# Housing Price Prediction

## Problem Statement:-

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

A US-based housing company named **Surprise Housing** has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below.

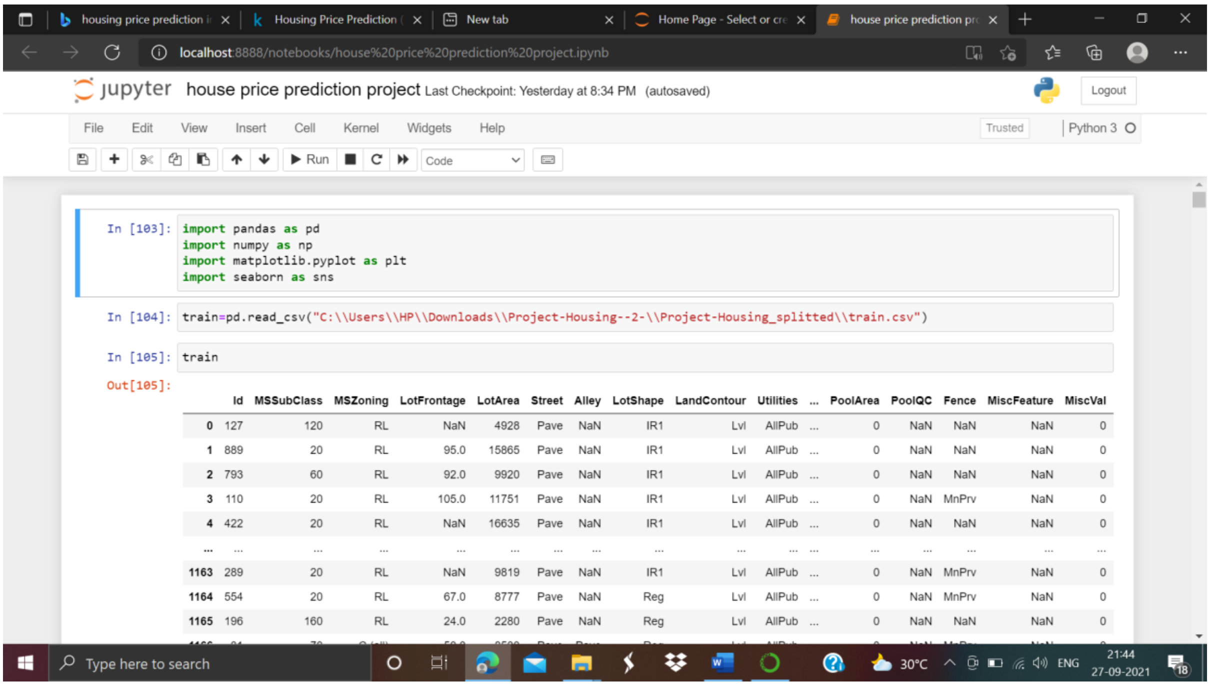
The company is looking at prospective properties to buy houses to enter the market. You are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. For this company wants to know:

- • Which variables are important to predict the price of variable?
- • How do these variables describe the price of the house?

**Data**

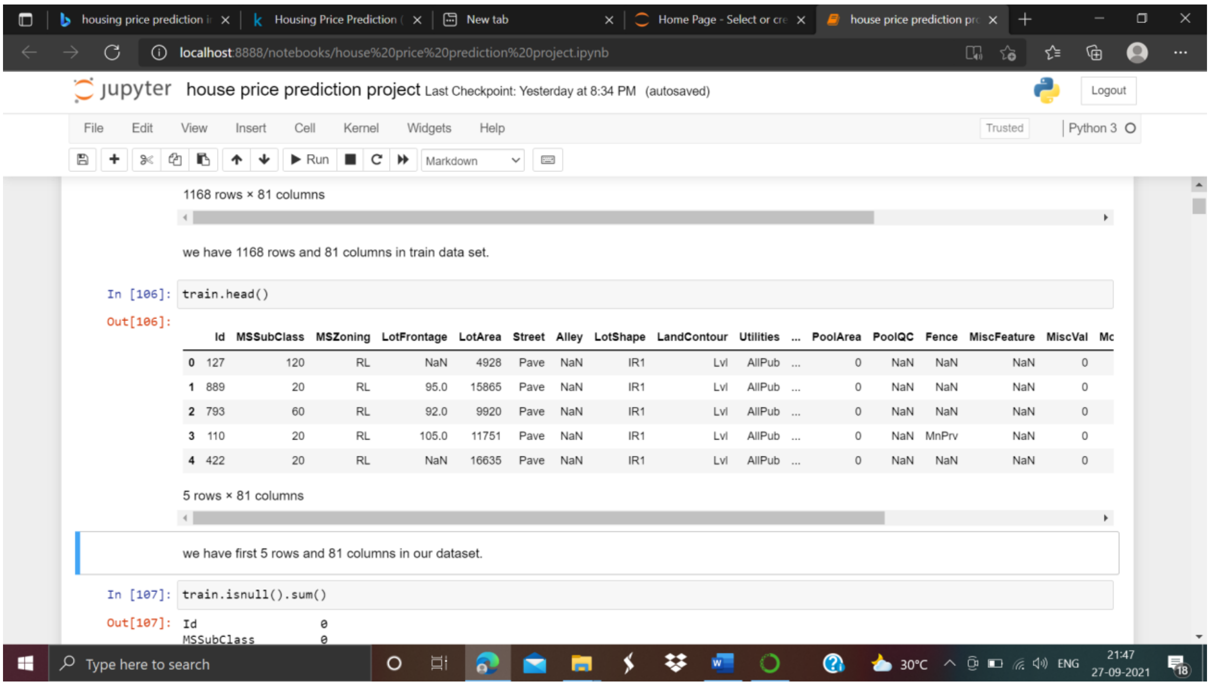
Use housing dataset.

# Reading and Understanding the Data




## <u>Data Inspection</u>

Let’s have  a look to inspect the our dataset.



1168 rows × 81 columns

we have 1168 rows and 81 columns in train data set.

```
In [106]: train.head()
```

Out[106]:

| | Id | MSSubClass | MSZoning | LotFrontage | LotArea | Street | Alley | LotShape | LandContour | Utilities | ... | PoolArea | PoolQC | Fence | MiscFeature | MiscVal | Mc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 127 | 120 | RL | NaN | 4928 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | |
| 1 | 889 | 20 | RL | 95.0 | 15865 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | |
| 2 | 793 | 60 | RL | 92.0 | 9920 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | |
| 3 | 110 | 20 | RL | 105.0 | 11751 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | MnPrv | NaN | 0 | |
| 4 | 422 | 20 | RL | NaN | 16635 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | |

5 rows × 81 columns

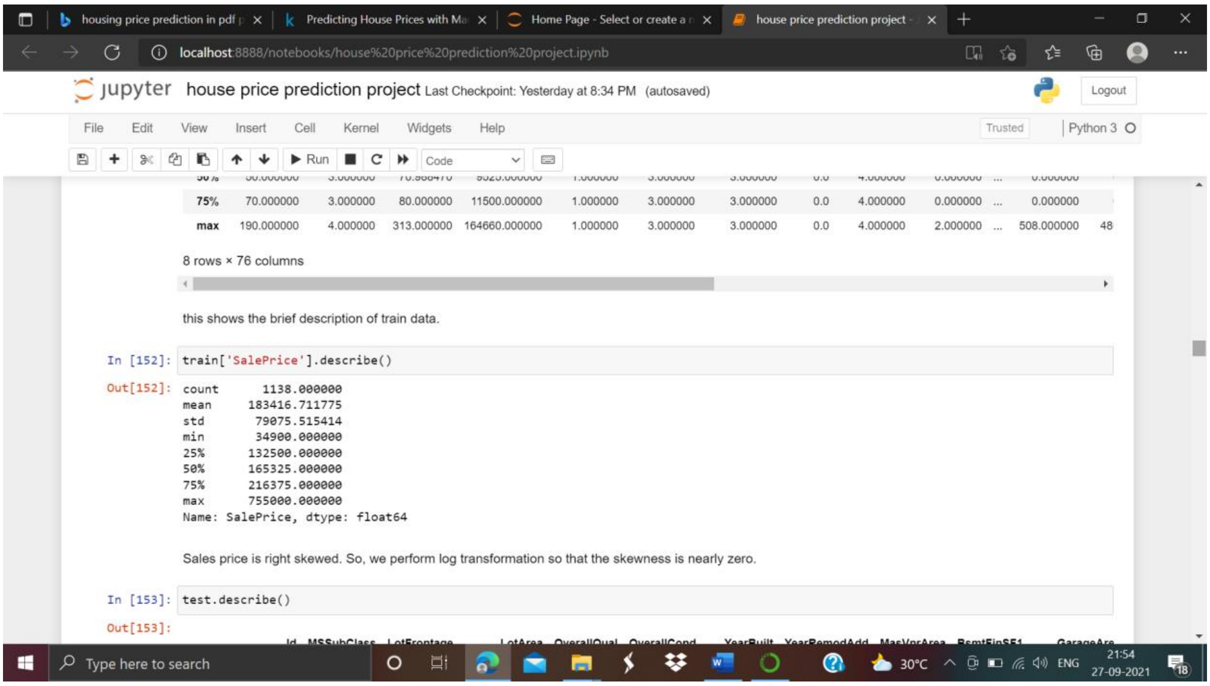
we have first 5 rows and 81 columns in our dataset.

```
In [107]: train.isnull().sum()
```

```
Out[107]: Id            0
MSSubClass    0
```

# Analyzing the Test Variable (Sale Price):-

Let's check out the most interesting feature in this study: Sale Price. Important Note: This data is from Ames, Iowa. The location is extremely correlated with Sale Price. (I had to take a double-take at a point, since I consider myself a house-browsing enthusiast)



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 75% | 70.000000 | 3.000000 | 80.000000 | 11500.000000 | 1.000000 | 3.000000 | 3.000000 | 0.0 | 4.000000 | 0.000000 | ... | 0.000000 | |
| max | 190.000000 | 4.000000 | 313.000000 | 164660.000000 | 1.000000 | 3.000000 | 3.000000 | 0.0 | 4.000000 | 2.000000 | ... | 508.000000 | 48 |

8 rows × 76 columns

this shows the brief description of train data.

```
In [152]: train['SalePrice'].describe()
```

```
Out[152]: count      1138.000000
mean     183416.711775
std       79075.515414
min       34900.000000
25%      132500.000000
50%      165325.000000
75%      216375.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

Sales price is right skewed. So, we perform log transformation so that the skewness is nearly zero.

```
In [153]: test.describe()
```

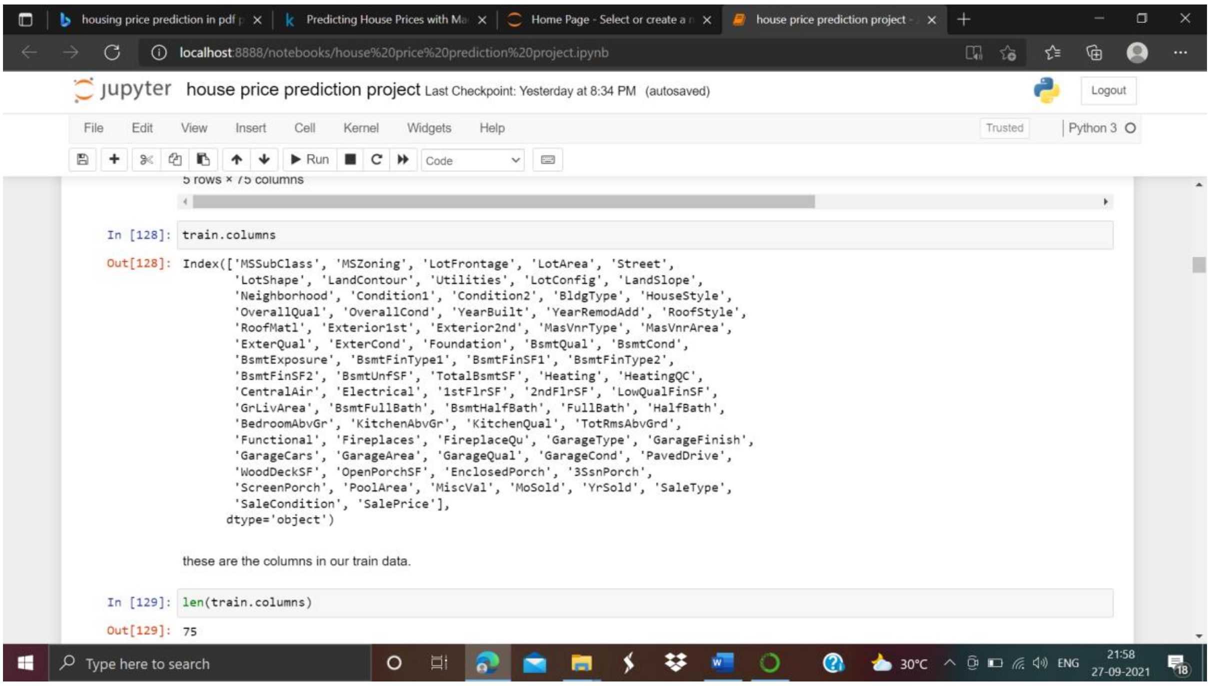
Out[153]:

## **Multivariable Analysis:-**

Let's check out all the variables! There are two types of features in housing data, categorical and numerical.

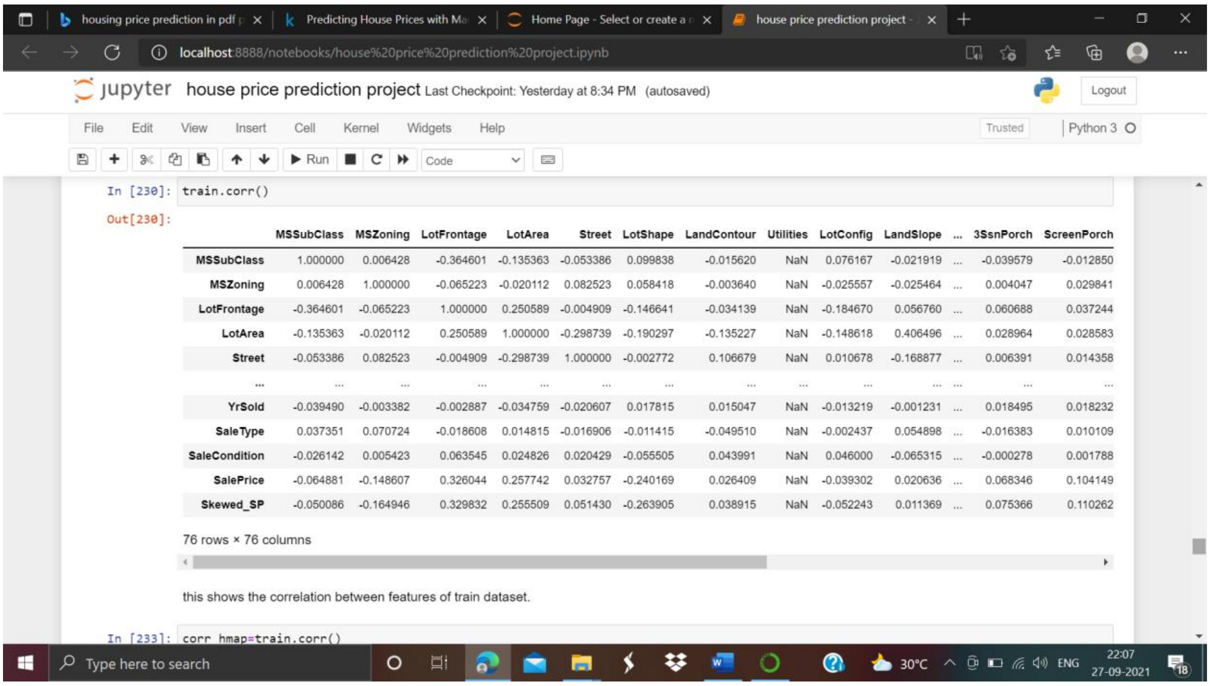
Categorical data is just like it sounds. It is in categories. It isn't necessarily linear, but it follows some kind of pattern. For example, take a feature of "Downtown". The response is either "Near", "Far", "Yes", and "No". Back then, living in downtown usually meant that you couldn't afford to live in uptown. Thus, it could be implied that downtown establishments cost less to live in. However, today, that is not the case. (Thank you, hipsters!) So we can't really establish any particular order of response to be "better" or "worse" than the other.

Numerical data is data in number form. (Who could have thought!) These features are in a linear relationship with each other. For example, a 2,000 square foot place is 2 times "bigger" than a 1,000 square foot place. Plain and simple. Simple and clean.




## Find the correlation

It's a very important to find the correlation among the variables, but oh man is that a lot of data to look at. Let's zoom into the top 10 features most related to Sale Price.



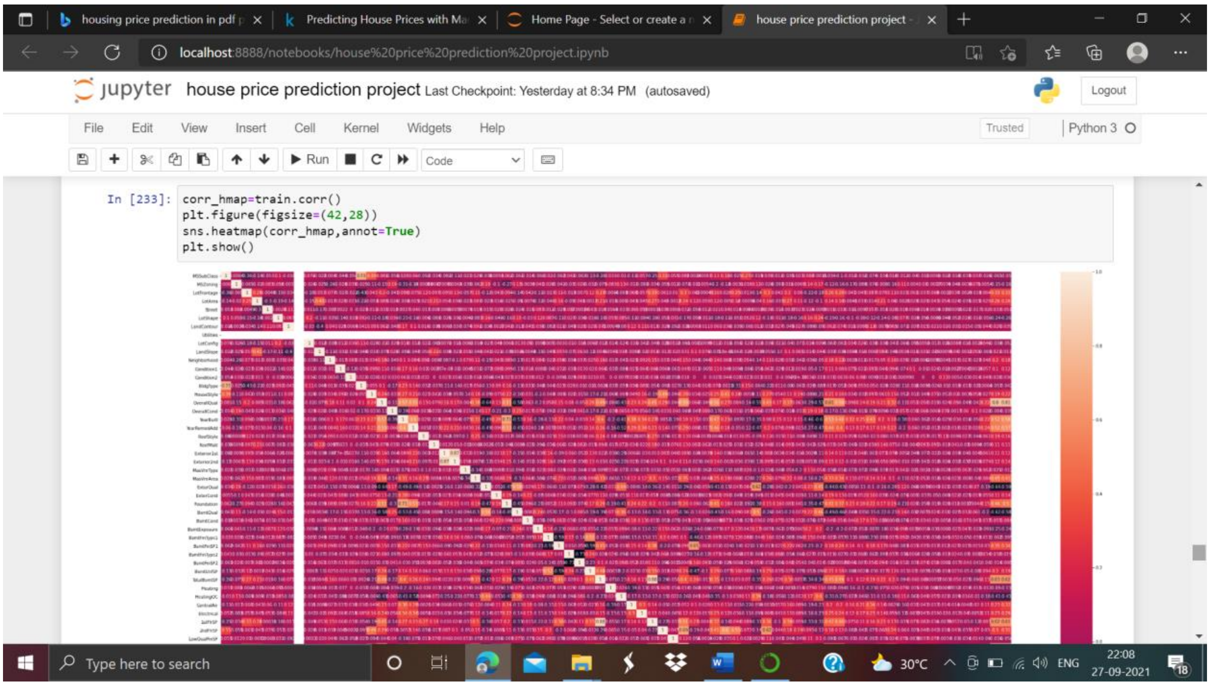
In [230]:
```
train.corr()
```

Out[230]:

| | MSSubClass | MSZoning | LotFrontage | LotArea | Street | LotShape | LandContour | Utilities | LotConfig | LandSlope | ... | 3SsnPorch | ScreenPorch |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **MSSubClass** | 1.000000 | 0.006428 | -0.364601 | -0.135363 | -0.053386 | 0.099838 | -0.015620 | NaN | 0.076167 | -0.021919 | ... | -0.039579 | -0.012850 |
| **MSZoning** | 0.006428 | 1.000000 | -0.065223 | -0.020112 | 0.082523 | 0.058418 | -0.003640 | NaN | -0.025557 | -0.025464 | ... | 0.004047 | 0.029841 |
| **LotFrontage** | -0.364601 | -0.065223 | 1.000000 | 0.250589 | -0.004909 | -0.146641 | -0.034139 | NaN | -0.184670 | 0.056760 | ... | 0.060688 | 0.037244 |
| **LotArea** | -0.135363 | -0.020112 | 0.250589 | 1.000000 | -0.298739 | -0.190297 | -0.135227 | NaN | -0.148618 | 0.406496 | ... | 0.028964 | 0.028583 |
| **Street** | -0.053386 | 0.082523 | -0.004909 | -0.298739 | 1.000000 | -0.002772 | 0.106679 | NaN | 0.010678 | -0.168877 | ... | 0.006391 | 0.014358 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **YrSold** | -0.039490 | -0.003382 | -0.002887 | -0.034759 | -0.020607 | 0.017815 | 0.015047 | NaN | -0.013219 | -0.001231 | ... | 0.018495 | 0.018232 |
| **SaleType** | 0.037351 | 0.070724 | -0.018608 | 0.014815 | -0.016906 | -0.011415 | -0.049510 | NaN | -0.002437 | 0.054898 | ... | -0.016383 | 0.010109 |
| **SaleCondition** | -0.026142 | 0.005423 | 0.063545 | 0.024826 | 0.020429 | -0.055505 | 0.043991 | NaN | 0.046000 | -0.065315 | ... | -0.000278 | 0.001788 |
| **SalePrice** | -0.064881 | -0.148607 | 0.326044 | 0.257742 | 0.032757 | -0.240169 | 0.026409 | NaN | -0.039302 | 0.020636 | ... | 0.068346 | 0.104149 |
| **Skewed_SP** | -0.050086 | -0.164946 | 0.329832 | 0.255509 | 0.051430 | -0.263905 | 0.038915 | NaN | -0.052243 | 0.011369 | ... | 0.075366 | 0.110262 |

76 rows × 76 columns

this shows the correlation between features of train dataset.



In [233]:
```
corr_hmap=train.corr()
plt.figure(figsize=(42,28))
sns.heatmap(corr_hmap,annot=True)
plt.show()
```

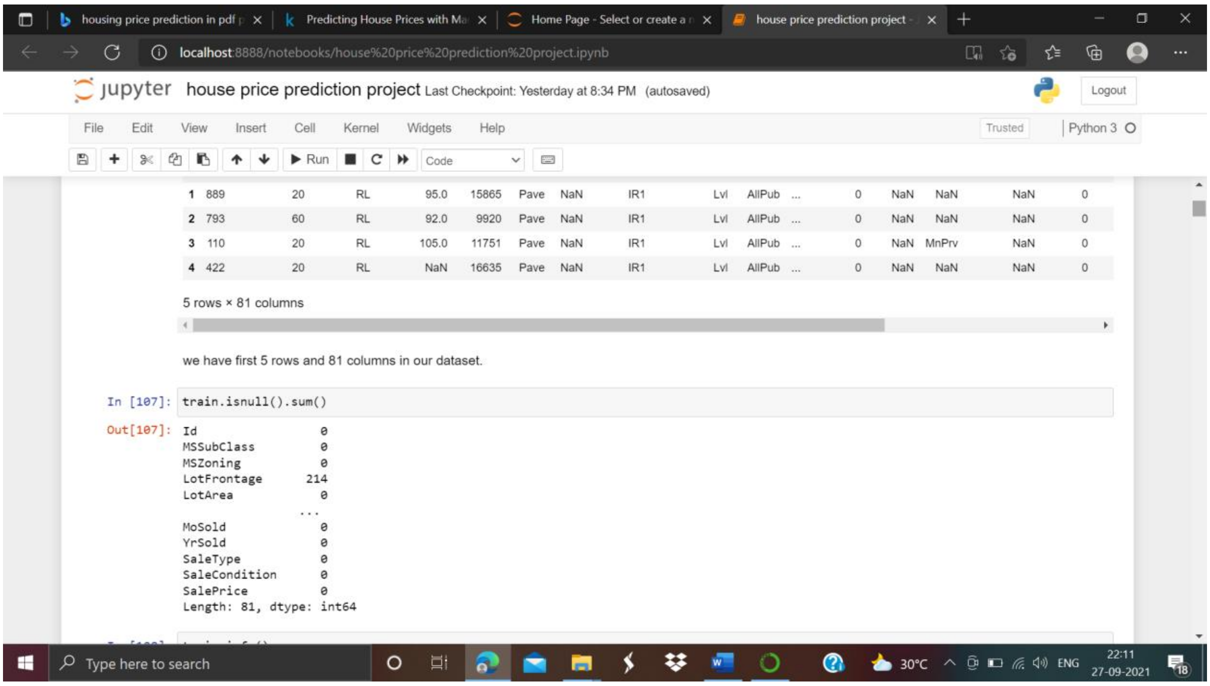
## . Impute Missing Data and Clean Data:-

Important questions when thinking about missing data:

- How prevalent is the missing data?
- Is missing data random or does it have a pattern?

The answer to these questions is important for practical reasons because missing data can imply a reduction of the sample size. This can prevent us from proceeding with the analysis. Moreover, from a substantive perspective, we need to ensure that the missing data process is not biased and hiding an inconvenient truth.

Let's combine both training and test data into one dataset to impute missing values and do some cleaning.



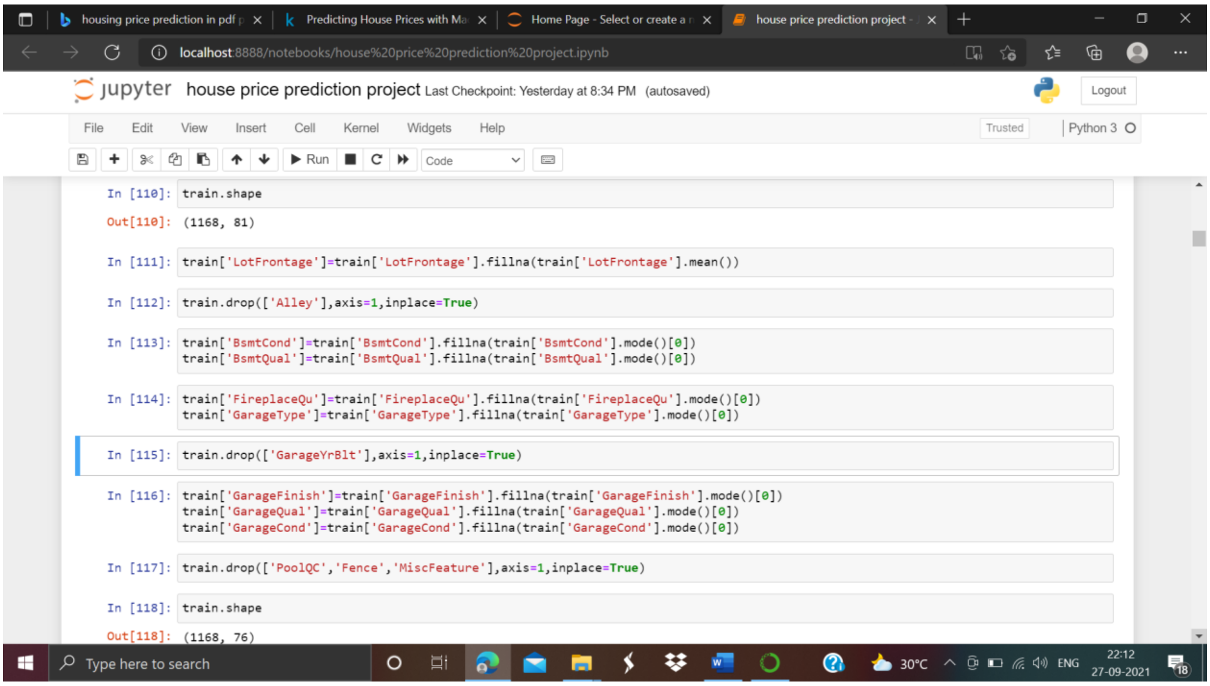

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 889 | 20 | RL | 95.0 | 15865 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 |
| 2 | 793 | 60 | RL | 92.0 | 9920 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 |
| 3 | 110 | 20 | RL | 105.0 | 11751 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | MnPrv | NaN | 0 |
| 4 | 422 | 20 | RL | NaN | 16635 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 |

5 rows × 81 columns

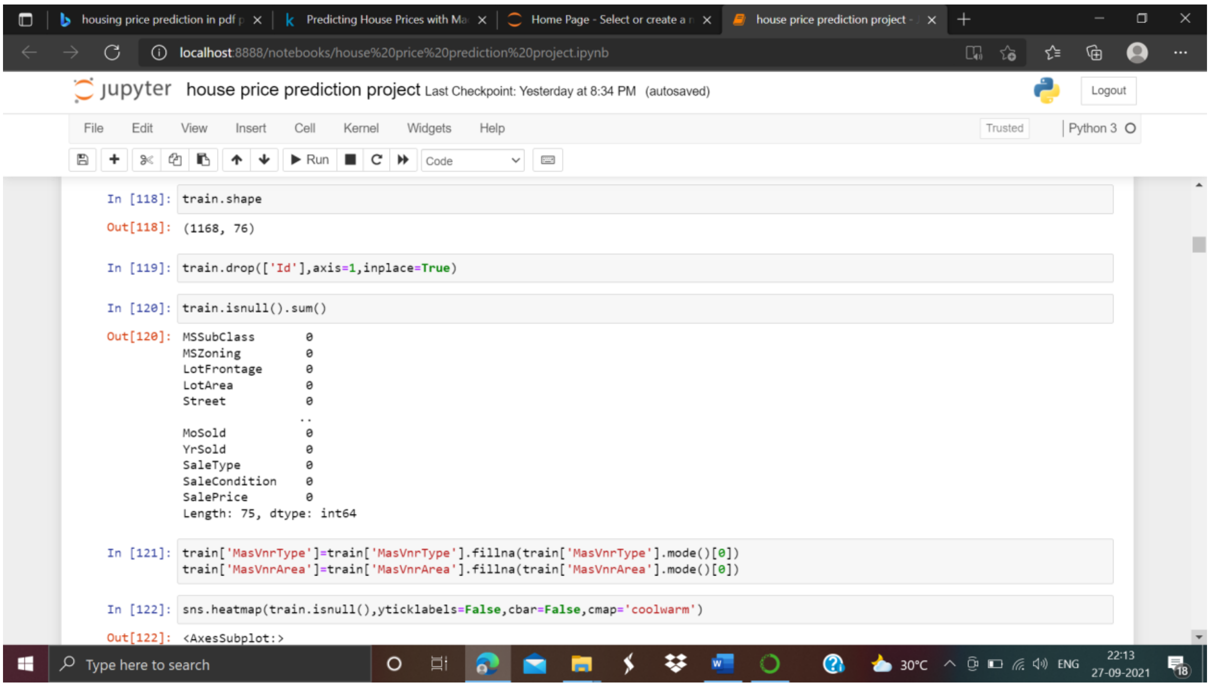
we have first 5 rows and 81 columns in our dataset.

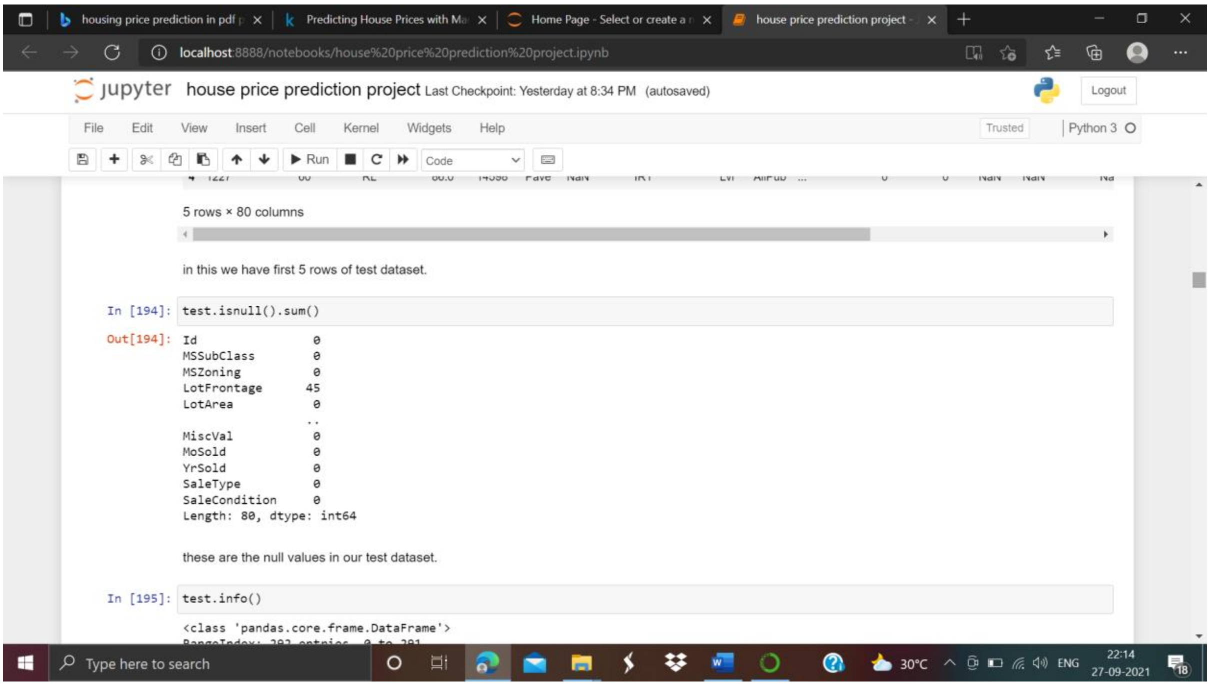
```
In [107]: train.isnull().sum()
Out[107]: Id                0
          MSSubClass        0
          MSZoning          0
          LotFrontage     214
          LotArea           0
                         ...
          MoSold            0
          YrSold            0
          SaleType          0
          SaleCondition     0
          SalePrice         0
          Length: 81, dtype: int64
```

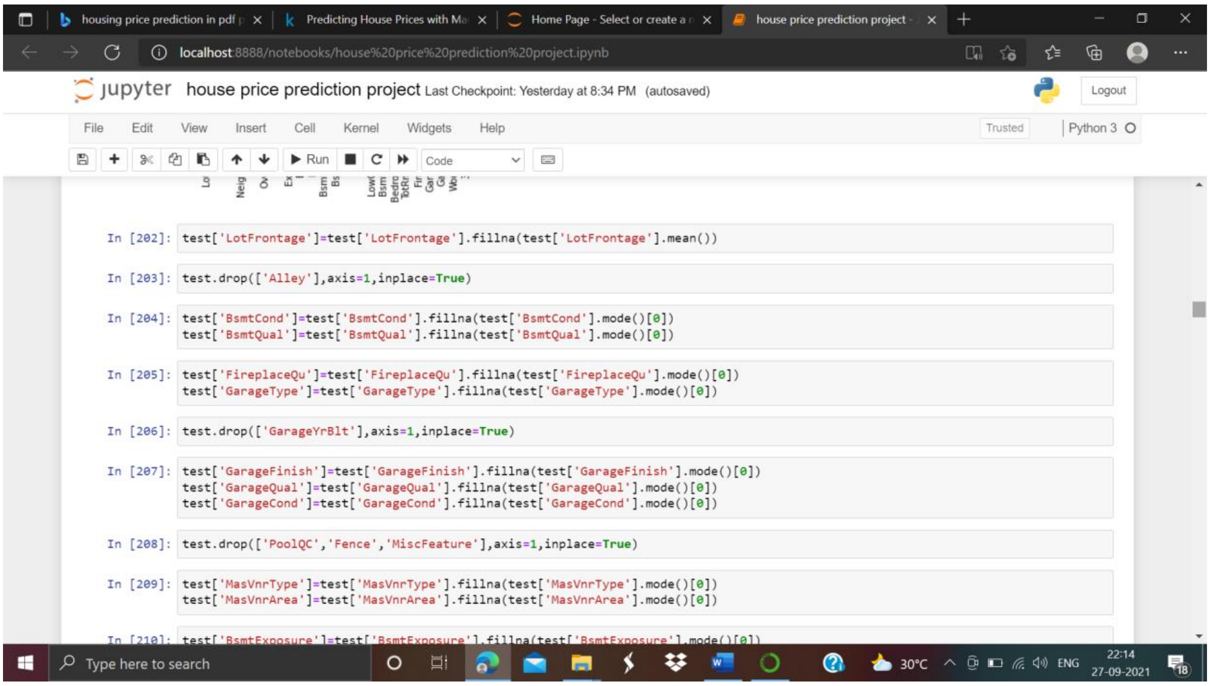


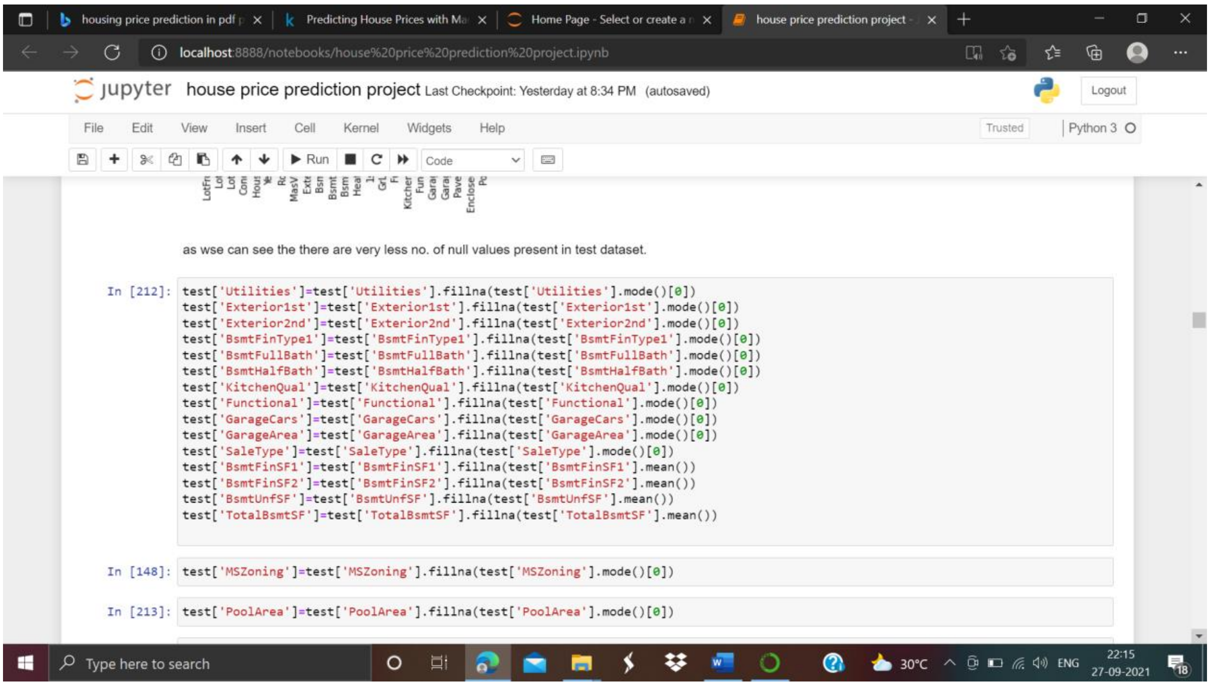



```
In [110]: train.shape
Out[110]: (1168, 81)

In [111]: train['LotFrontage']=train['LotFrontage'].fillna(train['LotFrontage'].mean())

In [112]: train.drop(['Alley'],axis=1,inplace=True)

In [113]: train['BsmtCond']=train['BsmtCond'].fillna(train['BsmtCond'].mode()[0])
          train['BsmtQual']=train['BsmtQual'].fillna(train['BsmtQual'].mode()[0])

In [114]: train['FireplaceQu']=train['FireplaceQu'].fillna(train['FireplaceQu'].mode()[0])
          train['GarageType']=train['GarageType'].fillna(train['GarageType'].mode()[0])

In [115]: train.drop(['GarageYrBlt'],axis=1,inplace=True)

In [116]: train['GarageFinish']=train['GarageFinish'].fillna(train['GarageFinish'].mode()[0])
          train['GarageQual']=train['GarageQual'].fillna(train['GarageQual'].mode()[0])
          train['GarageCond']=train['GarageCond'].fillna(train['GarageCond'].mode()[0])

In [117]: train.drop(['PoolQC','Fence','MiscFeature'],axis=1,inplace=True)

In [118]: train.shape
Out[118]: (1168, 76)
```


housing price prediction in pdf
Predicting House Prices with Ma
Home Page - Select or create a
house price prediction project
localhost:8888/notebooks/house%20price%20prediction%20project.ipynb
jupyter house price prediction project Last Checkpoint: Yesterday at 8:34 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [118]: train.shape
Out[118]: (1168, 76)
In [119]: train.drop(['Id'],axis=1,inplace=True)
In [120]: train.isnull().sum()
Out[120]: MSSubClass 0
MSZoning 0
LotFrontage 0
LotArea 0
Street 0
..
MoSold 0
YrSold 0
SaleType 0
SaleCondition 0
SalePrice 0
Length: 75, dtype: int64
In [121]: train['MasVnrType']=train['MasVnrType'].fillna(train['MasVnrType'].mode()[0])
train['MasVnrArea']=train['MasVnrArea'].fillna(train['MasVnrArea'].mode()[0])
In [122]: sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='coolwarm')
Out[122]: <AxesSubplot:>
Type here to search
30°C
ENG
22:13
27-09-2021


housing price prediction in pdf
Predicting House Prices with Ma
Home Page - Select or create a
house price prediction project
localhost:8888/notebooks/house%20price%20prediction%20project.ipynb
jupyter house price prediction project Last Checkpoint: Yesterday at 8:34 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
5 rows × 80 columns
in this we have first 5 rows of test dataset.
In [194]: test.isnull().sum()
Out[194]: Id 0
MSSubClass 0
MSZoning 0
LotFrontage 45
LotArea 0
..
MiscVal 0
MoSold 0
YrSold 0
SaleType 0
SaleCondition 0
Length: 80, dtype: int64
these are the null values in our test dataset.
In [195]: test.info()
<class 'pandas.core.frame.DataFrame'>
Type here to search
30°C
ENG
22:14
27-09-2021



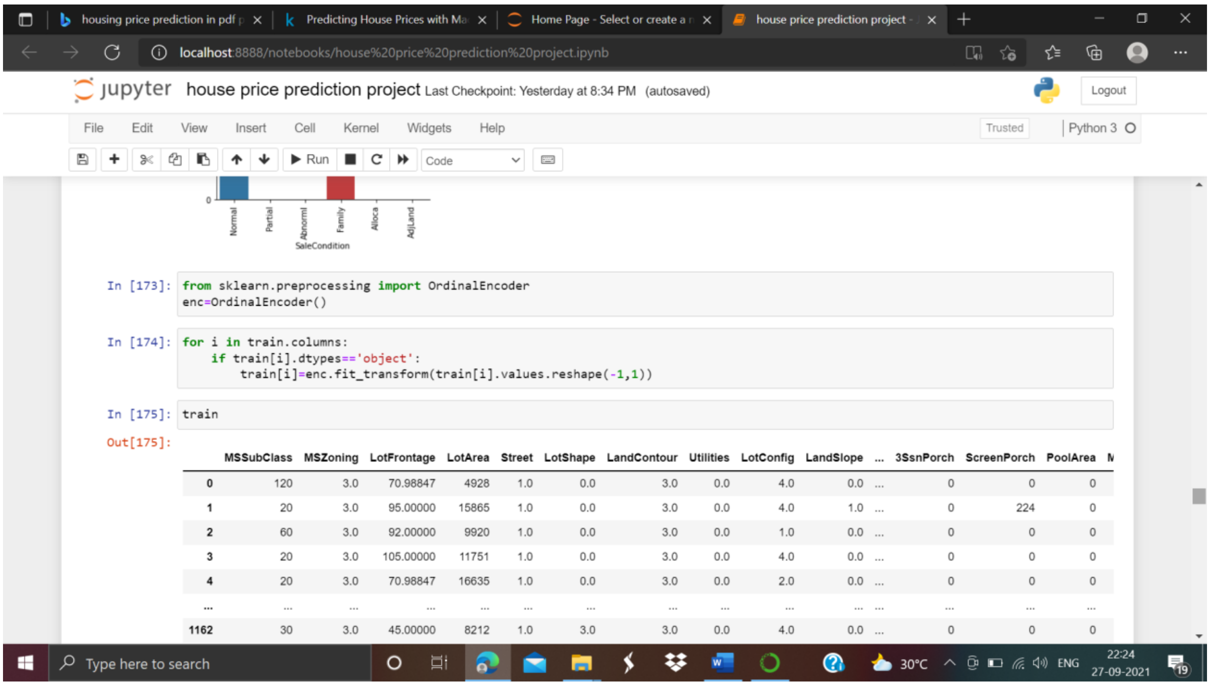

## Imputing Missing Values:-

- PoolQC : data description says NA means "No Pool"
- MiscFeature : data description says NA means "no misc feature"
- Alley : data description says NA means "no alley access"
- Fence : data description says NA means "no fence"
- FireplaceQu : data description says NA means "no fireplace"

- LotFrontage : Since the area of each street connected to the house property most likely have a similar area to other houses in its neighborhood , we can fill in missing values by the median LotFrontage of the neighborhood.
- GarageType, GarageFinish, GarageQual and GarageCond : Replacing missing data with "None".
- GarageYrBlt, GarageArea and GarageCars : Replacing missing data with 0.
- BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath and BsmtHalfBath: Replacing missing data with 0.
- BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1 and BsmtFinType2 : For all these categorical basement-related features, NaN means that there isn't a basement.
- MasVnrArea and MasVnrType : NA most likely means no masonry veneer for these houses. We can fill 0 for the area and None for the type.
- MSZoning (The general zoning classification) : 'RL' is by far the most common value. So we can fill in missing values with 'RL'.
- Utilities : For this categorical feature all records are "AllPub", except for one "NoSeWa" and 2 NA . Since the house with 'NoSewa' is in the training set, this feature won't help in predictive modelling. We can then safely remove it.
- Functional : data description says NA means typical.
- Electrical : It has one NA value. Since this feature has mostly 'SBrkr', we can set that for the missing value.
- KitchenQual: Only one NA value, and same as Electrical, we set 'TA' (which is the most frequent) for the missing value in KitchenQual.
- Exterior1st and Exterior2nd : Both Exterior 1 & 2 have only one missing value. We will just substitute in the most common string
- SaleType : Fill in again with most frequent which is "WD"
- MSSubClass : Na most likely means No building class. We can replace missing values with None

## <u>**Feature Transformation/Engineering:-**</u>

Let's take a look at some features that may be misinterpreted to represent something it's not.MSSubClass: Identifies the type of dwelling involved in the sale. Also we apply Label encoding for both the train and test dataset.



jupyter house price prediction project Last Checkpoint: Yesterday at 8:34 PM (autosaved)

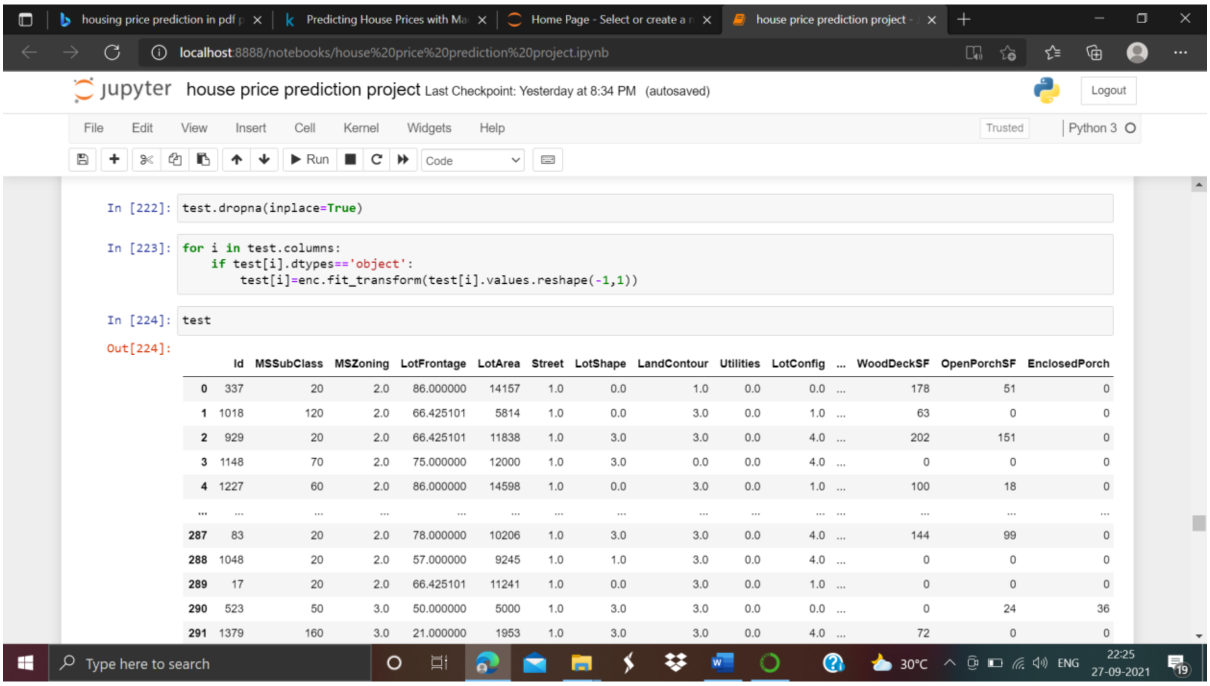
In [173]:
```python
from sklearn.preprocessing import OrdinalEncoder
enc=OrdinalEncoder()
```

In [174]:
```python
for i in train.columns:
    if train[i].dtypes=='object':
        train[i]=enc.fit_transform(train[i].values.reshape(-1,1))
```

In [175]:
```python
train
```

Out[175]:

| | MSSubClass | MSZoning | LotFrontage | LotArea | Street | LotShape | LandContour | Utilities | LotConfig | LandSlope | ... | 3SsnPorch | ScreenPorch | PoolArea |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 120 | 3.0 | 70.98847 | 4928 | 1.0 | 0.0 | 3.0 | 0.0 | 4.0 | 0.0 | ... | 0 | 0 | 0 |
| 1 | 20 | 3.0 | 95.00000 | 15865 | 1.0 | 0.0 | 3.0 | 0.0 | 4.0 | 1.0 | ... | 0 | 224 | 0 |
| 2 | 60 | 3.0 | 92.00000 | 9920 | 1.0 | 0.0 | 3.0 | 0.0 | 1.0 | 0.0 | ... | 0 | 0 | 0 |
| 3 | 20 | 3.0 | 105.00000 | 11751 | 1.0 | 0.0 | 3.0 | 0.0 | 4.0 | 0.0 | ... | 0 | 0 | 0 |
| 4 | 20 | 3.0 | 70.98847 | 16635 | 1.0 | 0.0 | 3.0 | 0.0 | 2.0 | 0.0 | ... | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1162 | 30 | 3.0 | 45.00000 | 8212 | 1.0 | 3.0 | 3.0 | 0.0 | 4.0 | 0.0 | ... | 0 | 0 | 0 |



jupyter house price prediction project Last Checkpoint: Yesterday at 8:34 PM (autosaved)

In [222]:
```python
test.dropna(inplace=True)
```

In [223]:
```python
for i in test.columns:
    if test[i].dtypes=='object':
        test[i]=enc.fit_transform(test[i].values.reshape(-1,1))
```

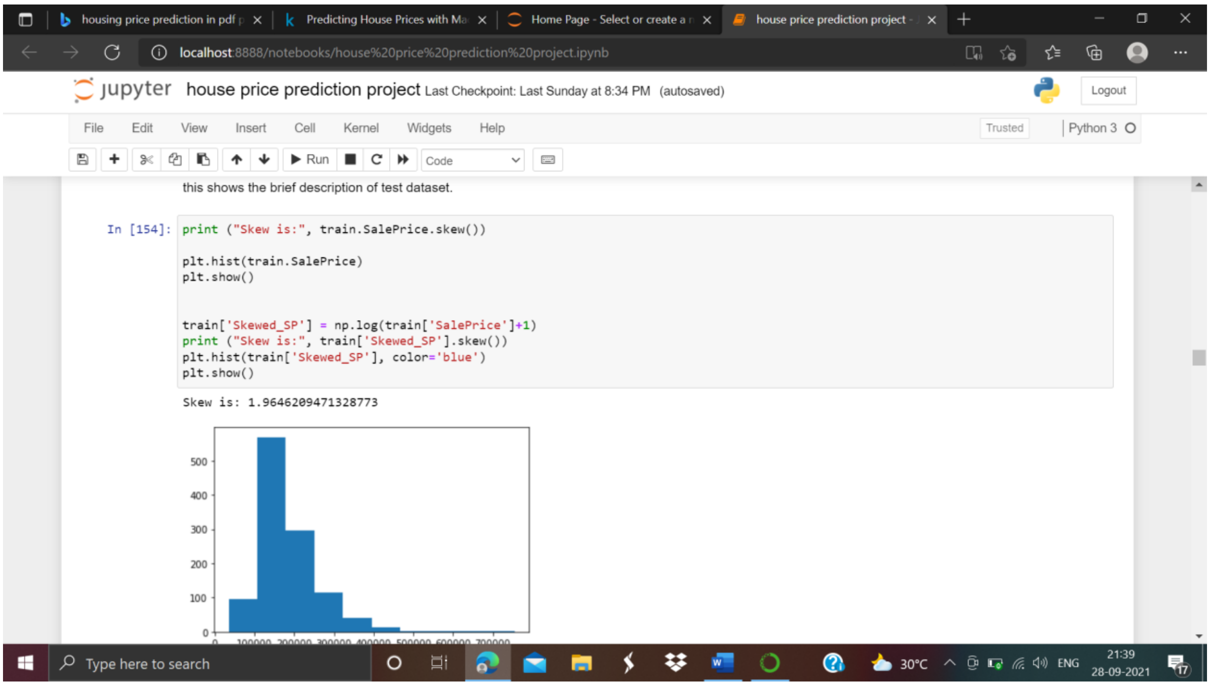
In [224]:
```python
test
```

Out[224]:

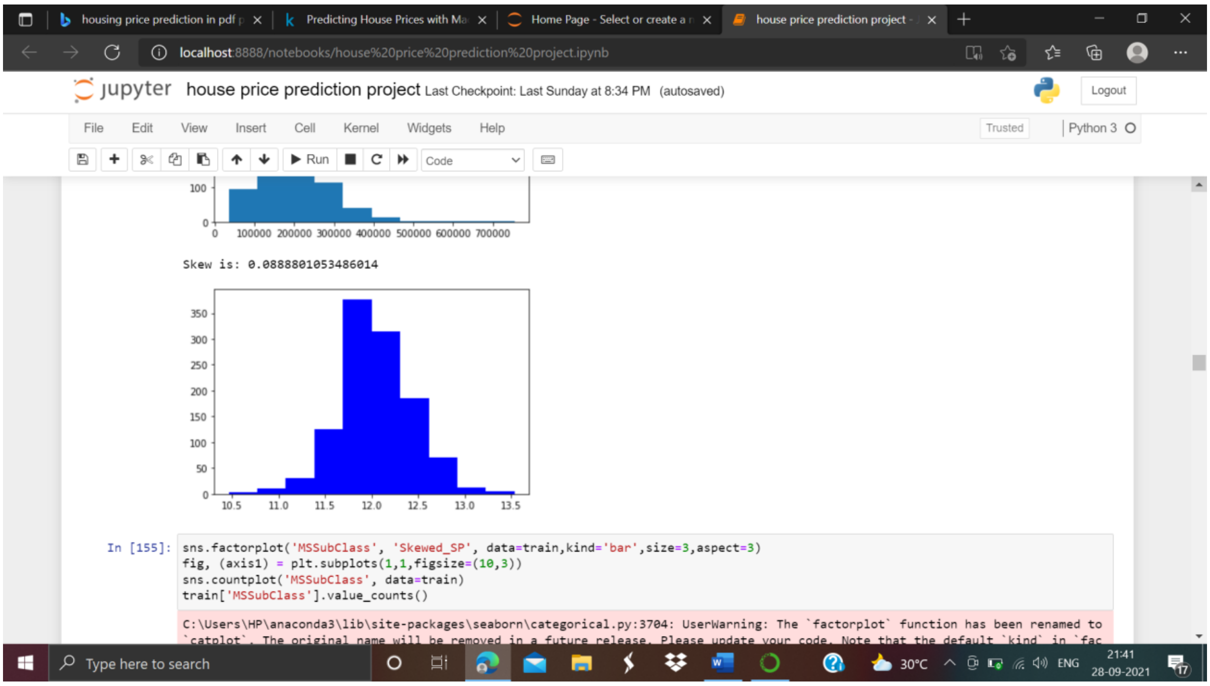
| | Id | MSSubClass | MSZoning | LotFrontage | LotArea | Street | LotShape | LandContour | Utilities | LotConfig | ... | WoodDeckSF | OpenPorchSF | EnclosedPorch |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 337 | 20 | 2.0 | 86.000000 | 14157 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | ... | 178 | 51 | 0 |
| 1 | 1018 | 120 | 2.0 | 66.425101 | 5814 | 1.0 | 0.0 | 3.0 | 0.0 | 1.0 | ... | 63 | 0 | 0 |
| 2 | 929 | 20 | 2.0 | 66.425101 | 11838 | 1.0 | 3.0 | 3.0 | 0.0 | 4.0 | ... | 202 | 151 | 0 |
| 3 | 1148 | 70 | 2.0 | 75.000000 | 12000 | 1.0 | 3.0 | 0.0 | 0.0 | 4.0 | ... | 0 | 0 | 0 |
| 4 | 1227 | 60 | 2.0 | 86.000000 | 14598 | 1.0 | 0.0 | 3.0 | 0.0 | 1.0 | ... | 100 | 18 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 287 | 83 | 20 | 2.0 | 78.000000 | 10206 | 1.0 | 3.0 | 3.0 | 0.0 | 4.0 | ... | 144 | 99 | 0 |
| 288 | 1048 | 20 | 2.0 | 57.000000 | 9245 | 1.0 | 1.0 | 3.0 | 0.0 | 4.0 | ... | 0 | 0 | 0 |
| 289 | 17 | 20 | 2.0 | 66.425101 | 11241 | 1.0 | 0.0 | 3.0 | 0.0 | 1.0 | ... | 0 | 0 | 0 |
| 290 | 523 | 50 | 3.0 | 50.000000 | 5000 | 1.0 | 3.0 | 3.0 | 0.0 | 0.0 | ... | 0 | 24 | 36 |
| 291 | 1379 | 160 | 3.0 | 21.000000 | 1953 | 1.0 | 3.0 | 3.0 | 0.0 | 4.0 | ... | 72 | 0 | 0 |

## Fixing "skewed" features:-

Here, we fix all of the skewed data to be more normal so that our models will be more accurate when making predictions.



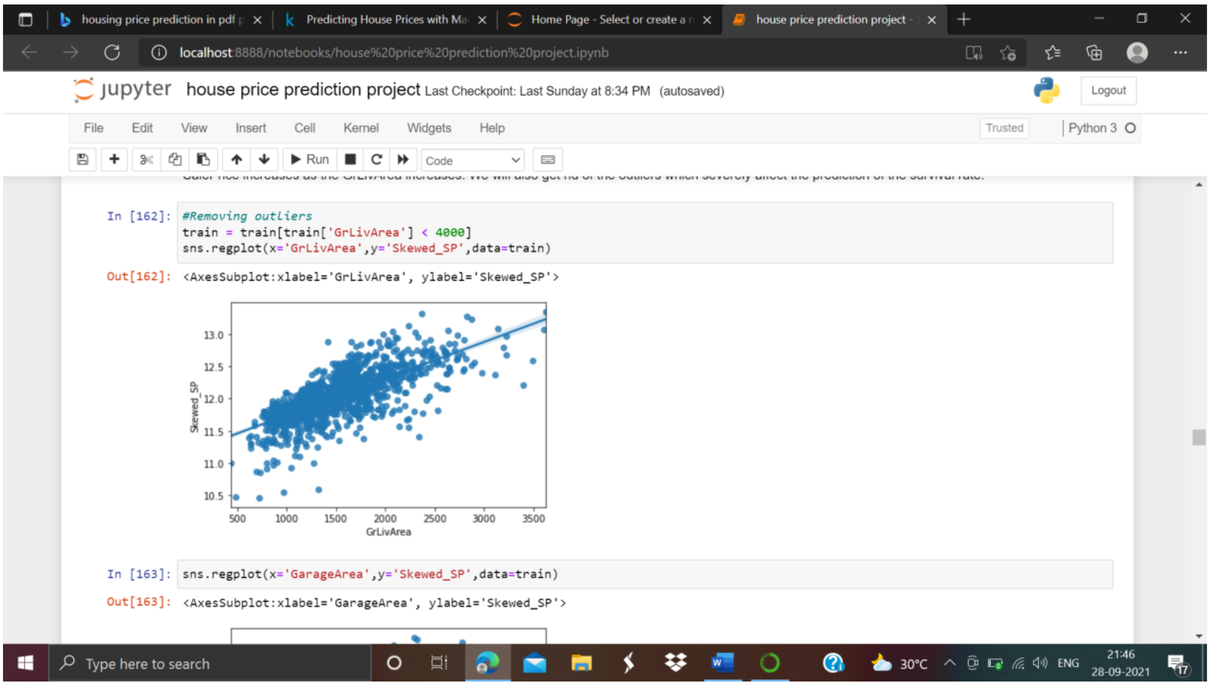

## Here we can see the skewness is approx. 1.9646

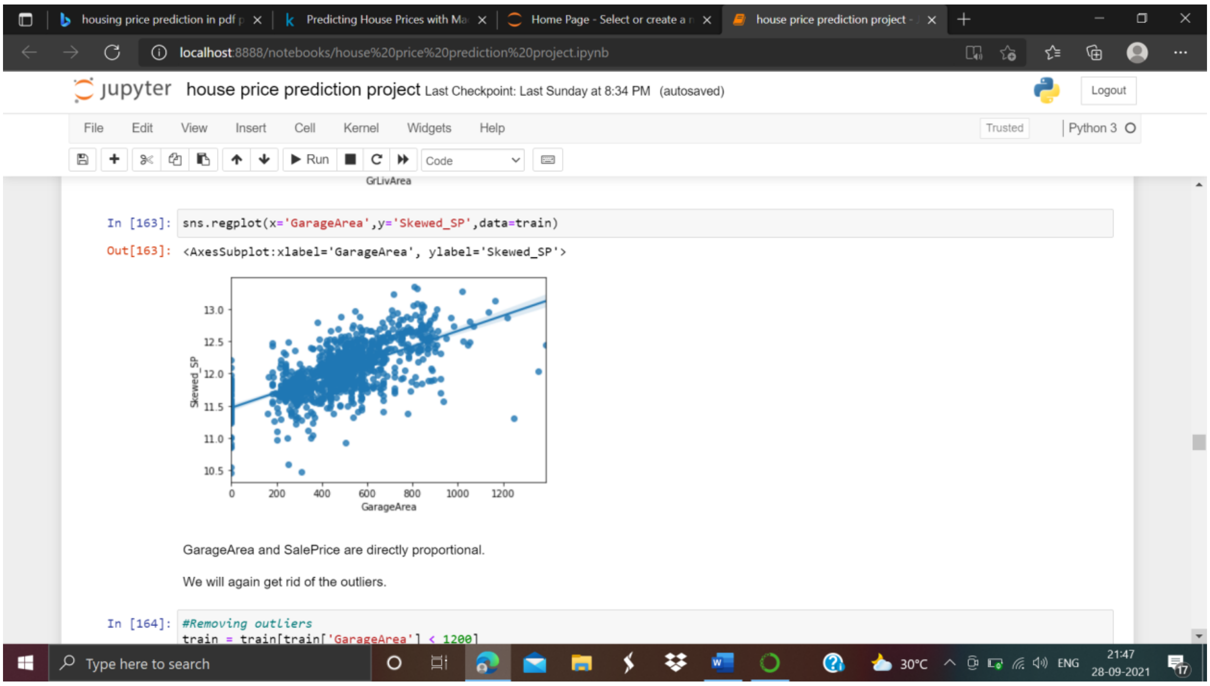


**Here the skewness is approx. 0.0888.**

## Removing outliers:-

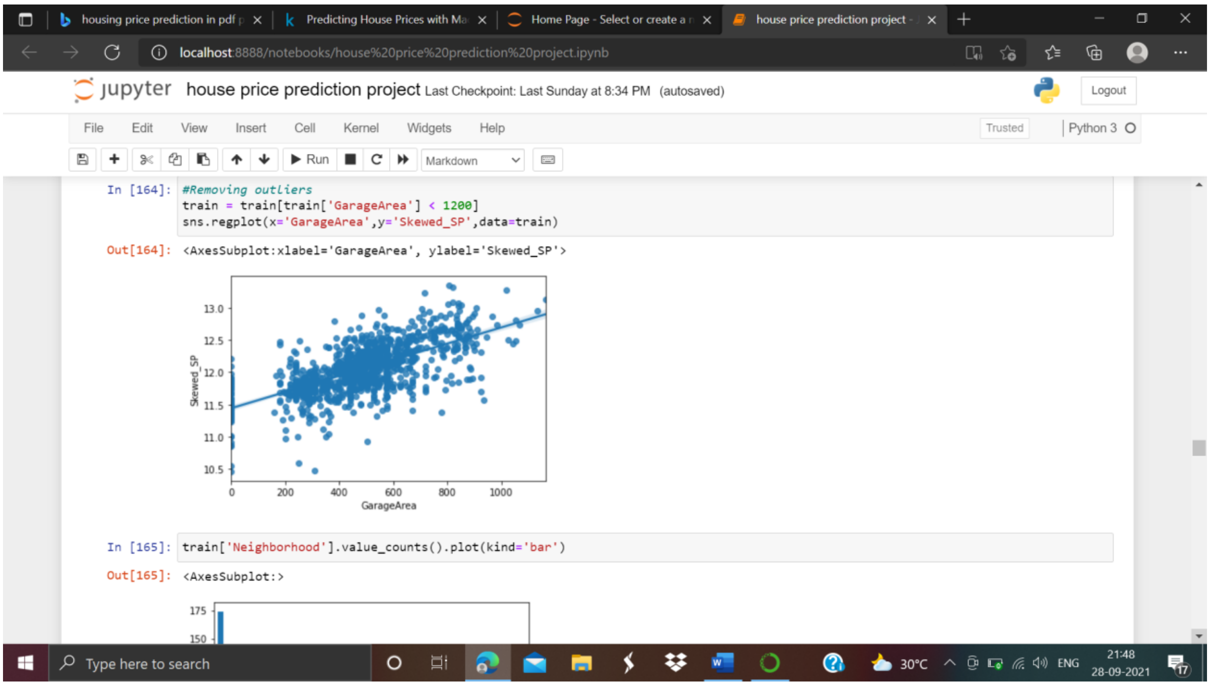
**Now we will move to outliers. We have to remove these outliers from the dataset.**



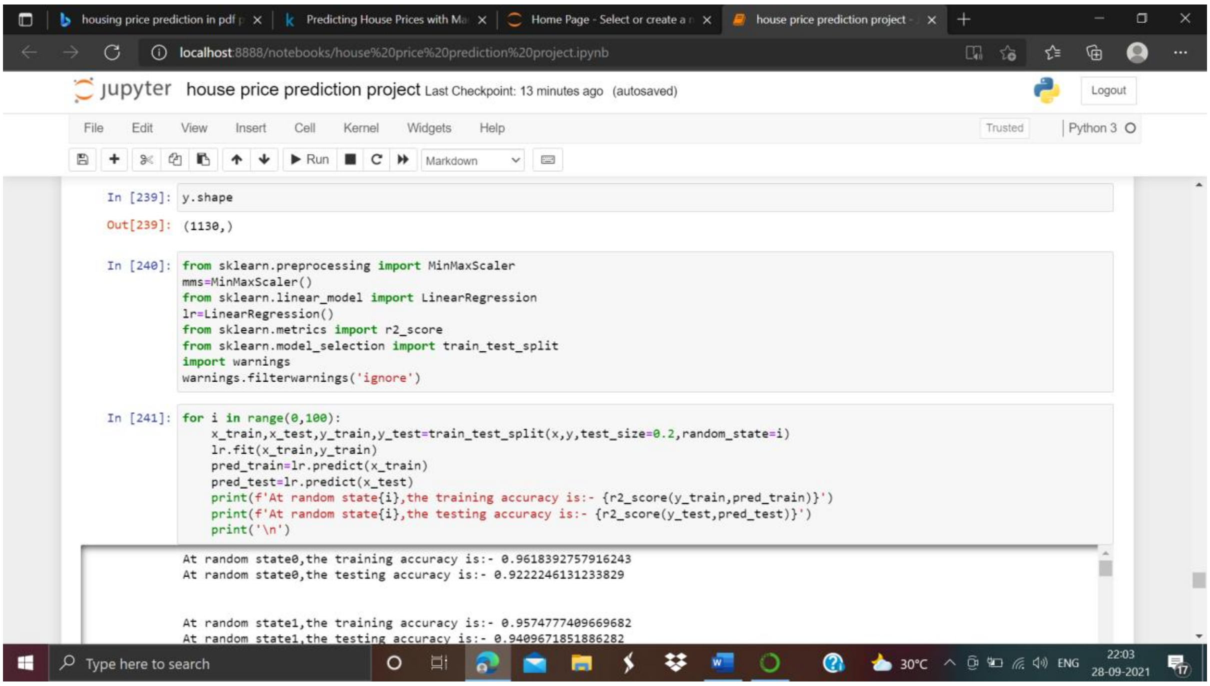

Garage Area and Sale Price are directly proportional.

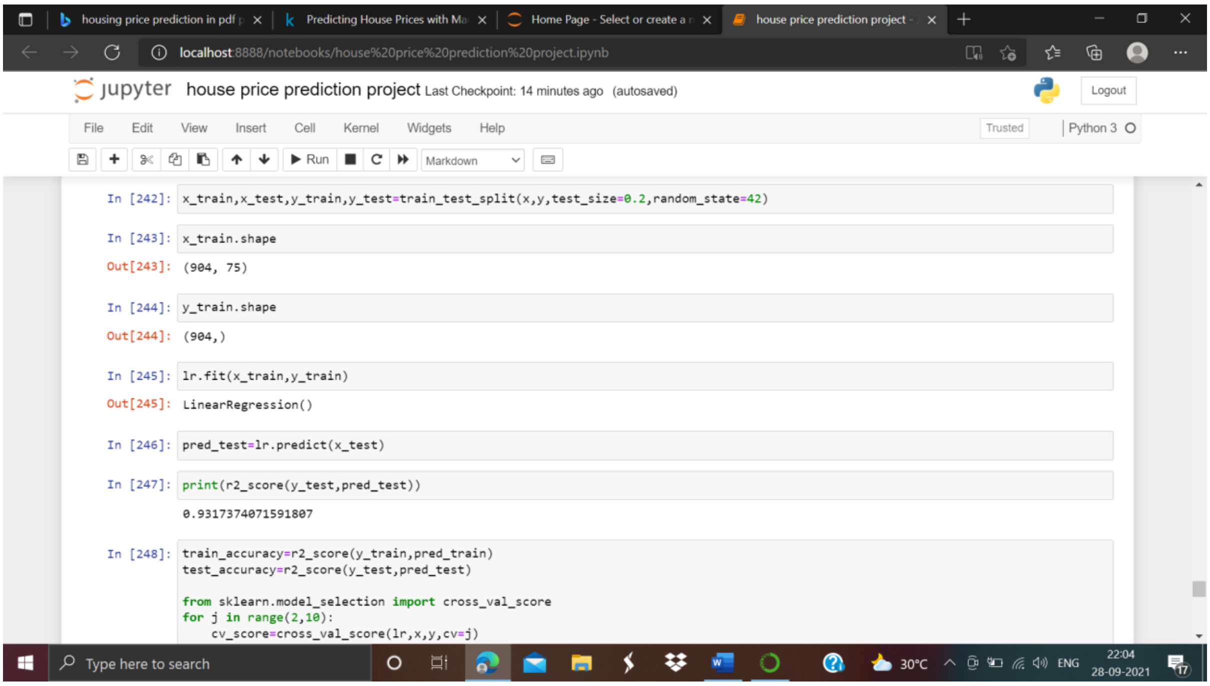
We will again get rid of the outliers.




## Spiliting the data and find the accuracy:-

Now we are going to split the our dataset and find the accuracy for both train and test dataset. We have used min max scaler also.

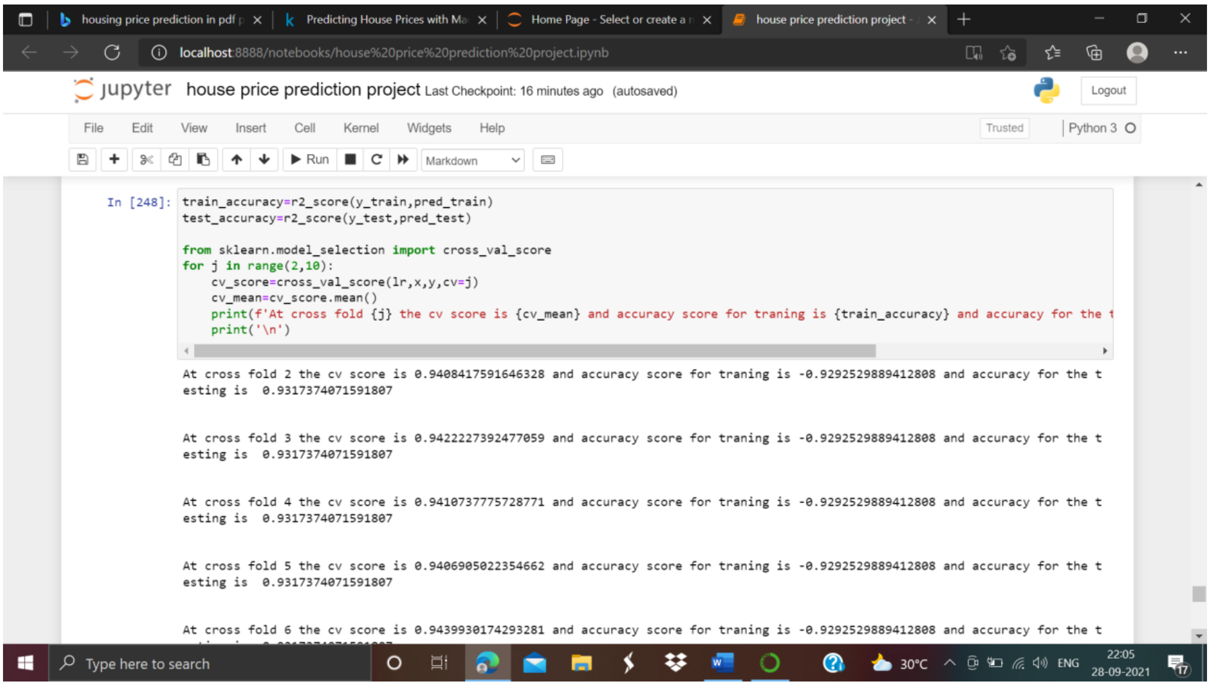


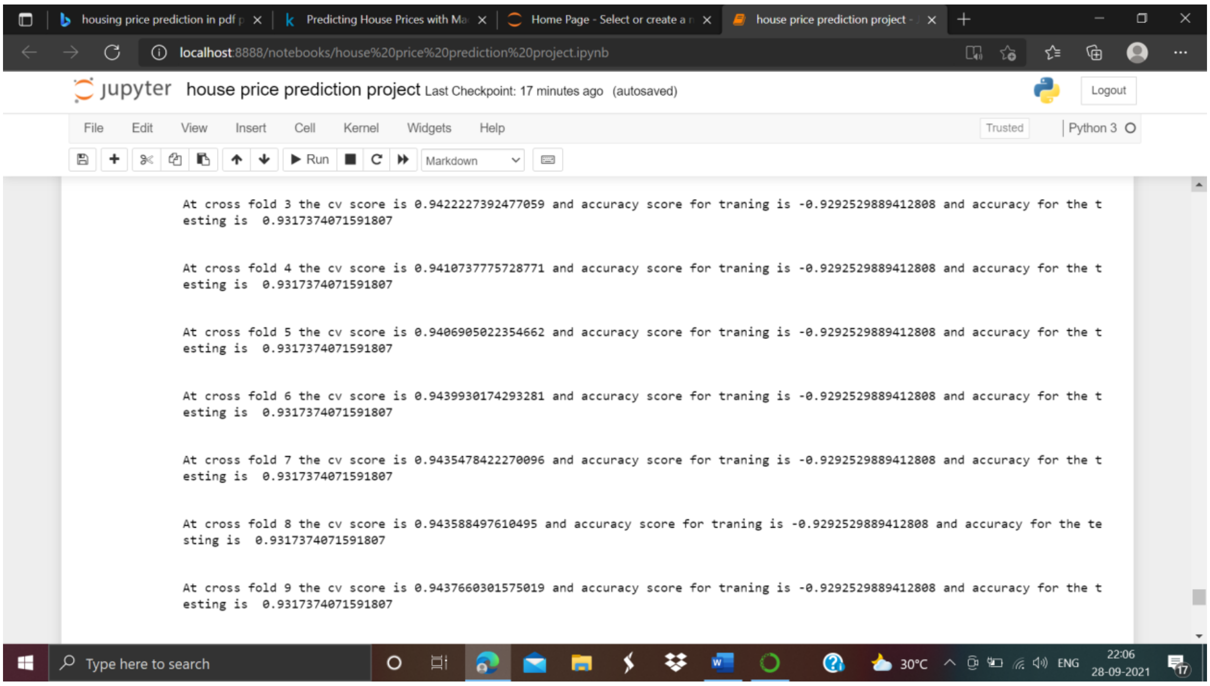



# Modeling and Predictions:-

For our models, we are going to use Random forest regressor, Decision tree regressor and linear regression.

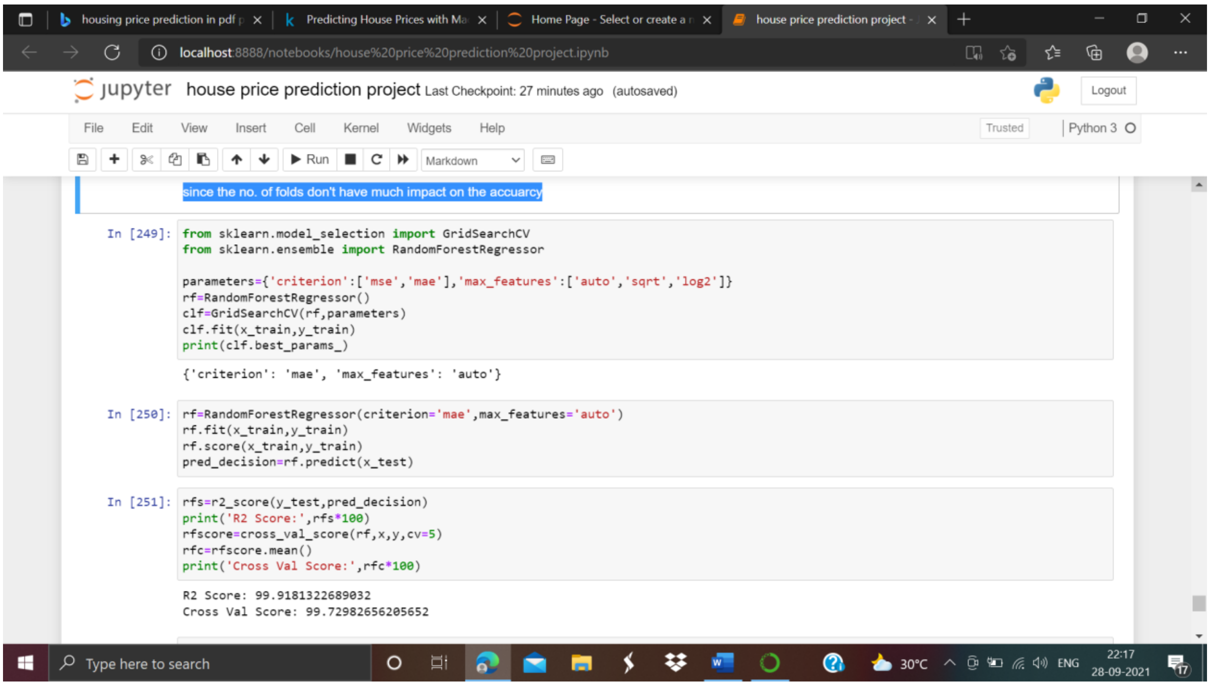
We have also calculated the r2 score and cross validation score.

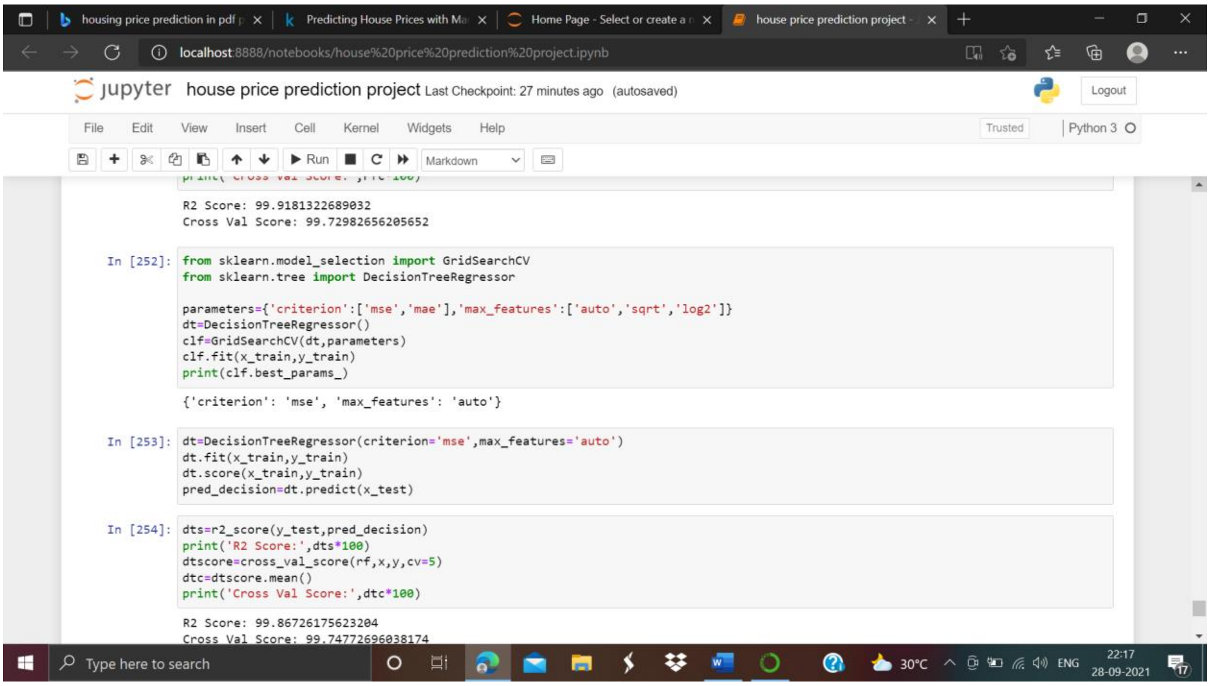



As above we can see that since the no. of folds don't have much impact on the accuracy.

We have to check the  performance of base models by evaluating the cross-validation.

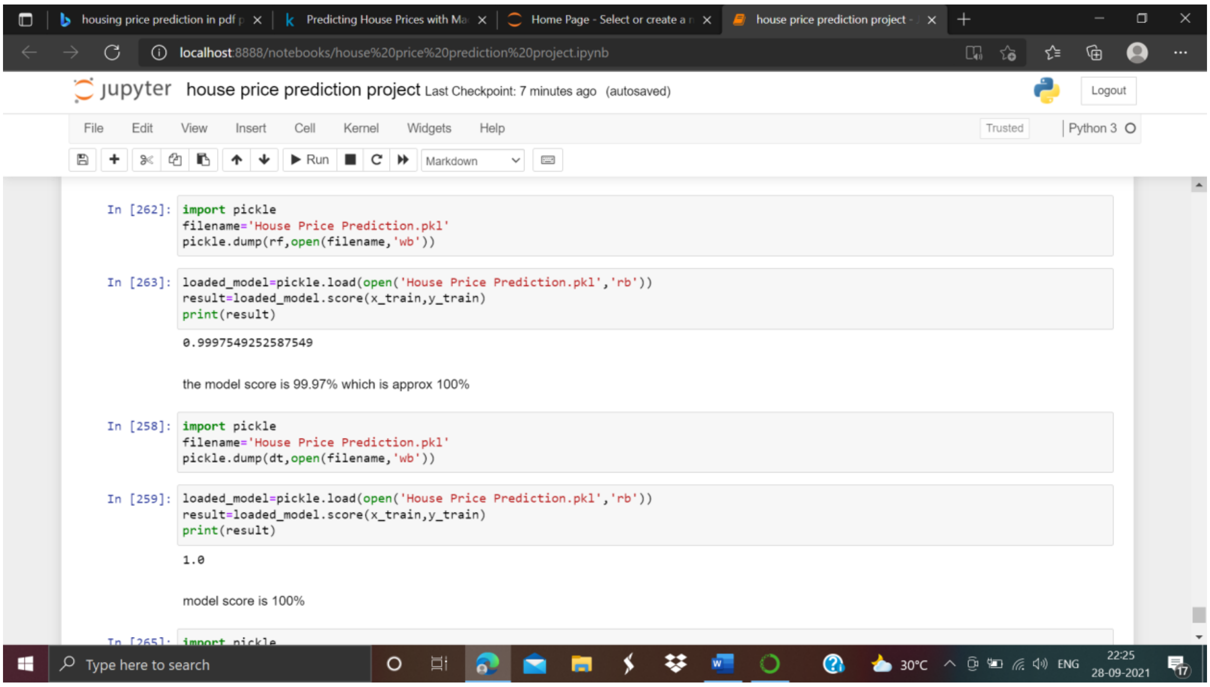



As we can see we have applied different -different regression techniques.

## Saving the model:-

Let’s we will save the our model with file name “House price prediction” in .pkl  format. As we can see below:-




## Conclusion:-

As we can see the our train and test data are balanced Decision tree Regressor score is 100% and Random forest regressor test score also comes with approx 100 %

linear regression model score is 96%.

Hence Decision tree and random forest regressor model's are the our best model.