

Daegu Apartment

Price Prediction

EDWARDO KRISNA SEMBIRING

Powered by : Purwadhika





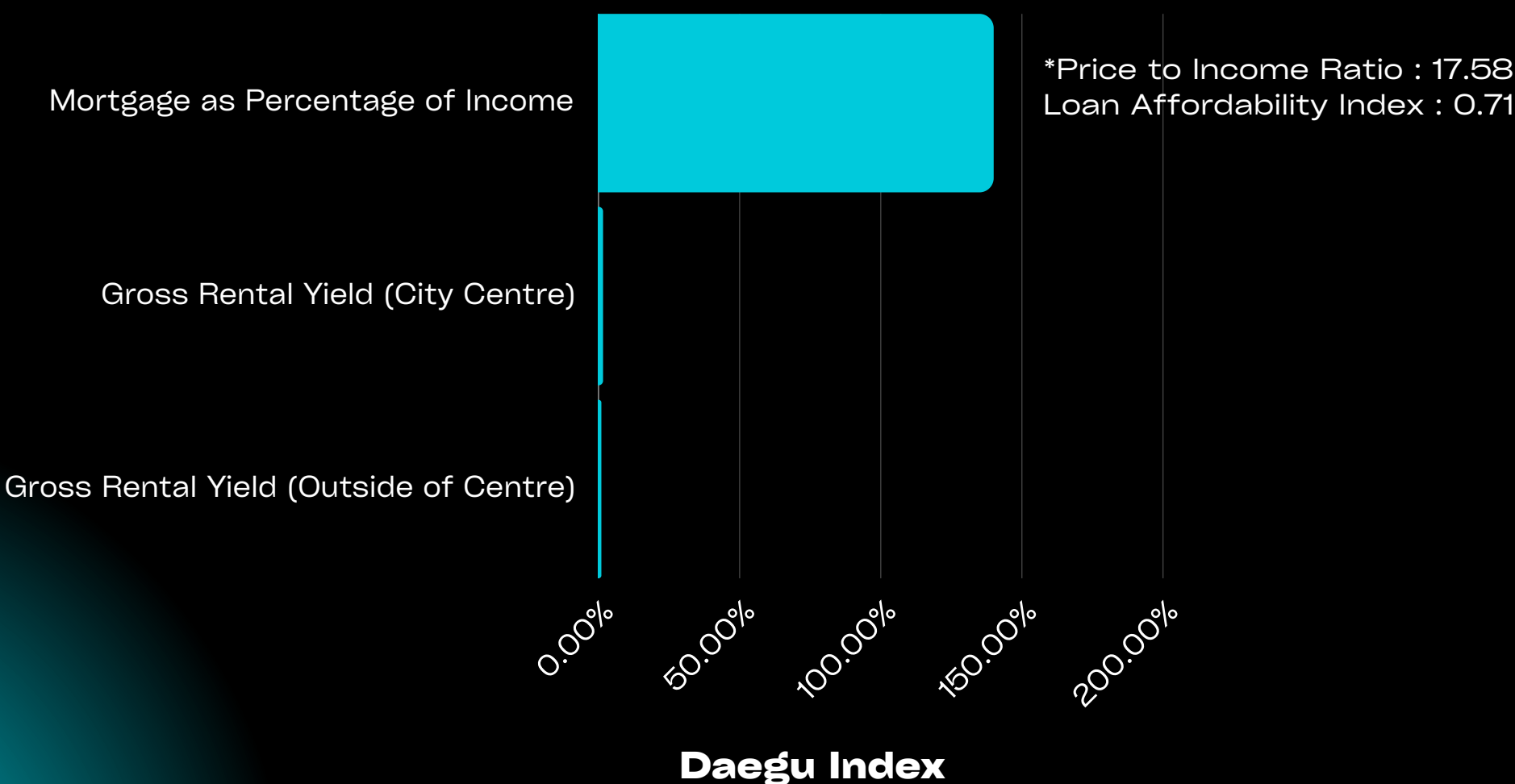
Daegu, the fourth-largest city in South Korea, boasts a population of approximately 2.5 million residents. Known for its vibrant culture, historical significance, and economic dynamism, Daegu is also home to a highly active real estate market. In particular, the apartment sector stands out due to its popularity among residents seeking convenience, modern amenities, and strategic urban locations.



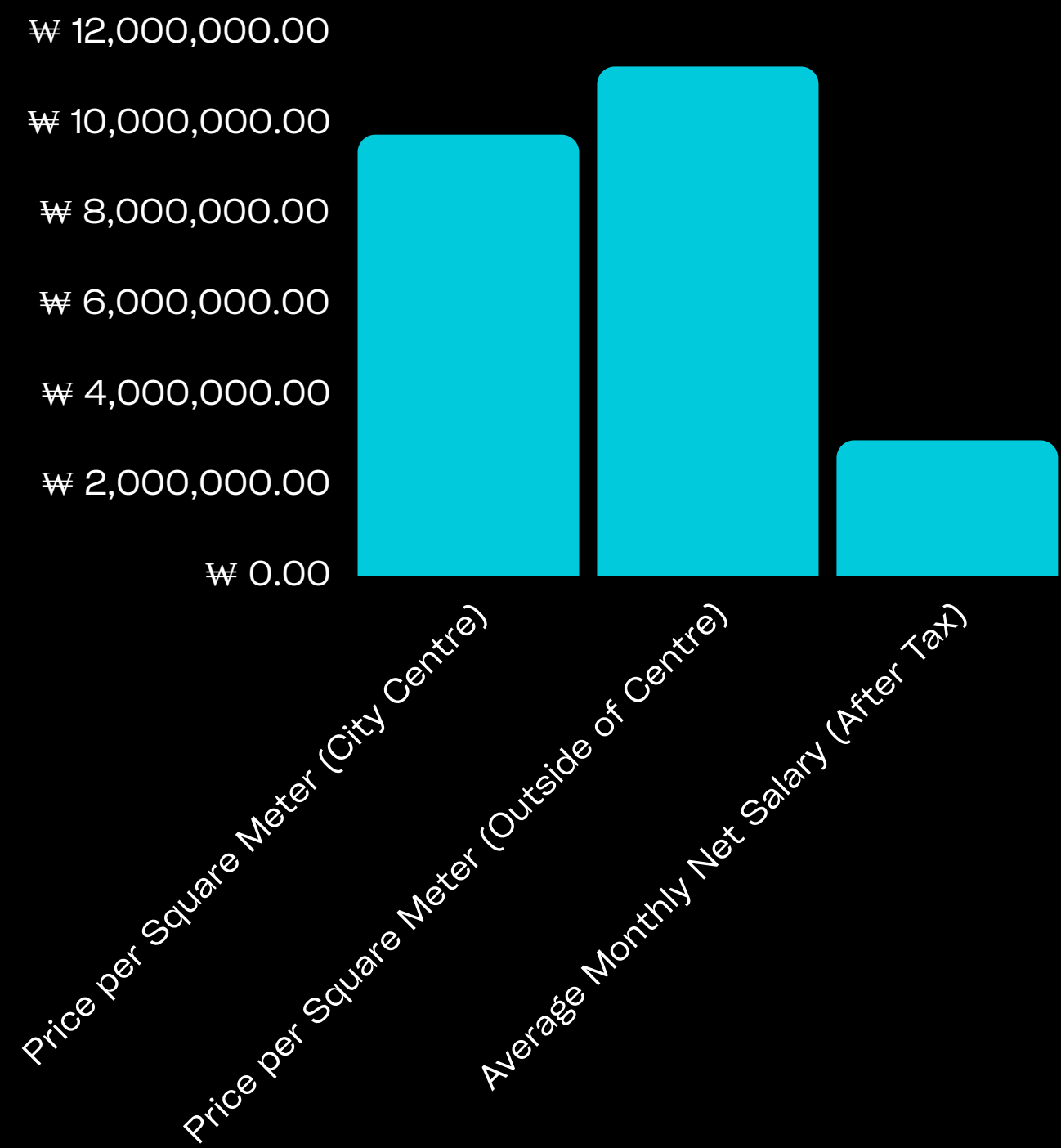
Property Industry Analysis in Daegu, South Korea

Country Data

GDP Per Capita	\$45,600.00
GDP Growth Rate	2.61%
Population Growth Rate	0.23%



Daegu Apartment Price, Salaries And Financing



*Mortgage Interest Rate (Yearly, Fixed-Rate Options) : 5.07%



Problem

Market Efficiency

Affordability Crisis

Investment Decisions

Urban Planning

Financial Risk Assessment



My Goal

“To develop a machine learning model capable of accurately predicting apartment prices in Daegu. This model will leverage various features such as property size, location, nearby facilities, and other relevant characteristics.”



Why ML and how ML Regression will help solve the problem ?

Complex
Relationships

Large Dataset

Adaptability

Feature
Importance

Handling
Multiple
Variables



Workflow

- Data Collection and Understanding
- Exploratory Data Analysis (EDA)
- Data Preprocessing and Feature Engineering

- Model Selection and Training
- Hyperparameter Tuning
- Model Evaluation
- SHAP Analysis

- Key Findings and Insights
- Business Implications
- Conclusion

Metric Evaluation

- R^2 Score (Coefficient of Determination)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)
- Mean Absolute Percentage Error (MAPE)

For the detailed workflow you can press [here](#)



Data Dictionary

Attribute	Data Type	Description
Hallway Type	Object	Apartment type
TimeToSubway	Object	Time needed to the nearest subway station
SubwayStation	Object	The name of the nearest subway station
N_FacilitiesNearBy(ETC)	Float	The number of facilities nearby
N_FacilitiesNearBy(PublicOffice)	Float	The number of public office facilities nearby
N_SchoolNearBy(University)	Float	The number of universities nearby
N_Parkinglot(Basement)	Float	The number of the parking lot
YearBuilt	Integer	The year the apartment was built
N_FacilitiesInApt	Integer	Number of facilities in the apartment
Size(sqft)	Integer	The apartment size (in square feet)
SalePrice	Integer	The apartment price (Won)



Data Preprocessing

Handling null values

No action needed as there were no null values.

Handling duplicates

Duplicate rows : 1422

I chose to **keep** the duplicates data.

Handling outliers

Outliers in Size(sqf) : 98

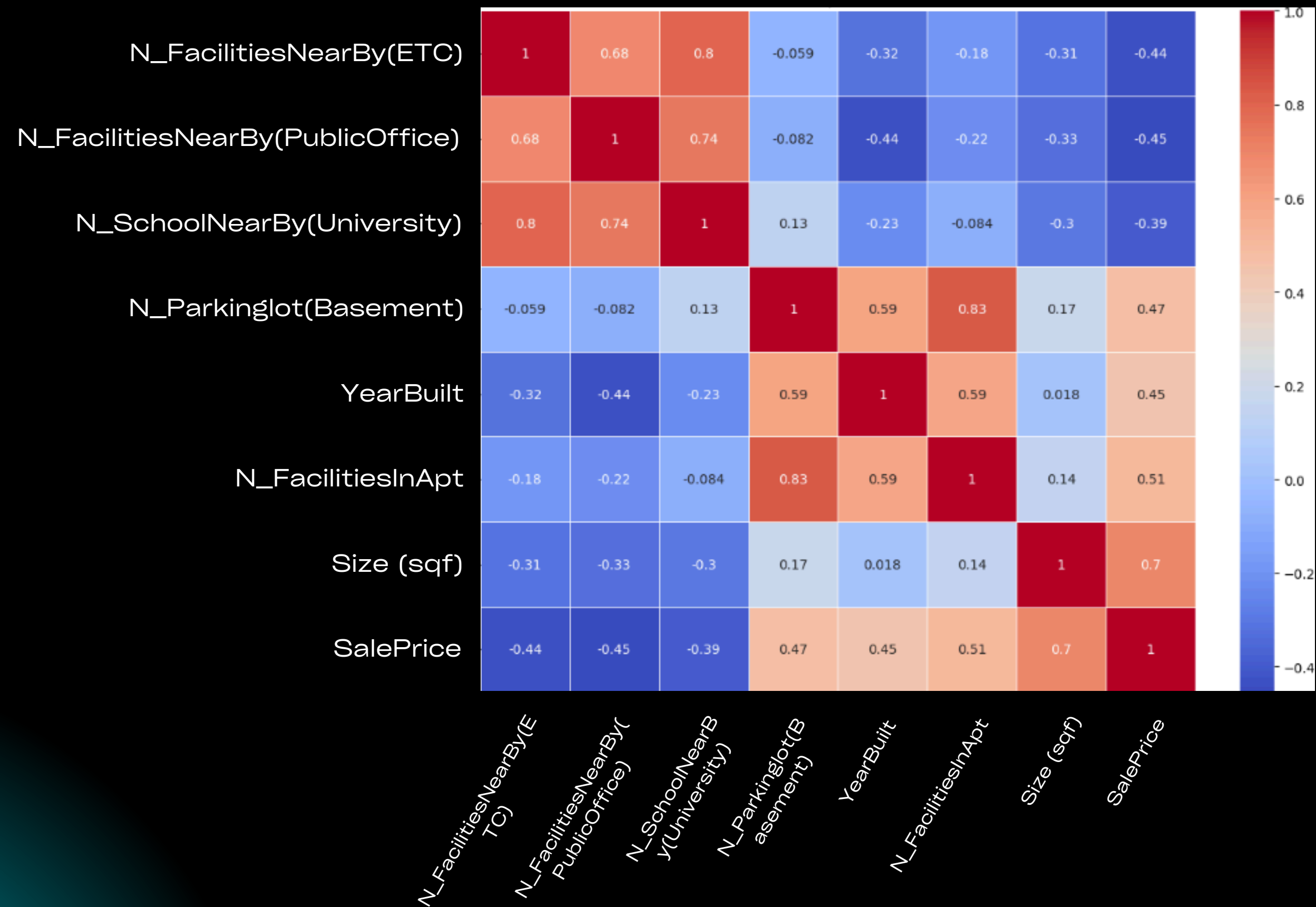
Outliers in SalePrice : 27

Outliers in other column : 0

I **didn't remove** the outliers.



Feature Selection



Feature Engineering

Implemented a custom transformer called SubwayDataCleaner:

```
class SubwayDataCleaner(BaseEstimator,
TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_ = X.copy()
        if 'SubwayStation' in X_.columns and
        'TimeToSubway' in X_.columns:
            mask = (X_['SubwayStation'] ==
        'no_subway_nearby') & (X_['TimeToSubway'] !=
        'no_bus_stop_nearby')
            X_.loc[mask, 'TimeToSubway'] =
        'no_bus_stop_nearby'
        return X_
```

Feature scaling and encoding :

- StandardScaler for numerical features
- OrdinalEncoder for categorical features



	Linear Regression	Decision Tree	K-Nearest Neighbors (KNN)	Random Forest	XGBoost	Gradient Boosting
Diversity in Approach	A simple, interpretable baseline model	A non-linear model with high interpretability	A non-parametric, instance-based learning method	An ensemble of decision trees using bagging	Advanced boosting techniques	Advanced boosting techniques
Complexity Spectrum	the simplest	moderate complexity	moderate complexity	more complex ensemble methods	more complex ensemble methods	more complex ensemble methods
Handling of Non-linearity	X	✓	✓	✓	✓	✓
Interpretability vs. Performance	highly interpretable	highly interpretable	moderately interpretable	often provide better performance but are less interpretable	often provide better performance but are less interpretable	often provide better performance but are less interpretable
Toughness to Outliers	more sensitive to outliers	more tough to outliers	more sensitive to outliers	more tough to outliers	more tough to outliers	more tough to outliers



How XGBoost compared with other models

- Performance 🐍 **highest** cross-validation R2 score of **0.8378** after hyperparameter tuning.
- Consistency 🐍 performed well in both cross-validation (R2 = 0.8379) and test set evaluation (R2 = 0.8432) before tuning.

*I only tuned top 3 model

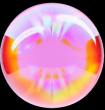
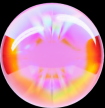
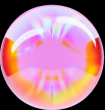




- Toughness
- Feature importance
- Handling of outliers and non-linear relationships

Model	Cross-Validation R ² Score before tuning	Cross-Validation R ² Score after tuning
Linear Regression	0.7398	-
Decision Tree	0.8380	0.8356
KNN	0.8074	-
Random Forest	0.8380	0.8365
Gradient Boosting	0.8379	-
XGBoost	0.8379	0.8378

Cross-val R2 score comparison with other models :



How XGBoost works

-  Ensemble Method
-  Sequential Learning
-  Gradient Boosting
-  Regularization
-  Feature Importance
-  Handling of Missing Values
-  Parallel Processing

In our case, the best XGBoost model had this following parameters :

- `n_estimators` : 621 (number of trees)
- `max_depth` : 4 (maximum depth of each tree)
- `learning_rate` : 0.02259 (step size shrinkage used to prevent overfitting)
- `subsample` : 0.62339 (fraction of samples used for training each tree)
- `colsample_bytree` : 0.66327 (fraction of features used for training each tree)



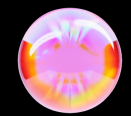
How's the Hyperparameter Tuning

OPTUNA ,uses advanced algorithms to suggest hyperparameters, learning from previous trials to focus on promising areas of the hyperparameter space.

Breakdown of the process:

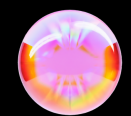


Objective Function



Hyperparameter Search Space 🐛

- n_estimators : Integer value between 100 and 1000
- max_depth : Integer value between 3 and 10
- learning_rate : Continuous value between 0.001 and 1.0, on a logarithmic scale
- subsample : Continuous value between 0.6 and 1.0
- colsample_bytree : Continuous value between 0.6 and 1.0



Optimization Process 🐛

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```



What metrics to measure the model performance?

Metrics	Final Score
R2 Score (Coefficient of Determination)	0.8435
Mean Squared Error (MSE)	1687934365.2237
Root Mean Squared Error (RMSE)	41084.4784
Mean Absolute Error (MAE)	32223.9079
Mean Absolute Percentage Error (MAPE)	0.1783 (17.83%)



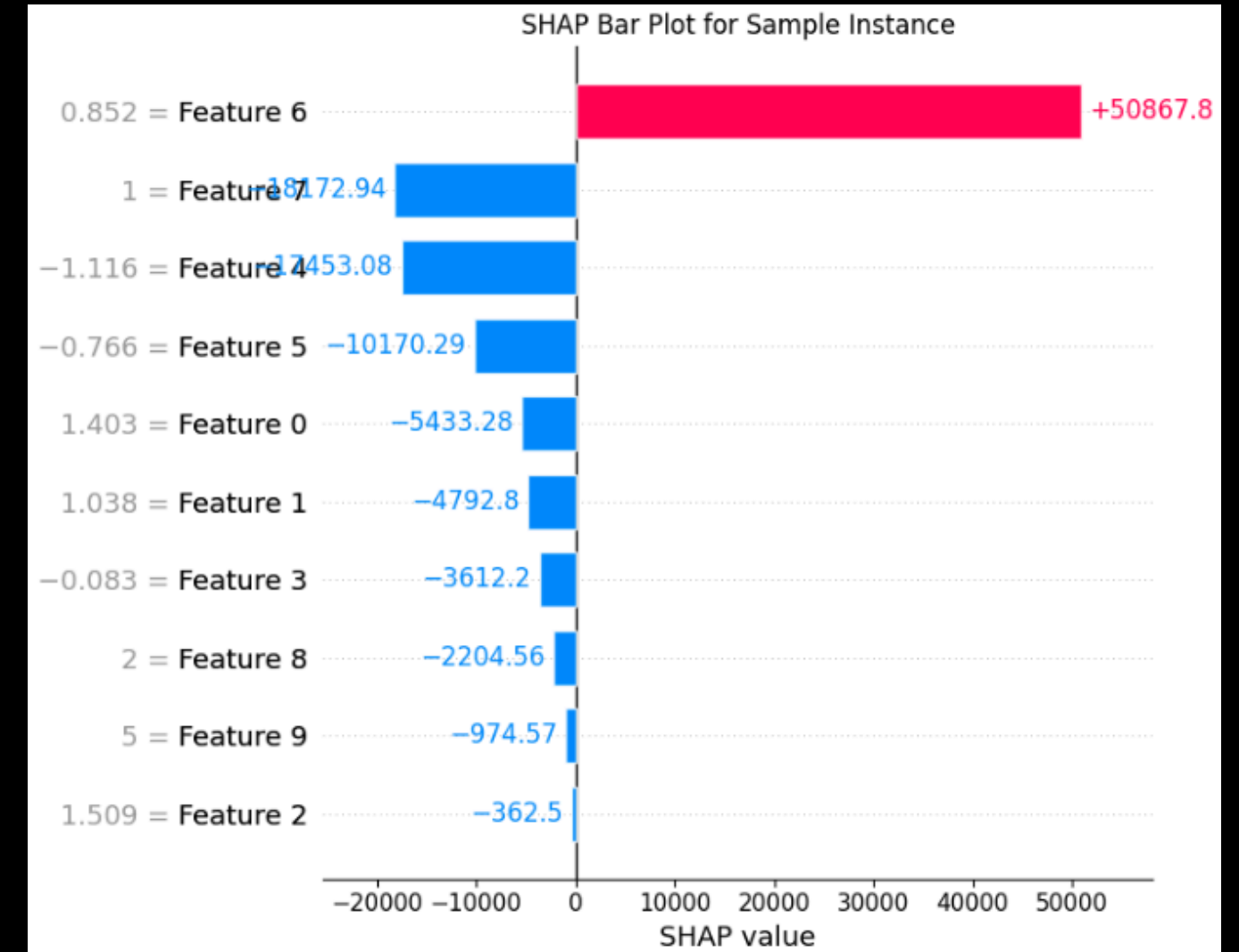
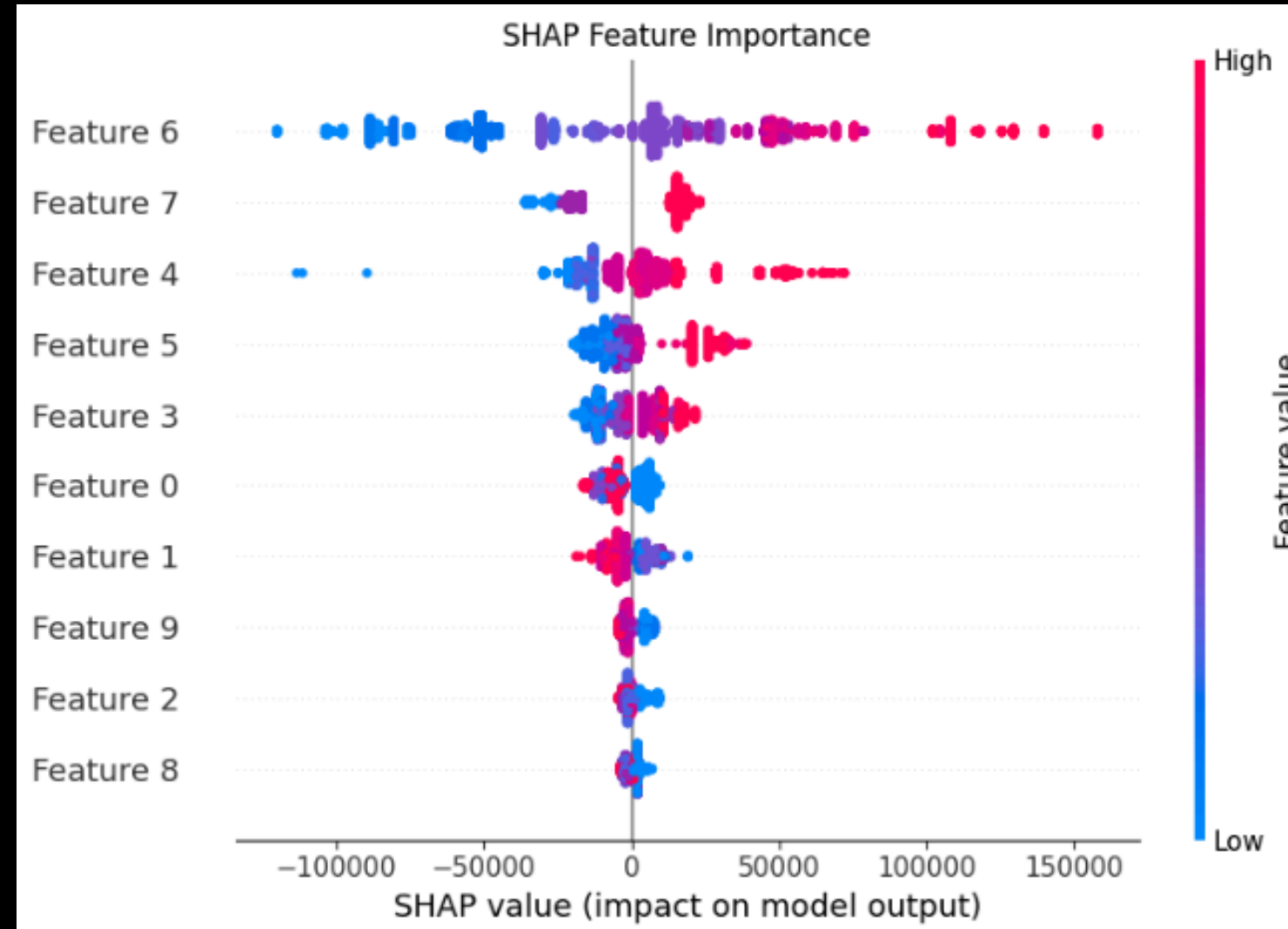
This model explaining about **84%**
of the price variance.

On average, the predicted price of
an apartment deviates from the
actual price by **32.224 WON**.



SHAP

0	Hallway Type
1	TimeToSubway
2	N_FacilitiesNearBy(PublicOffice)
3	N_FacilitiesNearBy(ETC)
4	N_FacilitiesInApt
5	N_Parkinglot(Basement)
6	Size(sqft)
7	YearBuilt
8	N_SchoolNearBy(University)
9	SubwayStation



Limitations

**Data
Timeframe**

**Economic
Factors**

**Localized
Factors**

**Market
Sentiment**



Conclusion and Impact

Business Problems

Market Efficiency

Affordability Crisis

Investment Decisions

Urban Planning

Financial Risk Assessment

Impact of Model Deployment

Revenue Increase

Cost Reduction

Operational Efficiency

Competitive Advantage

Customer Satisfaction

Risk Mitigation



Thankyou !

