Exploring Toronto's Rental : Kijiji Data analysis

Aaron Chen, Rosa Lee, Chloe Li, Danika Mariam

TABLE OF CONTENTS

01

OBJECTIVE

Our goal was to illuminate the key drivers of rental pricing. Our insights guide real estate professionals, investors, and tenants through the complexities of the market; which helps to inform and proactive decisions in the fast-paced world of real estate.

02

CODE

CODE: Removing Null Values

```
# count null value of all columns
df.isnull().sum()
```

```
Price($)                    256
Address                      12
Date Posted                 608
Building Type              1025
Bedrooms                   1025
Bathrooms                  1025
Utilities                  1968
Wi-Fi and More             1025
Parking Included           1025
Agreement Type             1025
Move-In Date               2098
Pet Friendly                347
Size (sqft)                1023
Furnished                    47
Air Conditioning           1025
Personal Outdoor Space     1025
Smoking Permitted          1025
Appliances                 1535
Amenities                  2510
Description                 186
Visit Counter              1677
url                           0
dtype: int64
```

CODE: Removing Null Values

```
Price($)                   0
Address                    0
Date Posted              364
Building Type              0
Bedrooms                   0
Bathrooms                  0
Utilities                880
Wi-Fi and More             0
Parking Included           0
Agreement Type             0
Move-In Date            1064
Pet Friendly               0
Size (sqft)                0
Furnished                  0
Air Conditioning           0
Personal Outdoor Space     0
Smoking Permitted          0
Appliances               469
Amenities               1396
Description              165
Visit Counter           1595
url                        0
dtype: int64
```

```
# Amenities, Visit Counter has too much null, dropped
df = df.drop(columns=['Amenities', 'Visit Counter'])
# Appliances can be re-visited later if needed
# maybe categorical variable for Freezer/Laundry/Dishwasher?
df = df.drop(columns=['Appliances'])
# Description not needed, move in date and date posted can be re-visit later if need
df = df.drop(columns=['Date Posted', 'Move-In Date', 'Description'])
# don't need address, url
df = df.drop(columns=['Address', 'url'])
```

CODE: Mapping Categorical Variables

```python
# original column unique values
'''
array([nan, 'Hydro_No,Heat_No,Water_Yes', 'Hydro_No,Heat_Yes,Water_Yes',
       'Hydro_No,Heat_Yes,Water_No', 'Hydro_Yes,Heat_Yes,Water_Yes',
       'Hydro_Yes,Heat_No,Water_Yes', 'Hydro_Yes,Heat_Yes,Water_No',
       'Hydro_Yes,Heat_No,Water_No'], dtype=object)
'''
# can assume nan = No, No, No, split utilities to three columns
# Hydro, Heat, and Water all with 0 being no, 1 being yes
df[['Hydro', 'Heat', 'Water']] = df['Utilities'].str.split(',', expand=True)
df['Hydro'] = df['Hydro'].str.split('_').str[1].map({'Yes': 1, 'No': 0}).fillna(0)
df['Heat'] = df['Heat'].str.split('_').str[1].map({'Yes': 1, 'No': 0}).fillna(0)
df['Water'] = df['Water'].str.split('_').str[1].map({'Yes': 1, 'No': 0}).fillna(0)
df = df.drop(columns=['Utilities'])
```

CODE: Mapping Categorical Variables

```
# original column unique values
'''
array(['Apartment', 'Condo', 'Basement', 'House', 'Duplex/Triplex',
       'Townhouse'], dtype=object)
'''
# Basement < Apartment < Condo < Duplex/Triplex < Townhouse < House
mapping = {
    'Basement': 0,
    'Apartment': 1,
    'Condo' : 2,
    'Duplex/Triplex': 3,
    'Townhouse' : 4,
    'House' : 5
}
# reduce one-hot dimension since we have limited data
# treat apartment and condo as same
df['Building Type'] = df['Building Type'].replace(mapping)
```

CODE: Mapping Categorical Variables

```
# original column unique values
'''
array(['Not Included', 'Balcony', 'Yard', 'YardBalcony'], dtype=object)
'''
# treat outdoor space as ordinal (No < Bal < Yard < YardBal)
mapping = {
    'Not Included' : 0,
    'Balcony' : 1,
    'Yard' : 2,
    'YardBalcony' : 3
}
df['Personal Outdoor Space'] = df['Personal Outdoor Space'].replace(mapping)
df['Personal Outdoor Space'].unique()
```

CODE: Mapping Categorical Variables

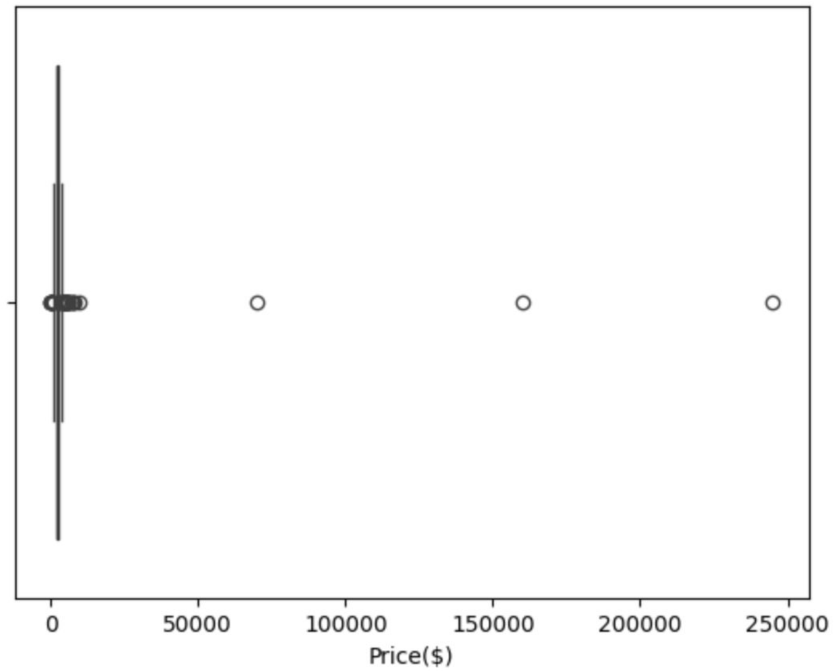
```python
# original column unique values
'''
array(['2', 'Bachelor/Studio', '1', '3', '2 + Den', '1 + Den', '3 + Den',
       '4', '5+', '4 + Den'], dtype=object)
'''
# treat den as one more room
mapping = {
    'Bachelor/Studio':0,
    '1': 1,
    '1 + Den': 2,
    '2': 3,
    '2 + Den': 4,
    '3': 5,
    '3 + Den': 6,
    '4' : 7,
    '4 + Den' : 8,
    '5+' : 9,
}

df['Bedrooms'] = df['Bedrooms'].replace(mapping)
df['Bedrooms'].unique()
```

```
array([3, 0, 1, 5, 4, 2, 6, 7, 9, 8])
```

CODE: Removing Outliers

```python
# three datapoint that is too much of a outlier
sns.boxplot(data=df, x='Price($)')
```

```
<Axes: xlabel='Price($)'>
```

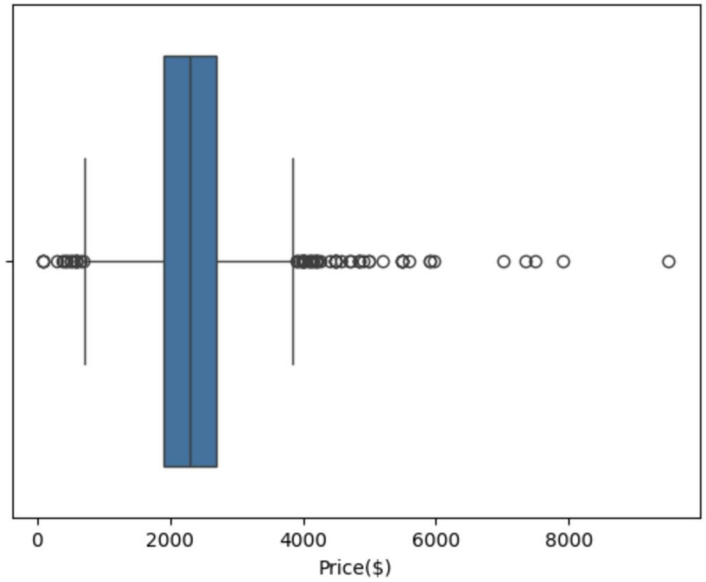


CODE: Removing Outliers

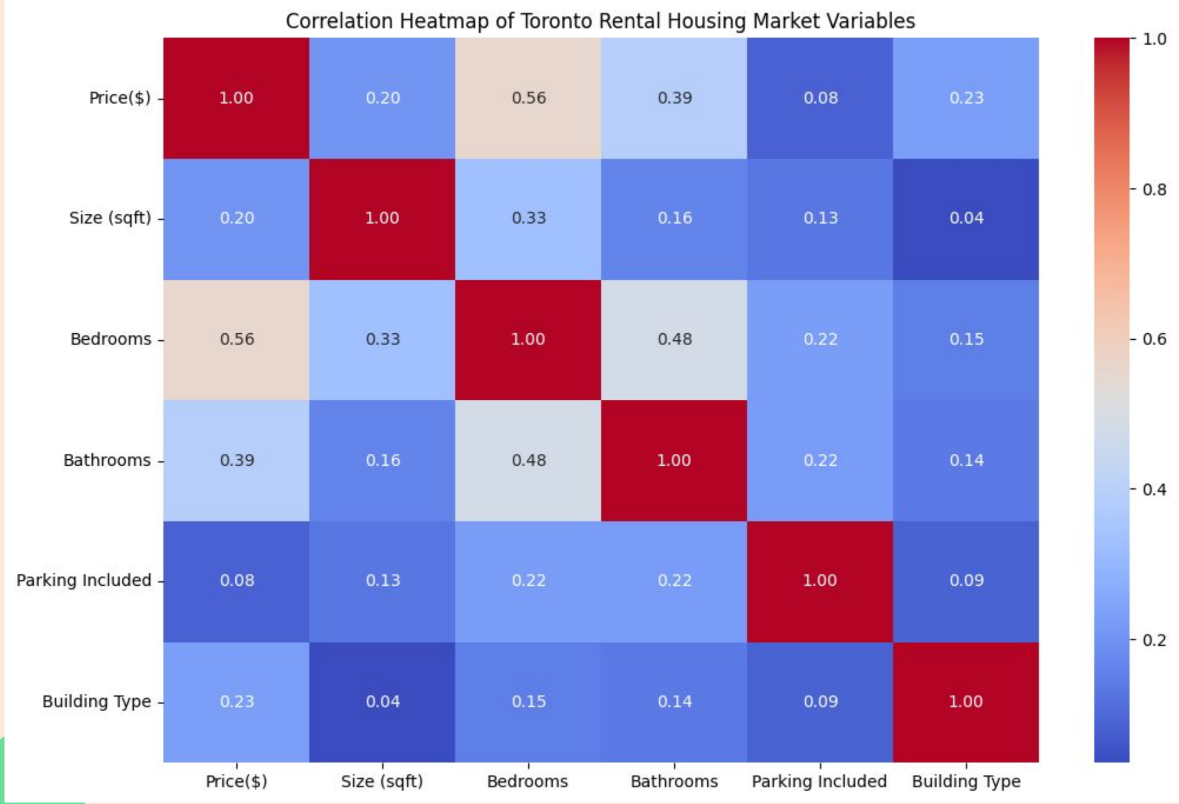
```python
# there are three datapoint with a very high price, remove it (higher then 50000)
df = df.loc[df['Price($)'] < 50000]
```

```python
sns.boxplot(data=df, x='Price($)')
```

```
<Axes: xlabel='Price($)'>
```



CODE: Creating a Heatmap



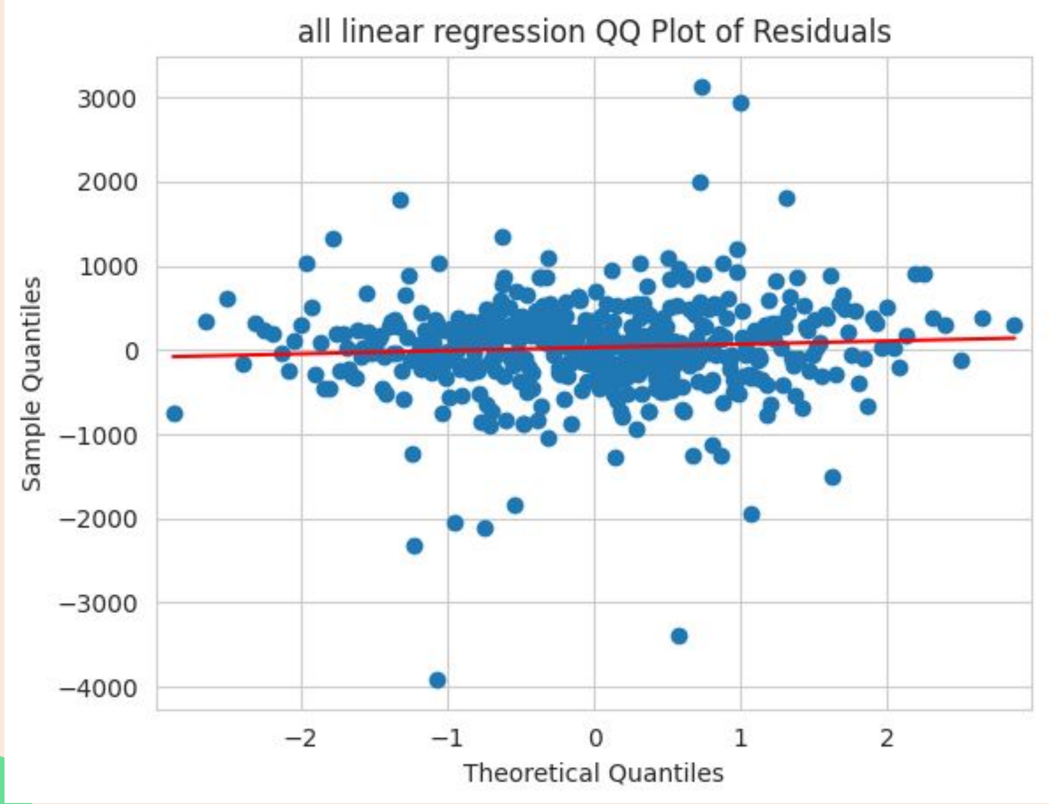
CODE: Linear Regression

```
for train_idx, test_idx in kfold.split(X_all):
  # we don't want to normalize categorical data
  X_train_cat = X_all[cat_columns].values[train_idx]
  X_test_cat = X_all[cat_columns].values[test_idx]

  # get numeric value from data set
  X_train, X_test = X_all[num_columns].values[train_idx], X_all[num_columns].values[test_idx]
  y_train, y_test = y_all.values[train_idx], y_all.values[test_idx]

  # normalize
  X_train, X_test = normalize_data(X_train, X_test)

  # add the categorical data back
  X_train = np.append(X_train, X_train_cat, axis=1)
  X_test = np.append(X_test, X_test_cat, axis=1)

  result = linear_regression(X_train, X_test, y_train, y_test, num_columns + cat_columns, str(count+1) + 'fold', False)

  mse[count] = result

  count += 1

  # Print the accuracy for the current fold
  print("Fold {}: MSE: {}".format(count, result))

# Print the average accuracy across all folds
print("Average Score: {} ({})".format(np.mean(mse), np.std(mse)))
```

CODE: Linear Regression

```
mean square error:  484.84907816822164
Fold 1: MSE: 484.84907816822164
mean square error:  840.3657109164114
Fold 2: MSE: 840.3657109164114
mean square error:  536.690685282092з
Fold 3: MSE: 536.6906852820923
mean square error:  708.15361332279 4
Fold 4: MSE: 708.15361332279 4
mean square error:  920.2071872759722
Fold 5: MSE: 920.2071872759722
Average Score: 698.0532549930983 (168.04827990632302)
mean square error:  717.0330612310703
```

CODE: Linear Regression



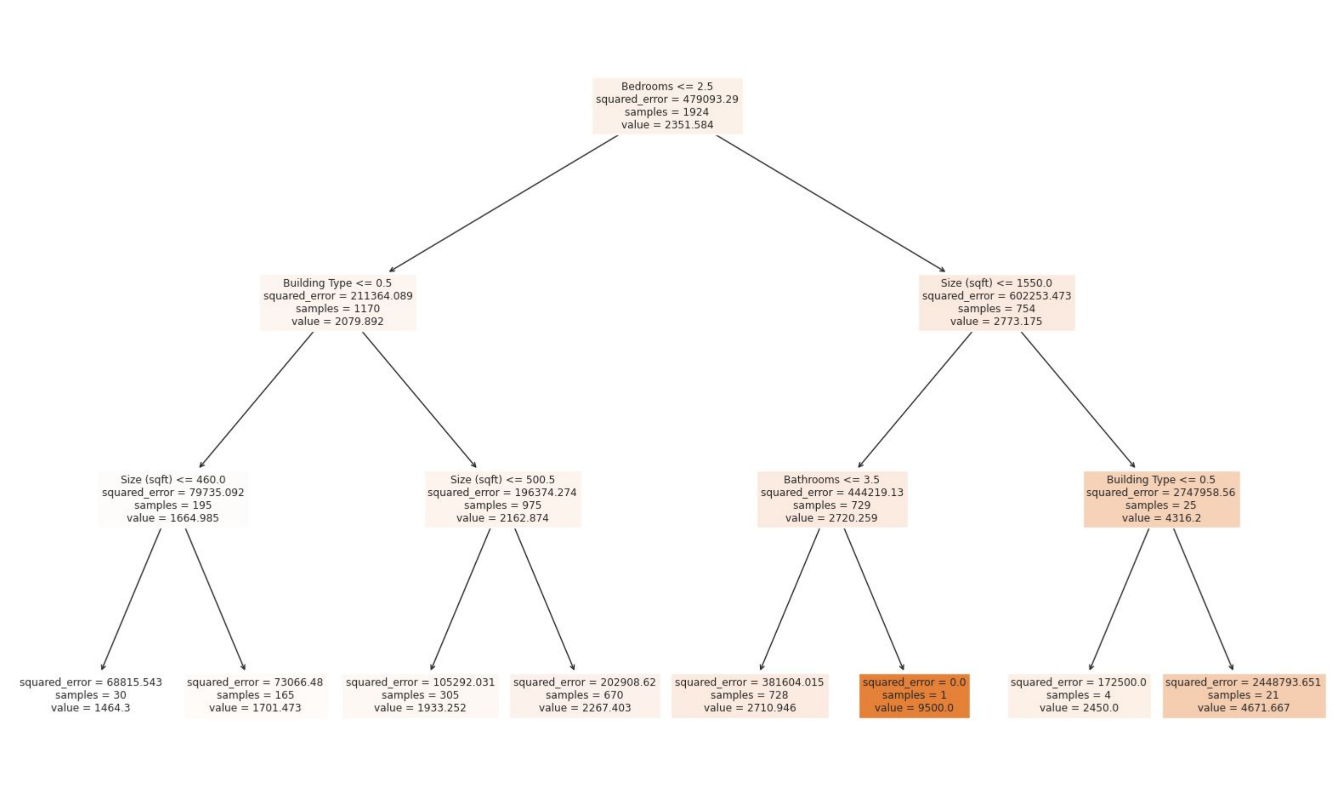
CODE: Decision Tree

```
#####################################DECISION TREES###########################

# defince x column and y column for current model
columns = ['Size (sqft)', 'Bedrooms', 'Bathrooms', 'Parking Included', 'Building Type',
           'Agreement Type', 'Personal Outdoor Space', 'Smoking Permitted',
           'Hydro', 'Heat', 'Water', 'Pet Friendly', 'Air Conditioning', 'Furnished',
           'Wi-Fi and More']

# k-fold cross validation
num_folds = 5
kfold = KFold(n_splits = num_folds)
kfold.get_n_splits(X_all)

best_mse = float('inf')
best_depth = 3
best_std = 0

for depth in [3, 5, 50, 500]:
  mse = np.zeros(num_folds)
  count = 0
  for train_idx, test_idx in kfold.split(X_all):
    # dont normalize data for better visualization
    X_train, X_test = X_all[columns].values[train_idx], X_all[columns].values[test_idx]
    y_train, y_test = y_all.values[train_idx], y_all.values[test_idx]

    # run the decision tree model
    result = decisionTreeRegression(X_train, X_test, y_train, y_test, columns, False, depth)

    mse[count] = result
    count += 1
```

CODE: Decision Tree

```
  # find the best parameter
  if np.mean(mse) < best_mse:
    best_mse = np.mean(mse)
    best_depth = depth
    best_std = np.std(mse)

# Print the best parameters and the corresponding score
print("The optimal decision tree model uses depth={}, achieving a cross-validation mse of {} with a standard d

###############################################

# run model on all data

# get numeric value from data set
X_all_num, X_not_seen_num = X_all[columns].values, X_not_seen[columns].values

# convert to numpy
y_all_num, y_not_seen_num = y_all.values, y_not_seen.values

# regression tree
decisionTreeRegression(X_all_num, X_not_seen_num, y_all_num, y_not_seen_num, columns, True, depth=best_depth)
```

CODE: Decision Tree



CODE: Decision Tree

- The optimal decision tree model uses depth=3, achieving a cross-validation mse of 868.027850800712 with a standard deviation of 197.0257722409644l.

- mean square error: 1016.4675278747675

CODE: Random Forest

```
# k-fold cross validation
num_folds = 5
kfold = KFold(n_splits = num_folds)
kfold.get_n_splits(X_all)

best_mse = float('inf')
best_depth = 3
best_std = 0

for depth in [3, 5, 50, 500]:
  mse = np.zeros(num_folds)
  count = 0
  for train_idx, test_idx in kfold.split(X_all):
    # dont normalize data for better visualization
    X_train, X_test = X_all[columns].values[train_idx], X_all[columns].values[test_idx]
    y_train, y_test = y_all.values[train_idx], y_all.values[test_idx]

    # run the decision tree model
    result = randomForestRegression(X_train, X_test, y_train, y_test, columns, False, depth)

    mse[count] = result
    count += 1

  # find the best parameter
  if np.mean(mse) < best_mse:
    best_mse = np.mean(mse)
    best_depth = depth
    best_std = np.std(mse)

# Print the best parameters and the corresponding score
print("The optimal random forest model uses depth={}, achieving a cross-validation mse of {} with a standard deviation of {}.".
```

CODE: Random Forest

	Features	Importances
0	Size (sqft)	0.280696
1	Bedrooms	0.305019
2	Bathrooms	0.068343
3	Parking Included	0.031840
4	Building Type	0.099116
5	Agreement Type	0.037654
6	Personal Outdoor Space	0.029152
7	Smoking Permitted	0.005621
8	Hydro	0.023328
9	Heat	0.015275
10	Water	0.022462
11	Pet Friendly	0.020649
12	Air Conditioning	0.016996
13	Furnished	0.026022
14	Wi-Fi and More	0.017826

CODE: Random Forest

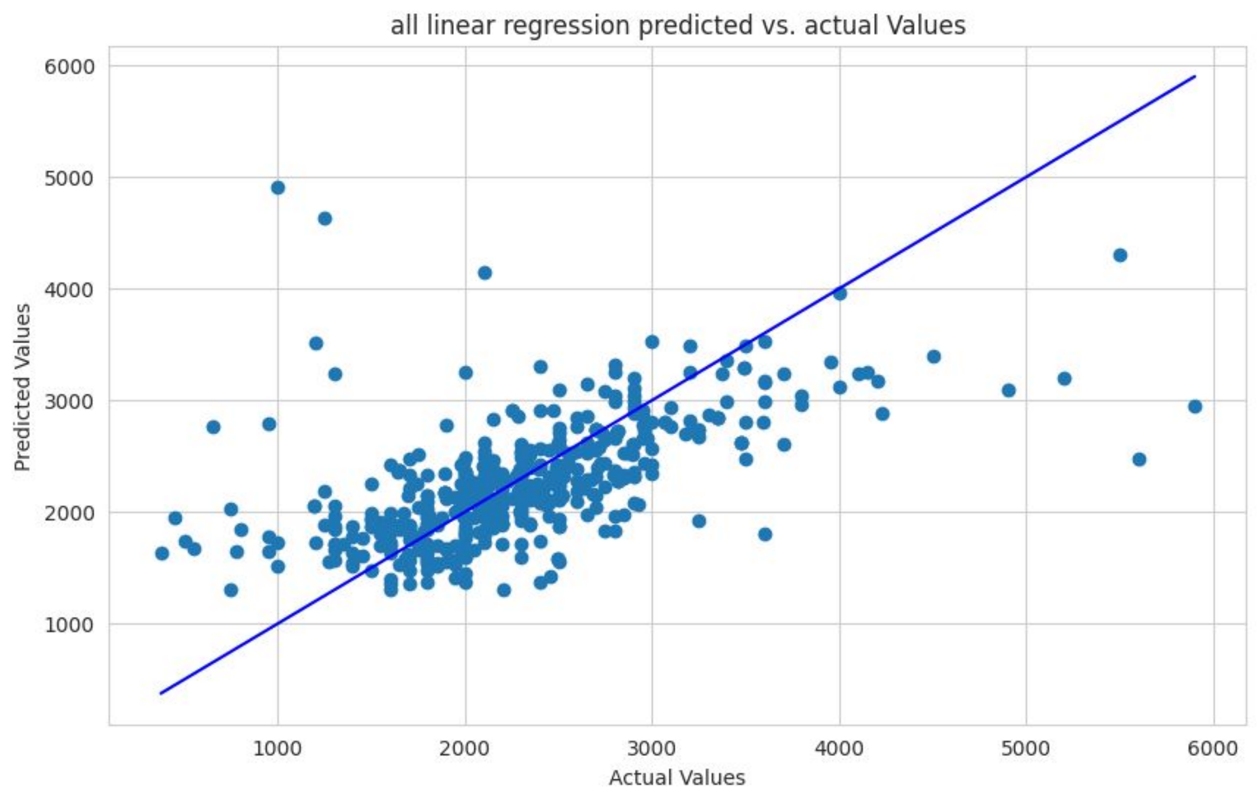
- The optimal random forest model uses depth=50, achieving a cross-validation mse of 656.5471401685129 with a standard deviation of 200.93439158553892.

- mean square error: 648.9013065548285

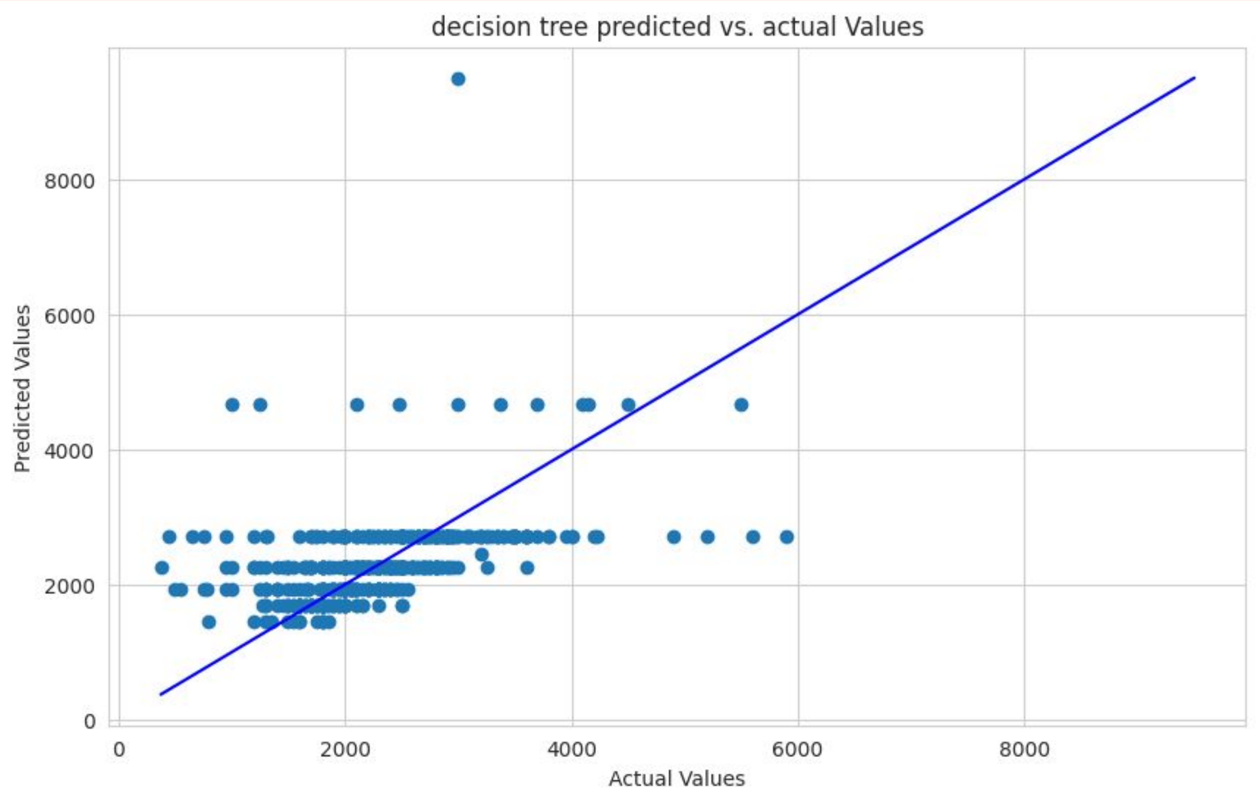
03

RESULTS

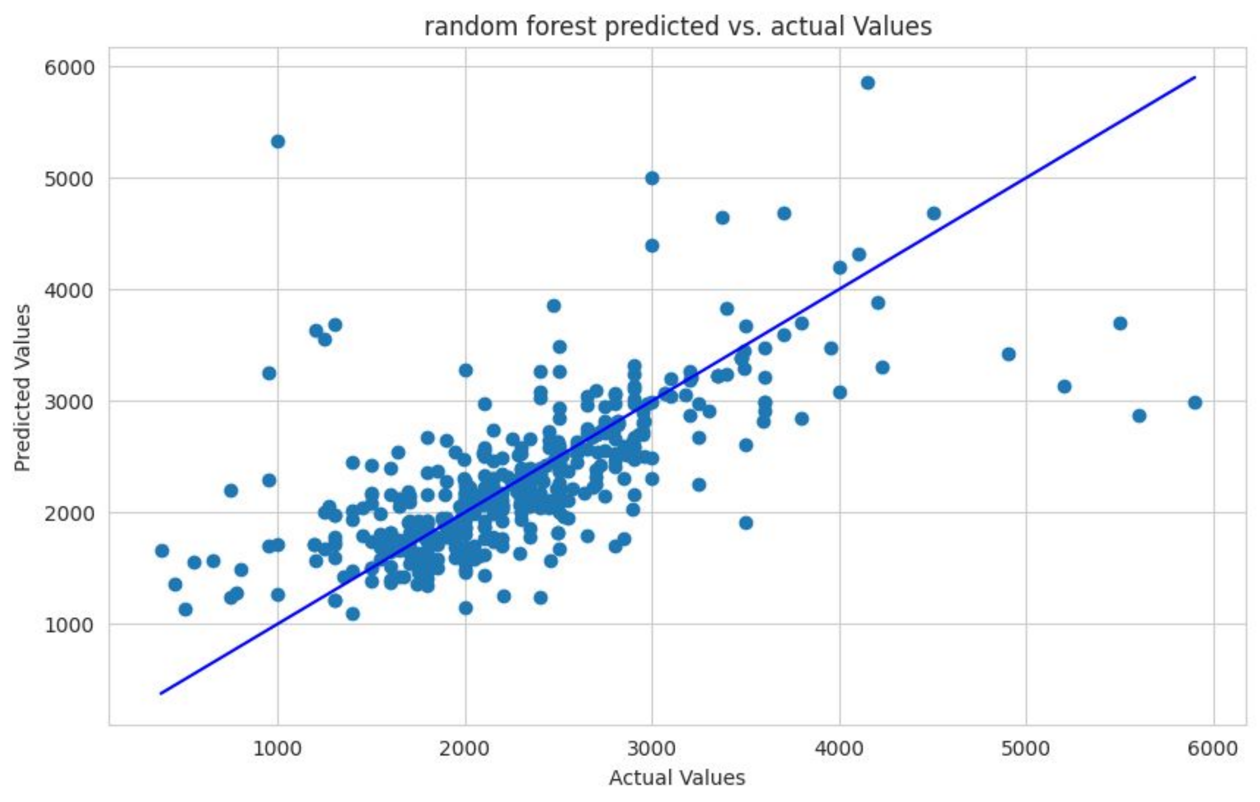
Linear Regression



Decision Tree



Random Forest



Influential Features

Bedroom count significantly affects rental prices; size matters less.

Model Performance

Random Forest outperforms other models in prediction accuracy.

Amenity Impact

Traditional amenities like Wi-Fi have minimal effect on pricing.

Strategic Use of Insights

Models reflect current trends, inform future market strategies.

04

CONCLUSION

Data-Driven Decisions

Our analysis has equipped stakeholders with the knowledge to make informed, strategic decisions in Toronto's dynamic rental market.

Predictive Power

We have demonstrated the predictive power of machine learning in real estate, translating complex data into understandable trends.

Market Trends

Our findings underscore the importance of focusing on key property features that align with market expectations and demand.

Forward-Looking

The insights we provide are a foundation for anticipating shifts in the rental landscape, ensuring stakeholders are prepared for the future.

Limitations

Area Data

Our model does not account for area-related pricing. As we know, postal code has an affect on housing prices due to school districts and community resources.

Removed Values

We removed the columns with many null values in the beginning of our analysis. This included: Appliances, Amenities, and Move-in date. A future model could incorporate this information for more accurate results.

THANK YOU