Study Case: Jakarta, Indonesia

RENT-VALUE CAPTURE (RVC)

Unlocking the Potential of Boarding House Rent through Predictive Modeling

Brahma P. | brahma.workmail@gmail.com
brahma-work.netlify.app

Problem Definition

Why Do We Need Rent Price Prediction?

Boarding house owners or managers need to set rental prices that are competitive and aligned with the market. Without a predictive model, they may either **overprice, leading to vacancies, or underprice, leading to lost income.**

Predicting boarding house rent prices helps ensure tenants (often students or workers on tight budgets) find accommodations that **suit their financial abilities, preventing overcharging while still allowing landlords to remain profitable.**

Rent predictions provide investors with insights into high-demand areas, **helping them identify profitable locations for new boarding houses.**

Suitable Price | Continous Improvement | High-value Area Site-Selection

Analysis Approach

Why ML-Based Predictive modeling rather than traditional-analytics?

Machine learning algorithms automatically uncover complex interactions between features, improving prediction accuracy by capturing relationships that traditional methods often miss.

In traditional statistical models, relationships between variables (or features) are often assumed to be linear or simple, which limits their ability to capture complex interactions between multiple factors. In the case of rent price prediction for boarding houses, however, **the factors influencing the rent price are multifaceted and interdependent, making the relationships between them highly non-linear and dynamic.**

For example, the effect of proximity to a university on rent price may vary depending on room size, amenities, or neighborhood characteristics—something machine learning can uncover.

ML algorithms, like Gradient Boosting or Random Forests, can automatically learn the complex, non-linear relationships between features. For instance, the model might learn that the increase in rent due to proximity to a university is more significant when the room size is smaller, but less significant when the room is larger.

Goals

What are the end-product of this predictive modeling analysis?

1

Build prediction model and demonstrate Price Prediction Interactive Map with Arcgis Javascript SDK that consumes prediction model from Python Flask API

User will be able to click any area on the map inside DK. Jakarta, system automatically detect it’s surrounding Point of Interest within 1 kilometers. Then, user needs to input their desired property parameter such as room size, air-conditioner availability, road-type, etc.

Output: Analysis report consist of **Price Prediction Result** and list of **Surrounding Point of Interest**

Site-selection analysis, showing Distributed Hexagonal Point of Interest area

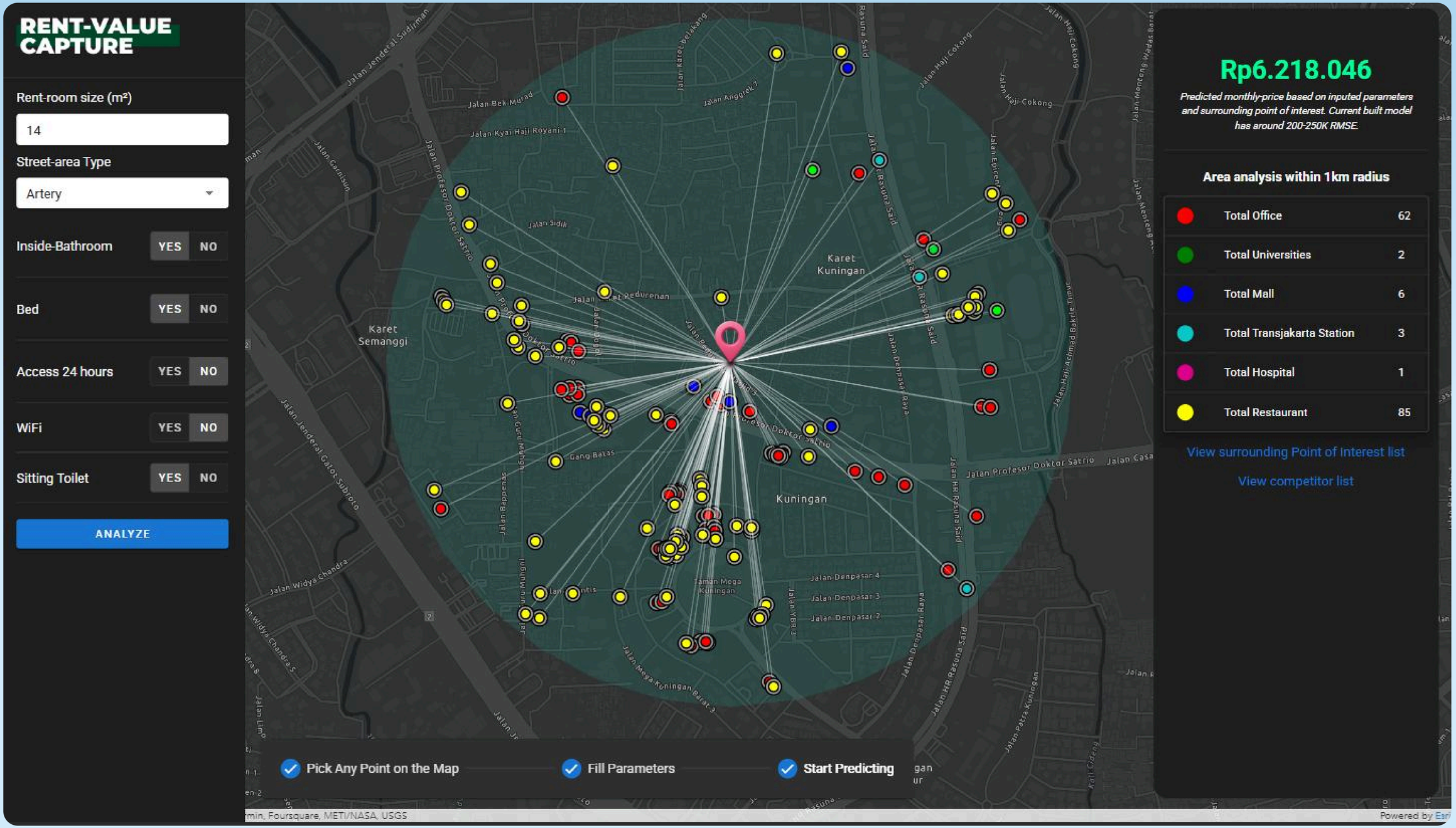
User will be able to generate a hexagon-grid based on selected Point of interest accross jakarta, each hexagon has a different color based on their score.

Output: **Scored Area.**

2

End-Product Solution Preview

ArcGIS Maps SDK for JavaScript



Data Availability

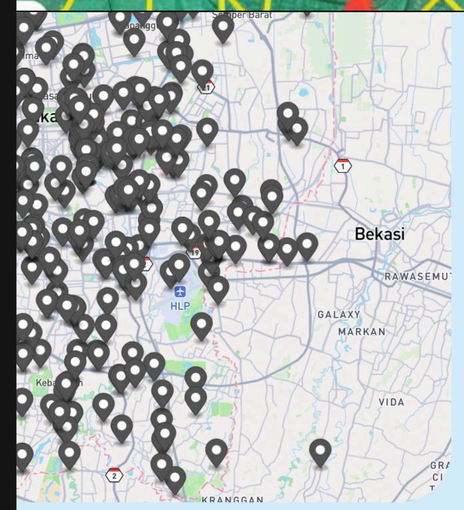
What data is used in this project?

1. 3000+ fields of raw boarding house json data across Jakarta, complete with its Coordinate (lng, lat), facility, room-size, monthly-price, etc. this data is gathered from a boarding-house online services.


```
  "price_title_time": "2.025.000 / bulan",
  "rating": 4.9,
  "rating_string": "4.9",
  "review_count": 65,
  "room-title": "Kost Keagungan 20A Tipe A",
 "size": "2.8x2.7",
  "status": 0,
  "status-title": "null",
  "status_kos": "verified",
  "subdistrict": "Taman Sari",
  "top_facility": [
      "K. Mandi Dalam",
      "WiFi",
      "AC",
      "Kloset Duduk",
      "Kasur",
      "Akses 24 Jam"
  ],
  "unique_code": "9NB148KS",
  "unit_type": "Tipe A",
  "unit_type_rooms": null,
  "updated": "2025-01-24 15:21:49",
```


- {} apartment-jaksel.geojson
- {} fasilitas-kesehatan.geojson
- {} fasilitas-pendidikan.geojson
- {} hospital.geojson
- {} krl.geojson
- {} lrt.geojson
- {} mall.geojson
- {} market.geojson
- {} mrt_timur_barat.geojson
- {} mrt.geojson
- {} office.geojson
- {} place-of-worship.geojson
- {} ptn.geojson
- {} pts.geojson
- {} restaurant.geojson
- {} sd.geojson
- {} slb.geojson
- {} sma.geojson
- {} smk.geojson
- {} smp.geojson
- {} spbu.geojson
- {} stasiun-krl.geojson
- {} stasiun-lrt.geojson
- {} stasiun-mrt.geojson
- {} supermarket.geojson
- {} transjakarta.geojson

27+ Point of Interest category in DK. Jakarta.



Methodology How the process work? (CRISP-DM flow)

Data Understanding

Rent Price Data (JSON)

PoI Data (8 PoI GeoJSON Layer)

Data Preprocessing 1

- excluding unecessary field from Rent Price Data, leave only coordinate point, 6 boolean, 2 categorical, 1 numerical features and 1 target (price)
- Load All required PoI: office, universities, mall, transjakarta, lrt, hospital, restaurant, and krl
- With utilizing TurfJS, create Buffer for each point within 1 kilometers, then count every PoI that intersect with the buffer.
- Save the end-result as CSV data

CSV Dataset

Data Preprocessing 2 & Data Transformation

- cleaning data such as filling the empty cell
- Normalize numerical data with StandardScaler
- covert true/false boolean field to 0 and 1
- One-Hot encoding categorical data
- Spliting Training and Testing Data by 80% - 20%

preprocessed data

Modelling

- Defining features (X) and target (Y)
- Train the dataset with Gradient Boosting Regressor

Model Evaluation

- Predict the testing Data
- Analyse Root Mean Squared Error (RMSE)

model

Model Improvement (Hyperparameter Tuning) & Model Interpretation)

- modify n_parameters, learning_rate, and max_depth of Gradient Boosting Regressor to achieve better RMSE

model

Model Deployment

Deploy the model using Python Flask to be accessible

- Running WebService to have an API Endpoint for front-end consumption

ArcGIS Notebooks

WebService (API)

Data Visualization

Data Visualization
Data Interpretation
End-Prouct Consumption

- Integrate Model with Interactive ArcGIS Javascript Map that allow user to click any area on the map inside DK. Jakarta, system automatically detect it's surrounding Point of Interest within 1 kilometers. Then, user needs to input their desired property parameter such as room sizing, air-conditioner availability, road-type, etc.
- PoI Spatial Plotting
- Hexagonal-Grid score distribution

ArcGIS Maps SDK for JavaScript

ArcGIS Living Atlas of the World

Data Preprocessing 1 (In Javascript)

RAW Data

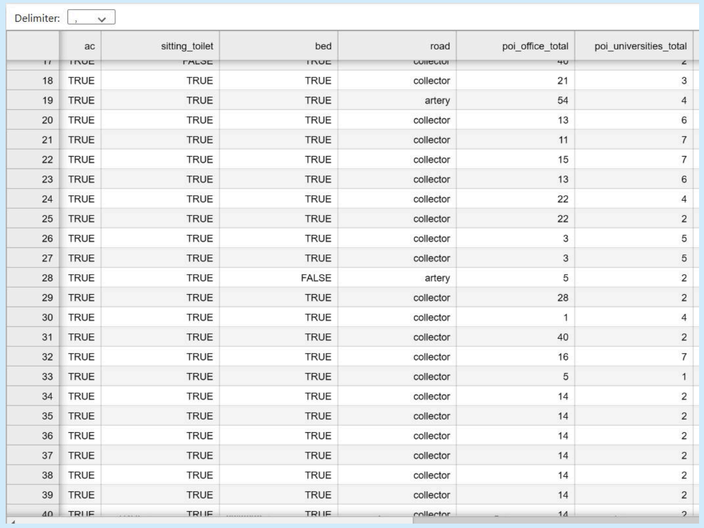

```
"price_title": "1.9 jt",
"price_title_format": {
    "currency_symbol": "Rp",
    "price": "2.025.000",
    "discount_price": "1.925.000",
    "discount": "5%",
    "discount_label": "100rb",
    "rent_type_unit": "bulan"
},
"price_title_time": "2.025.000 / bulan",
"rating": 4.9,
"rating_string": "4.9",
"review_count": 65,
"room-title": "Kost Keagungan 20A Tipe A",
"size": "2.8x2.7",
"status": 0,
"status-title": "null",
"status_kos": "verified",
"subdistrict": "Taman Sari",
"top_facility": [
    "K. Mandi Dalam",
    "WiFi",
    "AC",
    "Kloset Duduk",
    "Kasur",
    "Akses 24 Jam"
],
```


- excluding unecessary field from Rent Price Data, leave only coordinate point, 6 boolean, 2 categorical, 1 numerical features and 1 target (price)


```
[
    {
        "name": "Kost Keagungan 20A Tipe A",
        "price": 2025000,
        "size": 7.56,
        "access_24": true,
        "inside_bathroom": true,
        "wifi": true,
        "ac": true,
        "sitting_toilet": true,
        "bed": true,
        "gender": "Campur",
        "coordinate": "-6.1506025,106.8123706",
        "road": "collector"
    },
```


- Restructure data to be FeatureCollection of **GeoJSON Format**


```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    106.8123706,
                    -6.1506025
                ]
            },
            "properties": {
                "name": "Kost Keagungan 20A Tipe A",
                "price": 2025000,
                "size": 7.56,
                "access_24": true,
                "inside_bathroom": true,
                "wifi": true,
                "ac": true,
                "sitting_toilet": true,
                "bed": true,
                "gender": "Campur",
                "road": "collector"
            }
        },
```

Delimiter: ,

	ac	sitting_toilet	bed	road	poi_office_total	poi_universities_total
18	TRUE	TRUE	TRUE	collector	21	3
19	TRUE	TRUE	TRUE	artery	54	4
20	TRUE	TRUE	TRUE	collector	13	6
21	TRUE	TRUE	TRUE	collector	11	7
22	TRUE	TRUE	TRUE	collector	15	7
23	TRUE	TRUE	TRUE	collector	13	6
24	TRUE	TRUE	TRUE	collector	22	4
25	TRUE	TRUE	TRUE	collector	22	2
26	TRUE	TRUE	TRUE	collector	3	5
27	TRUE	TRUE	TRUE	collector	3	5
28	TRUE	TRUE	FALSE	artery	5	2
29	TRUE	TRUE	TRUE	collector	28	2
30	TRUE	TRUE	TRUE	collector	1	4
31	TRUE	TRUE	TRUE	collector	40	2
32	TRUE	TRUE	TRUE	collector	16	7
33	TRUE	TRUE	TRUE	collector	5	1
34	TRUE	TRUE	TRUE	collector	14	2
35	TRUE	TRUE	TRUE	collector	14	2
36	TRUE	TRUE	TRUE	collector	14	2
37	TRUE	TRUE	TRUE	collector	14	2
38	TRUE	TRUE	TRUE	collector	14	2
39	TRUE	TRUE	TRUE	collector	14	2

- Convert into tabular data (CSV)

- For each points, Create a 1000 meters buffer, and count every Point of Interest category within the buffer.

```
datasetPolygon.features.forEach((item) => {
  let point = turf.point(item.geometry.coordinates)
  let buffered = turf.buffer(point, 1000, { units: "meters" });
  let ptsWithinOffice = turf.pointsWithinPolygon(this.poi_office, buffered);
  item.properties.poi_office = ptsWithinOffice
  item.properties.poi_office_total = ptsWithinOffice.features.length
```

Data Preprocessing 2 & Data Transformation (In Python)

1. Checking for missing values
2. Checking and removing outliers field

```
#--------- Plot boxplots for price columns
plt.figure(figsize=(5, 1))
sns.boxplot(x=data["price"])
plt.title("Boxplot of Price")
plt.show()
```

Split data training and data testing in 8 : 2 ratio

```
#--------- Import Numpy
import numpy as np

#--------- Import train_test_split package for splitting datasets
from sklearn.model_selection import train_test_split

#--------- Splitting test-train by 20% : 80%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#--------- If y values range from 500K to 7500K
#--------- a model might struggle to learn patterns effectively.
#--------- Applying log1p(y) compresses large values, making the dataset more balanced.

y_train_log = np.log1p(y_train)
```

Data Transformation

- Standardize the numerical columns using StandardScaler

poi_office_total	poi_universities_total	poi_mall_total	\
5.693897	-0.097388	0.984572	
4.425723	7.582026	-0.781407	
3.059996	0.142593	0.984572	
2.962444	-0.337370	0.984572	
1.889374	7.582026	-0.781407	
...	...	...	

- Convert boolean columns (True/False) to 1/0

access_24	inside_bathroom	wifi	sitting_toilet	bed
1	1	1	1	1
0	1	1	1	1
1	1	1	1	1
1	1	1	1	0
0	1	1	1	1
...	...	...	...	...
1	0	0	0	1

- Encode categorical value

	road_encoded
0	1
1	0
2	1
3	1
4	0

Modelling Create a prediction model

Train model using **Gradient Boosting Regressor** with **learning_rate=0.5, max_depth=7, n_estimators=500**

```
##### --------- load GradientBoostingRegressor for modelling
from sklearn.ensemble import GradientBoostingRegressor

#--------- load Mean Square Error
from sklearn.metrics import mean_squared_error

#--------- Create a Modelling variable
model = GradientBoostingRegressor(n_estimators=500, learning_rate=0.5, max_depth=7)

#--------- Start modelling with X parameter but using loged Y as target
model.fit(X_train, y_train_log)
```

Predict testing data

```
#--------- Predicting with Testing Data
y_pred_log = model.predict(X_test)

#--------- # Convert back to original scale
y_pred = np.expm1(y_pred_log)

#--------- Print Results
print(y_pred)
```

```
[1977624.84302129 3825000.00000126 2275530.00655661 2193627.45148826
 1463952.80671868 1100761.80293643 3007083.66884384 1949358.86928387
 1628812.64261311  850188.63788718 1327833.14411977 1248999.60048166
  637978.40868382 2049281.80715155  851470.77140018 3250000.00000151
 1433466.61006301 2028982.16296315 1775427.16498753 1503291.24447421
 2125000.00000625  776898.9375032  2413320.70777997 1413096.74918073
 1573447.90712554 1943622.20679722 2525036.35164177  815314.05456232
  997934.63360621 1153753.41767716 1321654.44747947 2299315.57865141
 1548134.26238387 2306681.73202569  805845.05907528 2431116.92080333
```

Model Evaluation

Calculate Root Mean Squared Error & Error Percentage

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f'Root Mean Squared Error (RMSE): {rmse}')

#--------- Checking Error percentage

#--------- Define range actual
range_actual = np.max(data.price) - np.min(data.price)

#--------- Calculate error percentage using mean
error_percentage_mean = (rmse / range_actual) * 100

print(error_percentage_mean, '%')
```

with range price 500K - 7500K, the model resulting 6.44% Error

```
Root Mean Squared Error (RMSE): 386497.85236870946
6.441630872811824 %
```

Hyperparameter Tuning

to Improve Accuracy by minimalizing RSME and Error Percentage, we decreasing learning_rate, increasing ma_depth and also increasing n_estimators:

learning_rate=0.1, max_depth=8, n_estimators=1000, random_state=42

```
#--------- Hyperparameter Tuning
#--------- Lowering learning-rate, Increasing n_estimator, set random_state value, increate max-depth
model = GradientBoostingRegressor(learning_rate=0.1, max_depth=8, n_estimators=1000, random_state=42)

#--------- Train the model
model.fit(X_train, y_train)

#--------- Predict and evaluate again
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

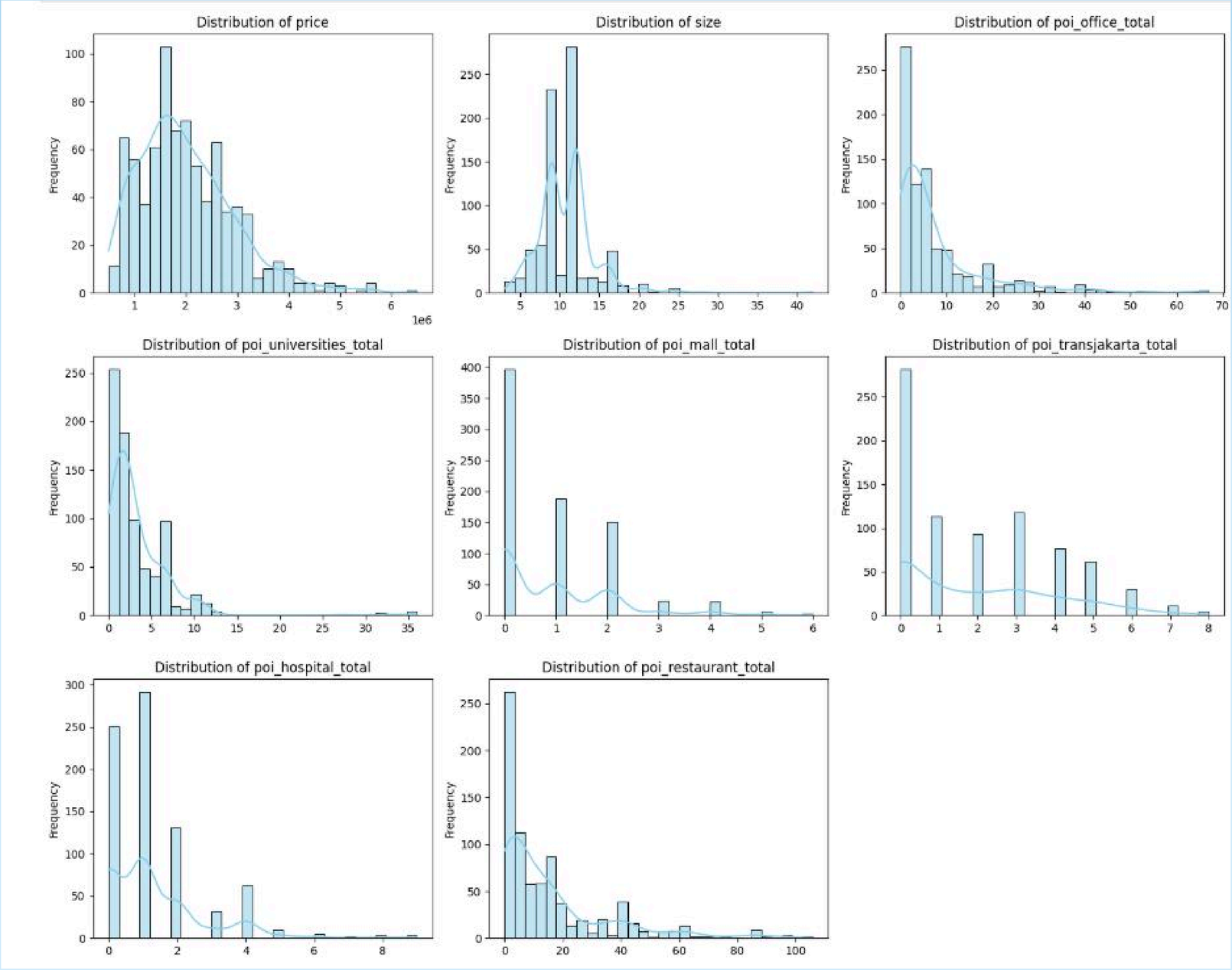
Result after Hyperparameter Tuning

```
Root Mean Squared Error (RMSE): 355348.9270177617
5.922482116962695 %
```

Conclusion: after hyperparameter tuning, error percentage are decreased from 6.44 % to 5.92% (-0,54%)

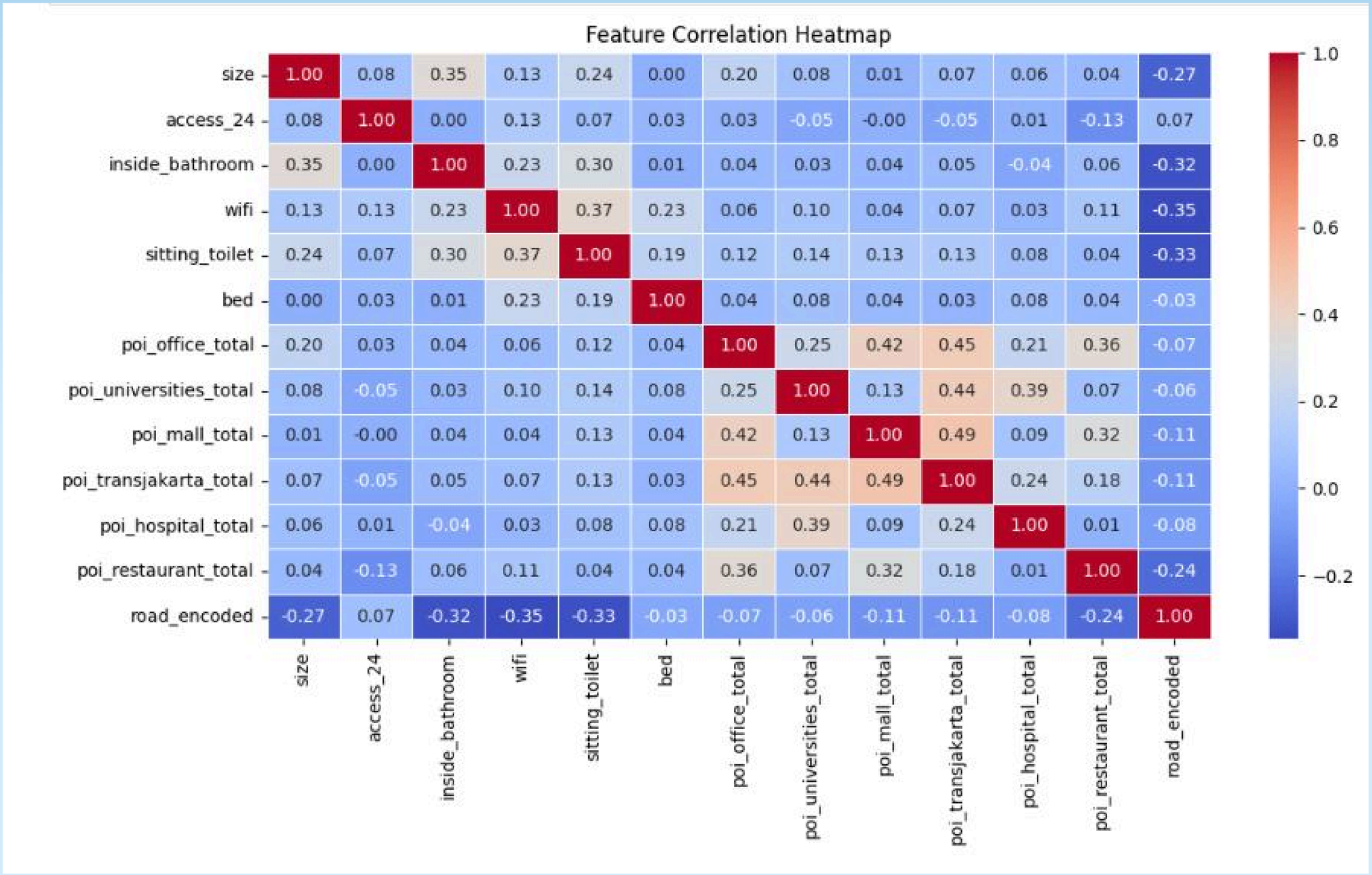
Data Visualization

Feature Distribution



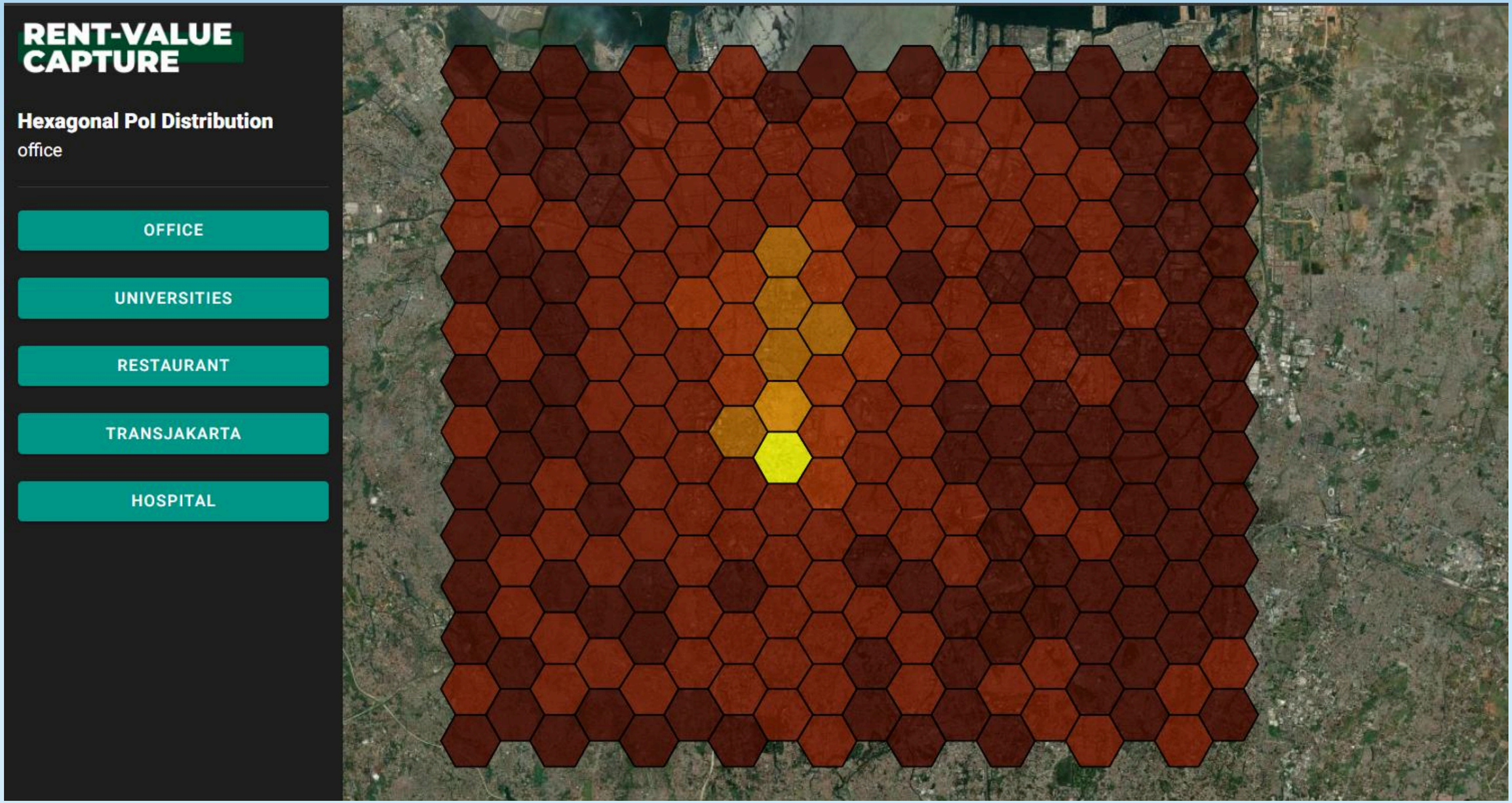
Data Visualization

Feature Correlaction Heatmap

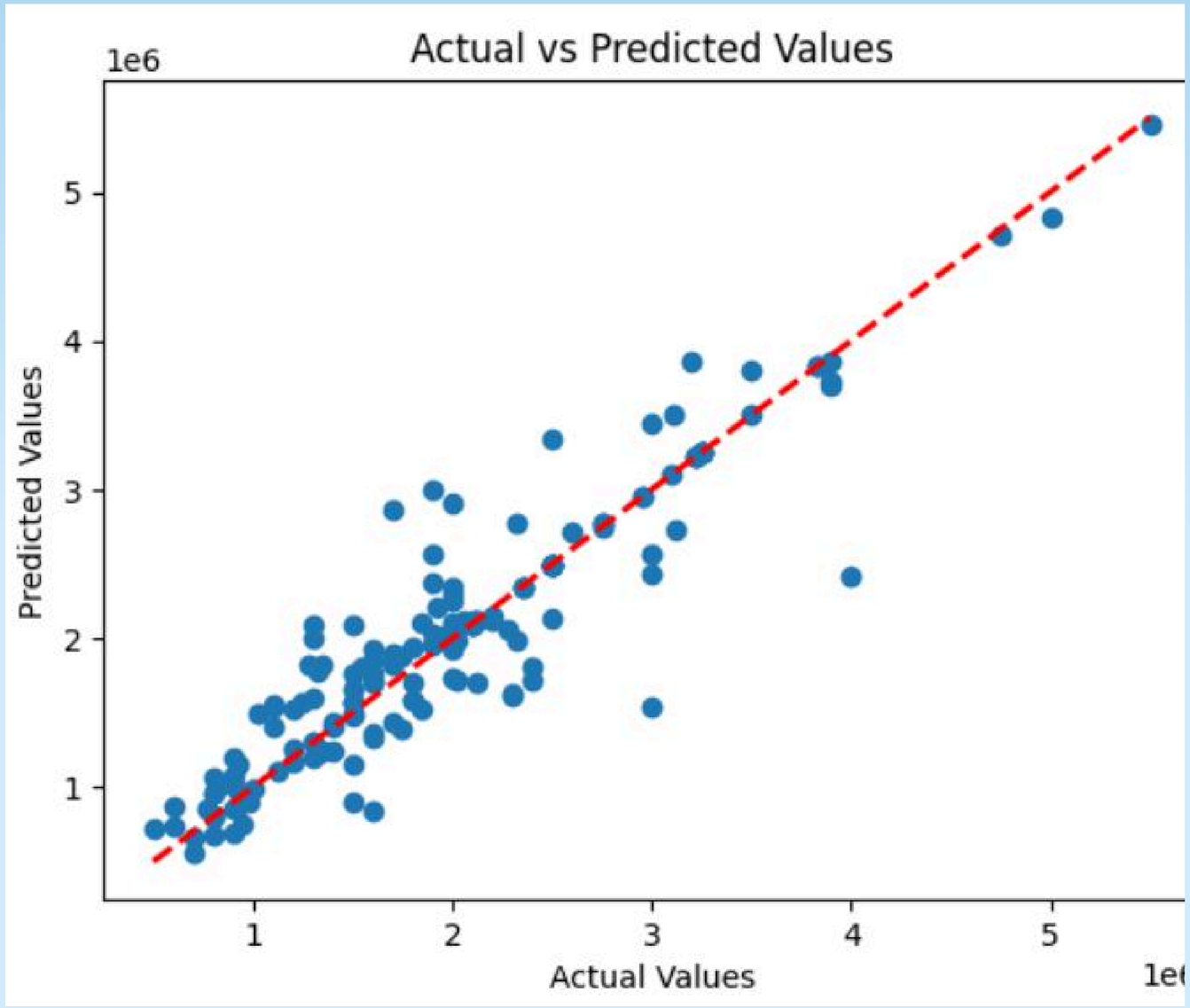


Data Visualization

Hexagonal PoI Distribution Scoring



Model Interpretation



Axes:

- X-axis (Actual Values): This represents the actual observed values from dataset.
- Y-axis (Predicted Values): This represents the values predicted by model.

Data Points:

- Each blue dot on the plot represents a pair of actual and predicted values for a single data point.

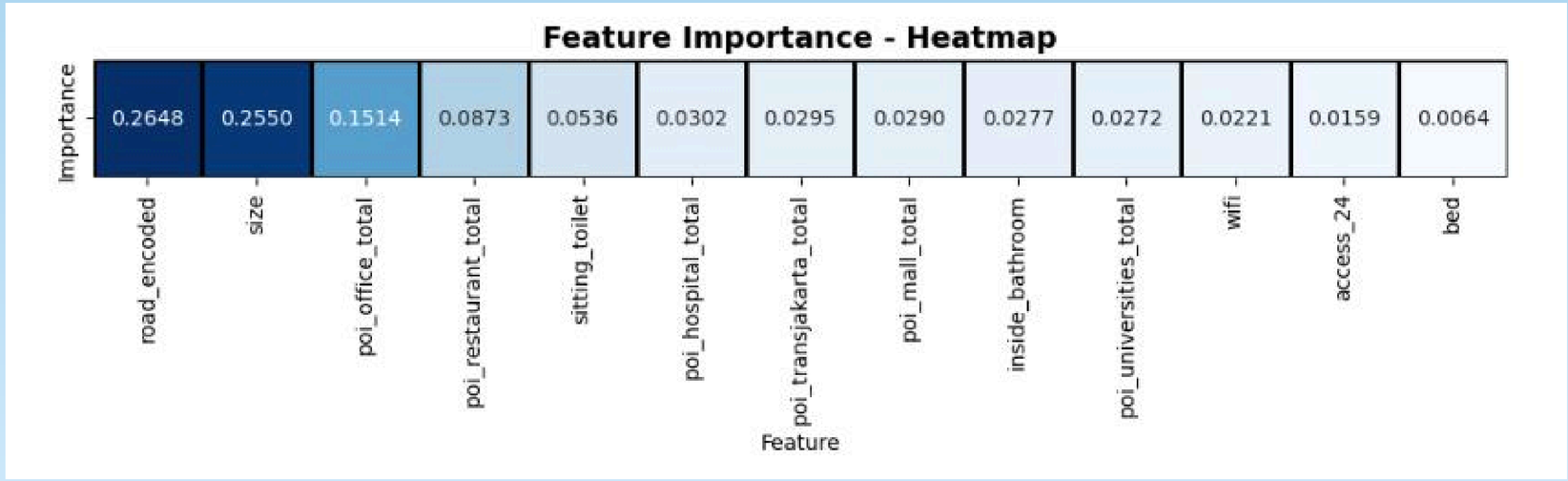
Red Dashed Line:

- The red dashed line is the line of perfect prediction where the predicted values would exactly match the actual values (y = x). If all points were on this line, it would indicate a perfect model where every prediction was spot on.

Interpretation from the Plot:

- The points are somewhat scattered around the red line but generally follow its trend, suggesting that the **model has captured the overall trend in the data but with some degree of error.**
- There's a noticeable spread, particularly at higher values, indicating that the model's predictions become less accurate as the actual values increase.
- The plot also shows that while there's some deviation, most points are not too far from the line, which is a positive sign for model performance.

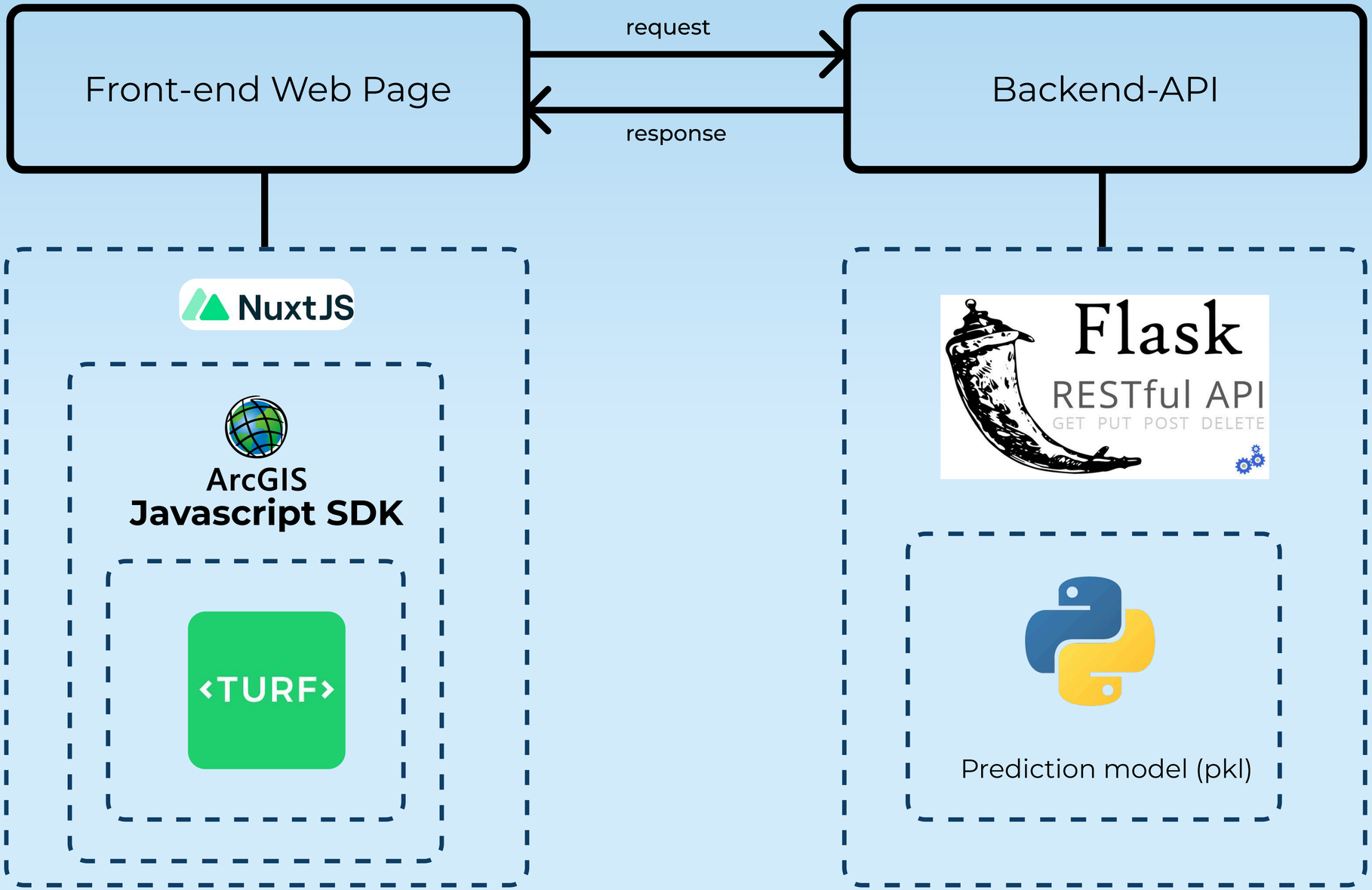
Model Interpretation



Interpretation:

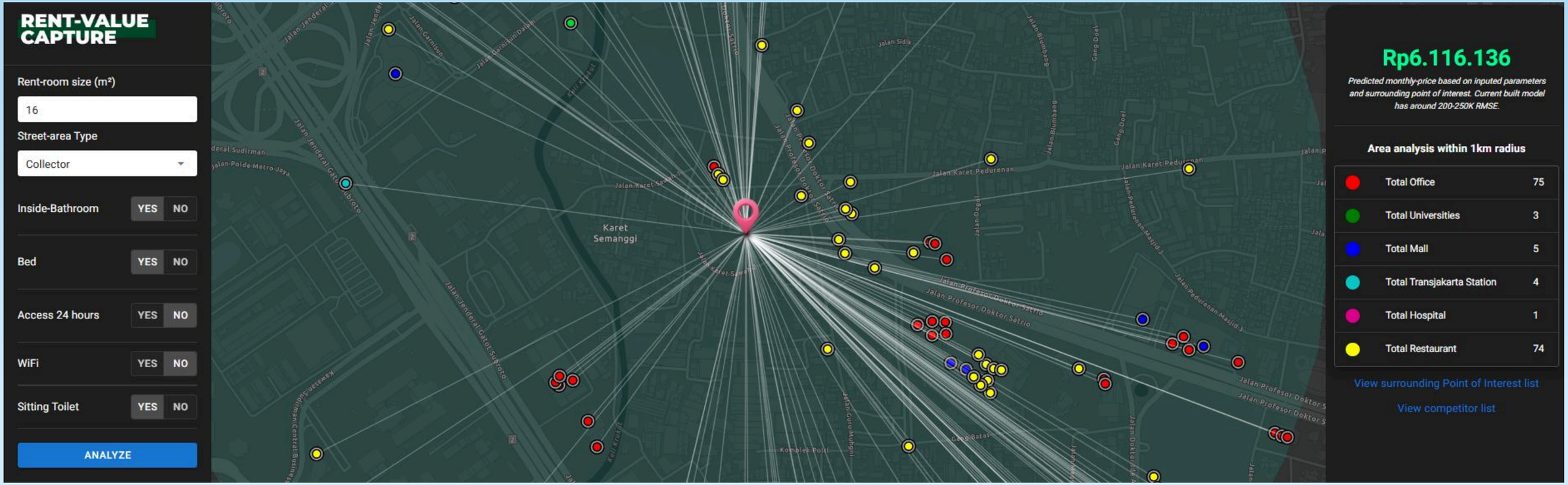
- High Importance: Features like **road_encoded** and **size** are crucial in determining the model's output. This might suggest that the location's accessibility (encoded road information) and the physical size of the property are key factors in rent pricing.
- Moderate Importance: Features like **poi_office_total**, **poi_restaurant_total**, and **sitting_toilet** indicate that amenities and nearby facilities influence the model's decision.
- Low Importance: Features like **bed and access_24 have less impact,** suggesting they might not be as critical for the model's predictions in this context.

GIS App Building With ArcGIS Javascript



End-Product Experiment 1

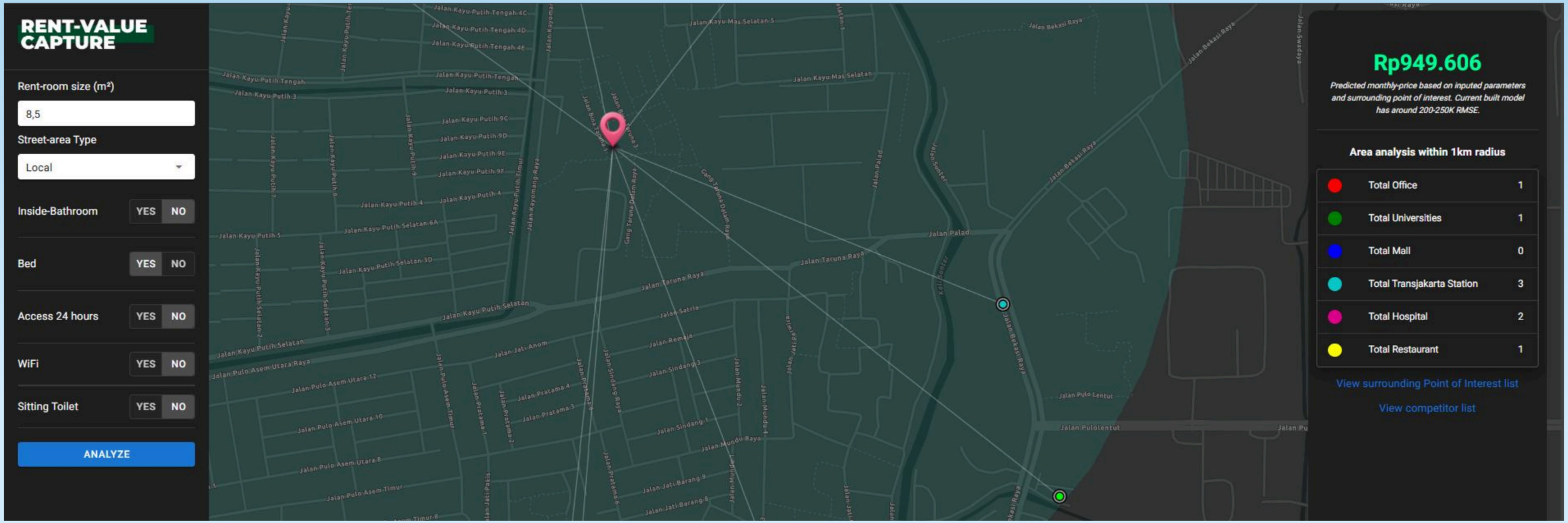
ArcGIS Maps SDK for JavaScript



The predicted monthly rent for a property with the specified characteristics (16 m², street-area type, inside bathroom, bed, 24-hour access, WiFi, and sitting toilet) located in Karet Semanggi, Jakarta, is approximately **Rp6,116,136.** This prediction is influenced by the surrounding points of interest within a 1km radius, which include **75 offices, 3 universities, 5 malls, 4 Transjakarta stations, 1 hospital, and 74 restaurants, showcasing the area's amenities and accessibility as significant factors in determining rental value.**

End-Product Experiment 2

ArcGIS Maps SDK for JavaScript



(8.5 m², small-local alley road type, no inside bathroom, bed, no 24-hour access, no WiFi, and no sitting toilet) located in Gang Taruna Dalam, Jakarta, is approximately **Rp949,606.** This prediction is influenced by the surrounding points of interest within a 1km radius, **which include 1 offices, 1 universities, 0 malls, 3 Transjakarta stations, 2 hospital, and 1 restaurants, showcasing the area's amenities and accessibility as significant factors in determining rental value.**