Predicting house prices in King County

1 Introduction

The price of an apartment is affected by several factors and the purchase decision is rarely easy. In this project, we try to facilitate this process and aim to predict the house prices for houses with different features in King County.

We aim to make predictions about the price based on some simple parameters. Our model should learn from the data and be able to predict the price of the house with a new combination of features. The predictions could be done using different models such as linear regression and polynomial regression.

The structure of this report is as follows: the machine learning problem is explained in more detail in the problem formulation section. In the methods section, the feature selection and used machine learning models are described. More detailed information on data processing is also provided in the methods section. In the result section, the results are compared and there is a summary of the results and the report in the conclusion section.

2 Problem formulation

The data points are houses in King County (USA). There are 16 different features. There is a lot of basic information such as the number of bedrooms, floors, and bathrooms. 0.5 bathrooms means that there is a toilet in the room without a shower. Building year, possible last renovation, and zip code are also included. There is also information about square footage of land and interior living space. Interior living space is divided into above square footage and basement square footage. Above square footage means the square footage above ground level and basement square footage the opposite. Waterfront is described with a dummy variable for whether there is a water view or not. Other features of the apartment are described on different scales. The view is described with an index from 1 to 4 of how good the view is. Condition is an index from 1 to 5 and grade of the apartment from 1 to 13. Grade 1-3 has poor construction and design, 7 is average and from 11 to 13 is high-level quality. There is also included the average square footage of interior and land of the 15 nearest neighbours.

2.1 Summary of the problem

Label: the price of the house. **Features**: number of bedrooms, number of bathrooms, square footage of the apartments interior living space, square footage of the land space, number of floors, whether there is a water view or not, view(0-4), condition(1-5), grade(1-13), the square footage of interior living space that is above ground level, the square footage of interior living space that is below ground level, building year, year of the house's last renovation, zip code, the square footage of interior living space that for the nearest 15 neighbours and the square footage of land of the nearest 15 neighbours. The data types are shown in figure 1.

```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
```

Figure 1. Datatypes of the features.

3 Methods

3.1 Dataset

For the project, I found the data from the Kaggle [1]. The dataset includes data for 21 614 apartments in different locations in King County, so we have enough data to solve this problem with machine learning methods.

3.2 Feature selection

We use price as a label and 21 other columns could be used as a feature. To get an idea of the correlation of different properties and to find out which features have high correlations with the price, we are using a correlation matrix that shows correlations between different variables. The correlations are shown in figure 2.

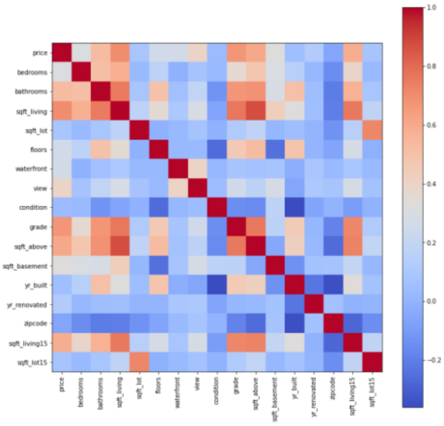


Figure 2. Correlation heatmap.

Based on the correlation matrix, the price has a high positive correlation with number of bathrooms, square feet of living, grade, the square footage of interior living space that is above ground level and, the square footage of interior living space for the nearest 15 neighbour. Price has a low correlation with the number of bedrooms, floors, waterfront, and sqft basement. We take a closer look at correlations between price and square foot living, price and grade and price and number of bedrooms in figure 3. Based on figure 3, we make the same conclusion about the correlations as from the correlation matrix.

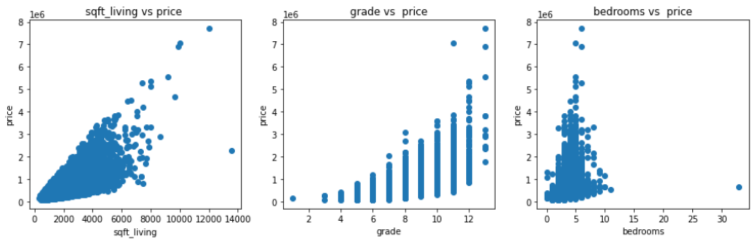


Figure 3. Correlations between different features with price.

Some features are dropped because they are not relevant to the problem. The id, date of sale, latitude, and longitude was removed from the dataset because we do not want to study their effect on price. If we wanted to include the date, we should have data from the longer term. We also get a good understanding of the apartment's location from the postal code and thus we do not need the exact location of the apartment. There were no missing features or labels in the dataset. All the features were scaled between 0 and 1 because it makes it easier to compare different features. After comparing the correlations in the correlation matrix and deleting the extra features we are left with 16 features.

3.3. Construction of training and validation sets

Datapoints were split into training (80%), validation (10%), and test (10%) sets. The validation set is a separate section of the dataset that is used to evaluate the model trained on the training set to get a little sense of how the model works. In this case for example we can compare the success of the training process with different training sizes with the validation set. We can evaluate the final model performance of the chosen model with the test set. We want to keep the training set as big as possible because the more samples we have in the training set the better opportunity the algorithm has to understand the dependencies of the features.

3.4 Linear regression model

The first model I used was the linear regression model. We try first a linear model because if thinking with common sense, there could be a relatively strong correlation between these different features and price.

In the linear regression model, we assume that there is a linear relationship between continuous variable y and one or more independent variables X. The idea is to find the best fit linear line and coefficients so that the error is as small as possible. The model takes the form:

$$h(x) = w_0 + w_1 x_1 + \cdots + w_n x_n$$

where y is the predicted label, x's are the values of different labels, w_0 is a constant term and w_n's are weights. We use linear regression to search optimal weights w_0 and w_n from a linear hypothesis space. As a loss function, we use mean squared error. We try to minimize the error by trying different weights w_n. Mean squared error is a convenient way to determine how "good" a model is. Also, according to the literature, it is smart to use mean squared error with linear regression [2]. Mean squared error is defined as:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h_w(x^i)\right)^2$$

3.5 Polynomial regression

Polynomial regression is the next model to be used. The problem may be too complex for the linear model so we could assume that polynomial regression performs better. [4]

The idea is to reuse the algorithm for linear regression. Polynomial regression is a form of linear regression where due to the non-linear relationship, we add polynomial terms to the model to convert it to polynomial regression. [5] We fit the data to polynomial regression models with different degrees. The model takes the form:

$$h(x) = w_0 + w_1 {x_1}^2 + \cdots + w_n {x_n}^n$$

Mean squared error is used also with this model because it gives a good picture of the performance of the model.

4 Results

Errors and inaccuracies of linear regression are shown in figure 4.

```
The training error: 0.0007486762075210716
Accuracy: 0.6515992474941443
The validation error: 0.0009771946533612364
Validation accuracy: 0.657430305749561
```

Figure 4. The result from linear regression.

From the results, we can observe that we can analyse and predict housing market prices relatively well with linear regression. However, we have to admit that there is room for improvement in accuracy.

The results from polynomial regression are shown in figure 5.

	poly degree	linear_train_errors	linear_val_errors
0	1	0.000749	0.000978
1	2	0.000541	0.000642
2	3	0.000413	0.001514
3	4	0.000281	607.860308

Figure 5. The results from polynomial regression.

From the polynomial regression results, we notice that the model starts dramatically to overfit after when poly degree is 3 but there is a little overfitting already when poly degree is 3.

Errors and inaccuracies of polynomial regression with poly degree 2 are shown in figure 6.

```
The training error: 0.000541190799430368
training accuracy: 0.6515992474941443
The validation error: 0.0006423009949871493
Validation accuracy: 0.657430305749561
```

Figure 6. Results from polynomial regression with polynomial 2.

5 Conclusion

If thinking about what we want from the model, there are a couple of things we value. The model should maintain its performance when applied to new data. The model should also be parsimonious and as simple as possible. [6] We have now tried two models with different properties. Accuracy from polynomial regression was better compared to linear regression. Also, the mean squared errors are smaller in polynomial regression. Thus, polynomial regression with polynomial 2 was the model that performed the best and that is the model we choose to use.

Many things could be developed further in this project. It might be that very high house prices distort the operation of the model and we get some weird results. Because of that, the model could only use apartments of a certain price. Also, it would be interesting to study the effect of distance to schools, kindergartens, and shops on house prices.

6 Appendices

The code can be found from Git:

https://github.com/Valdde/Machine-Learning.git

References

[1] https://www.kaggle.com/harlfoxem/housesalesprediction

[2] https://vitalflux.com/mean-square-error-r-squared-which-one-to-use/

[3] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

[4] https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/

[5] https://www.slideshare.net/PawanShivhare1/predicting-king-county-house-prices

[6] https://jofalu.github.io/

In [19]:
```python
import numpy as np #numerical computations
import pandas as pd #data manipulation and analysis
import matplotlib.pyplot as plt #plotting data
from sklearn.linear_model import LinearRegression, HuberRegressor #tools for linear regression
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.model_selection import train_test_split
from mpl_toolkits import mplot3d #plotting
from sklearn.preprocessing import StandardScaler #scaling
from sklearn.preprocessing import PolynomialFeatures
%matplotlib inline #3D plotting
```

```
UsageError: unrecognized arguments: #3D plotting
```

In [20]:
```python
#getting and preprocessing the data

rawdata = pd.read_csv("kc_house_data.csv")
```

In [21]:
```python
rawdata.dtypes
```

Out[21]:
```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [22]:
```python
#removing the columns 'id', 'date', 'lat', 'long'
data = rawdata.drop(['id', 'date', 'lat', 'long'], axis=1) # axis=1: dropping info from columns
data.columns
```

Out[22]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'sqft_living15',
       'sqft_lot15'],
      dtype='object')
```

In [23]:

```
#Now it is the time to select specific properties as features and labels:

X = data[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above'
y = data['price']

#dividing all the prices with the max price to get smaller values
y = y/y.max()

#NumPy representations of the features and labels
y = y.to_numpy()
X = X.to_numpy()
```

In [24]:

```
#an overview of the data

data.describe()
```

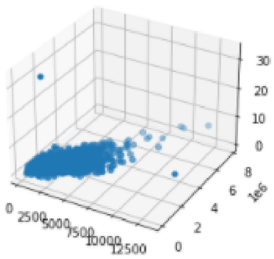
Out[24]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_a
count	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.00
mean	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.39
std	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.09
min	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.00
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.00
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.00
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.00
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.00

In [25]:

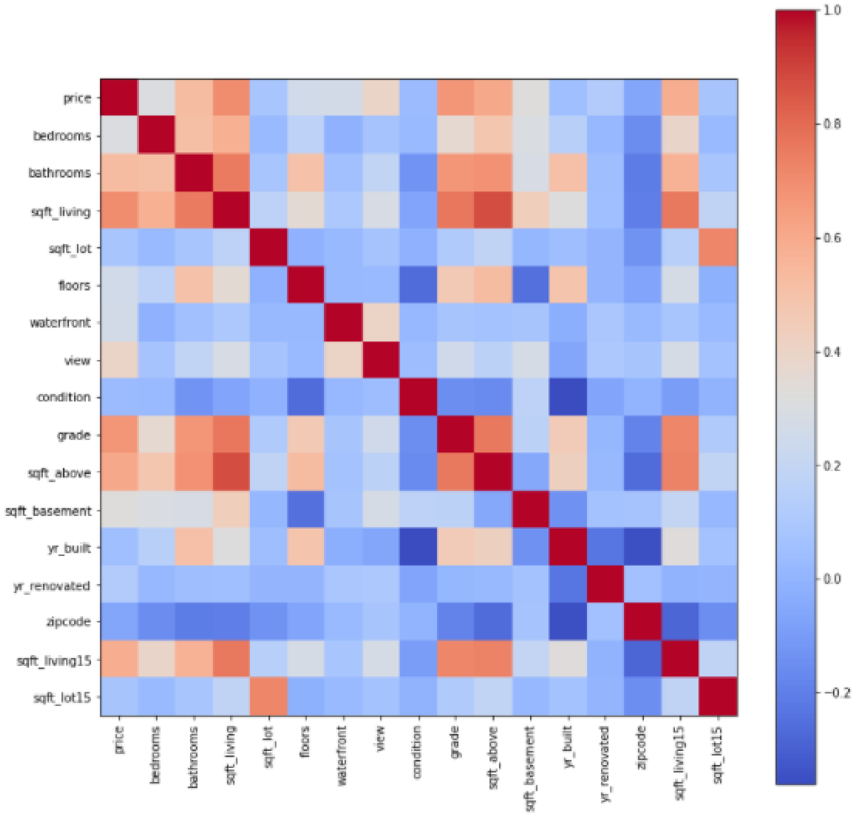
```
#examine the correaltion of different properties with price

ax = plt.axes(projection='3d')

# Data for a three-dimensional line
zline = data['bedrooms']
xline = data['sqft_living']
yline = data['price']
ax.scatter3D(xline, yline, zline, 'gray')

print(data['price'])
```

```
0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           ...
21608    360000.0
21609    400000.0
21610    402101.0
21611    400000.0
21612    325000.0
Name: price, Length: 21613, dtype: float64
```



In [26]:
```python
#Building correalation heatmap to examine the correaltion of different properties with price

plt.imshow(data.corr(), cmap='coolwarm', interpolation='none')
plt.colorbar()
plt.xticks(range(len(data.columns)), data.columns, rotation=90)
plt.yticks(range(len(data.columns)), data.columns)
plt.gcf().set_size_inches(12,12)

labels = data.corr().values
for y_ in range(labels.shape[0]):
    for x_ in range(labels.shape[1]):
        plt.text(x_, y_, "{:.2f}".format(labels(x_,y_)), ha='center', va='center', color='white')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8240/335871422.py in <module>
     10 for y_ in range(labels.shape[0]):
     11     for x_ in range(labels.shape[1]):
---> 12         plt.text(x_, y_, "{:.2f}".format(labels(x_,y_)), ha='center', va='center', color='white')

TypeError: 'numpy.ndarray' object is not callable
```

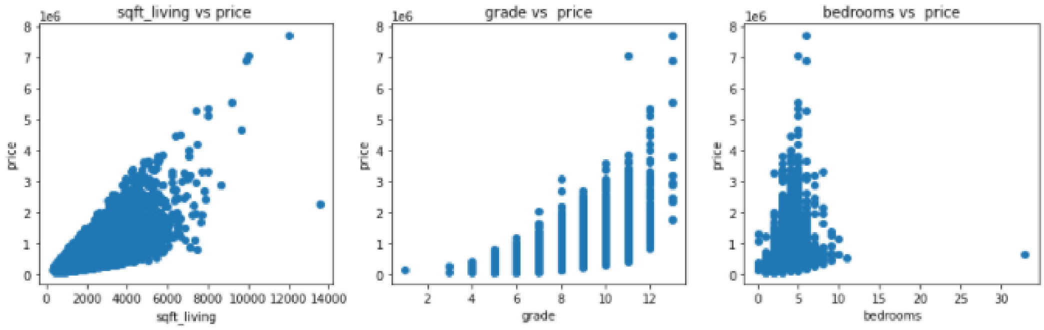


In [27]:

```
#We take a closer look to correlations between price and square foot living
#We also take look to correlation between price and number of bedrooms

# Visualize data
fig, axes = plt.subplots(1, 3, figsize=(15,4))
axes[0].scatter(data['sqft_living'],data['price']);
axes[0].set_xlabel("sqft_living")
axes[0].set_ylabel("price")
axes[0].set_title("sqft_living vs price ")

axes[1].scatter(data['grade'],data['price']);
axes[1].set_xlabel("grade")
axes[1].set_ylabel("price")
axes[1].set_title("grade vs  price")

axes[2].scatter(data['bedrooms'],data['price']);
axes[2].set_xlabel("bedrooms")
axes[2].set_ylabel("price")
axes[2].set_title("bedrooms vs  price")
```

Out[27]:

```
Text(0.5, 1.0, 'bedrooms vs  price')
```



In [28]:

```
#scaling the X-values
scaler = StandardScaler().fit(X)
X_scaled = scaler.transform(X)
```

In [29]:
```
#testing how model performs wih only one feature
#Linear regression with only price and sqft_living
X_ = data[['sqft_living']]
y_ = data['price']

X_ = X_.to_numpy()
y_ = y_.to_numpy()

lin_regr = LinearRegression()
lin_regr.fit(X_, y_)

y_pred_train = lin_regr.predict(X_)
tr_error = mean_squared_error(y_, y_pred_train)
acc_train = lin_regr.score(X_, y_)

print(y_pred_train[:10])
print(y_[:10])

#y_pred_val = lin_regr.predict(X_val)
#val_error = mean_squared_error(y_val, y_pred_val)
#acc_val = lin_regr.score(X_val, y_val)

print("The training error:", tr_error)
print("Accuracy:", acc_train)
print("\nw1 = ", lin_regr.coef_)
print("\nw0 = ", lin_regr.intercept_)

#print("The validation error:", val_error)
#print("Validation accuracy:", acc_val)
```

```
[ 287555.06702452  677621.82640197  172499.40418656  506441.44998452
  427866.85097324 1477398.99490969  437688.67584965  253880.23887682
  455929.20776298  486797.8002317 ]
[ 221900.  538000.  180000.  604000.  510000. 1225000.  257500.  291850.
  229500.  323000.]
The training error: 68351286833.039825
Accuracy: 0.4928532179037931

w1 =  [280.6235679]

w0 =  -43580.743094473146
```

In [30]:
```
#splitting the datset into training and remaining set

X_train, X_rem, y_train, y_rem = train_test_split(X_scaled, y, test_size=0.2, random_state=41)
```

In [31]:
```
#splitting the remaining dataset into validation set and test set

X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.1, random_state=41)
```

In [32]:
```
#now we have three datasets

#X_train and y_train
#X_val and y_val
#X_test and y_test
```

In [33]:
```
#first we use linear regression model

#now we fit linear regression model and predict label values based on features
#we also calculate training error

lin_regr = LinearRegression()
lin_regr.fit(X_train, y_train)

y_pred_train = lin_regr.predict(X_train)
tr_error = mean_squared_error(y_train, y_pred_train)
acc_train = lin_regr.score(X_train, y_train)

y_pred_val = lin_regr.predict(X_val)
val_error = mean_squared_error(y_val, y_pred_val)
acc_val = lin_regr.score(X_val, y_val)

print("The training error:", tr_error)
print("Accuracy:", acc_train)
#print("\nw1 = ", lin_regr.coef_)
#print("\nw0 = ", lin_regr.intercept_)

print("The validation error:", val_error)
print("Validation accuracy:", acc_val)
```

```
The training error: 0.0007486762075210716
Accuracy: 0.6515992474941443
The validation error: 0.0009771946533612364
Validation accuracy: 0.657430305749561
```

In [34]:

```python
#next we try polynomial regression
#this is done wih help from assigment 4.2

#define a list of values for polynomial degreees
degrees = [1,2,3,4]

#making lists where we can store the errors
tr_errors = []
val_errors = []

for i,degree in enumerate(degrees): #looping with different polynomial degrees

    #print("We are using polynomial degree", degree)

    lin_regr = LinearRegression(fit_intercept=False) #generating linearregression model
    poly = PolynomialFeatures(degree=degree) #generating polynomial features
    X_train_poly = poly.fit_transform(X_train) #fit and transform features
    lin_regr.fit(X_train_poly, y_train) #applying linear regression to new features

    y_pred_train = lin_regr.predict(X_train_poly) #predict values for training data using linear model
    tr_error = mean_squared_error(y_train, y_pred_train)

    X_val_poly = poly.fit_transform(X_val) #fit and transform the raw features for validation data
    y_pred_val = lin_regr.predict(X_val_poly)
    val_error = mean_squared_error(y_val, y_pred_val)

    #print("\nWegihts: \n",lin_regr.coef_)

    tr_errors.append(tr_error)
    val_errors.append(val_error)

    print("The training error:", tr_error)
    print("training accuracy:", acc_train)
    print("The validation error:", val_error)
    print("Validation accuracy:", acc_val)
```

```
The training error: 0.000748704087067833
training accuracy: 0.651599247494l443
The validation error: 0.0009783992372056987
Validation accuracy: 0.657430305749561
The training error: 0.000541190799430368
training accuracy: 0.6515992474941443
The validation error: 0.0006423009949871493
Validation accuracy: 0.657430305749561
The training error: 0.00041307201301284997
training accuracy: 0.6515992474941443
The validation error: 0.001514012304052485
Validation accuracy: 0.657430305749561
The training error: 0.00028095810734668356
training accuracy: 0.6515992474941443
The validation error: 607.8603083610866
Validation accuracy: 0.657430305749561
```

In [35]:
```
y_max = data['price'].max()

print("y max:", y_max)
print(y_pred_train[:10]*y_max)
print("Real values:", y_train[:10]*y_max)
```

```
y max: 7700000.0
[1111505.754649    279121.2696167   476200.14812765  187027.67112664
  272550.29333438  359446.0714659   327331.50890443  625807.233341
  984044.6110823   701826.4012787 ]
Real values: [1100000.  372000.  535000.  176000.  265000.  280000.  630000.  550000.
  957000.  805000.]
```

In [36]:
```
# create a table to compare training and validation errors
errors = {"poly degree":degrees,
          "linear_train_errors":tr_errors,
          "linear_val_errors":val_errors,
         }
pd.DataFrame({ key:pd.Series(value) for key, value in errors.items()})
```

Out[36]:

	poly degree	linear_train_errors	linear_val_errors
0	1	0.000749	0.000978
1	2	0.000541	0.000842
2	3	0.000413	0.001514
3	4	0.000281	607.860308