Introduction

In this presentation, we are going to look at regression modelling on house sales in NorthWestern county so as to provide advice to homeowners and the real estate agency on some varibles that have a strong relationship with housing prices.

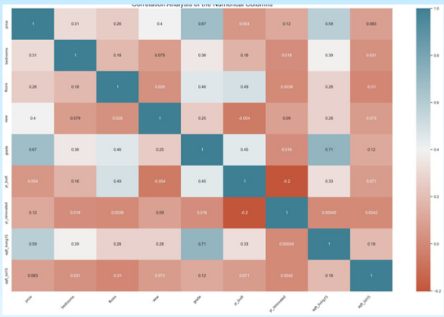
Objectives
You will be able to:
*Perform a full linear regression analysis with iterative model development
*Perform and prepare a train-test split data for modelling.
*Compare training and testing errors to determine if model is over or underfitting
*Apply an inferential lens to interpret relationships between variables identified by the model

We load our data to have a better understanding on it.

	id	date	price	bedrooms	bathroom	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0.0	1955	0.0	98178	47.5112	-122.257	1340	5650
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	1951	1991.0	98125	47.7210	-122.319	1690	7639
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	1933	NaN	98028	47.7379	-122.233	2720	8062
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	1965	0.0	98136	47.5208	-122.393	1360	5000
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	1987	0.0	98074	47.6168	-122.045	1800	7503

After we have narrowing down to the columns with high correlation with prices we get.

	price	bedrooms	floors	view	grade	yr_built	yr_renovated	sqft_living15	sqft_lot15
0	221900.0	3	1.0	0.0	7	1955	0.0	1340	5650
1	538000.0	3	2.0	0.0	7	1951	1991.0	1690	7639
2	180000.0	2	1.0	0.0	6	1933	NaN	2720	8062
3	604000.0	4	1.0	0.0	7	1965	0.0	1360	5000
4	510000.0	3	1.0	0.0	8	1987	0.0	1800	7503



:price 1.000000

grade 0.668050

sqft_living15 0.585179

view 0.396067

bedrooms 0.308865

floors 0.257337

yr_renovated 0.117910

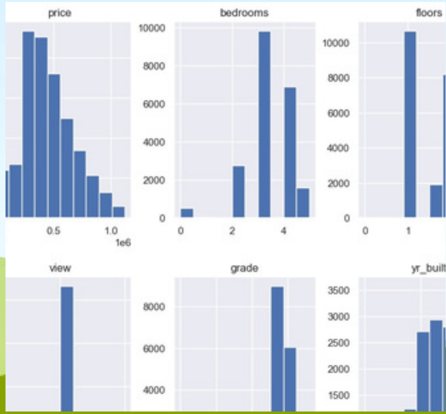
sqft_lot15 0.082778

yr_built 0.054042

aAfter removing the outlier, missing value and infite values we get this

	price	bedrooms	floors	view grade	yr_built	yr_renovated	sqf t_living15	sqft_lot15
0	221900.0	3.0	1.0	0.0 7.0	1955	0.0	1340.0	5650.0
1	538000.0	3.0	2.0	0.0 7.0	1951	0.0	1690.0	7639.0
2	180000.0	2.0	1.0	0.0 6.0	1933	0.0	2720.0	8062.0
3	604000.0	4.0	1.0	0.0 7.0	1965	0.0	1360.0	5000.0
4	510000.0	3.0	1.0	0.0 8.0	1987	0.0	1800.0	7503.0
...	...	...	...	... ...	...	...	...	...
21592	360000.0	3.0	3.0	0.0 8.0	2009	0.0	1530.0	1509.0
21593	400000.0	4.0	2.0	0.0 8.0	2014	0.0	1830.0	7200.0
21594	402101.0	2.0	2.0	0.0 7.0	2009	0.0	1020.0	2007.0
21595	400000.0	3.0	2.0	0.0 8.0	2004	0.0	1410.0	1287.0
21596	325000.0	2.0	2.0	0.0 7.0	2008	0.0	1020.0	1357.0

21574 rows × 9 columns



```
y=new_data[['price']]
x=new_data.drop(['price'], axis=1)

# Split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
x_train
```

	bedrooms	floors	view grade	yr_built	yr_renovated	sqft_living15	sqft_lot15
1385	3.0	1.0	0.0 8.0	1959	0.0	1900.0	14400.0
1580	4.0	1.0	0.0 7.0	1979	0.0	2500.0	10120.0
18086	3.0	1.0	0.0 7.0	1926	0.0	1380.0	3750.0
1582	3.0	1.0	0.0 8.0	1953	0.0	3030.0	12752.0
14503	3.0	1.0	0.0 7.0	1951	0.0	1590.0	9000.0
...	...	...	... ...	...	...	...	...
482	3.0	2.0	0.0 8.0	1973	0.0	2230.0	11553.0
10108	3.0	2.0	0.0 7.0	2005	0.0	1760.0	1916.0
232	0.0	1.0	0.0 8.0	1978	0.0	2120.0	8236.0
42	5.0	2.0	0.0 9.0	2014	0.0	3625.0	5639.0
3309	2.0	1.0	0.0 0.0	1948	0.0	1450.0	4400.0

17259 rows × 8 columns

```
y_train
```

	price
1385	381000.0
1580	420000.0
18086	505400.0
1582	0.0
14503	345000.0
...	...
482	849950.0
10108	280000.0
232	315000.0
42	861990.0
3309	296000.0

17259 rows × 1 columns

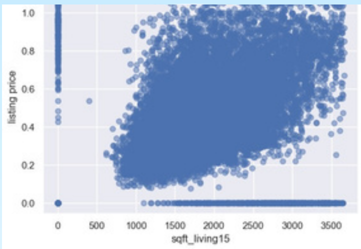
```
Data columns (total 8 columns):
 # Column Non-Null Count Dtype
--- ------ -------------- -----
          0 bedrooms 17259 non-null float64
 1 floors 17259 non-null float64
 2 view 17259 non-null float64
 3 grade 17259 non-null float64
 4 yr_built 17259 non-null int64
      5 yr_renovated 17259 non-null float64
 6 sqft_living15 17259 non-null float64
 7 sqft_lot15 17259 non-null float64
dtypes: float64(7), int64(1)
memory usage: 1.2 MB
```

Having no object dtype in our dataset we won't neccessarily need to carry out One_hot encoding or NLP to convert the non_numeric columns to numbers

```
#from sklearn.preprocessing import OneHotEncoder
#ohe=OneHotEncoder(handle_unknown='ignore',sparse=False)
#cat_columns=['grade','sqft_living15','sqft_lot15']
#ohe.fit(x_train[cat_columns])
#new_cat_columns=ohe.get_feature_names(input_features=cat_columns)
 # for training set
#x_train_ohe=pd.DataFrame(ohe.fit_transform(x_train[cat_columns]),columns=n
#x_train=pd.concat([x_train.drop(cat_columns,axis=1),x_train_ohe],axis=1)
#x_train


 # for the testing set
#x_test_ohe=pd.DataFrame(ohe.transform(x_test[cat_columns]),columns=new_cat
#x_test=pd.concat([x_test.drop(cat_columns,axis=1),x_test_ohe],axis=1)
#x_test
```

```
X_train_numeric = x_train.select_dtypes(exclude=['object'])

X_train_numeric
```

	bedrooms	floors	view grade	yr_built	yr_renovated	sqft_living15	sqft_lot15
1385	3.0	1.0	0.0 8.0	1959	0.0	1900.0	14400.0
1580	4.0	1.0	0.0 7.0	1979	0.0	2500.0	10120.0




```
most_correlated_feature=['grade']
most_correlated_feature
```

Let's build our baseline model that we will use as a comparison on the training model.

```
from sklearn.linear_model import LinearRegression

baseline_model = LinearRegression()
```

```
from sklearn.model_selection import cross_validate, ShuffleSplit

splitter = ShuffleSplit(n_splits=3, test_size=0.25, random_state=0)

baseline_scores = cross_validate(
 estimator=baseline_model,
 X=x_train[most_correlated_feature],
 y=y_train,
 return_train_score=True,
 cv=splitter
)

print("Train score: ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

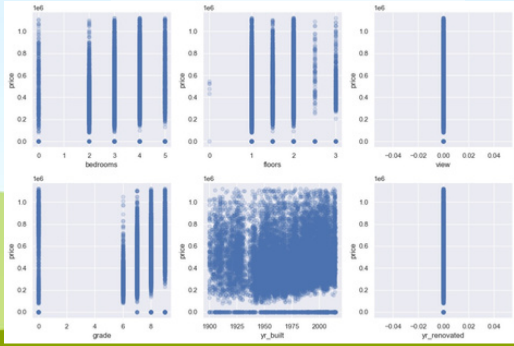
```
Train score: 0.02589269025727825
Validation score: 0.026981476449032076
```

This seems like a general weak model already having the training r-squared scores slightly lower than its validation score.

18086	3.0	1.0	0.0	7.0	1926	0.0	1380.0	3750.0
1582	3.0	1.0	0.0	8.0	1953	0.0	3030.0	12752.0
14503	3.0	1.0	0.0	7.0	1951	0.0	1590.0	9000.0
...	...	...	...	...	...	...	...	...
482	3.0	2.0	0.0	8.0	1973	0.0	2230.0	11553.0
10108	3.0	2.0	0.0	7.0	2005	0.0	1760.0	1916.0
232	0.0	1.0	0.0	8.0	1978	0.0	2120.0	8236.0
42	5.0	2.0	0.0	9.0	2014	0.0	3625.0	5639.0
3309	2.0	1.0	0.0	0.0	1948	0.0	1450.0	4400.0

17259 rows × 8 columns

```
scatterplot_data = X_train_numeric#.drop("grade", axis=1)

fig, axes = plt.subplots(ncols=3, nrows=2, figsize=(12, 8))
fig.set_tight_layout(True)
for index, col in enumerate(scatterplot_data.columns):
 ax = axes[index//3][index%3]
 ax.scatter(X_train_numeric[col], y_train, alpha=0.2)
 ax.set_xlabel(col)
 ax.set_ylabel("price")
```



```
print("Current Model")
print("Train score: ", second_model_scores["train_score"].mean())
print("Validation score:", second_model_scores["test_score"].mean())
print()
print("Train score: ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Current Model
Train score: 0.15071091477887869
Validation score: 0.14814230966779107

Train score: 0.027057957124270504
Validation score: 0.021088515908451955
```

the current model happens to perform better than our baseline model after dropping some of the columns this happens to be causing a underfitting as the validation score tend to be doing better than our current model train score.

```
import statsmodels.api as sm
```

```
sm.OLS(y_train, sm.add_constant(X_train_second_model)).fit().summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.150
Model:	OLS	**Adj. R-squared:**	0.150
Method:	Least Squares	**F-statistic:**	508.2
Date:	Sun, 09 Apr 2023	**Prob (F-statistic):**	0.00
Time:	01:52:20	**Log-Likelihood:**	-2.3610e+05
No. Observations:	17259	**AIC:**	4.722e+05
Df Residuals:	17252	**BIC:**	4.723e+05
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.056e+06	1.23e+05	16.688	0.000	1.81e+06	2.3e+06
bedrooms	2.025e+04	1802.222	11.235	0.000	1.67e+04	2.38e+04
floors	6.287e+04	3590.296	17.511	0.000	5.58e+04	6.99e+04
grade	1.364e+04	711.164	19.186	0.000	1.23e+04	1.5e+04
yr_built	-1025.4331	64.387	-15.926	0.000	-1151.638	-899.229
sqft_living15	100.9866	2.593	38.939	0.000	95.903	106.070
sqf t_lot15	-4.0619	0.449	-9.040	0.000	-4.943	-3.181

Omnibus:	985.091	**Durbin-Watson:**	2.000
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2600.794
Skew:	0.319	**Prob(JB):**	0.00
Kur tosis:	4.792	**Cond. No.**	6.06e+05

```
significant_features =['bedrooms','floors','sqft_living15','sqft_lot15']
```

```
print("Current Model")
print("Train score: ", third_model_scores["train_score"].mean())
print("Validation score:", third_model_scores["test_score"].mean())
print()
print("Second Model")
print("Train score: ", second_model_scores["train_score"].mean())
print("Validation score:", second_model_scores["test_score"].mean())
print()
print("Baseline Model")
print("Train score: ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Current Model
Train score: 0.11854003105380606
Validation score: 0.12676387853162338

Second Model
Train score: 0.15071091477887869
Validation score: 0.14814230966779107

Baseline Model
Train score: 0.027057957124270504
Validation score: 0.021088515908451955
```

removing those features led to the better scores even though they are slightly lower than those of the second model.

"RFE" stands for "recursive feature elimination", meaning that it repeatedly scores the model, finds and removes the feature with the lowest "importance", then scores

27/40

the model again. If the new score is better than the previous score, it continues removing features until the minimum is reached. "CV" stands for "cross validation" here, and we can use the same splitter we have been using to test our data so far.

```
Was the column selected?
bedrooms: True
floors: True
grade: True
yr_built: True
sqft_living15: True
sqft_lot15: True
```

Intresting, so this algorithm says that all our features are the best we could use to fit well with our target.

```
best_features=['bedrooms','floors','grade','yr_built','sqft_living15','sqft
```

```
X_train_final = x_train[best_features]
X_test_final = x_test[best_features]
```

```
final_model = LinearRegression()

# Fit the model on X_train_final and y_train
final_model.fit(X_train_final,y_train)

# Score the model on X_test_final and y_test
# (use the built-in .score method)
final_model.score(X_test_final, y_test)
```

```
=
0.11760973856941015
```

```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_test, final_model.predict(X_test_final), squared=False
```

```
214299.69809371408
```

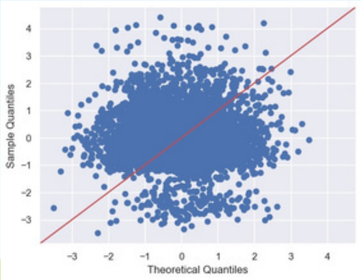
Below, we display the coefficients and intercept for the final model:

Coefficients:
bedrooms: [1.94345408e+04 6.26733765e+04 1.39463304e+04
-9.62117288e+02
9.72708882e+01 -4.05965851e+00]
Intercept: [1938070.74045687]
['bedrooms', 'floors', 'grade', 'yr_built', 'sqft_living15', 'sqft_lot15

According to our model, we can say that the base price for a house in North western county is set to be about (model intercept) = 1,938,070.74 and for each additional bedroom the price changes with a margin of 1.94, for every floor the price changes with a range of 6.267. For every change in grade the price increases by 1.394, the older the building the price is decreased by -9.62 while the sqft_living adjusts the prices by 9.727 and lastly the sqft_lot changes the house price by -4.0596

Investigating Normality

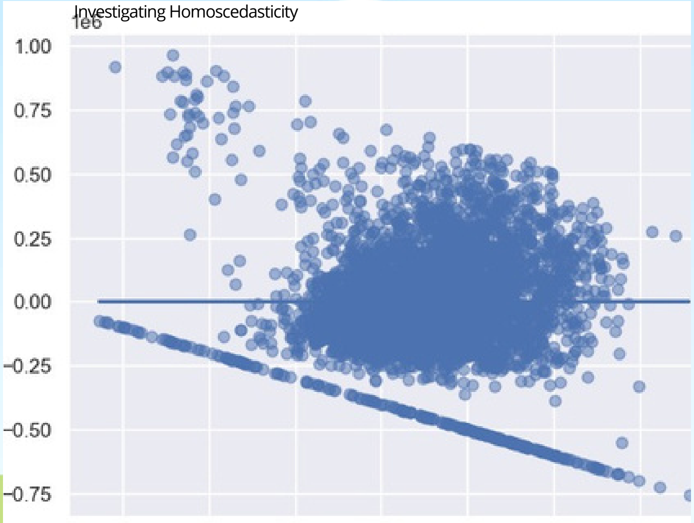
```
import scipy.stats as stats

residuals = (y_test - preds)
sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True);
```



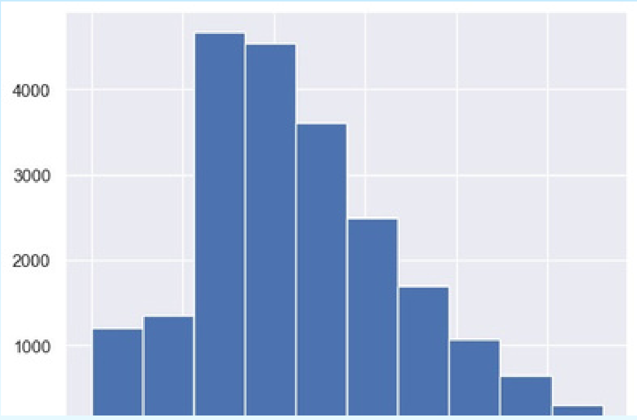
Investigating Multicollinearity (Independence Assumption)

A VIF that is 5 is too high

```
Out[96]:bedrooms 14.252220
floors 10.263416
grade 10.288106
yr_built 35.527799
sqft_living15 10.256843
sqft_lot15 4.465978
Name: Variance Inflation Factor, dtype: float64
```



From our graph we can say that our dependent variability is not equal across the values of the independent variable. Thus we are violating strict defination of homoscedasticity.



Our confidence in these coefficients should not be too high, since we are violating or close to violating more than one of the assumptions of linear regression. This really only should be used for predictive purposes.t