  • Locations: {' ', '.join(cities)}")
print(f"  • Property Types: {' ', '.join(property_types)}")
print(f"  • Price Range: Rp {df['price'].min():,.0f} – {df['price'].max():,.0f} juta")

print("\n🧠 MODELS TRAINED:")
print("  1. Linear Regression")
print("  2. Random Forest")
print("  3. Decision Tree")

print("\n🏆 RESULTS COMPARISON:")
print(f"  • Linear Regression: {r2*100:.2f}% accuracy (MAE: Rp {mae:.0f} juta)")
print(f"  • Random Forest: {r2_rf*100:.2f}% accuracy (MAE: Rp {mae_rf:.0f} juta) ⭐ BEST")
print(f"  • Decision Tree: {r2_dt*100:.2f}% accuracy (MAE: Rp {mae_dt:.0f} juta)")

print("\n🔍 KEY INSIGHTS:")
print(f"  • Building Area is the most important factor (85% importance)")
print(f"  • City location affects price significantly (Surabaya most expensive)")
print(f"  • Random Forest achieved 93.48% accuracy – very strong performance!")

print("\n💡 WHAT I LEARNED:")
print("  • How to prepare and clean data for machine learning")
print("  • Data exploration and visualization techniques")
print("  • Training and comparing multiple ML algorithms")
print("  • Evaluating model performance (MAE, RMSE, R² Score)")
print("  • Making practical predictions with trained models")

print("\n✅ PROJECT COMPLETED SUCCESSFULLY!")
print("="*60)
```

```
=====
INDONESIA PROPERTY PRICE PREDICTION PROJECT SUMMARY
=====

📊 DATASET:
  • Total Properties: 1000
  • Locations: Surabaya, Sidoarjo, Gresik, Malang, Batu
  • Property Types: Rumah, Apartemen, Ruko, Villa
  • Price Range: Rp 300 – 7,850 juta

🧠 MODELS TRAINED:
  1. Linear Regression
  2. Random Forest
  3. Decision Tree

🏆 RESULTS COMPARISON:
  • Linear Regression: 85.72% accuracy (MAE: Rp 444 juta)
  • Random Forest: 93.48% accuracy (MAE: Rp 276 juta) ⭐ BEST
  • Decision Tree: 88.68% accuracy (MAE: Rp 372 juta)

🔍 KEY INSIGHTS:
  • Building Area is the most important factor (85% importance)
  • City location affects price significantly (Surabaya most expensive)
  • Random Forest achieved 93.48% accuracy – very strong performance!

💡 WHAT I LEARNED:
  • How to prepare and clean data for machine learning
  • Data exploration and visualization techniques
  • Training and comparing multiple ML algorithms
  • Evaluating model performance (MAE, RMSE, R² Score)
  • Making practical predictions with trained models

✅ PROJECT COMPLETED SUCCESSFULLY!
=====
```

