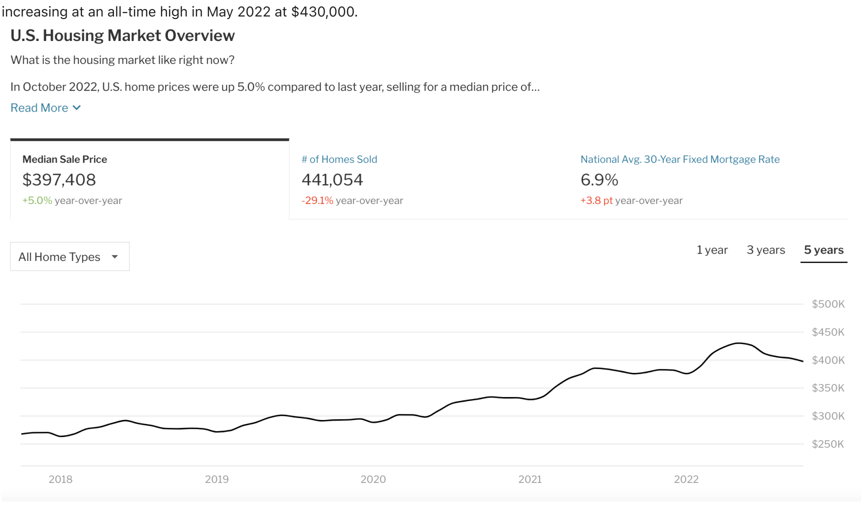
Regression model: Housing Price

1. Introduction

1.1 Domain specific area

Housing price increases every year, and with an increasing demand for houses and lack of space, it is easy to see why it is getting more expensive especially if people are planning to live in the city. Even though some people might choose to live in the suburb or rural areas, prices are still subject to demand and location. According to Redfin, housing prices in the last five years have been slowly increasing at an all-time high in May 2022 at $430,000.



Hence, there is a need for a system to predict house prices in the future. The regression model used will take in a set of data containing multiple factors to predict an estimated price range for the specific house. House price prediction can help developers determine the selling price of a house and can help future homeowners to arrange the right time to purchase a home.

1.2 Dataset

The dataset will be of residential homes in Ames, Iowa. It is split into 2 files, training and testing, in the CSV format allowing it to be easily used with Pandas and it consists of many information such as sales price, street, condition etc. There are more than enough features that the regression model can use to predict the values.

The training file will be used to train the regression model and the testing file will be used to predict the values. This allows me to see whether the model is effective on unseen data.

1.3 Objectives

The main objective of this project:

- To predict the sales price of each house

For each id in the set, i will be using a regression model to predict the value of the SalePrice variable

There are 4 main steps i have to follow through:

1. cleaning the data to make it usable
2. visualizing the data
3. training the model
4. Making predictions using the model

This regression model will allow us to predict the house prices in the area which is good for developers who want to estimate how much houses will be sold for and homeowners who want to purchase a new house.

1.4 Disclaimer

As this project is used for personal and educational purpose, any conclusions from this project will be based on assumptions after looking at the predicted values and should not be used as a professional analytical judgement.

2. Implementation

Importing necessary Data and libraries

In [1]:
```python
#importing necessary libriaries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
```

Loading the training data

In [2]:
```python
df_train = pd.read_csv('data/train.csv')
df_train
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fe
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	N
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	N
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	Mr
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	Gc
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N

1460 rows × 81 columns

2.1 Cleaning the training dataset

Checking the percentage of missing values.

In [3]:
```python
missingSum = df_train.isna().sum().sort_values(ascending=False)
percentage = (df_train.isna().sum()/df_train.isna().count() * 100).sort_values(ascending=False)
missingData = pd.concat([missingSum, percentage], axis=1, keys=['missingSum', 'percentage'])
missingData.head(10)
```

Out[3]:

	missingSum	percentage
PoolQC	1453	99.520548
MiscFeature	1406	96.301370
Alley	1369	93.767123
Fence	1179	80.753425
FireplaceQu	690	47.260274
LotFrontage	259	17.739726
GarageYrBlt	81	5.547945
GarageCond	81	5.547945
GarageType	81	5.547945
GarageFinish	81	5.547945

Remove all columns with more than 10% of missing values as there is too many missing data

In [4]:
```python
#calculates columns with more than 10% of values missingColumn
missingColumn = df_train.isna().sum()
missingColumn = missingColumn[missingColumn > 0.1*len(df_train.index)]

#removes column
df_train2 = df_train.drop(columns=missingColumn.index)
df_train2
```

Out[4]:

	Id	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3Ss
0	1	60	RL	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1	2	20	RL	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	...	0	
2	3	60	RL	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	0	
3	4	70	RL	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	...	272	
4	5	60	RL	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	7917	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1456	1457	20	RL	13175	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1457	1458	70	RL	9042	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1458	1459	20	RL	9717	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	112	
1459	1460	20	RL	9937	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	

1460 rows × 75 columns

Look at the remaining data info

In [5]:
```python
df_train2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 75 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotArea        1460 non-null   int64
 4   Street         1460 non-null   object
 5   LotShape       1460 non-null   object
 6   LandContour    1460 non-null   object
 7   Utilities      1460 non-null   object
 8   LotConfig      1460 non-null   object
 9   LandSlope      1460 non-null   object
 10  Neighborhood   1460 non-null   object
 11  Condition1     1460 non-null   object
 12  Condition2     1460 non-null   object
 13  BldgType       1460 non-null   object
 14  HouseStyle     1460 non-null   object
 15  OverallQual    1460 non-null   int64
 16  OverallCond    1460 non-null   int64
 17  YearBuilt      1460 non-null   int64
 18  YearRemodAdd   1460 non-null   int64
 19  RoofStyle      1460 non-null   object
 20  RoofMatl       1460 non-null   object
 21  Exterior1st    1460 non-null   object
 22  Exterior2nd    1460 non-null   object
 23  MasVnrType     1452 non-null   object
 24  MasVnrArea     1452 non-null   float64
 25  ExterQual      1460 non-null   object
 26  ExterCond      1460 non-null   object
 27  Foundation     1460 non-null   object
 28  BsmtQual       1423 non-null   object
 29  BsmtCond       1423 non-null   object
 30  BsmtExposure   1422 non-null   object
 31  BsmtFinType1   1423 non-null   object
 32  BsmtFinSF1     1460 non-null   int64
 33  BsmtFinType2   1422 non-null   object
 34  BsmtFinSF2     1460 non-null   int64
 35  BsmtUnfSF      1460 non-null   int64
 36  TotalBsmtSF    1460 non-null   int64
 37  Heating        1460 non-null   object
 38  HeatingQC      1460 non-null   object
 39  CentralAir     1460 non-null   object
 40  Electrical     1459 non-null   object
 41  1stFlrSF       1460 non-null   int64
 42  2ndFlrSF       1460 non-null   int64
 43  LowQualFinSF   1460 non-null   int64
 44  GrLivArea      1460 non-null   int64
 45  BsmtFullBath   1460 non-null   int64
 46  BsmtHalfBath   1460 non-null   int64
 47  FullBath       1460 non-null   int64
 48  HalfBath       1460 non-null   int64
 49  BedroomAbvGr   1460 non-null   int64
 50  KitchenAbvGr   1460 non-null   int64
 51  KitchenQual    1460 non-null   object
 52  TotRmsAbvGrd   1460 non-null   int64
 53  Functional     1460 non-null   object
 54  Fireplaces     1460 non-null   int64
 55  GarageType     1379 non-null   object
 56  GarageYrBlt    1379 non-null   float64
 57  GarageFinish   1379 non-null   object
 58  GarageCars     1460 non-null   int64
 59  GarageArea     1460 non-null   int64
 60  GarageQual     1379 non-null   object
 61  GarageCond     1379 non-null   object
 62  PavedDrive     1460 non-null   object
 63  WoodDeckSF     1460 non-null   int64
 64  OpenPorchSF    1460 non-null   int64
 65  EnclosedPorch  1460 non-null   int64
 66  3SsnPorch      1460 non-null   int64
 67  ScreenPorch    1460 non-null   int64
 68  PoolArea       1460 non-null   int64
 69  MiscVal        1460 non-null   int64
 70  MoSold         1460 non-null   int64
 71  YrSold         1460 non-null   int64
 72  SaleType       1460 non-null   object
 73  SaleCondition  1460 non-null   object
 74  SalePrice      1460 non-null   int64
dtypes: float64(2), int64(35), object(38)
memory usage: 855.6+ KB
```

Looking at columns only with int or float

In [6]:
```python
intOrFloat = [x for x in df_train2.columns if df_train2.dtypes[x] != 'object'] #list of columns with int or flo
intOrFloat
```

Out[6]:
```
['Id',
 'MSSubClass',
 'LotArea',
 'OverallQual',
 'OverallCond',
 'YearBuilt',
 'YearRemodAdd',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'GrLivArea',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath',
 'BedroomAbvGr',
 'KitchenAbvGr',
 'TotRmsAbvGrd',
 'Fireplaces',
 'GarageYrBlt',
 'GarageCars',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'PoolArea',
 'MiscVal',
 'MoSold',
 'YrSold',
 'SalePrice']
```

A new dataframe built using only integer or float values for the regression model.

In [7]:
```python
df_train3 = df_train2[intOrFloat]
df_train3
```

Out[7]:

	Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	Wo
0	1	60	8450	7	5	2003	2003	196.0	706	0	...	
1	2	20	9600	6	8	1976	1976	0.0	978	0	...	
2	3	60	11250	7	5	2001	2002	162.0	486	0	...	
3	4	70	9550	7	5	1915	1970	0.0	216	0	...	
4	5	60	14260	8	5	2000	2000	350.0	655	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	7917	6	5	1999	2000	0.0	0	0	...	
1456	1457	20	13175	6	6	1978	1988	119.0	790	163	...	
1457	1458	70	9042	7	9	1941	2006	0.0	275	0	...	
1458	1459	20	9717	5	6	1950	1996	0.0	49	1029	...	
1459	1460	20	9937	5	6	1965	1965	0.0	830	290	...	

1460 rows × 37 columns

there are still some missing values

In [8]:
```python
missing2 = df_train3.isnull().sum()
missing2 = missing2[missing2 > 0] #columns with missing values
missing2
```

Out[8]:
```
MasVnrArea      8
GarageYrBlt    81
dtype: int64
```

Fill missing columns with mean

In [9]:
```python
#calculates the mean in the column and fills missing value with it
def fillMissing(dataframe,column):
    mean = dataframe[column].mean()
    dataframe[column].fillna(value=mean,inplace=True)
    return dataframe

fillMissing(df_train3,'MasVnrArea')
fillMissing(df_train3,'GarageYrBlt')
```

```
/Users/ronaldgohjingwei/opt/anaconda3/lib/python3.9/site-packages/pandas/core/generic.py:6392: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return self._update_inplace(result)
```

Out[9]:

	Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	Wo
0	1	60	8450	7	5	2003	2003	196.0	706	0	...	
1	2	20	9600	6	8	1976	1976	0.0	978	0	...	
2	3	60	11250	7	5	2001	2002	162.0	486	0	...	
3	4	70	9550	7	5	1915	1970	0.0	216	0	...	
4	5	60	14260	8	5	2000	2000	350.0	655	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	7917	6	5	1999	2000	0.0	0	0	...	
1456	1457	20	13175	6	6	1978	1988	119.0	790	163	...	
1457	1458	70	9042	7	9	1941	2006	0.0	275	0	...	
1458	1459	20	9717	5	6	1950	1996	0.0	49	1029	...	
1459	1460	20	9937	5	6	1965	1965	0.0	830	290	...	

1460 rows × 37 columns

Training dataset after cleaning

In [10]:
```python
df_train3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 37 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   LotArea        1460 non-null   int64
 3   OverallQual    1460 non-null   int64
 4   OverallCond    1460 non-null   int64
 5   YearBuilt      1460 non-null   int64
 6   YearRemodAdd   1460 non-null   int64
 7   MasVnrArea     1460 non-null   float64
 8   BsmtFinSF1     1460 non-null   int64
 9   BsmtFinSF2     1460 non-null   int64
 10  BsmtUnfSF      1460 non-null   int64
 11  TotalBsmtSF    1460 non-null   int64
 12  1stFlrSF       1460 non-null   int64
 13  2ndFlrSF       1460 non-null   int64
 14  LowQualFinSF   1460 non-null   int64
 15  GrLivArea      1460 non-null   int64
 16  BsmtFullBath   1460 non-null   int64
 17  BsmtHalfBath   1460 non-null   int64
 18  FullBath       1460 non-null   int64
 19  HalfBath       1460 non-null   int64
 20  BedroomAbvGr   1460 non-null   int64
 21  KitchenAbvGr   1460 non-null   int64
 22  TotRmsAbvGrd   1460 non-null   int64
 23  Fireplaces     1460 non-null   int64
 24  GarageYrBlt    1460 non-null   float64
 25  GarageCars     1460 non-null   int64
 26  GarageArea     1460 non-null   int64
 27  WoodDeckSF     1460 non-null   int64
 28  OpenPorchSF    1460 non-null   int64
 29  EnclosedPorch  1460 non-null   int64
 30  3SsnPorch      1460 non-null   int64
 31  ScreenPorch    1460 non-null   int64
 32  PoolArea       1460 non-null   int64
 33  MiscVal        1460 non-null   int64
 34  MoSold         1460 non-null   int64
 35  YrSold         1460 non-null   int64
 36  SalePrice      1460 non-null   int64
dtypes: float64(2), int64(35)
memory usage: 422.2 KB
```

In [11]:
```python
pd.set_option('max_columns',None) # show all columns
df_train3
```

Out[11]:

	Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtU
0	1	60	8450	7	5	2003	2003	196.0	706	0	
1	2	20	9600	6	8	1976	1976	0.0	978	0	
2	3	60	11250	7	5	2001	2002	162.0	486	0	
3	4	70	9550	7	5	1915	1970	0.0	216	0	
4	5	60	14260	8	5	2000	2000	350.0	655	0	
...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	7917	6	5	1999	2000	0.0	0	0	
1456	1457	20	13175	6	6	1978	1988	119.0	790	163	
1457	1458	70	9042	7	9	1941	2006	0.0	275	0	
1458	1459	20	9717	5	6	1950	1996	0.0	49	1029	
1459	1460	20	9937	5	6	1965	1965	0.0	830	290	

1460 rows × 37 columns

In [12]:
```python
pd.reset_option('max_columns') #resets max number of columns displayed to default
```

In [ ]:

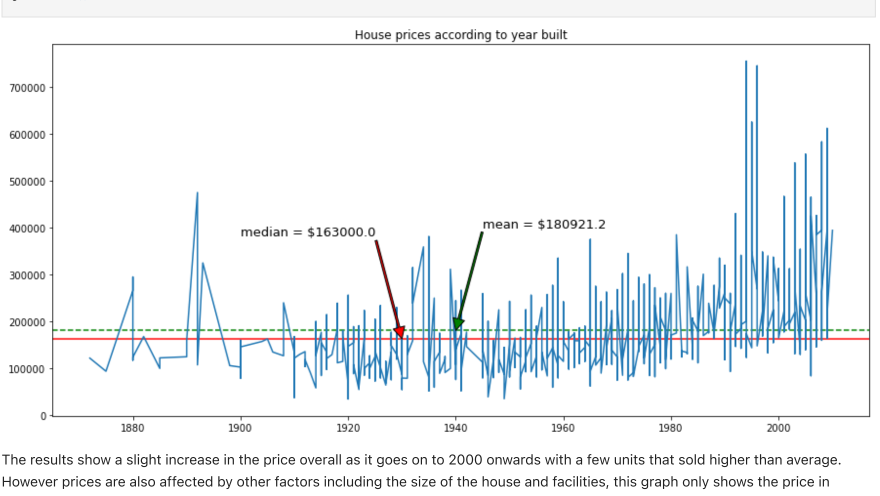
2.2 Identify key series, provide statistical summary and visualizing

As our main focus is the sales price, lets find out the the statistical value for sales price, comparing houseprice versus year the house was built in.

In [13]:
```python
#calculate mean
yearPrice = pd.concat([df_train3['YearBuilt'], df_train3['SalePrice']],
                      axis=1, keys=['YearBuilt', 'SalePrice'],sort=True)
yearPrice= yearPrice.sort_values('YearBuilt', ascending=True)
mean = round(yearPrice['SalePrice'].mean(),1)

#calculate median
median = yearPrice['SalePrice'].median()

plt.subplots(figsize=(15, 7))
plt.title('House prices according to year built')
#draws horizontal line
plt.axhline(mean,c='g',linestyle='dashed')
plt.axhline(median,c='r',linestyle='-')
plt.plot(yearPrice['YearBuilt'],yearPrice['SalePrice'])
#annotates on the graph
plt.annotate(f'mean = ${mean}',xy=(1940,mean),xytext=(1945,mean+220000),fontsize=13,
            arrowprops=dict(facecolor='g',width=2))
plt.annotate(f'median = ${median}',xy=(1930,median),xytext=(1900,median+220000),fontsize=13,
            arrowprops=dict(facecolor='r',width=2))
plt.show()
```



The results show a slight increase in the price overall as it goes on to 2000 onwards with a few units that sold higher than average. However prices are also affected by other factors including the size of the house and facilities, this graph only shows the price in relation to when the house is built and does not entirely depict the house price.

In [ ]:

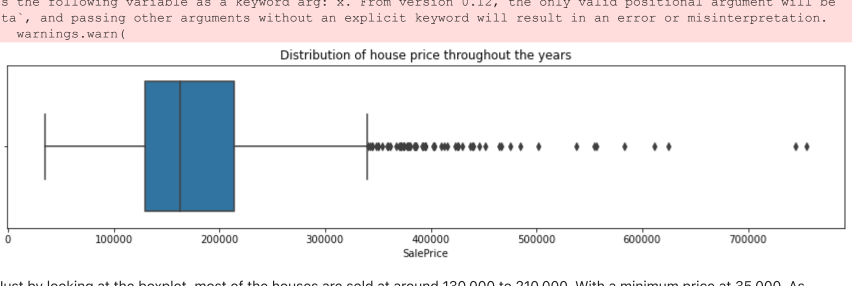
Using a box plot to show the distribution of numerical data and skewness through displaying the data quartiles.

In [14]:
```python
yearPrice['SalePrice'].describe()
```

Out[14]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [15]:
```python
plt.subplots(figsize=(15, 3))
ax = sns.boxplot( yearPrice['SalePrice'])
ax.set_title('Distribution of house price throughout the years')
plt.show()
```

```
/Users/ronaldgohjingwei/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `da
ta`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



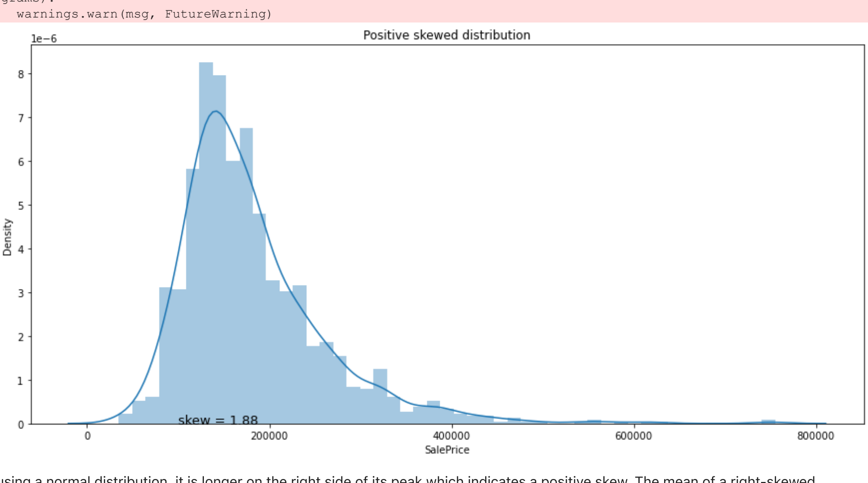
Just by looking at the boxplot, most of the houses are sold at around 130,000 to 210,000. With a minimum price at 35,000. As regards to the maximum price there are a some outliers but the highest amount sold is around 750,000

In [ ]:

Using a distibution plot to show the measure of the asymmetry of a plot.

In [16]:
```python
plt.subplots(figsize=(15, 7))
dist = sns.distplot(yearPrice['SalePrice'])
skew = yearPrice['SalePrice'].skew()
dist.set_title('Positive skewed distribution')
plt.annotate(f'skew = {round(skew,2)}',xy=(100000,0),fontsize=13)
plt.show()
```

```
/Users/ronaldgohjingwei/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
  warnings.warn(msg, FutureWarning)
```



using a normal distribution, it is longer on the right side of its peak which indicates a positive skew. The mean of a right-skewed distribution is almost always greater than its median which is proven true by comparing the values from yearPrice['SalePrice'].describe().

In [ ]:

2.3 Identifying core features using correlation

correlation is any statistical relationship, whether causal or not, between two random variables.

In [17]:
```python
correlationTable = df_train3.corr() #Compute pairwise correlation of columns
correlationTable
```

Out[17]:

	Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
Id	1.000000	0.011156	-0.033226	-0.028365	0.012609	-0.012713	-0.021998	-0.050199	-0.005024
MSSubClass	0.011156	1.000000	-0.139781	0.032628	-0.059316	0.027850	0.040581	0.022895	-0.069836
LotArea	-0.033226	-0.139781	1.000000	0.105806	-0.005636	0.014228	0.013788	0.103960	0.214103
OverallQual	-0.028365	0.032628	0.105806	1.000000	-0.091932	0.572323	0.550684	0.410238	0.239666
OverallCond	0.012609	-0.059316	-0.005636	-0.091932	1.000000	-0.375983	0.073741	-0.127788	-0.046231
YearBuilt	-0.012713	0.027850	0.014228	0.572323	-0.375983	1.000000	0.592855	0.314745	0.249503
YearRemodAdd	-0.021998	0.040581	0.013788	0.550684	0.073741	0.592855	1.000000	0.179186	0.128451
MasVnrArea	-0.050199	0.022895	0.103960	0.410238	-0.127788	0.314745	0.179186	1.000000	0.263582
BsmtFinSF1	-0.005024	-0.069836	0.214103	0.239666	-0.046231	0.249503	0.128451	0.263582	1.000000
BsmtFinSF2	-0.005968	-0.065649	0.111170	-0.059119	0.040229	-0.049107	-0.067759	-0.072302	-0.050117
BsmtUnfSF	-0.007940	-0.140759	-0.002618	0.308159	-0.136841	0.149040	0.181133	0.114184	-0.495251
TotalBsmtSF	-0.015415	-0.238518	0.260833	0.537808	-0.171098	0.391452	0.291066	0.362452	0.522396
1stFlrSF	0.010496	-0.251758	0.299475	0.476224	-0.144203	0.281986	0.240379	0.342160	0.445863
2ndFlrSF	0.005590	0.307886	0.050986	0.295493	0.028942	0.010308	0.140024	0.174019	-0.137079
LowQualFinSF	-0.044230	0.046474	0.004779	-0.030429	0.025494	-0.183784	-0.062419	-0.069068	-0.064503
GrLivArea	0.008273	0.074853	0.263116	0.593007	-0.079686	0.199010	0.287389	0.389893	0.208171
BsmtFullBath	0.002289	0.003491	0.158155	0.111098	-0.054942	0.187599	0.119470	0.085055	0.649212
BsmtHalfBath	-0.020155	-0.002333	0.048046	-0.040150	0.117821	-0.038162	-0.012337	0.026669	0.067418
FullBath	0.005587	0.131608	0.126031	0.550600	-0.194149	0.468271	0.439046	0.275730	0.058543
HalfBath	0.006784	0.177354	0.014259	0.273458	-0.060769	0.242656	0.183331	0.200802	0.004262
BedroomAbvGr	0.037719	-0.023438	0.119690	0.101676	0.012980	-0.070651	-0.040581	0.102417	-0.107355
KitchenAbvGr	0.002951	0.281721	-0.017784	-0.183882	-0.087001	-0.174800	-0.149598	-0.037364	-0.081007
TotRmsAbvGrd	0.027239	0.040380	0.190015	0.427452	-0.057583	0.095589	0.191740	0.280027	0.044316
Fireplaces	-0.019772	-0.045569	0.271364	0.396765	-0.023820	0.147716	0.112581	0.247906	0.260011
GarageYrBlt	0.000070	0.080187	-0.024812	0.518018	-0.306169	0.780555	0.618130	0.249367	0.150338
GarageCars	0.016570	-0.040110	0.154871	0.600671	-0.185758	0.537850	0.420622	0.363778	0.224054
GarageArea	0.017634	-0.098672	0.180403	0.562022	-0.151521	0.478954	0.371600	0.372567	0.296970
WoodDeckSF	-0.029643	-0.012579	0.171698	0.238923	-0.003334	0.224880	0.205726	0.159349	0.204306
OpenPorchSF	-0.000477	-0.006100	0.084774	0.308819	-0.032589	0.188686	0.226298	0.124965	0.111761
EnclosedPorch	0.002889	-0.012037	-0.018340	-0.113937	0.070356	-0.387268	-0.193919	-0.109849	-0.102303
3SsnPorch	-0.046635	-0.043825	0.020423	0.030371	0.025504	0.031355	0.045286	0.018795	0.026451
ScreenPorch	0.001330	-0.026030	0.043160	0.064886	0.054811	-0.050364	-0.038740	0.061453	0.062021
PoolArea	0.057044	0.008283	0.077672	0.065166	-0.001985	0.004950	0.005829	0.011723	0.140491
MiscVal	-0.006242	-0.007683	0.038068	-0.031406	0.068777	-0.034383	-0.010286	-0.029815	0.003571
MoSold	0.021172	-0.013585	0.001205	0.070815	-0.003511	0.012398	0.021490	-0.005940	-0.015727
YrSold	0.000712	-0.021407	-0.014261	-0.027347	0.043950	-0.013618	0.035743	-0.008184	0.014359
SalePrice	-0.021917	-0.084284	0.263843	0.790982	-0.077856	0.522897	0.507101	0.475241	0.386420

37 rows × 37 columns

In [ ]:

Creating a dataframe where only features with correlation of more than 0.5 with 'SalePrice'

In [18]:
```python
correlationSale = correlationTable[correlationTable['SalePrice']>0.5]
correlationSale
```

Out[18]:

	Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
OverallQual	-0.028365	0.032628	0.105806	1.000000	-0.091932	0.572323	0.550684	0.410238	0.239666
YearBuilt	-0.012713	0.027850	0.014228	0.572323	-0.375983	1.000000	0.592855	0.314745	0.249503
YearRemodAdd	-0.021998	0.040581	0.013788	0.550684	0.073741	0.592855	1.000000	0.179186	0.128451
TotalBsmtSF	-0.015415	-0.238518	0.260833	0.537808	-0.171098	0.391452	0.291066	0.362452	0.522396
1stFlrSF	0.010496	-0.251758	0.299475	0.476224	-0.144203	0.281986	0.240379	0.342160	0.445863
GrLivArea	0.008273	0.074853	0.263116	0.593007	-0.079686	0.199010	0.287389	0.389893	0.208171
FullBath	0.005587	0.131608	0.126031	0.550600	-0.194149	0.468271	0.439046	0.275730	0.058543
TotRmsAbvGrd	0.027239	0.040380	0.190015	0.427452	-0.057583	0.095589	0.191740	0.280027	0.044316
GarageCars	0.016570	-0.040110	0.154871	0.600671	-0.185758	0.537850	0.420622	0.363778	0.224054
GarageArea	0.017634	-0.098672	0.180403	0.562022	-0.151521	0.478954	0.371600	0.372567	0.296970
SalePrice	-0.021917	-0.084284	0.263843	0.790982	-0.077856	0.522897	0.507101	0.475241	0.386420

11 rows × 37 columns

In [ ]:

New dataframe built with features with correlation > 0.5 with SalePrice

In [19]:
```python
df_train4 = df_train3[correlationSale.index]
df_train4
```

Out[19]:

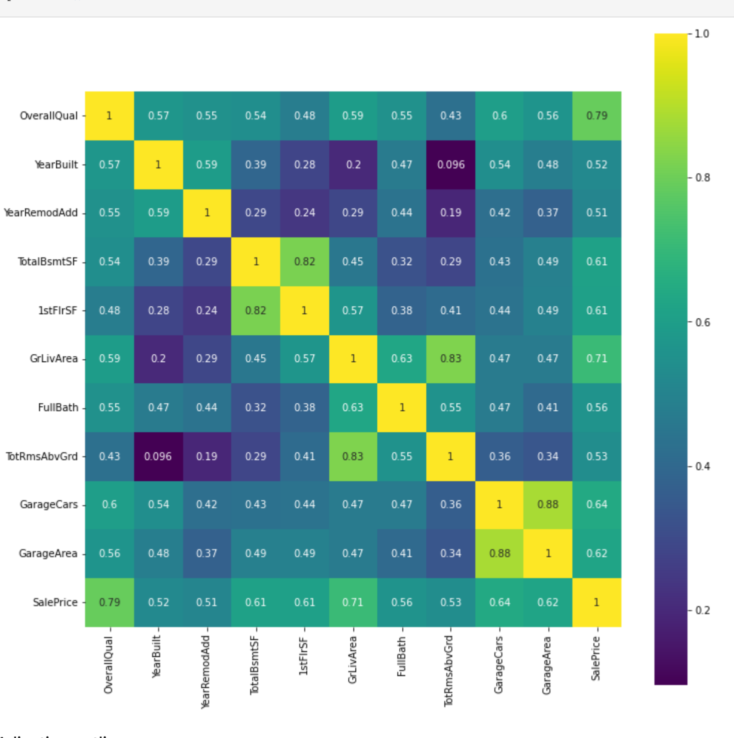
	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea
0	7	2003	2003	856	856	1710	2	8	2	548
1	6	1976	1976	1262	1262	1262	2	6	2	460
2	7	2001	2002	920	920	1786	2	6	2	608
3	7	1915	1970	756	961	1717	1	7	3	642
4	8	2000	2000	1145	1145	2198	2	9	3	836
...	...	...	...	...	...	...	...	...	...	...
1455	6	1999	2000	953	953	1647	2	7	2	460
1456	6	1978	1988	1542	2073	2073	2	7	2	500
1457	7	1941	2006	1152	1188	2340	2	9	1	252
1458	5	1950	1996	1078	1078	1078	1	5	1	240
1459	5	1965	1965	1256	1256	1256	1	6	1	276

1460 rows × 11 columns

Final correlation table with only features that are > 0.5 with 'Saleprice'

In [20]:
```python
correlationTable2 = df_train4.corr()
```

In [21]:
```python
plt.subplots(figsize=(12, 12))
sns.heatmap(correlationTable2,square=True,cmap='viridis',annot=True )
plt.show()
```



Adjusting outliers

As seen from the boxplot, there are several outliers value which are an exception, their values have to be adjusted or it will affect the result of the regression model when training it.

In [22]:
```python
df_train4.describe()
```

Out[22]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.00000
mean	6.099315	1971.267808	1984.865753	1057.429452	1162.626712	1515.463699	1.565068	6.517808	1.76712
std	1.382997	30.202904	20.645407	438.705324	386.587738	525.480383	0.550916	1.625393	0.74731
min	1.000000	1872.000000	1950.000000	0.000000	334.000000	334.000000	0.000000	2.000000	0.00000
25%	5.000000	1954.000000	1967.000000	795.750000	882.000000	1129.500000	1.000000	5.000000	1.00000
50%	6.000000	1973.000000	1994.000000	991.500000	1087.000000	1464.000000	2.000000	6.000000	2.00000
75%	7.000000	2000.000000	2004.000000	1298.250000	1391.250000	1776.750000	2.000000	7.000000	2.00000
max	10.000000	2010.000000	2010.000000	6110.000000	4692.000000	5642.000000	3.000000	14.000000	4.00000

One way to remove the outliers is to limit it based on the IQR. By removing adjusting values that lie outside the range defined by the quartiles +/- 1.5 * IQR.

In [23]:
```python
iqRange = df_train4.describe().SalePrice[6] - df_train4.describe().SalePrice[4]
```

I have tried different values of multiplying iqRange and found 1.5 to be the most optimal, as it has the best R-squared score, which will be shown below.

In [24]:
```python
boundary = iqRange * 1.5
boundary
```

Out[24]: 126037.5

In [25]:
```
upperBoundary = df_train4.describe().SalePrice[6] + boundary  #upper selling price boundary
upperBoundary
```

Out[25]: 340037.5

creating 2 dataframes to seperate Saleprice above the upperBoundary and lesser than the upperBoundary

In [26]:
```
df_train5 = (df_train4[df_train4['SalePrice']<upperBoundary])
```

In [27]:
```
df_train6 = (df_train4[df_train4['SalePrice']>upperBoundary])
```

As the outliers are mostly above the upper boundary, the outlier's value will be changed to the upperBoundary value.

In [28]:
```
df_train6['SalePrice'] = upperBoundary
df_train6
```

```
/var/folders/63/48hcdx8x3_s91h_x2yvfm0kc0000gn/T/ipykernel_2841/1041908506.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df_train6['SalePrice'] = upperBoundary
```

Out[28]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea	Sa
11	9	2005	2006	1175	1182	2324	3	11	3	736	34
53	9	1981	1987	1842	1842	1842	0	5	3	894	34
58	10	2006	2006	1410	1426	2945	3	10	3	641	34
112	7	2007	2007	1264	1282	2696	2	10	3	792	34
151	8	2007	2008	1710	1710	1710	2	6	3	866	34
...	...	...	...	...	...	...	...	...	...	...	
1268	8	1935	1997	728	1968	3447	3	11	3	1014	34
1353	8	1995	1996	2033	2053	3238	2	9	3	666	34
1373	10	2001	2002	2633	2633	2633	2	8	3	804	34
1388	9	2006	2007	1746	1746	1746	2	7	3	758	34
1437	8	2008	2008	1932	1932	1932	2	7	3	774	34

61 rows × 11 columns

Combine the 2 dataframes with outliers adjusted.

In [29]:
```
df_train7 = pd.concat([df_train5,df_train6])
```

Ensuring that the ouliers have been adjusted by checking the max value of 'Saleprice'

In [30]:
```
df_train7['SalePrice'].describe()
```

Out[30]:
```
count      1460.000000
mean     177331.526370
std       67205.835915
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      340037.500000
Name: SalePrice, dtype: float64
```

as seen from the max price of SalePrice the upper boundary has been set proeprly

In [31]:
```
df_train7.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 0 to 1437
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   OverallQual   1460 non-null   int64  
 1   YearBuilt     1460 non-null   int64  
 2   YearRemodAdd  1460 non-null   int64  
 3   TotalBsmtSF   1460 non-null   int64  
 4   1stFlrSF      1460 non-null   int64  
 5   GrLivArea     1460 non-null   int64  
 6   FullBath      1460 non-null   int64  
 7   TotRmsAbvGrd  1460 non-null   int64  
 8   GarageCars    1460 non-null   int64  
 9   GarageArea    1460 non-null   int64  
 10  SalePrice     1460 non-null   float64
dtypes: float64(1), int64(10)
memory usage: 136.9 KB
```

In [ ]:

Model training

with the training data modified, we can now split the data into training set and testing set.

In [32]:
```
x = df_train7.drop('SalePrice', axis=1) # removing the 'SalePrice' as thats is what i am training the model on
y = df_train7['SalePrice']

# splitting data into train and test data set
x_train, x_test,y_train,y_test = train_test_split(x,y,test_size=0.3, random_state=0)
```

using polynomial regression and linear regression to train the model to predict y.

In [33]:
```
polyRegress = PolynomialFeatures(degree = 2) # not using a high degree to prevent overfitting
x_poly = polyRegress.fit_transform(x_train) #adding x^2 to the feature set
regression = LinearRegression()
regression.fit(x_poly,y_train) #train x_poly beacuse we have additional information

x_test_polyRegress = polyRegress.fit_transform(x_test)
y_pred = regression.predict(x_test_polyRegress)
y_pred
```

Out[33]:
```
array([109490.04290584, 198401.83800896, 183051.28304756, 251839.77867655,
       256749.86424462, 150921.34086749, 276431.54342893, 114082.66795201,
       191252.62517434, 198151.9429883 ,  95010.66064509, 189008.70705185,
       178580.28292495, 140074.83916597, 147375.05401192, 103738.88117345,
       198747.49441399, 127827.89923728, 207644.82935299, 222580.06578013,
       102574.32572919, 146715.94021172, 136791.14975238, 246348.54539633,
        75856.08930946,  85422.50244164, 135055.82809152, 219539.61530853,
       298169.41165207, 147896.08475747, 173361.31392929, 177641.04041121,
       268574.02518588, 285738.00151856, 210474.9946175 , 137971.64473948,
       287004.66824406, 204780.52074055, 121346.67124245, 182729.05589037,
        99061.89246649, 218235.35513316, 194548.41612074, 191324.76039823,
        98373.61235796, 143795.86257744, 113705.84597463, 129350.98952957,
       242292.02849521, 108631.67690599, 311997.37161289, 153024.91072173,
       300649.21150898, 217838.63344025, 221419.97127369, 121145.69653734,
       177083.14508997,  99387.72989715, 268637.36414461, 171345.63845792,
        97289.77353268, 122592.07031627, 119684.60780076, 280817.85282359,
        77121.60263729, 133613.74303716, 274036.06786798, 252415.8147199 ,
       122427.94857917, 268648.8514615 , 158020.12322452, 118451.7920736 ,
        56305.242736  , 193329.4489251 , 197123.6832372 ,  99529.71759549,
       174619.23064338, 312774.36508642, 123484.53769276, 151694.30664381,
       255428.79349952, 132899.25161337, 166295.57088984, 131217.06451398,
       225337.60940243, 188958.00041828, 341828.67717732, 110666.90843318,
       188780.14264608, 129177.68912225, 121420.61343321, 224441.92830949,
       239673.60243678, 167790.1430672 , 152691.64546878, 142550.60063724,
       153537.45743287, 192083.04045409, 139324.07403774, 112038.69549231,
       192107.10738917, 137515.90763343, 279088.07243434,  98438.35064481,
       189596.47547415, 101518.36806684, 167514.71044519, 140186.66069536,
       126485.12123279, 169683.80182986, 175858.41957342,  78643.84906202,
       258363.91287023, 166839.68841779, 209990.96643056, 121933.93585661,
       208297.55926473, 152012.08664444, 125921.20824572, 342157.26794451,
       181222.2878738 , 146313.18002059, 167713.64717732, 212705.77598668,
       109902.91500774, 147343.64228322, 141844.46958713, 211573.2979557 ,
       116399.75512255, 133991.84122341, 134327.42889743,  94300.88445211,
       158409.63571338, 308739.1967174 , 135157.30346211, 269545.51338665,
       129213.70721326, 156431.23052122, 222215.35318825, 243702.19578947,
       145580.24552869, 321443.06261538, 162956.01222113, 111277.96002054,
       170473.56631144,  91742.64326431,  72144.65464986, 115914.34396509,
       254825.54261952, 106770.40817777, 209830.58183416, 130933.76134094,
       107343.74879224, 125883.55789168, 313154.84519569, 250947.95247835,
       239018.00895897, 132879.49619803, 160194.79544952, 285504.4183331 ,
       116892.22911628, 137967.01077427, 273672.10227157, 196173.85422113,
       197723.97072329, 157263.73130814, 154765.3127086 , 123996.00387144,
       291961.58315175, 261500.76474361, 281317.7445121 , 126849.63155316,
       149112.19345865, 116830.58873171, 154283.89372988, 225532.9362108 ,
       222742.42620386, 201681.3684731 ,  95939.59061632, 245328.91261151,
       126496.04513558, 229509.96273157, 113822.70678719, 181881.89398718,
       124101.84423456, 166282.00380841, 211107.2174413 , 122267.57796404,
       143560.25705086, 228053.31309357, 209287.8307372 , 260245.59320603,
       285599.2007581 , 137784.75756841, 210305.42462121, 258637.39547722,
       201112.07794244, 145363.95863953, 174032.01944788, 190865.7874865 ,
       289810.53432098, 157373.81843262, 216278.18082751, 156869.24428999,
       133684.6704713 , 219704.41455059, 143477.08028722, 255476.35715854,
       156620.79797017, 121898.31222511, 138143.48200295, 126503.68761229,
       116524.18142279, 112140.45328769, 166132.34887502, 218556.02215996,
       215846.46490641, 214087.89470901, 123922.04749205, 171176.67845714,
       186971.37770001, 205788.6998785 , 107862.32748148, 166546.75951808,
       217395.9661329 , 180244.62900358, 136649.87486277, 185057.8218722 ,
       162000.79785914, 109745.56057109, 236744.31546078, 135346.34982956,
       146035.65908188, 126882.08711936, 183143.07749296, 259132.39775157,
       246572.81178805, 178054.8750076 , 158138.73261026, 174154.41015687,
       228460.40373645, 202296.92577418, 126716.13900198, 215693.57050887,
       159999.78206745, 147509.33206917, 170181.49685468, 293621.24446855,
       133927.70935878, 138008.37433632, 114781.65360314, 181617.52819753,
       105752.11246022,  99853.800117  , 108889.26871812, 136748.34534663,
       244148.56350351, 352903.82684175, 129166.19435323, 110413.85174651,
       106674.89649584, 202045.05505653, 175856.41735777, 150429.18450889,
       224694.30840416, 127408.18916312, 117706.61528956,  98994.66246258,
       182345.90397836, 118438.00415924, 154497.73582543, 251801.59066093,
       200231.92987263, 145664.54168873, 141462.56104378, 146195.49104614,
       159737.00062154, 161404.77091705, 121190.67621633, 282007.18807211,
       227499.91111899, 198792.61743275, 326521.97019897, 301847.8467222 ,
       309949.50885392, 224014.22952   , 139151.13014713,  83529.20118665,
       237747.57645258, 103644.36447546, 167901.35097543, 237388.66493032,
        73895.82550979,  52221.01907517, 162581.50701355, 126453.32199796,
       172753.54931597, 334443.32110003, 148032.39115482, 105452.71878035,
       150597.9870963 , 149108.57708557, 195308.16390609, 142234.40165016,
       288632.94399197, 231203.78530098, 213360.90493401, 218218.31491251,
       303536.2409756 , 114235.57685376, 223263.4899383 , 114993.61011592,
       177327.20830762, 282547.82177505, 173742.77336658, 349742.10536848,
       205740.21298373, 100153.66813743, 158356.17177756, 275078.76614436,
       199038.7721285 ,  91302.75599073, 159930.85795156, 182420.71382397,
       200900.48784977, 157289.81357083, 152752.90636158, 322731.10827118,
       116965.44205798, 161512.38463014, 163359.04867297,  70498.69072864,
       215163.03474952,  38239.56422355, 103047.83907318, 308388.37619887,
       194298.17783575, 273831.75601048, 186769.77706283, 111542.53278324,
       157724.91754118, 183973.05505633, 107394.55303393,  90164.05892707,
        43935.0133163 ,  58356.52108401, 137695.06282684, 144758.10055788,
       227388.63802316, 237247.86640046, 128200.5834338 , 173186.13514228,
       120200.62126318, 159052.76338739,  65675.5304099 , 223339.46996661,
       130792.2969999 , 174093.91677994, 322339.84449823,  89227.59033139,
       242965.36284034, 301600.37211999, 262042.22561376, 211292.24392663,
       167259.30701494, 154958.69870985, 164171.34415547, 206318.88793268,
       295310.54835324, 192137.42210079, 216842.5682981 , 190744.56584836,
       145173.65165149, 432104.23678144, 168526.02728989, 145196.98456023,
       240383.51582489, 342409.74489942, 122966.81169563,  86019.95664632,
       137080.72116845, 250570.0746314 , 140935.82117722, 284260.96549269,
       165674.54868931, 152105.63338637,  99449.24066921, 240635.88476641,
       206067.73874905, 304327.17278198, 233851.95117921, 184177.79619274,
       310526.08323765, 266447.36197588, 281135.93063465, 234270.05212304,
       123791.99877115, 189561.06308967, 117482.76288165, 354045.3524497 ,
       222727.19918124,  96871.23343659, 118966.31401593, 157947.8159758 ,
       213263.17173761, 105512.19943779, 329966.40572202, 171364.3497828 ,
       137675.51502026, 184130.96275552, 300309.2171446 , 112790.43370482,
       105419.21544906, 246686.33057752, 147353.82337771, 129862.47874471,
       163869.6168908 , 198037.63595346, 162346.30679636, 226951.13336746,
       226274.60719245, 170106.58435376, 209787.52790165, 139151.13014713,
       238618.71354429, 145049.8997422 , 150808.17344459, 118507.11435234,
       153261.56624896, 130168.62338898, 164404.09742556, 113393.61428505,
       201309.76790661, 203718.97988253, 205173.72003322, 139151.13014713,
        77322.39838054, 152956.17684705])
```

In [ ]:

Cleaning test dataset

Since we have actual testing data without the SalePrice, we can use it to predict the house value, however we firstly have to clean the test data which is the same steps as cleaning training dataset.

In [34]:
```
df_test = pd.read_csv('data/test.csv')
df_test
```

Out[34]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolAre
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	...	144	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	...	0	
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	AllPub	...	0	
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	AllPub	...	0	
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	AllPub	...	0	
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	AllPub	...	0	

1459 rows × 80 columns

Taken the same features from training data and using it to create test data with the same features

In [35]:
```
df_test2 = df_test[['OverallQual', 'YearBuilt', 'YearRemodAdd', 'TotalBsmtSF', '1stFlrSF',
       'GrLivArea', 'FullBath', 'TotRmsAbvGrd', 'GarageCars', 'GarageArea']]
df_test2
```

Out[35]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea
0	5	1961	1961	882.0	896	896	1	5	1.0	730.0
1	6	1958	1958	1329.0	1329	1329	1	6	1.0	312.0
2	5	1997	1998	928.0	928	1629	2	6	2.0	482.0
3	6	1998	1998	926.0	926	1604	2	7	2.0	470.0
4	8	1992	1992	1280.0	1280	1280	2	5	2.0	506.0
...	...	...	...	...	...	...	...	...	...	...
1454	4	1970	1970	546.0	546	1092	1	5	0.0	0.0
1455	4	1970	1970	546.0	546	1092	1	6	1.0	286.0
1456	5	1960	1996	1224.0	1224	1224	1	7	2.0	576.0
1457	5	1992	1992	912.0	970	970	1	6	0.0	0.0
1458	7	1993	1994	996.0	996	2000	2	9	3.0	650.0

1459 rows × 10 columns

checking for any missing values

In [36]:
```
df_test2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   OverallQual   1459 non-null   int64  
 1   YearBuilt     1459 non-null   int64  
 2   YearRemodAdd  1459 non-null   int64  
 3   TotalBsmtSF   1458 non-null   float64
 4   1stFlrSF      1459 non-null   int64  
 5   GrLivArea     1459 non-null   int64  
 6   FullBath      1459 non-null   int64  
 7   TotRmsAbvGrd  1459 non-null   int64  
 8   GarageCars    1458 non-null   float64
 9   GarageArea    1458 non-null   float64
dtypes: float64(3), int64(7)
memory usage: 114.1 KB
```

Fill the missing columns with mean

In [37]:
```
fillMissing(df_test2,'TotalBsmtSF')
fillMissing(df_test2,'GarageCars')
fillMissing(df_test2,'GarageArea')
```

```
/Users/ronaldgohjingwei/opt/anaconda3/lib/python3.9/site-packages/pandas/core/generic.py:6392: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return self._update_inplace(result)
```

Out[37]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea
0	5	1961	1961	882.0	896	896	1	5	1.0	730.0
1	6	1958	1958	1329.0	1329	1329	1	6	1.0	312.0
2	5	1997	1998	928.0	928	1629	2	6	2.0	482.0
3	6	1998	1998	926.0	926	1604	2	7	2.0	470.0
4	8	1992	1992	1280.0	1280	1280	2	5	2.0	506.0
...	...	...	...	...	...	...	...	...	...	...
1454	4	1970	1970	546.0	546	1092	1	5	0.0	0.0
1455	4	1970	1970	546.0	546	1092	1	6	1.0	286.0
1456	5	1960	1996	1224.0	1224	1224	1	7	2.0	576.0
1457	5	1992	1992	912.0	970	970	1	6	0.0	0.0
1458	7	1993	1994	996.0	996	2000	2	9	3.0	650.0

1459 rows × 10 columns

Test dataset after cleaning

In [38]:
```
df_test2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   OverallQual   1459 non-null   int64  
 1   YearBuilt     1459 non-null   int64  
 2   YearRemodAdd  1459 non-null   int64  
 3   TotalBsmtSF   1459 non-null   float64
 4   1stFlrSF      1459 non-null   int64  
 5   GrLivArea     1459 non-null   int64  
 6   FullBath      1459 non-null   int64  
 7   TotRmsAbvGrd  1459 non-null   int64  
 8   GarageCars    1459 non-null   float64
 9   GarageArea    1459 non-null   float64
dtypes: float64(3), int64(7)
memory usage: 114.1 KB
```

In [39]:
```
df_test2
```

Out[39]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea
0	5	1961	1961	882.0	896	896	1	5	1.0	730.0
1	6	1958	1958	1329.0	1329	1329	1	6	1.0	312.0
2	5	1997	1998	928.0	928	1629	2	6	2.0	482.0
3	6	1998	1998	926.0	926	1604	2	7	2.0	470.0
4	8	1992	1992	1280.0	1280	1280	2	5	2.0	506.0
...	...	...	...	...	...	...	...	...	...	...
1454	4	1970	1970	546.0	546	1092	1	5	0.0	0.0
1455	4	1970	1970	546.0	546	1092	1	6	1.0	286.0
1456	5	1960	1996	1224.0	1224	1224	1	7	2.0	576.0
1457	5	1992	1992	912.0	970	970	1	6	0.0	0.0
1458	7	1993	1994	996.0	996	2000	2	9	3.0	650.0

1459 rows × 10 columns

In [ ]:

Prediction on test data

using the regression model, we have trained to predict the sale price on the test data.

In [40]:
```
x_test_polyRegress2 = polyRegress.fit_transform(df_test2 )
x_test_polyRegress2
```

Out[40]:
```
array([[1.00000e+00, 5.00000e+00, 1.96100e+03, ..., 1.00000e+00,
        7.30000e+02, 5.32900e+05],
       [1.00000e+00, 6.00000e+00, 1.95800e+03, ..., 1.00000e+00,
        3.12000e+02, 9.73440e+04],
       [1.00000e+00, 5.00000e+00, 1.99700e+03, ..., 4.00000e+00,
        9.64000e+02, 2.32324e+05],
       ...,
       [1.00000e+00, 5.00000e+00, 1.96000e+03, ..., 4.00000e+00,
        1.15200e+03, 3.31776e+05],
       [1.00000e+00, 5.00000e+00, 1.99200e+03, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00],
       [1.00000e+00, 7.00000e+00, 1.99300e+03, ..., 9.00000e+00,
        1.95000e+03, 4.22500e+05]])
```

In [41]:
```
y_pred2 = regression.predict(x_test_polyRegress2)
y_pred2
```

Out[41]:
```
array([126957.47485102, 160884.34016817, 182580.30585962, ...,
       157422.39037056, 116243.87488677, 236934.54374853])
```

Predicted output for each house id

In [42]:
```
prediction = pd.DataFrame(df_test['Id'])
prediction['SalePrice'] = y_pred2
prediction
```

Out[42]:

	Id	SalePrice
0	1461	126957.474851
1	1462	160884.340168
2	1463	182580.305860
3	1464	183717.263677
4	1465	199915.113949
...	...	...
1454	2915	92399.674002
1455	2916	105452.718780
1456	2917	157422.390371
1457	2918	116243.874887
1458	2919	236934.543749

1459 rows × 2 columns

In [ ]:

3. Conclusion

3.1 Evaluate results

Using machine learning model fitting measures like R-Squared and root mean squared error

R-Squared

In [43]:
```
r2_score(y_test, y_pred)
```

Out[43]: 0.8658537666953896

Root mean squared error(RMSE)

In [44]:
```
mean_squared_error(y_test, y_pred)
```

Out[44]: 658804440.0468954

Based on the 2 measures, the R-Squared value tells us that the predictor variables in the model (square footage, # bathrooms, and # bedrooms) can explain 86% of the variation in the house prices.

The RMSE value tells us that the average deviation between the predicted house price made by the model and the actual house price is $638,645,937.

After researching online, a good score depends on the field, it was advised that an R squared value of between 0.7 to 0.9 would be a good score. This shows that the model can accurately predict the response variable's value using the predictor variable.

However, the RMSE value differs from the training set's maximum value, which is around $638,645,937. This much of a price difference tells me that the model is not a good choice for predicting house prices.

3.2 Reflection

Based on the results obtained from measuring the model, I think that the model did not succeed in predicting the house price well because of the RMSE score. If the model was able to predict the house price a bit more accurately, it will be able to transfer to other domain areas to predict different kinds of things as long as the inputs are numerical such as predicting a stock price or weather prediction.

R is a statistical language that is used for developing statistical software and data analysis. It contains many libraries such as Classification And REgression Training (caret) that streamline the process of creating predictive models. While R can do both data visualization and machine learning, R is better suited for data visualization and statistical function due to ease of use, while, python is better for machine learning. Hence, it depends on the goal when selecting which programming language to use. R is easier to use when it comes to statistical function and data visualizations

In [ ]:

4. References

- US house price trend: https://www.redfin.com/us-housing-market
- Data set: https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data
- Removing outliers: https://builtin.com/data-science/how-to-find-outliers-with-iqr
- polynomial regression: https://www.youtube.com/watch?v=H8kocPOT5v0
- seaborn api: https://seaborn.pydata.org/api.html
- sci-kit learn api: https://scikit-learn.org/stable/modules/classes.html
- R vs Python: https://www.projectpro.io/article/is-predictive-modelling-easier-with-r-or-with-python/245

In [ ]: