STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

HOUSE PRICE PREDECTION

CONTENT

01 Members

02 Abstract

03 Introduction

04 Related Work

05 Our Solution

06 Comparison of Models

07 Future Directions

08 Conclusion

TEAM MEMBERS - GROUP NO: 17

Visharad Ravi

Uday Kurella

Shivanshu Vinay Singh

Dani Martinez

ABSTRACT

- Real estate influenced by property features, location, and economic indicators.
- Improve the accuracy of using ML for house price prediction.
- Extensive data includes square footage, bedrooms, neighborhood attributes, and economic indicators.
- Preprocessing and feature engineering handle missing data and standardize variables.
- Linear Regression, XGB Regression, and Random Forest used for prediction models. Evaluated using metrics like MAE, MSE, and RMSE.

INTRODUCTION

- Accurate home price estimation requires a deep understanding of diverse factors influencing property values in the dynamic real estate market.
- Traditional methods struggle with complex patterns and non-linear relationships inherent in real estate data.
- Machine learning algorithms have transformed the industry, providing robust tools to accurately assess and forecast home prices.
- Goal is to develop a robust housing price forecast model using machine learning techniques.
- Aiming to enhance forecasting skills to aid investors, real estate agents, and homeowners in making informed decisions through sophisticated algorithms.

RELATED WORK - 1

01 Paper Title- House Resale Price Prediction Using Classification Algorithms

- Classification algorithms like K-Means, Random Forest, Decision Tree, and Linear Regression used for forecasting.
- Home prices affected by location, economic conditions, and physical characteristics.
- RMSE employed to evaluate accuracy across datasets, aiming for the most precise forecasting model.

RELATED WORK - 2

02 Paper Title- A hybrid regression technique for house prices prediction

- Developed a hybrid regression method for housing price prediction.
- A predictive model is constructed using several Machine Learning methods, such as Random Forest, Decision Tree, and Linear Regression.
- Random Forest demonstrated the highest accuracy, around 87%, in the training set.

RELATED WORK - 3

03 Paper Title- Housing Price Prediction Using Machine Learning Algorithms: The Case of Melbourne City, Australia

- Examines house price prediction in Melbourne using ML algorithms.
- Utilizes PCA for optimal data reduction, providing insightful information about Melbourne's housing sector.
- Applies Support Vector Machines (SVMs) and competes with various regression models, employing diverse techniques for optimal outcomes.

OUR SOLUTION

A Description of Dataset

B Machine Learning Algorithms

C Implementation Details

A - DESCRIPTION OF DATASET

- **Market Insight:** Analyzing Sydney and Melbourne's real estate values offers key economic and market indicators.

- **Data Organization:** A large real estate dataset is refined, emphasizing crucial factors like price, bedrooms, and bathrooms.

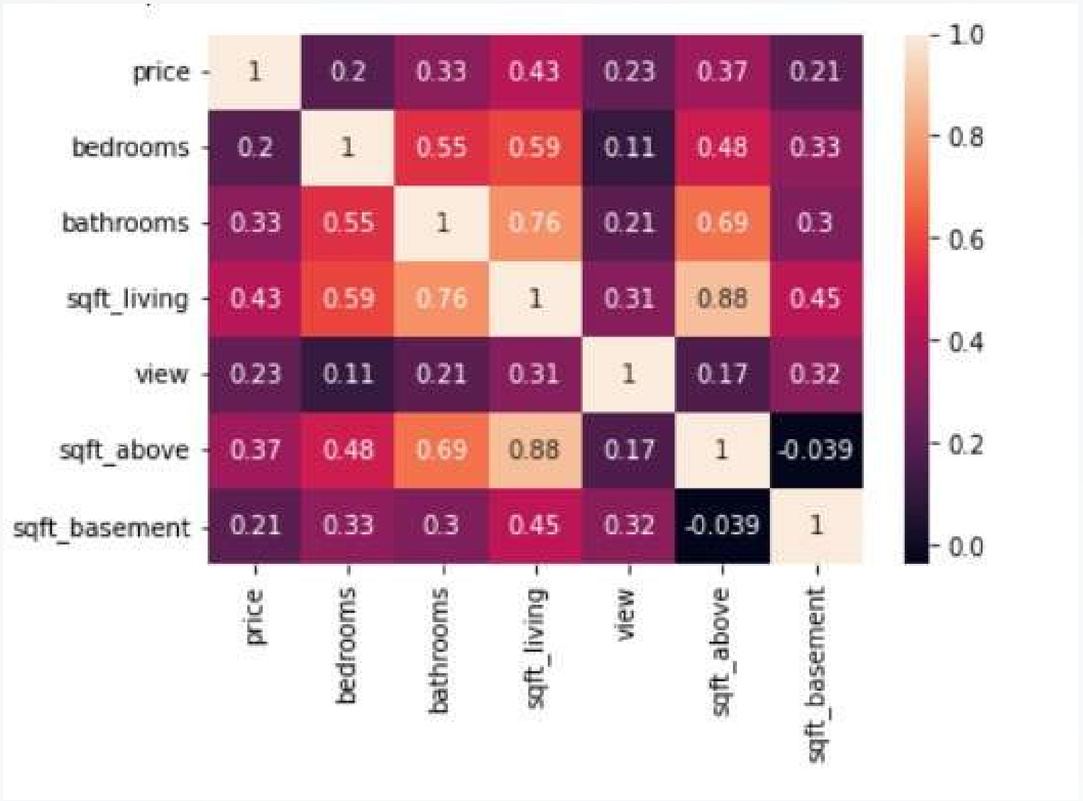
```
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          4600 non-null   float64
 1   bedrooms       4600 non-null   float64
 2   bathrooms      4600 non-null   float64
 3   sqft_living    4600 non-null   int64
 4   sqft_lot       4600 non-null   int64
 5   floors         4600 non-null   float64
 6   waterfront     4600 non-null   int64
 7   view           4600 non-null   int64
 8   condition      4600 non-null   int64
 9   sqft_above     4600 non-null   int64
 10  sqft_basement  4600 non-null   int64
 11  yr_built       4600 non-null   int64
 12  yr_renovated   4600 non-null   int64
 13  city           4600 non-null   object
```

A - DESCRIPTION OF DATASET

- **Heatmap Analysis:** Visualizing relationships reveals insights, with red squares indicating positive correlations and blue squares showing negative ones.

- **Positive Associations:** Significant positive links found in features like living area, bedrooms, and a view influence higher property prices.

- **Negative Correlations:** Basement square footage exhibits the most substantial negative correlation, impacting lower property prices.



B - MACHINE LEARNING ALGORITHMS

Linear Regression

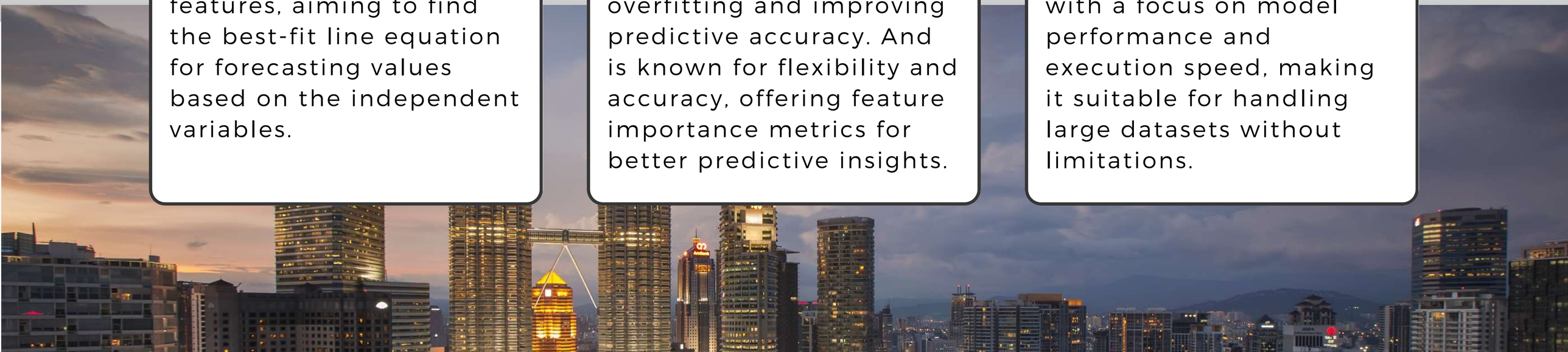
Linear regression calculates the linear relationship between a dependent variable and one or more independent features, aiming to find the best-fit line equation for forecasting values based on the independent variables.

Random Forest

Random Forest is an ensemble learning method that builds diverse decision trees during training, reducing overfitting and improving predictive accuracy. And is known for flexibility and accuracy, offering feature importance metrics for better predictive insights.

XGBoost

XGBoost, a robust open-source tool, combines decision trees and gradient boosting for creating superior models, with a focus on model performance and execution speed, making it suitable for handling large datasets without limitations.



- **Predictive Modeling**: Linear regression is a powerful predictive modeling technique that helps in estimating the relationship between the independent variables (features) and the dependent variable (house price).

- **Interpretability:** Linear regression clarifies variable impact, vital in transparent real estate.

- **Assumption of Linearity:** Linear regression assumes a linear relationship. In the context of house prices, many factors, such as square footage or the number of bedrooms, may exhibit a linear correlation with the property's value.

- **Simplicity and Speed:** In the dynamic real estate market, where quick and reliable predictions are essential, the simplicity and speed of linear regression make it a pragmatic choice for house price prediction.

- **Ensemble Learning:** Random Forest, as an ensemble method, combines multiple decision trees to improve prediction accuracy, making it effective for capturing complex relationships in house price data.

- **Feature Importance**: Random Forest provides a feature importance metric, aiding in the identification of key factors influencing house prices among numerous variables.

- **Robust to Overfitting**: The ensemble nature of Random Forest mitigates overfitting, enhancing generalization performance on diverse house price datasets.

- **Handles Non-Linearity**: Random Forest can model non-linear relationships, accommodating the intricate interactions between various features impacting house prices.

XG BOOST

- **Gradient Boosting Power:** XGBoost excels in capturing complex patterns for accurate house price predictions.
- **Efficiency and Scalability:** It efficiently handles large datasets, making it ideal for comprehensive house price prediction tasks.
- **Regularization Techniques:** Integrated regularization prevents overfitting, enhancing model robustness in house price forecasting.
- **Enhanced Speed:** Optimized for speed, XGBoost provides faster training and prediction for real-time applications.

C - IMPLEMENTATION DETAILS

In [1]:
```
import pandas as pd
import numpy as np

from __future__ import print_function
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

```
C:\Users\vrvis\AppData\Local\Programs\Python\Python310\lib\site-packages\xgboost\compat.py:36: FutureWarning: pandas.Int64Index
is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

In [2]:
```
data = pd.read_csv("data.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	st
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	18 Densr A
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	70 Blair
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26 2(143rd
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 1 P
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	9 170th

In [4]:
```
data.tail()
```

Out[4]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated
4595	2014-07-09 00:00:00	308166.666667	3.0	1.75	1510	6360	1.0	0	0	4	1510	0	1954	197
4596	2014-07-09 00:00:00	534333.333333	3.0	2.50	1460	7573	2.0	0	0	3	1460	0	1983	200
4597	2014-07-09 00:00:00	416904.166667	3.0	2.50	3010	7014	2.0	0	0	3	3010	0	2009	0
4598	2014-07-10 00:00:00	203400.000000	4.0	2.00	2090	6630	1.0	0	0	3	1070	1020	1974	0
4599	2014-07-10 00:00:00	220600.000000	3.0	2.50	1490	8102	2.0	0	0	4	1490	0	1990	0

In [8]:
```
plt.figure(figsize=(12, 10))
sns.heatmap(data.corr(), annot=True)
plt.show()
```

From the plot we can deduce that these's is some collinearity between 2 variables: i.e. **sqft_living and sqft_above**. Since they have correlation ratio more than 0.2 lets restrict the matrix.

In [9]:
```
cor_matrix = data.corr()
top_cor_feature = cor_matrix.index[abs(cor_matrix['price'])>0.2]
plt.figure()
sns.heatmap(data[top_cor_feature].corr(), annot=True)
```

Out[9]: <AxesSubplot:>

In [19]:
```
# Filtering Outliers
q1 = data.quantile(0.25)
q3 = data.quantile(0.75)
iqr = q3 - q1
data = data[~((data<(q1 - 1.5 * iqr))| (data > (q3 + 1.5 * iqr))).any(axis=1)]
```

In [20]:
```
# splitting the dataset into dependent and independent columns.
x = data.drop('price', axis=1)
y = data['price']
```

In [21]:
```
x.head()
```

Out[21]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	city
0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	0
2	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	2
3	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	3
4	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	4
5	2.0	1.00	880	6380	1.0	0	0	3	880	0	1938	1994	1

In [22]:
```
y.head()
```

Out[22]:
```
0    313000.0
2    342000.0
3    420000.0
4    550000.0
5    490000.0
Name: price, dtype: float64
```

In [15]:
```
# One Hot Encoding
data.city.unique()
```

Out[15]:
```
array(['Shoreline', 'Seattle', 'Kent', 'Bellevue', 'Redmond',
       'Maple Valley', 'North Bend', 'Lake Forest Park', 'Sammamish',
       'Auburn', 'Des Moines', 'Bothell', 'Federal Way', 'Kirkland',
       'Issaquah', 'Woodinville', 'Normandy Park', 'Fall City', 'Renton',
       'Carnation', 'Snoqualmie', 'Duvall', 'Burien', 'Covington',
       'Inglewood-Finn Hill', 'Kenmore', 'Newcastle', 'Mercer Island',
       'Black Diamond', 'Ravensdale', 'Clyde Hill', 'Algona', 'Skykomish',
       'Tukwila', 'Vashon', 'Yarrow Point', 'SeaTac', 'Medina',
       'Enumclaw', 'Snoqualmie Pass', 'Pacific', 'Beaux Arts Village',
       'Preston', 'Milton'], dtype=object)
```

In [16]:
```
data['city']=data['city'].apply({'Shoreline':0,'Seattle':1,'Kent':2,'Bellevue':3,'Redmond':4,'Maple Valley':5,'North Bend':6,'Lak
                                 'Sammamish':8,'Auburn':9,'Des Moines':10,'Bothell':11,'Federal Way':12,'Kirkland':13,'Issaquah'
                                 'Woodinville':15,'Normandy Park':16,'Fall City':17,'Renton':18,'Carnation':19,'Snoqualmie':20,
                                 'Duvall':21,'Burien':22,'Covington':23,'Inglewood-Finn Hill':24,'Kenmore':25,'Newcastle':26,'Mer
                                 'Black Diamond':28,'Ravensdale':29,'Clyde Hill':30,'Algona':31,'Skykomish':32,'Tukwila':33,'Vash
                                 'Yarrow Point':35,'SeaTac':36,'Medina':37,'Enumclaw':38,'Snoqualmie Pass':39,'Pacific':40,'Beau
                                'Preston':42,'Milton':43}.get)
```

In [23]:
```
# Splitting 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=40)
```

Linear Regressiong

In [24]:
```
lr = LinearRegression()
lr.fit(x_train,y_train)
```

Out[24]:
```
LinearRegression()
```

In [25]:
```
lr_pred = lr.predict(x_test)
lr_pred
```

```
408359.44951027, 470562.73372458, 321295.92187466, 440534.937979  ,
426879.45484514, 759361.03903134, 674253.12203565, 429952.67886884,
429426.27694835, 815892.405714  , 377786.838594  , 418351.10661561,
602880.36837821, 321770.97359078, 379857.96961852, 533843.69214801,
295707.98785172, 435714.76253843, 322126.47494122, 385045.81782514,
354013.73808113, 499423.14887871, 604898.16842492, 648538.68365585,
831754.50319458, 351890.86028119, 593959.36147272, 506291.43948191,
524138.86558661, 466554.73781148, 268573.81554902, 404642.72796466,
558866.66092414, 546423.30888605, 470631.43838952, 672120.24762372,
466195.35907408, 351373.19455178, 472773.614413  , 460568.34819116,
442805.20876466, 445681.73204391, 544465.88022653, 422183.73269039,
399056.12186521, 389327.88125582, 734599.79935715, 436199.2554065 ,
340558.02159566, 375921.69969104, 797313.85255456, 476977.72406464,
474433.75628409, 382786.43702889, 454604.2301973 , 353011.02304871,
683830.45720177, 459646.31906675, 298871.29143043, 384780.25856038,
223286.35609107, 343908.82464698, 342998.43048989, 602990.10110804,
535499.97400958, 407567.39462376, 600572.56189217, 488667.86137735,
380995.65605994, 519461.32672544, 426667.73662996, 395210.5726165 ,
530454.959328  , 606274.67253193, 231016.8284609 , 596768.06463318,
```

In [26]:
```
lr.score(x,y)
```

Out[26]:
```
0.43714954611069734
```

In [27]:
```
print('MAE:', mean_absolute_error(y_test, lr_pred))
print('MSE:', mean_squared_error(y_test, lr_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, lr_pred)))
```

```
MAE: 112531.24906249391
MSE: 21529119290.48216
RMSE: 146728.04534403828
```

Random Forest

In [28]:
```
regressor = RandomForestRegressor(n_estimators=300, random_state=0)
regressor.fit(x_train,y_train)
```

Out[28]:
```
RandomForestRegressor(n_estimators=300, random_state=0)
```

In [29]:
```
random_prediction = regressor.predict(x_test)
random_prediction
```

```
524606.43958333, 230771.82444444, 275397.47563492, 236858.53455159,
507064.19148148, 287856.49722221, 451833.45      , 359408.83338908,
318953.29653439, 377519.09333333, 536027.8       , 524555.03333333,
563313.4391207 , 357641.03333333, 603966.17904761, 424484.34      ,
623894.52313131, 461464.21333333, 309155.21666667, 498765.06444444,
692235.50333333, 163738.04269841, 274939.68754631, 492910.235     ,
753408.24666667, 697851.18402776, 279281.76744048, 527964.83619048,
579974.13512821, 183177.02066667, 395899.13370371, 198959.21333333,
397783.16666667, 618129.16666667, 530251.91666667, 325890.98717948,
544124.02666667, 283726.48000002, 680523.8847619 , 480580.08837607,
471111.5       , 453145.44333331, 494471.63827839, 667110.84952378,
288242.39138889, 390806.72614673, 211480.68222222, 530993.86166667,
302711.36547364, 487919.53888889, 308267.56115082, 829813.50833333,
779658.37311111, 386542.05555555, 422680.443     , 413792.77111111,
337237.32555555, 565533.91666667, 315625.73238095, 511732.95851852,
383329.98666667, 215433.84480341, 741976.44738095, 592419.29869047,
526994.38888889, 250337.46822222, 519150.67619048, 408256.99333333,
572286.46846154, 352507.83      , 354647.65465813, 558860.33333333,
620106.55      , 489931.73888888, 557188.56564103, 462912.96705686,
295111.41666667, 742804.125     , 429440.56888889, 318429.35166667,
```

In [30]:
```
regressor.score(x,y)
```

Out[30]:
```
0.8910318063520745
```

In [31]:
```
print('MAE:', mean_absolute_error(y_test, random_prediction))
print('MSE:', mean_squared_error(y_test, random_prediction))
print('RMSE:', np.sqrt(mean_squared_error(y_test, random_prediction)))
```

```
MAE: 80970.53329218418
MSE: 13752804373.197418
RMSE: 117272.35127342428
```

XGBoost

In [32]:
```
xgbr = XGBRegressor(n_estimators=5000, learning_rate=0.05, max_depth=4, random_state=5)
xgbr.fit(x_train, y_train)
```

```
C:\Users\vrvis\AppData\Local\Programs\Python\Python310\lib\site-packages\xgboost\data.py:262: FutureWarning: pandas.Int64Index
is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
```

Out[32]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.05, max_delta_step=0,
             max_depth=4, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=5000, n_jobs=12,
             num_parallel_tree=1, predictor='auto', random_state=5, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [33]:
```
xgbr_pred = xgbr.predict(x_test)
xgbr_pred
```

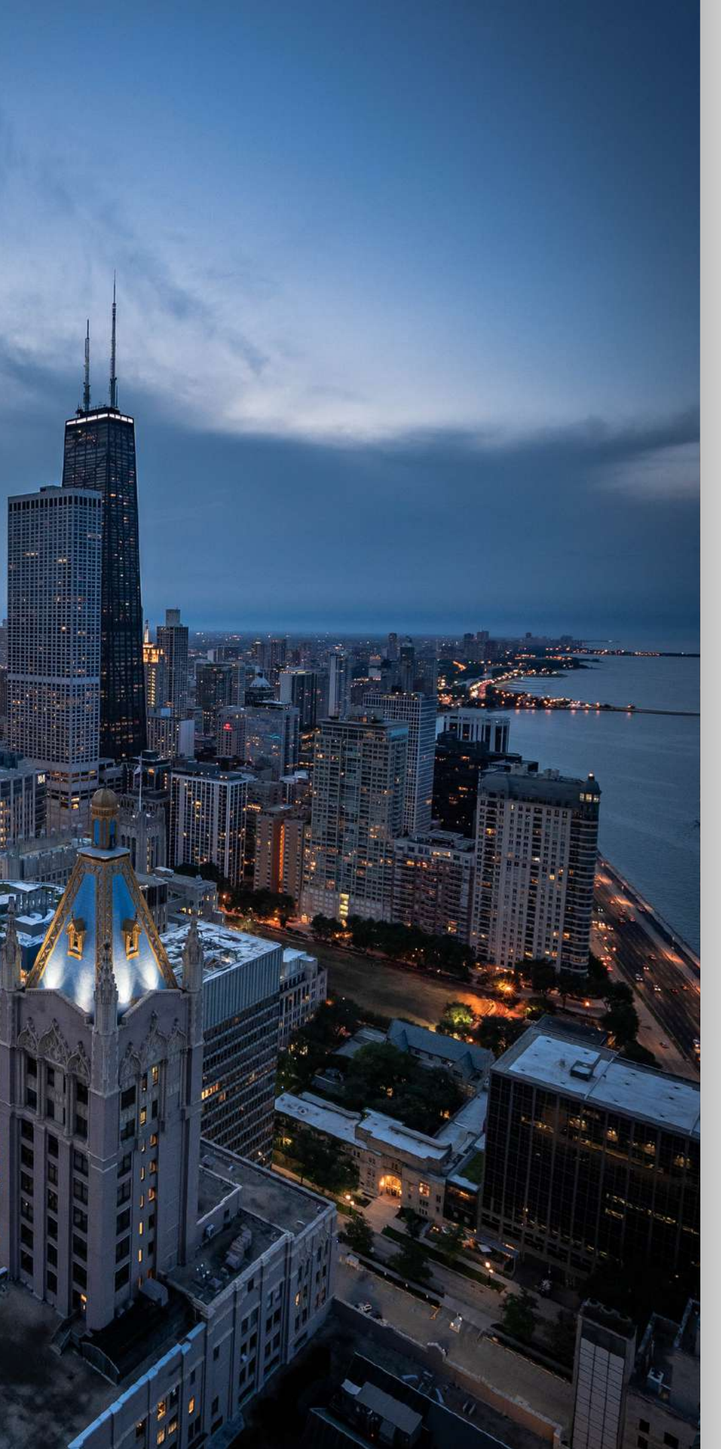
```
array([ 347175.2 ,  841044.7 ,  763898.1 ,  557451.94,  529822.94,
        523011.53,  383690.2 ,  259543.6 ,  417988.5 ,  465279.44,
        345866.12,  222198.4 ,  511211.97,  604636.25,  457043.34,
        241670.67,  280530.66,  523008.38,  224311.05,  639771.56,
        501068.56,  443498.5 ,  679442.44,  313645.97,  373285.72,
        250271.56,  208826.19,  520044.06,  229484.27,  225287.19,
        335588.75,  490471.25,  307754.78,  462636.47,  371285.2 ,
        276135.16,  378472.  ,  506644.2 ,  580201.6 ,  502949.66,
        384022.7 ,  548687.75,  409503.03,  621156.2 ,  526815.06,
        344586.38,  511061.2 ,  717528.  ,  171702.45,  275633.06,
        543881.5 ,  792367.5 ,  709584.25,  268765.84,  564921.3 ,
        618129.1 ,   72440.73,  376484.38,  161141.7 ,  396973.7 ,
        636633.44,  536055.9 ,  340899.66,  513287.22,  234924.3 ,
        744866.  ,  453995.53,  470133.62,  418145.7 ,  461990.62,
        666524.06,  267068.66,  389654.34,  195592.83,  576656.06,
        349116.53,  479090.44,  281390.38,  923861.06,  761387.7 ,
        372102.3 ,  417486.34,  421514.6 ,  260030.03,  714841.1 ,
        362044.75,  556972.3 ,  451453.16,  221582.19,  886612.06,
        460065.16,  422765.22,  272924.44,  499080.12,  349147.38,
```

In [34]:
```
xgbr.score(x,y)
```

Out[34]:
```
0.9174690322254982
```

In [35]:
```
print('MAE:', mean_absolute_error(y_test, xgbr_pred))
print('MSE:', mean_squared_error(y_test, xgbr_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, xgbr_pred)))
```

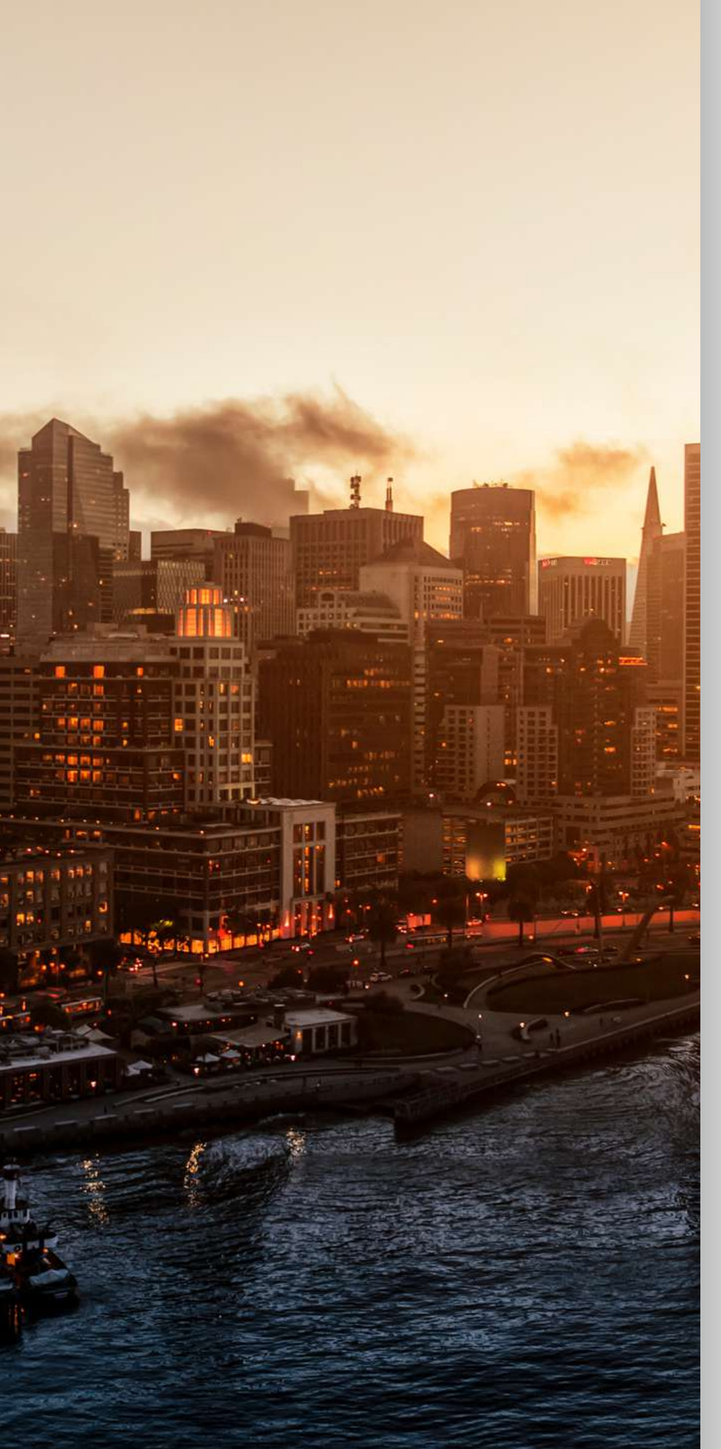
```
MAE: 86701.53270001253
MSE: 15325079752.977169
RMSE: 123794.50615022126
```



COMPARISON OF MODELS

	Model	MAE	MSE	RMSE	\
0	Linear Regression	112531.249062	2.152912e+10	146728.045344	
1	Random Forest	80970.533292	1.375280e+10	117272.351273	
2	XGBoost	86701.532700	1.532508e+10	123794.506150	

	R-squared Score
0	0.437150
1	0.891032
2	0.917469



CONCLUSION

- Aimed to estimate house prices using XGBoost, Random Forest, and Linear Regression.
- Linear Regression had notable error despite a reasonable R-squared score (approx. 0.44).
- Random Forest outperformed Linear Regression with lower errors and an impressive R-squared score (approx. 0.89).
- XGBoost emerged as the best model with an outstanding R-squared score (over 0.92) and balanced accuracy.
- Highlights the importance of advanced ensemble learning, like XGBoost, for accurate and robust predictions in complex tasks such as house price estimation.


THANK YOU