Final Project Presentation: NYC Property Sales

Markus Afonso

Problem Definition:

Problem Definition: What is the projected sale price of a 2000 sqft home?

Dataset:

1-10 38177 rows × 20 columns

	borough	neighborhood	building_class_category	tax_class_at_present	block	lot	building_class_at_present	
0	Bronx	Bathgate	01 One Family Dwellings	1	3030	62	A1	44
1	Bronx	Bathgate	01 One Family Dwellings	1	3030	70	A1	44
2	Bronx	Bathgate	01 One Family Dwellings	1	3039	63	A1	46

1-10 38177 rows × 20 columns

address	apartment_number	zip_code	residential_units	commercial_units	total_units	land_square_feet
4463 Park Avenue	NaN	10457.0	1	0	1	1578
4445 Park Avenue	NaN	10457.0	1	0	1	1694
469 E 185th St	NaN	10458.0	1	0	1	1650

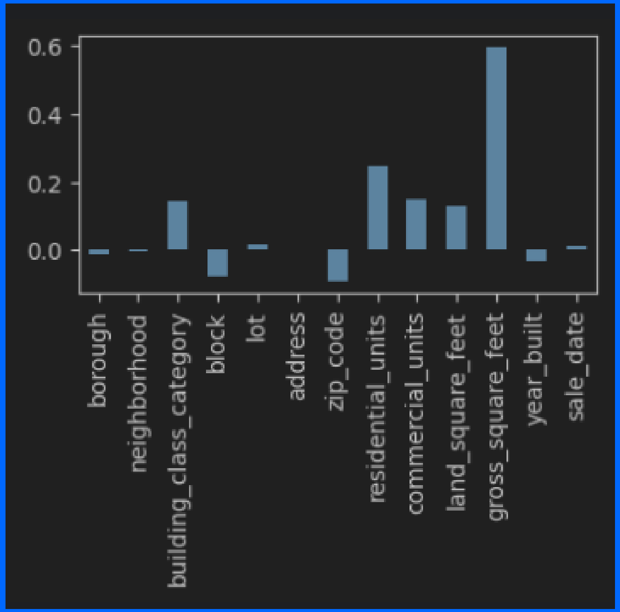
1-10 38177 rows × 20 columns

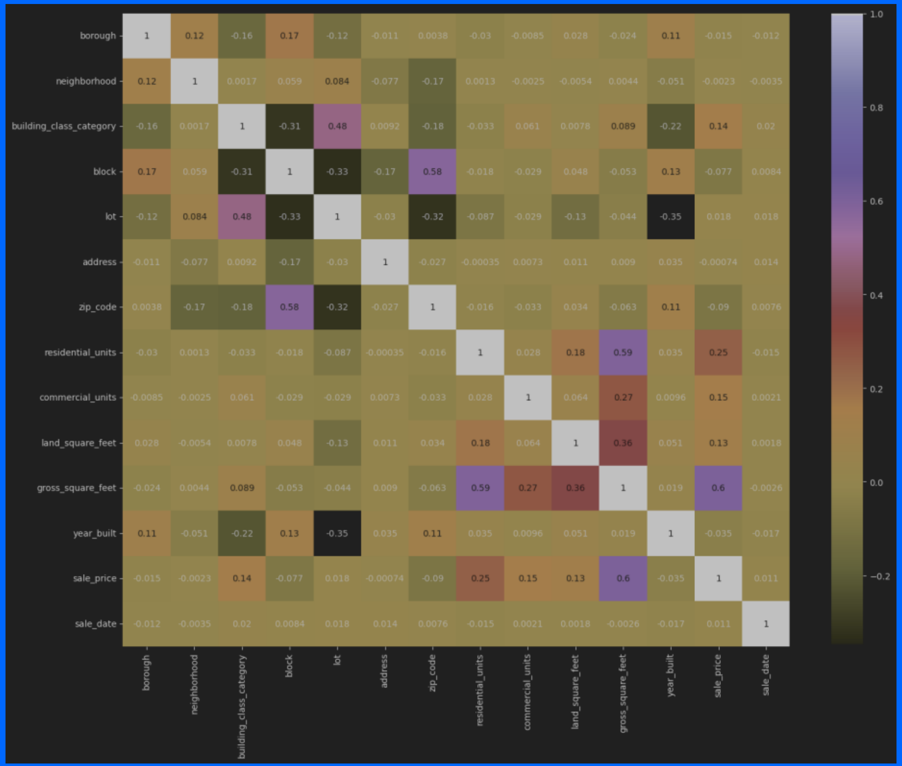
	gross_square_feet	year_built	tax_class_at_time_of_sale	building_class_at_time_of_sale	sale_price	sale_date
78	1470	1899	1	A1	455000	2018-11-28T00:00:00Z
94	1497	1899	1	A1	388500	2019-07-23T00:00:00Z
50	1296	1910	1	A1	419000	2018-12-20T00:00:00Z

Feature Scaling:

gross_square_feet, residential_units, commercial_units,

land_square_feet





Data Modeling - Model Selection

- Decision Tree
  - Simple and easy to understand
- Random Forest
  - multiple decision trees to improve accuracy and reduce overfitting
- Gradient Boosting Regression (GBR)
  - Like random forest
  - Seemed cool

Data Modeling – Implementation

```
X = df.iloc[:, 7:11].values # gross_square_feet, residential_units, commercial_units, land_square_feet
y = df.iloc[:, -2].values # sale_price

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

# feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Executed at 2024.04.11 00:59:25 in 8ms

```
# # decision tree
from sklearn.tree import DecisionTreeClassifier
classifier_dtc = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier_dtc.fit(X_train, y_train)
```

Data Modeling - Hyperparameter Tuning

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Decision Tree Regressor
param_grid_dtr = {
    'criterion': ['mse', 'friedman_mse', 'mae'],
    'max_depth': [None] + list(np.random.randint(1, 21, size=10)),
    'min_samples_split': list(np.random.randint(2, 11, size=10)),
    'min_samples_leaf': list(np.random.randint(1, 5, size=10))
}
regressor_dtr = DecisionTreeRegressor(random_state=0)
grid_search_dtr = GridSearchCV(estimator=regressor_dtr, param_grid=param_grid_dtr, cv=5,
grid_search_dtr.fit(X_train, y_train)
best_params_dtr = grid_search_dtr.best_params_

print("Best Parameters for Decision Tree Regressor:", best_params_dtr)
best_regressor_dtr = grid_search_dtr.best_estimator_

y_pred_dtr = best_regressor_dtr.predict(X_test)
mse_dtr = mean_squared_error(y_test, y_pred_dtr)
print("Mean Squared Error for Decision Tree Regressor:", mse_dtr)

from sklearn.metrics import r2_score
r2_dtr = r2_score(y_test, y_pred_dtr)
print("R-squared for Decision tree:", r2_dtr)
```

```
Maximum Memory Usage (MB): 12.349672786
Maximum CPU Usage (MB): 73.16%
```

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error
import numpy as np

param_dist_dtr = {
    'criterion': ['mse', 'friedman_mse', 'mae'],
    'max_depth': [None] + list(np.random.randint(1, 21, size=10)),
    'min_samples_split': list(np.random.randint(2, 11, size=10)),
    'min_samples_leaf': list(np.random.randint(1, 5, size=10))
}
regressor_dtr = DecisionTreeRegressor(random_state=0)
random_search_dtr = RandomizedSearchCV(estimator=regressor_dtr, param_distributions=param_dist_dtr,
                                       n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=0)
random_search_dtr.fit(X_train, y_train)
best_params_dtr = random_search_dtr.best_params_

print("Best Parameters for Decision Tree Regressor (Randomized Search):", best_params_dtr)
best_regressor_dtr = random_search_dtr.best_estimator_

y_pred_dtr = best_regressor_dtr.predict(X_test)
mse_dtr = mean_squared_error(y_test, y_pred_dtr)
print("Mean Squared Error for Decision Tree Regressor (Randomized Search):", mse_dtr)

from sklearn.metrics import r2_score
r2_dtr = r2_score(y_test, y_pred_dtr)
print("R-squared for Decision Tree (Randomized Search):", r2_dtr)
```

```
Maximum Memory Usage (MB): 1.579681396484375
Maximum CPU Usage (MB): 24.25%
```

Evaluation

```
Best Parameters for Random Forest Regressor: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 10}
Mean Squared Error for Random Forest Regressor: 28060265946.457245
R-squared for Random Forest Regressor: 0.9883415855038089
```

28,060,265,946

```
Best Parameters for Decision Tree Regressor: {'criterion': 'friedman_mse', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2}
Mean Squared Error for Decision Tree Regressor: 15367446607.233301
R-squared for Decision Tree Regressor: 0.9936151687714908
```

15,367,446,607

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
Mean Squared Error: 10672820290.66902
R-squared for GRB: 0.9883415855038089
```

10,672,820,290

Predicted price of a 2000 gross_square_feet_home

```
Predicted price using Random Forest Regressor: 25,288,175.11
Predicted price using Decision Tree Regressor: 940,000.00
Predicted price using Gradient Boosting Regressor: 38,920,038.05
```

Conclusion:

- data analytics techniques
- Clean and transform data
- models using statistical techniques
- data visualization tools
- Evaluate and interpret results
- Machine Learning

Thank you