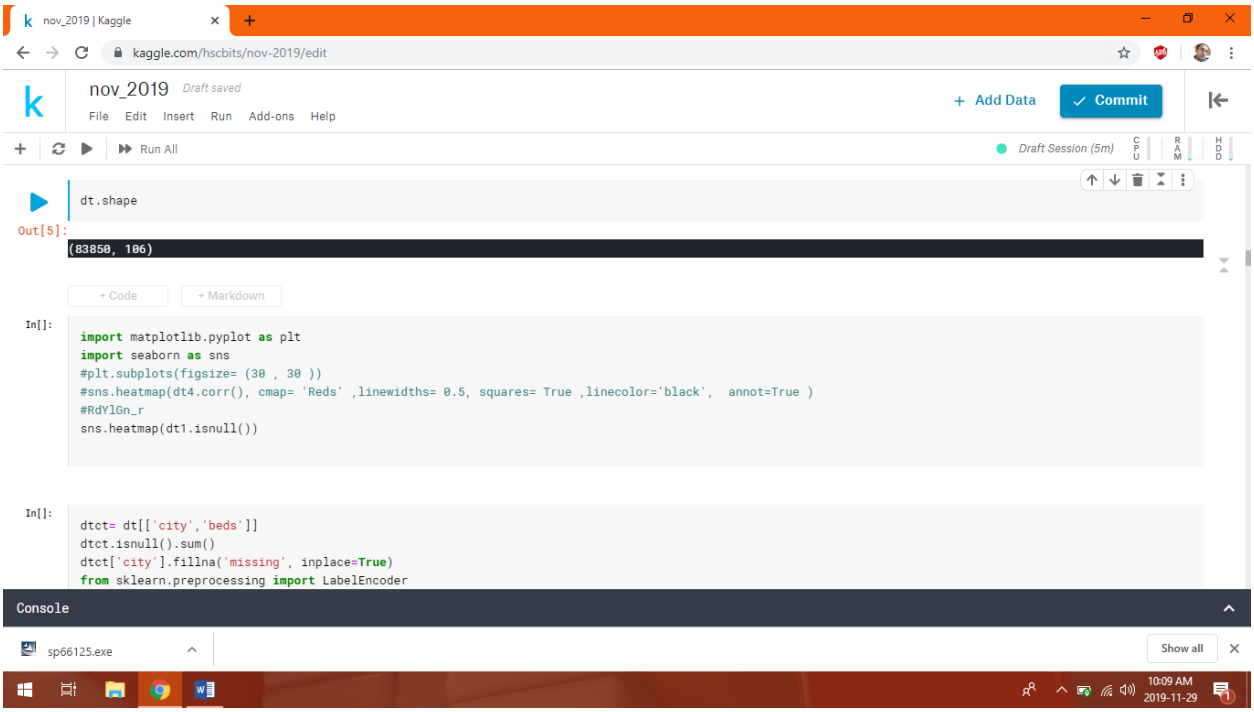
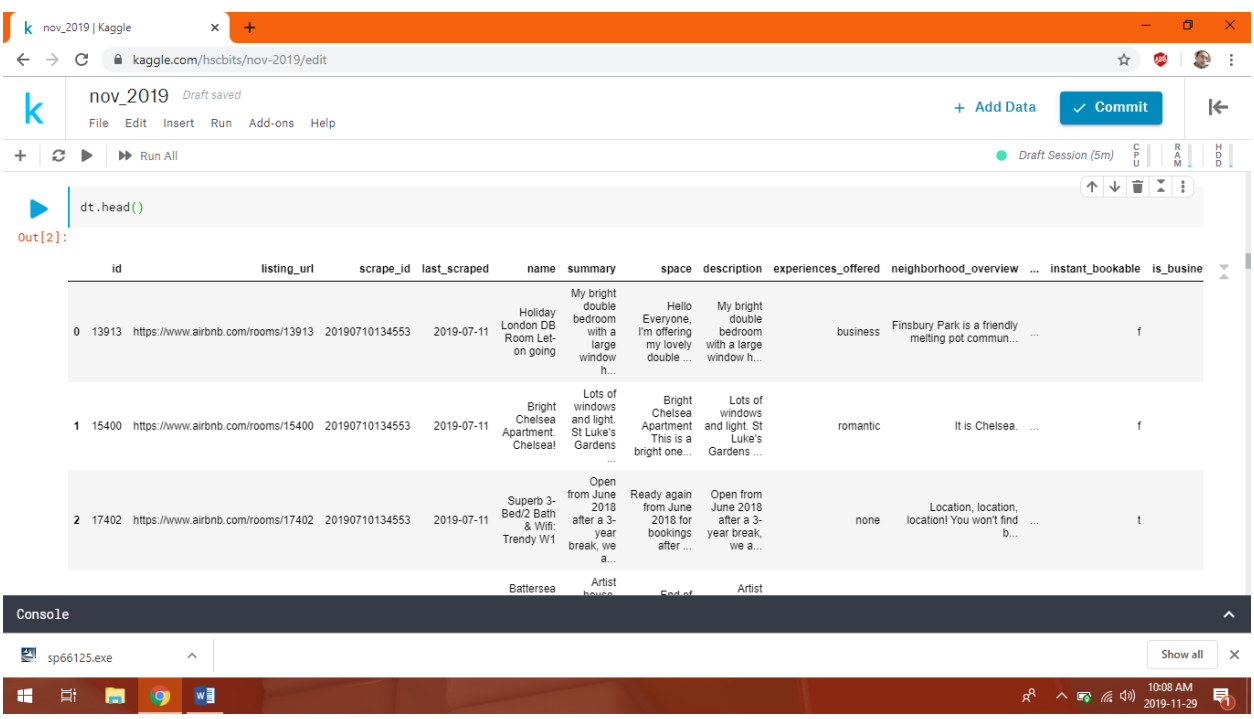
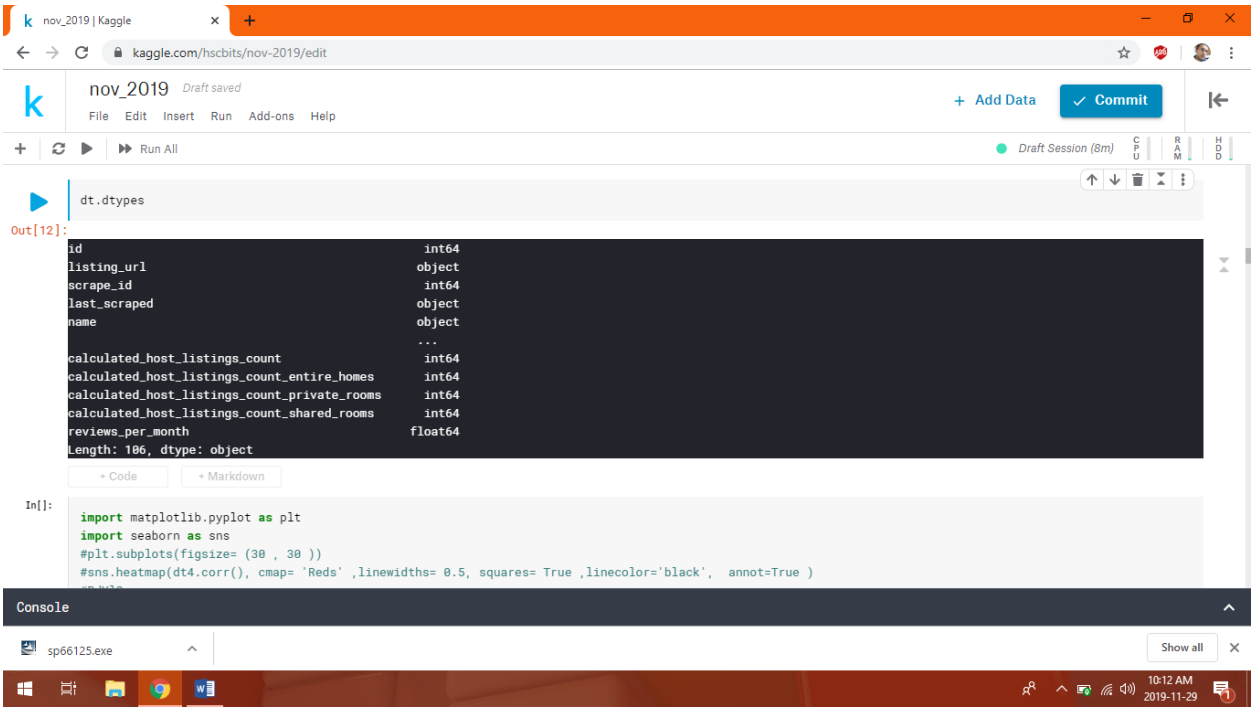
The attached dataset is of 83850 x 106 dimension as can be observed in the following screenshot.



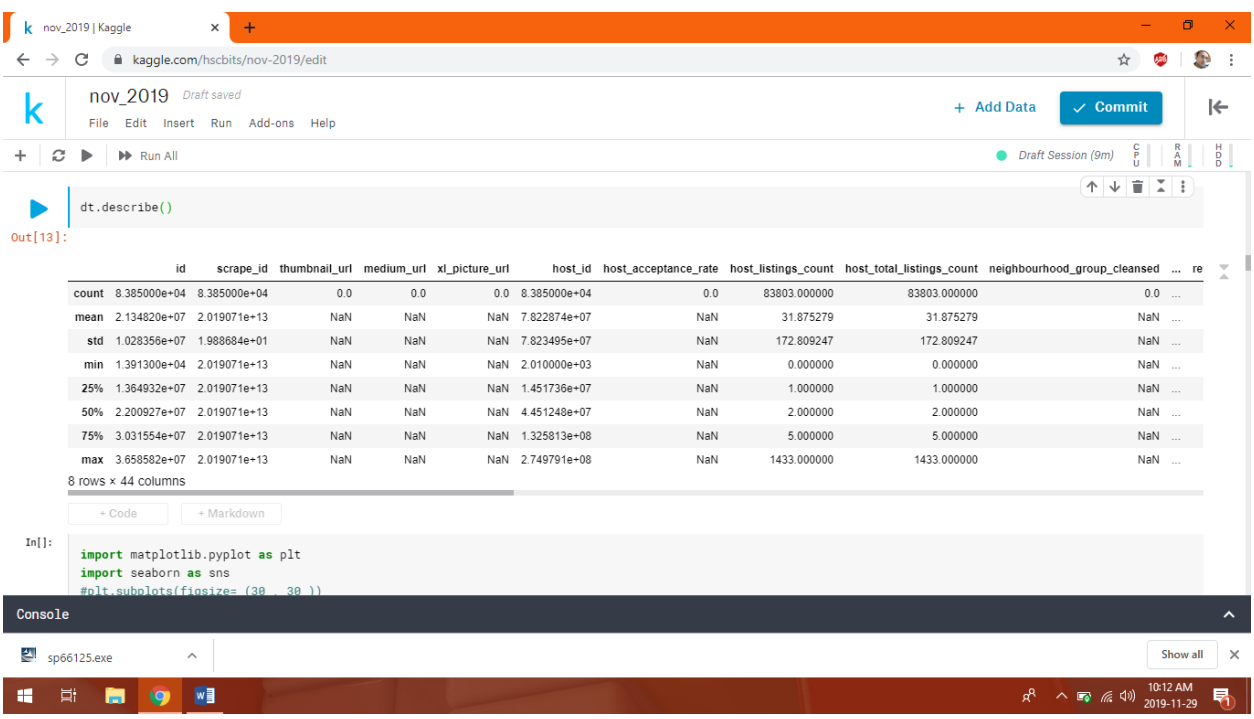
Top rows of the dataset can be observed for a quick idea of the entries. It shows the variety of data present in the dataset and raises curiosity level for further exploration of the data variety in the dataset.



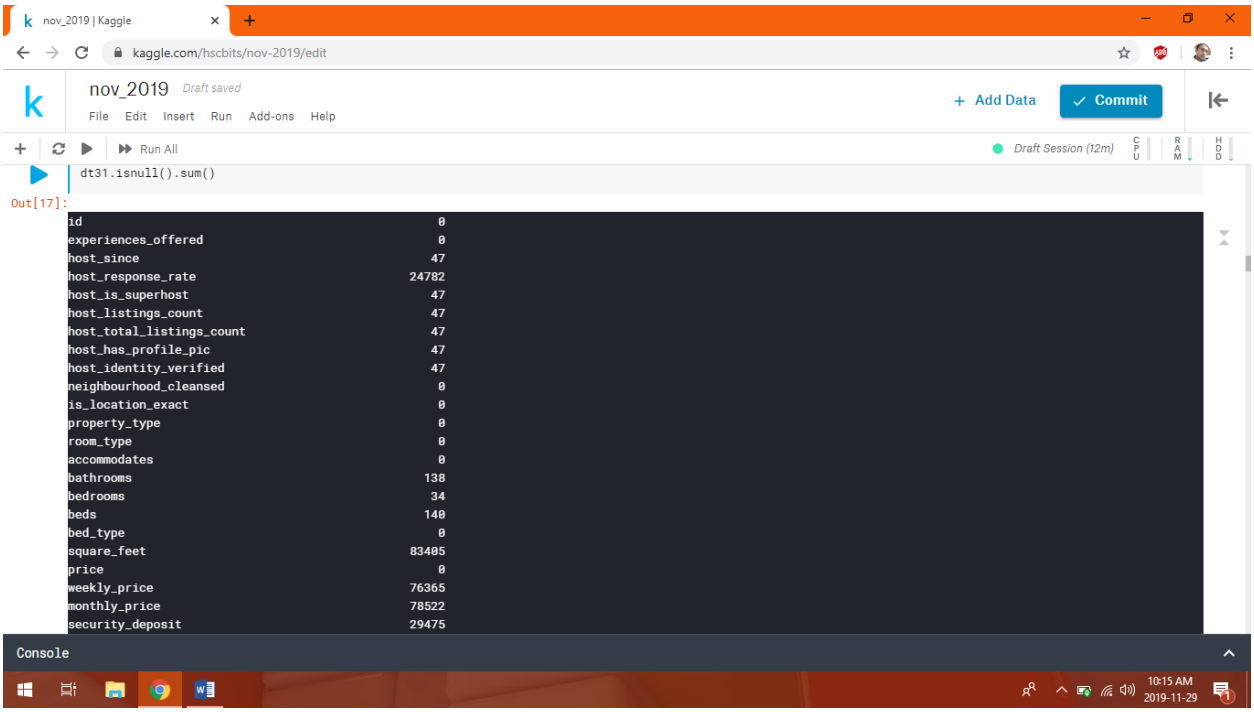
Data types of the column entries can be observed as done in the screenshot displayed below. It shows the presence of numerical as well as categorical data columns.



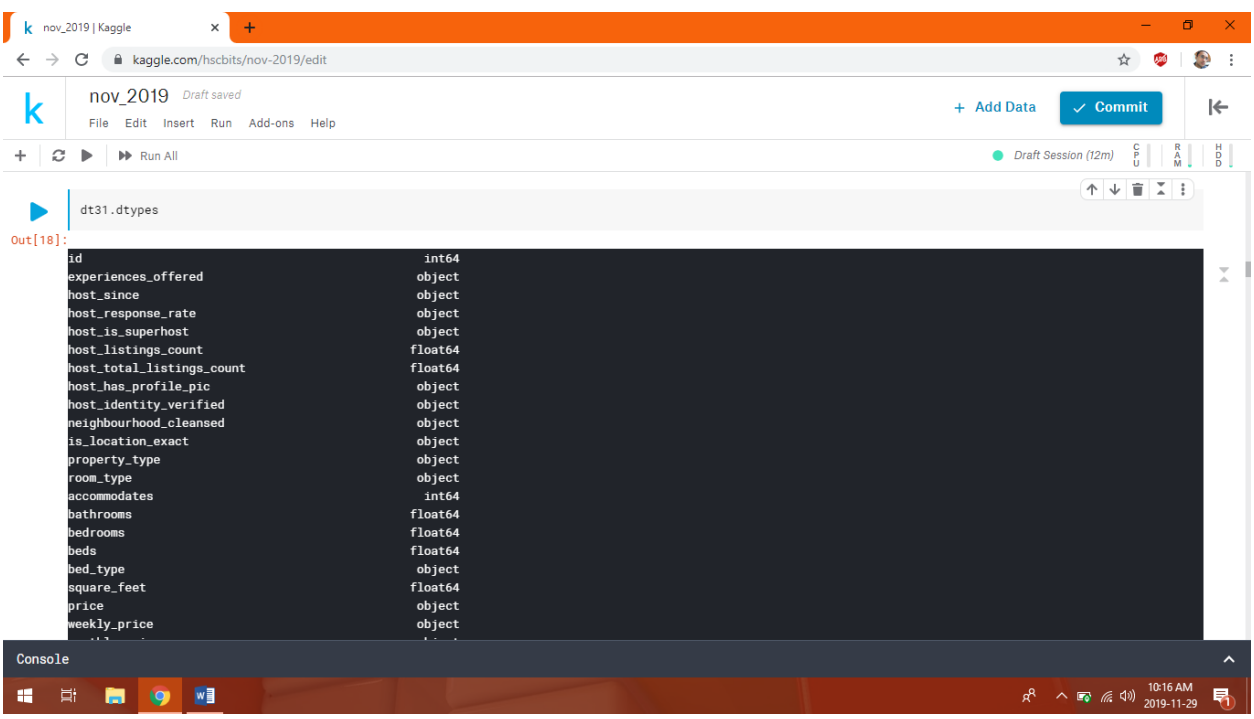
Dataset data description showing the distribution of values can be analysed as below.



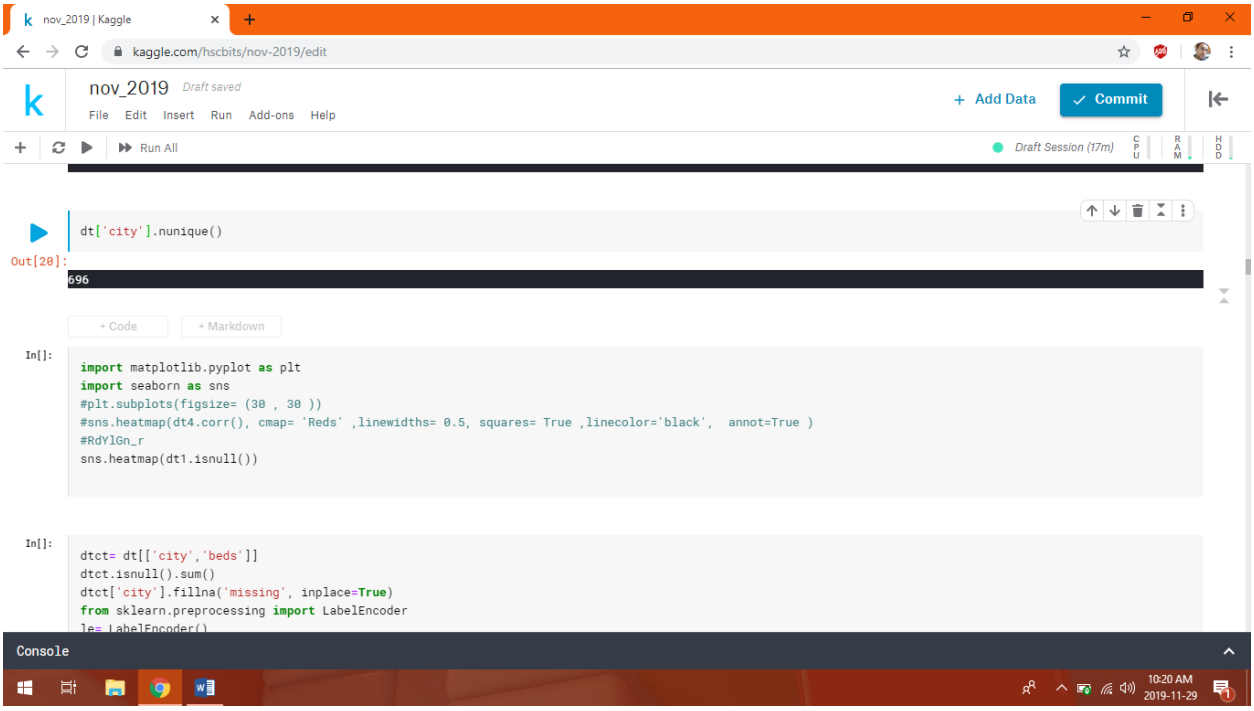
Presence of null values in the dataset checked as below prompting for cleaning and preprocessing.



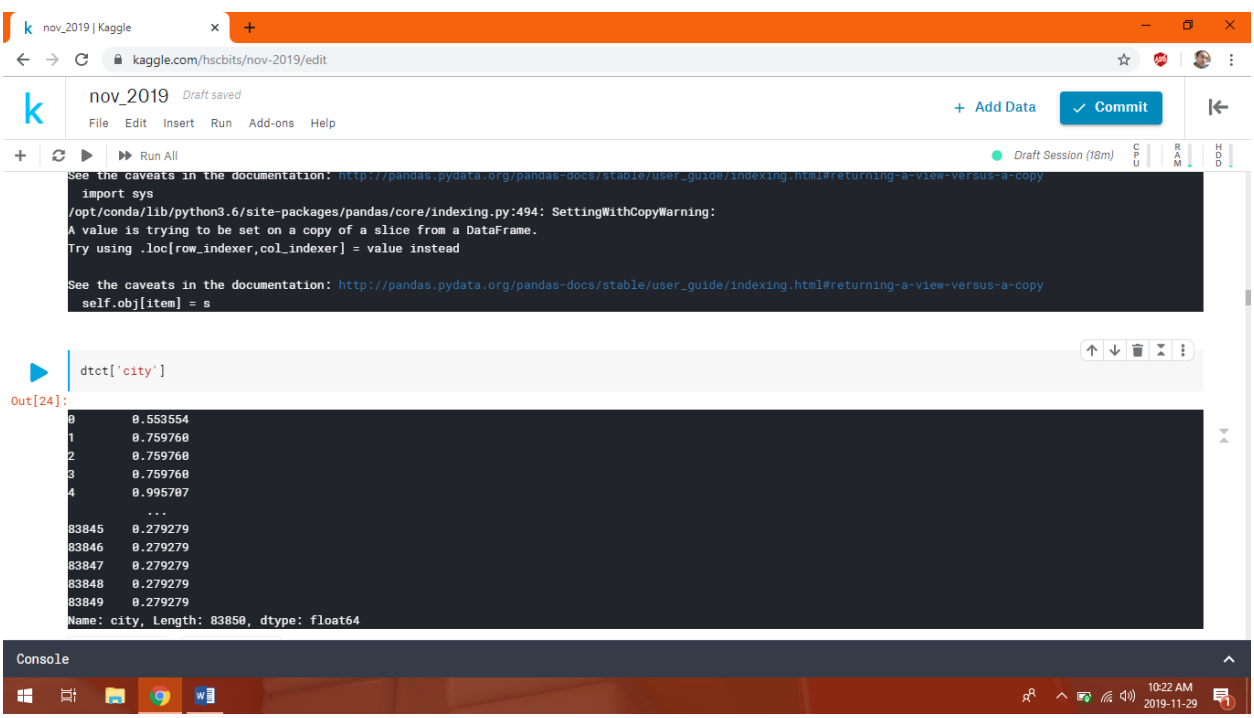
Data types of columns of the dataset we are going further with before cleaning and preprocessing can be observed as follows showing variety of values.



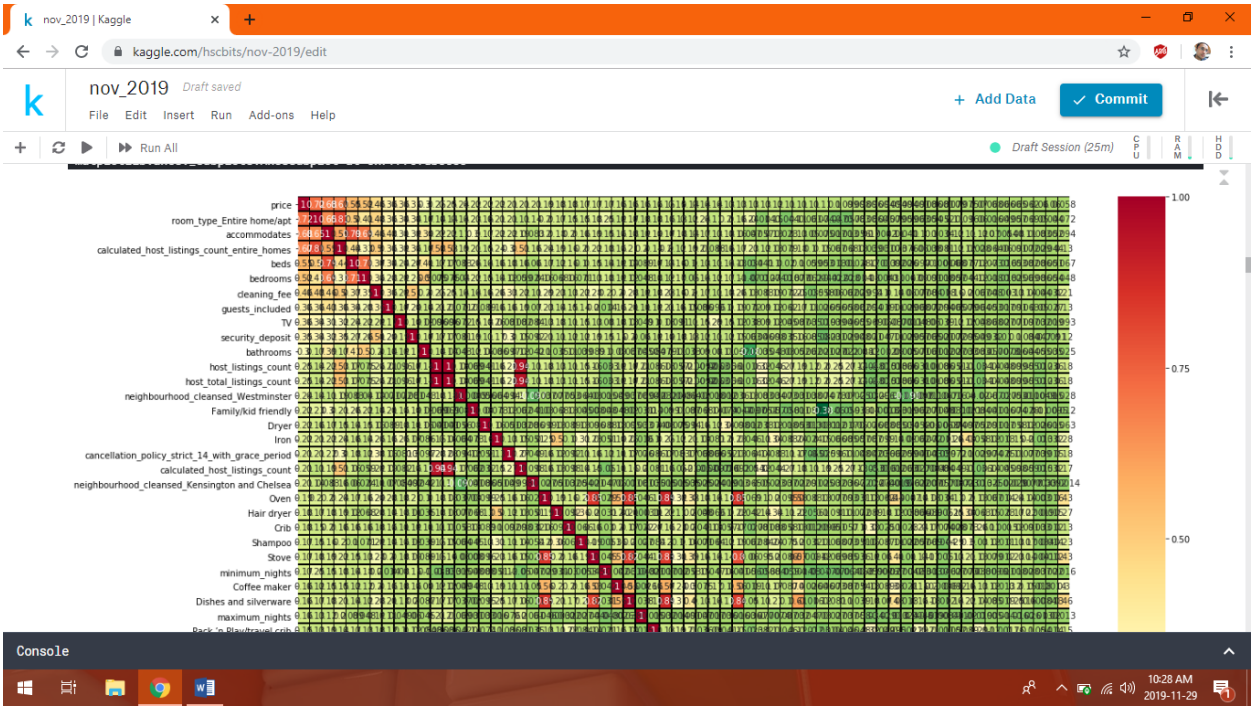
“city” column has 696 categorical values so we need to make it numeric for regression analysis.



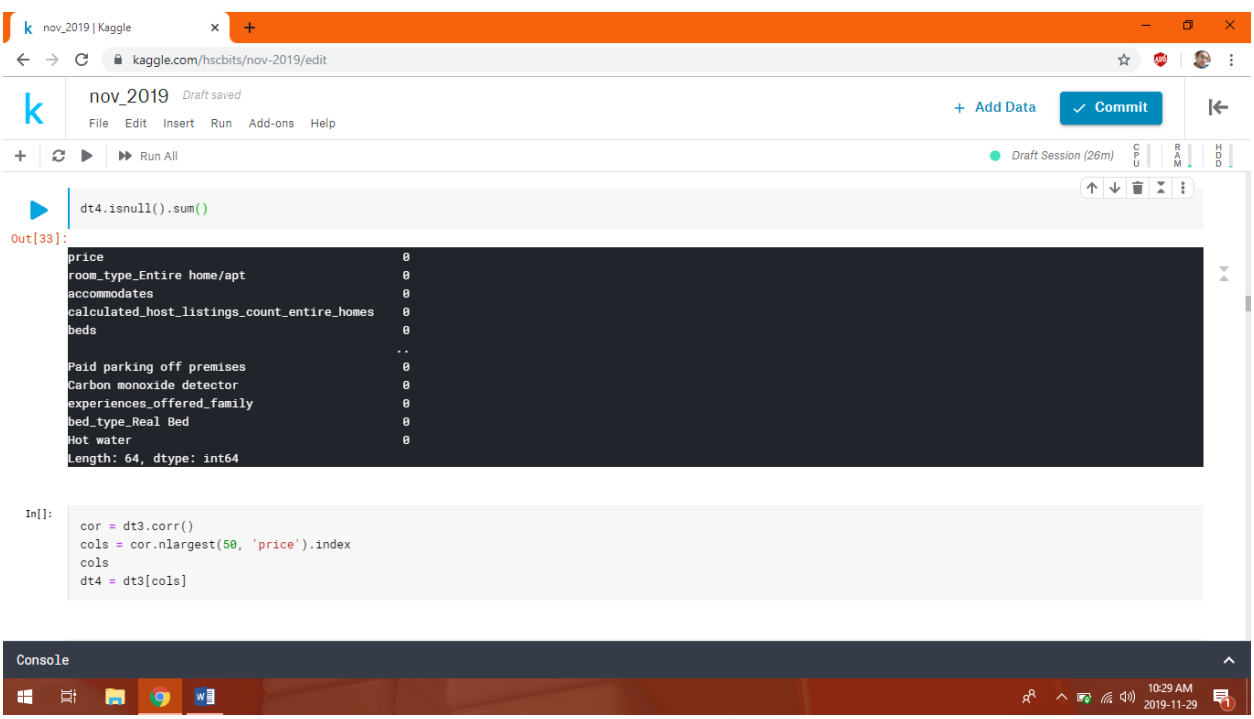
“city” column after label encoding can be observed as below for use in tree based algorithms but if we use Euclidean measure based algorithms, this won’t make sense as there is not an explicit ordinality in the original data as finalised in the label encoded data.



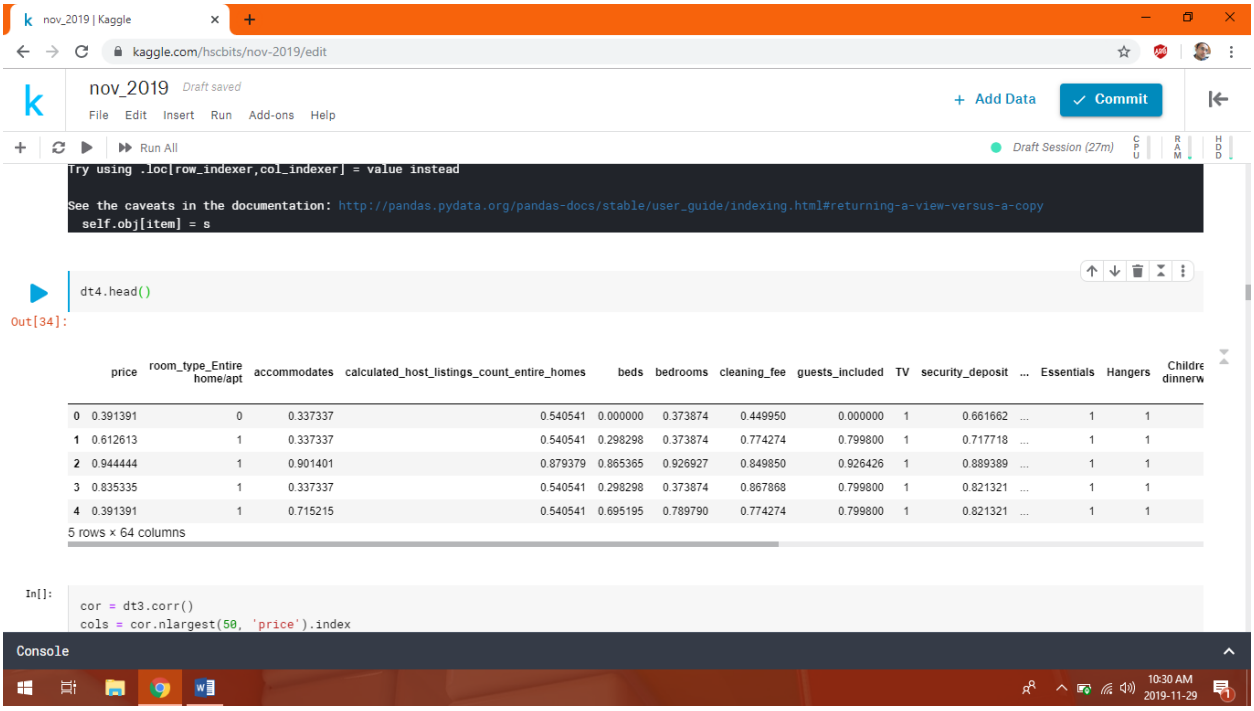
Selection of the top most price influencing features from the dataset. Top 150 features don't contain the label encoded "city" column so we don't need exclusive final data preparation for Euclidean measure based algorithms as of now, in case we want to go with all the features, we can proceed with dropping city and performing final data preparation separately.



null values treatment removing all null entries can be observed as below.



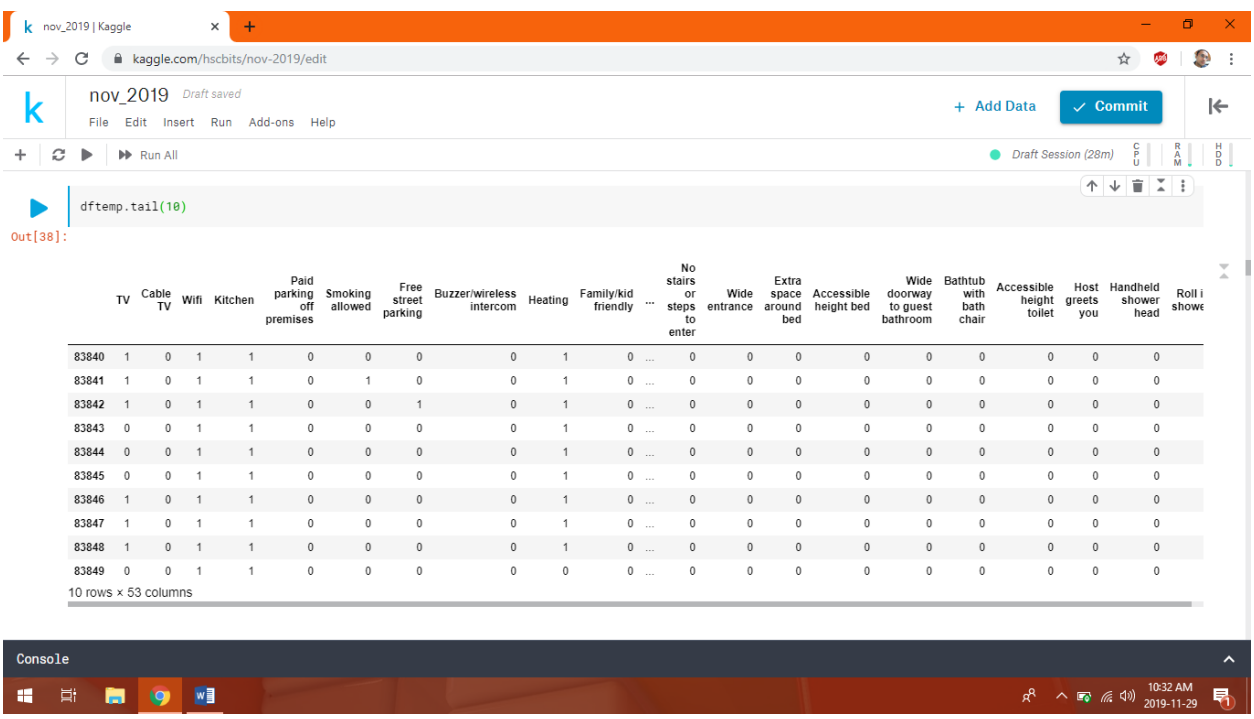
Final data showing top rows can be observed as below.



	price	room_type_Entire home/apt	accommodates	calculated_host_listings_count_entire_homes	beds	bedrooms	cleaning_fee	guests_included	TV	security_deposit	...	Essentials	Hangers	Childre dinnerw
0	0.391391	0	0.337337	0.540541	0.000000	0.373874	0.449950	0.000000	1	0.661662	...	1	1	
1	0.612613	1	0.337337	0.540541	0.298298	0.373874	0.774274	0.799800	1	0.717718	...	1	1	
2	0.944444	1	0.901401	0.879379	0.865365	0.926927	0.849850	0.926426	1	0.889389	...	1	1	
3	0.835335	1	0.337337	0.540541	0.298298	0.373874	0.867868	0.799800	1	0.821321	...	1	1	
4	0.391391	1	0.715215	0.540541	0.695195	0.789790	0.774274	0.799800	1	0.821321	...	1	1	

5 rows × 64 columns

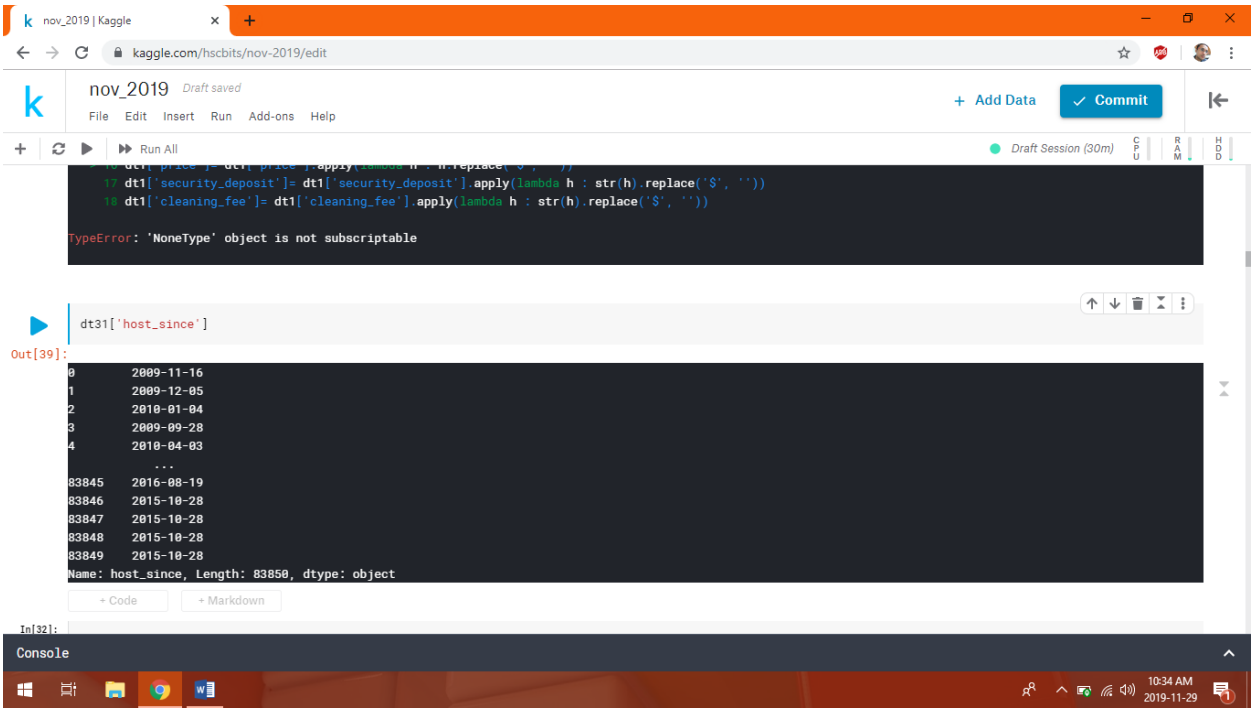
New features created from the amenities column showing the presence of absence of each particular amenity in the houses can be observed as follows.



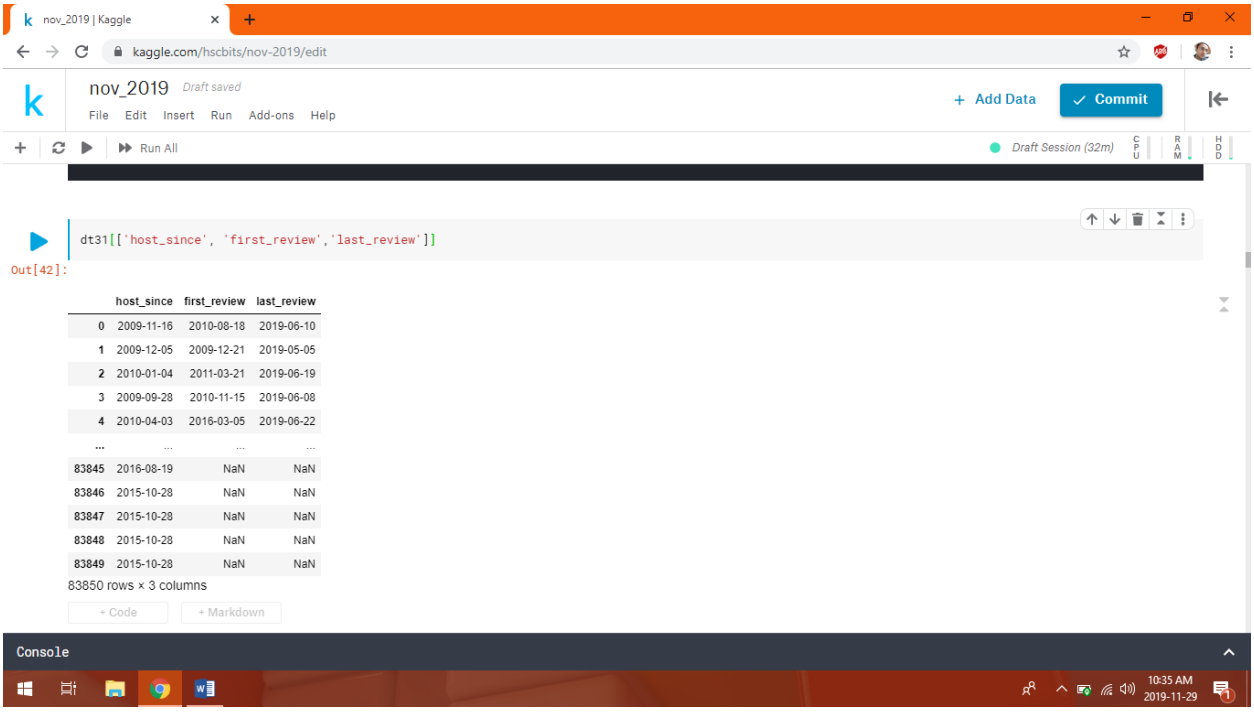
	TV	Cable TV	Wifi	Kitchen	Paid parking off premises	Smoking allowed	Free street parking	Buzzer/wireless intercom	Heating	Family/kid friendly	...	No stairs or steps to enter	Wide entrance	Extra space around bed	Accessible height bed	Wide doorway to guest bathroom	Bathtub with bath chair	Accessible height toilet	Host greets you	Handheld shower head	Roll i showe
83840	1	0	1	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83841	1	0	1	1	0	1	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83842	1	0	1	1	0	0	1	0	1	0	...	0	0	0	0	0	0	0	0	0	
83843	0	0	1	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83844	0	0	1	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83845	0	0	1	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83846	1	0	1	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83847	1	0	1	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83848	1	0	1	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	
83849	0	0	1	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	

10 rows × 53 columns

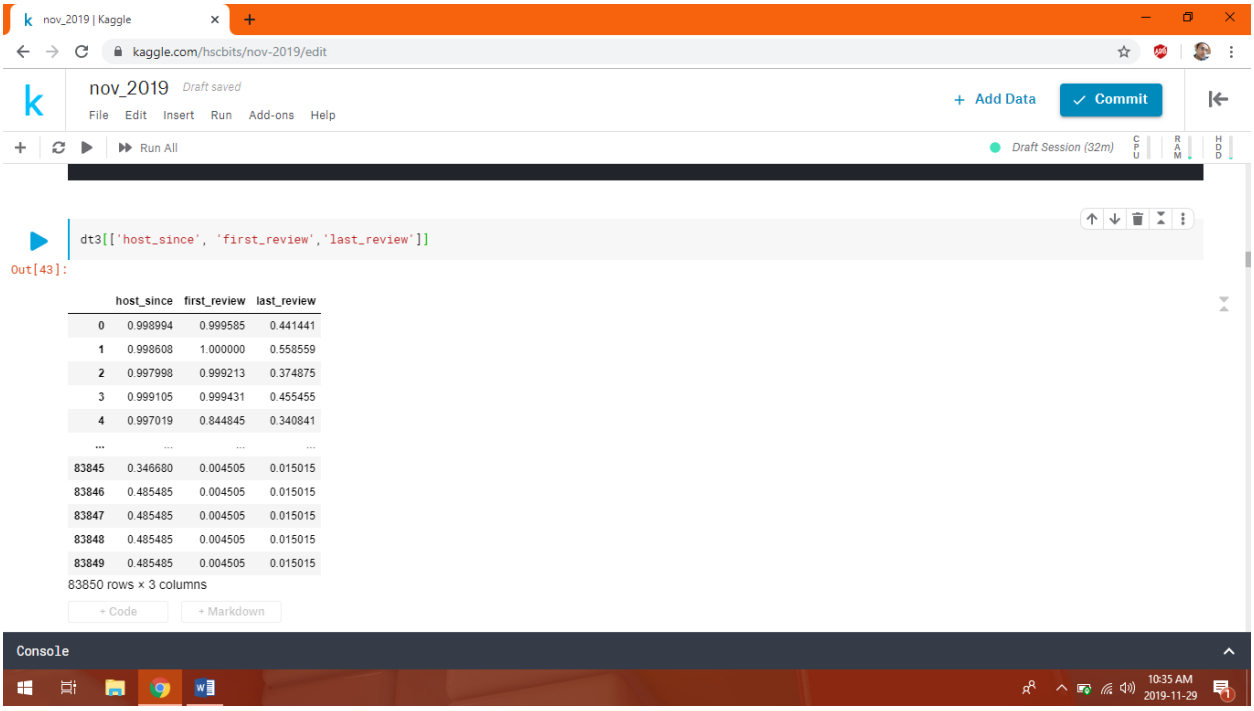
date format columns were converted to duration days till the data collection day as shown below.



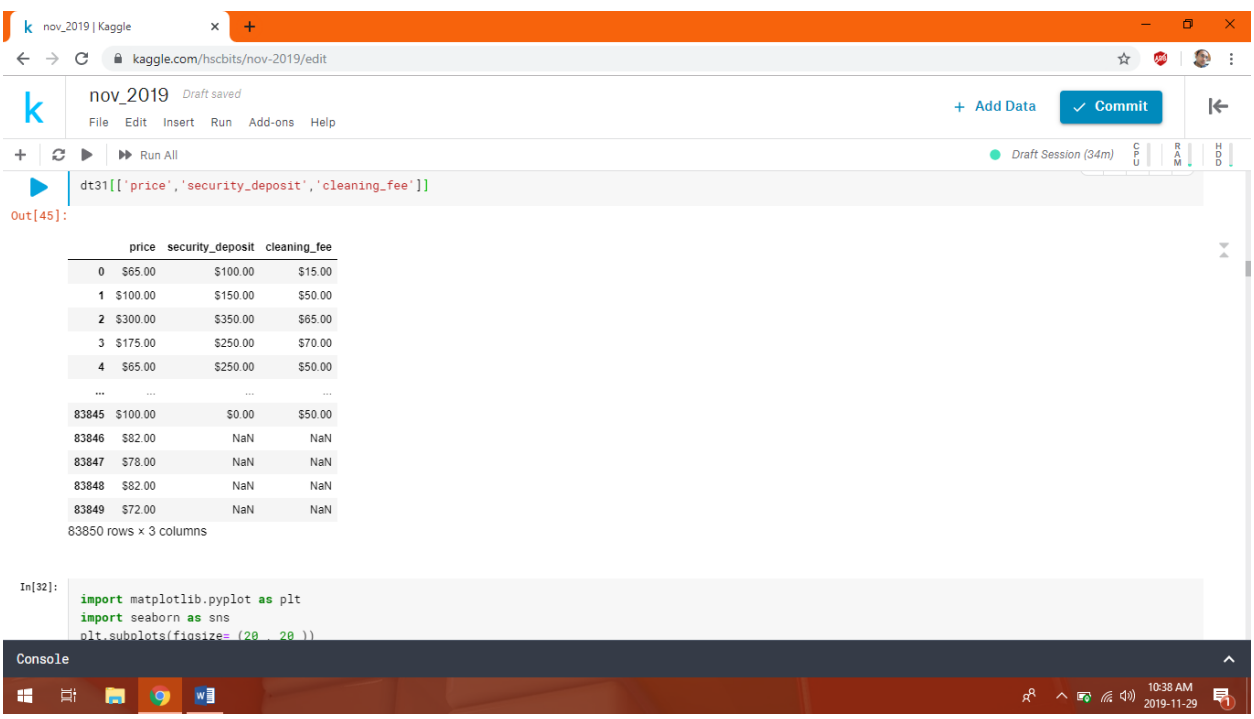
Columns having date values



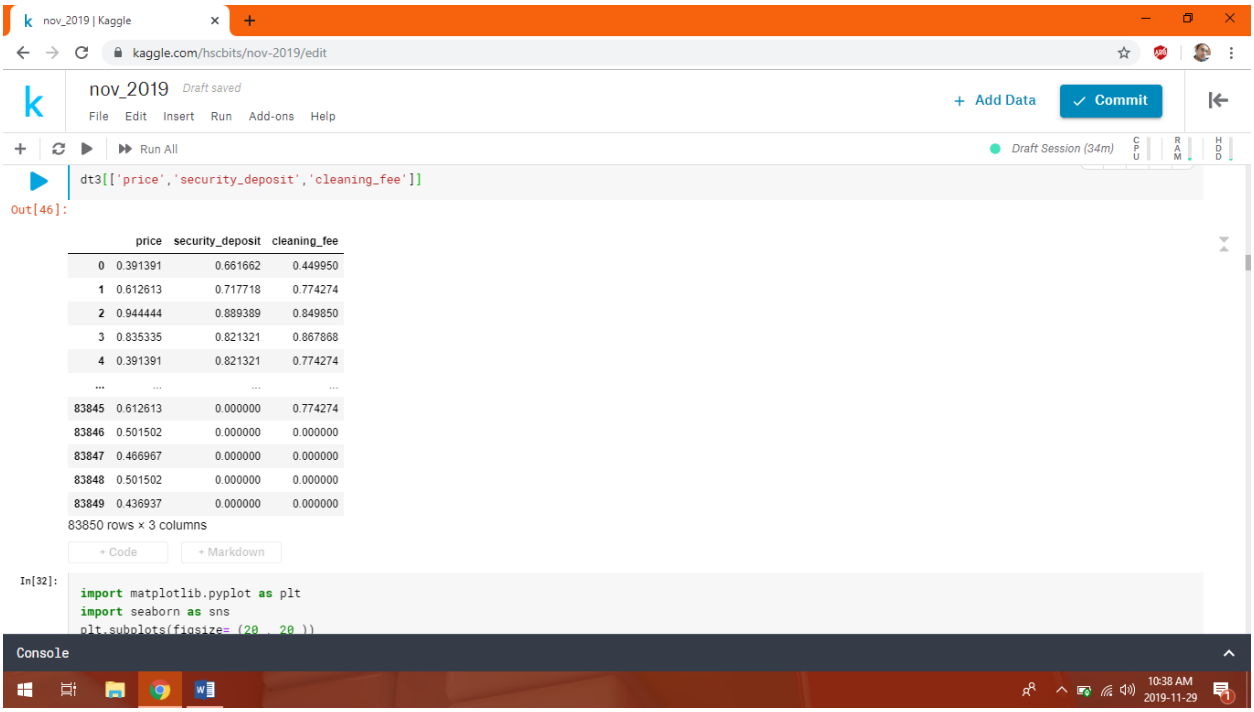
Processed column values making sense for regression



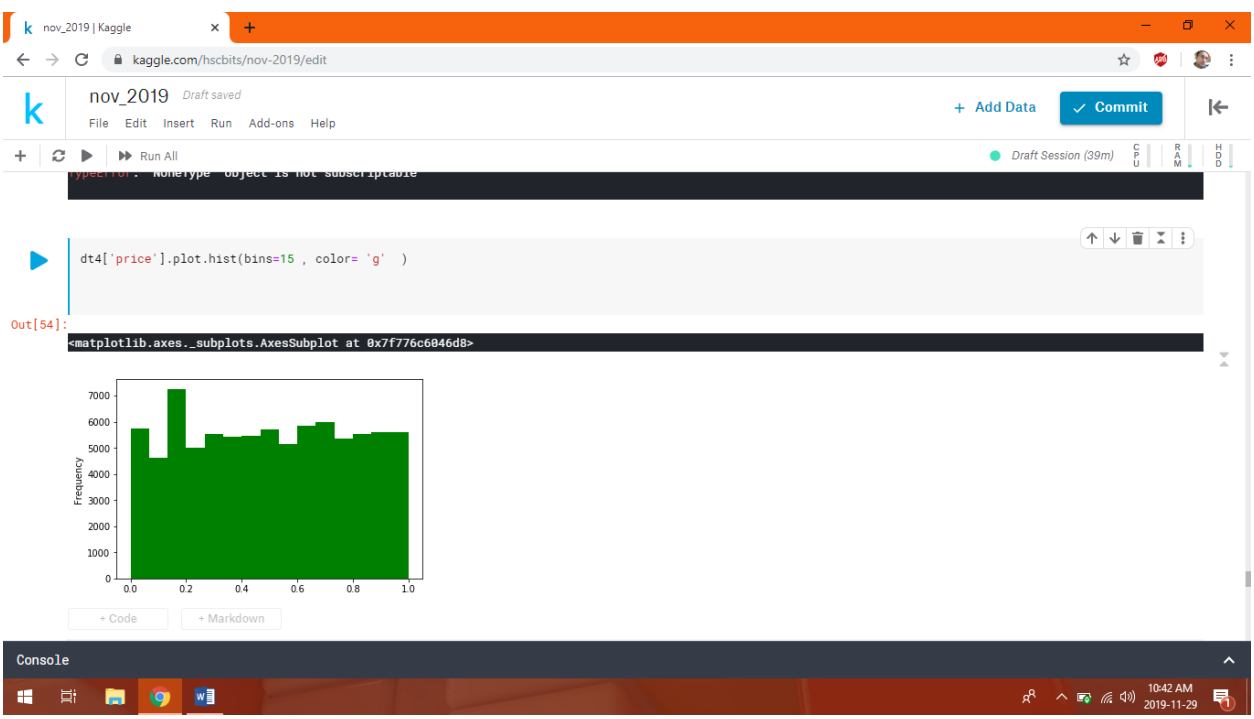
The columns having currency symbols were processed and finalised as shown below.



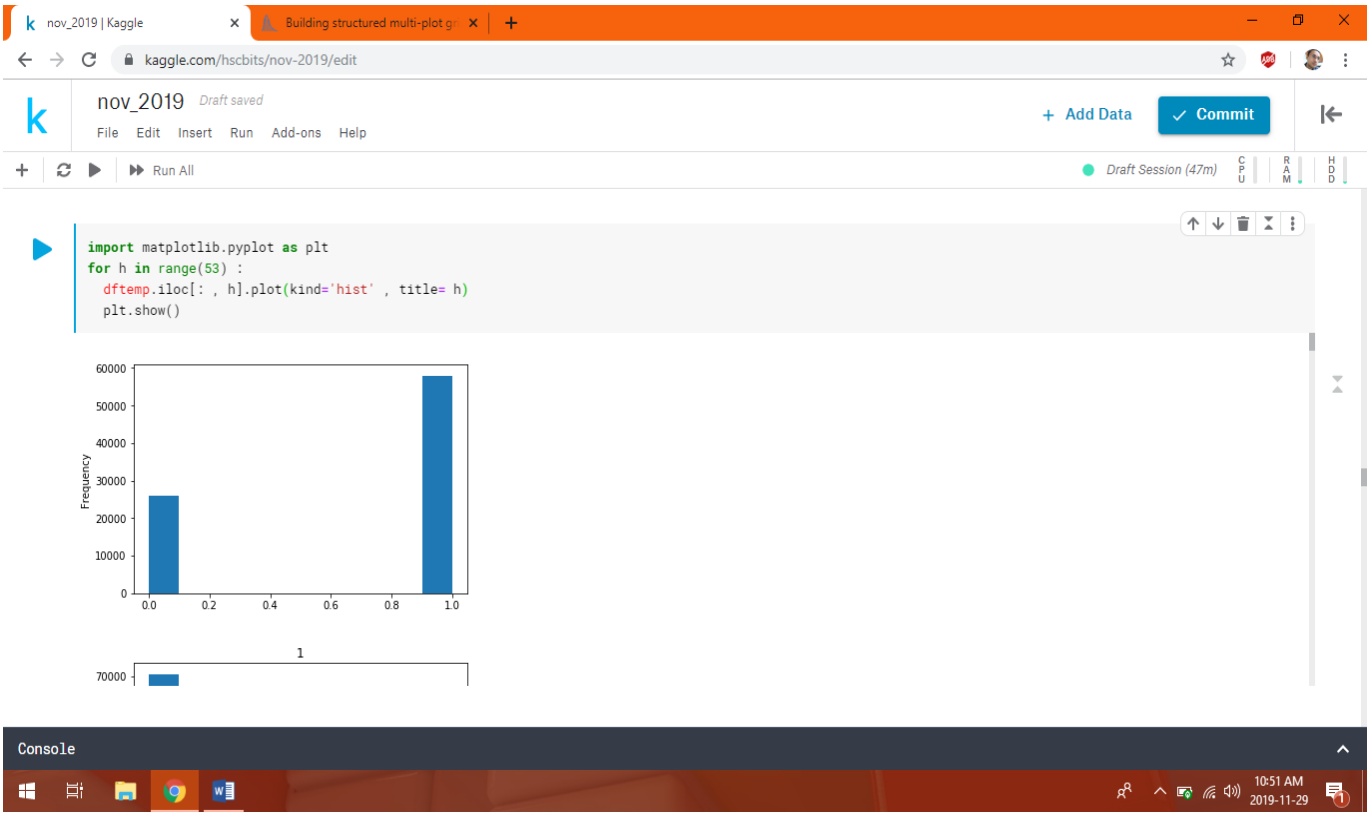
Processed currency columns

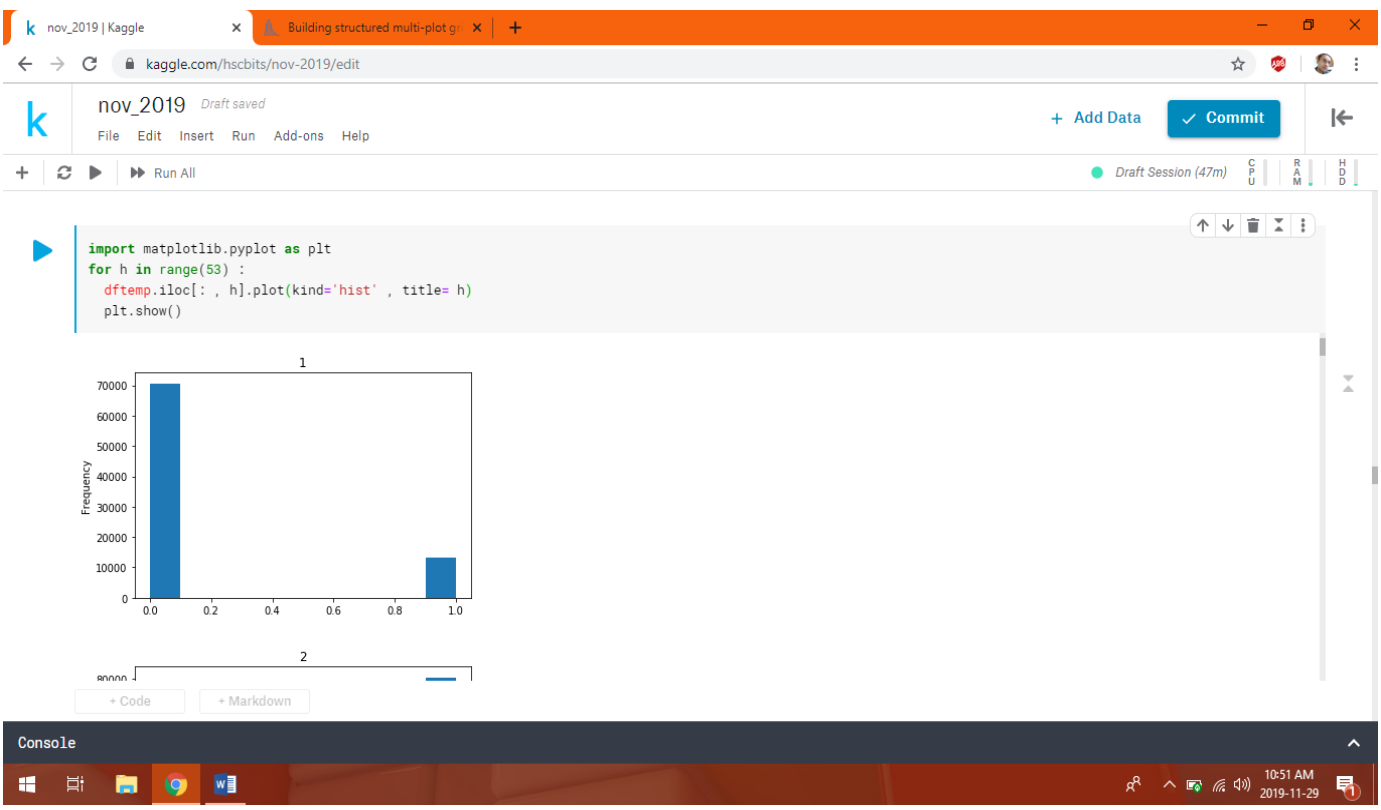


price distribution can be observed as below in the final data to be used for training going forward.



Distribution of the new features created for amenities' availability in the houses is shown graphically below in the attached screenshots.





nov_2019 | Kaggle

kaggle.com/hscbits/nov-2019/edit

nov_2019 *Draft saved*

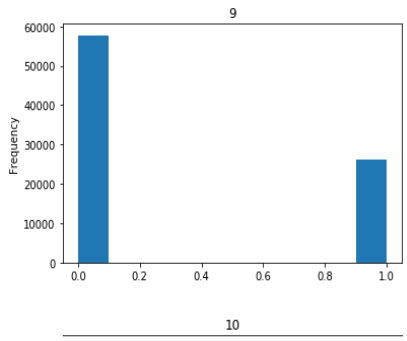
File Edit Insert Run Add-ons Help

+ Add Data ✓ Commit

Run All

Draft Session (47m)

```
import matplotlib.pyplot as plt
for h in range(53) :
  dftemp.iloc[: , h].plot(kind='hist' , title= h)
  plt.show()
```



+ Code + Markdown

Console

10:51 AM
2019-11-29

nov_2019 | Kaggle

kaggle.com/hscbits/nov-2019/edit

nov_2019 *Draft saved*

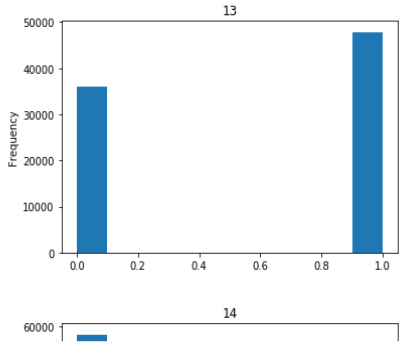
File Edit Insert Run Add-ons Help

+ Add Data ✓ Commit

Run All

Draft Session (48m)

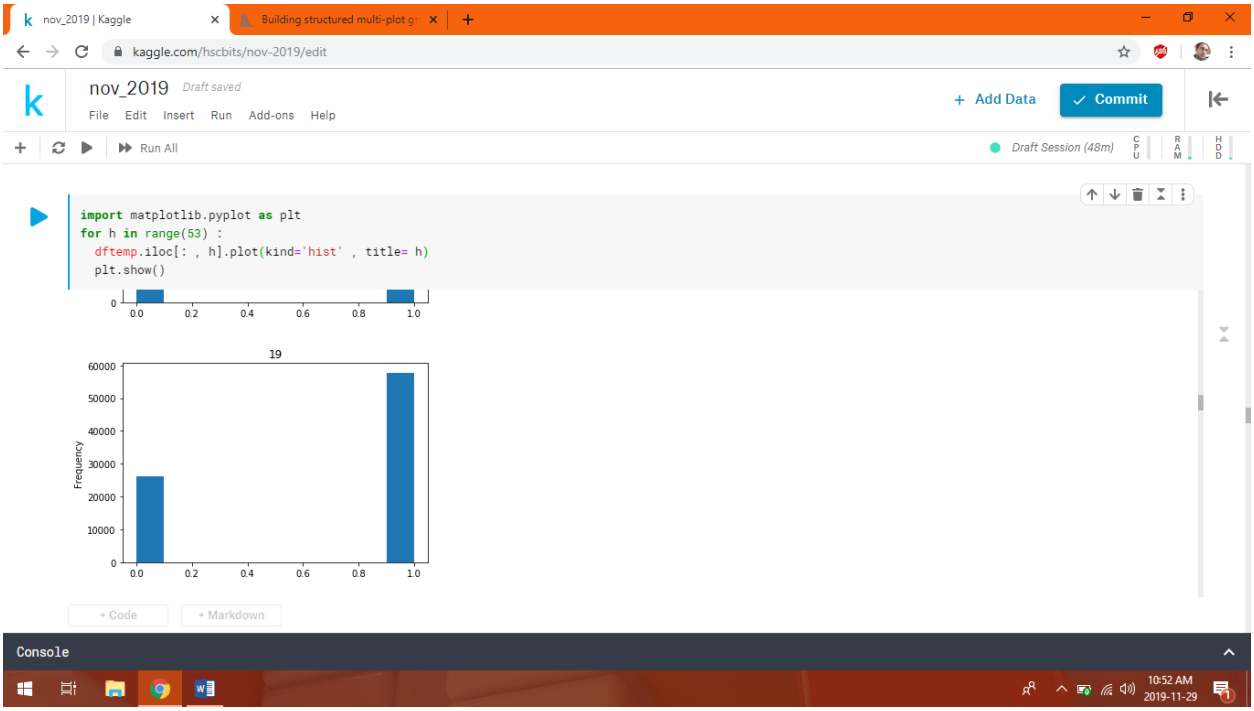
```
import matplotlib.pyplot as plt
for h in range(53) :
  dftemp.iloc[: , h].plot(kind='hist' , title= h)
  plt.show()
```



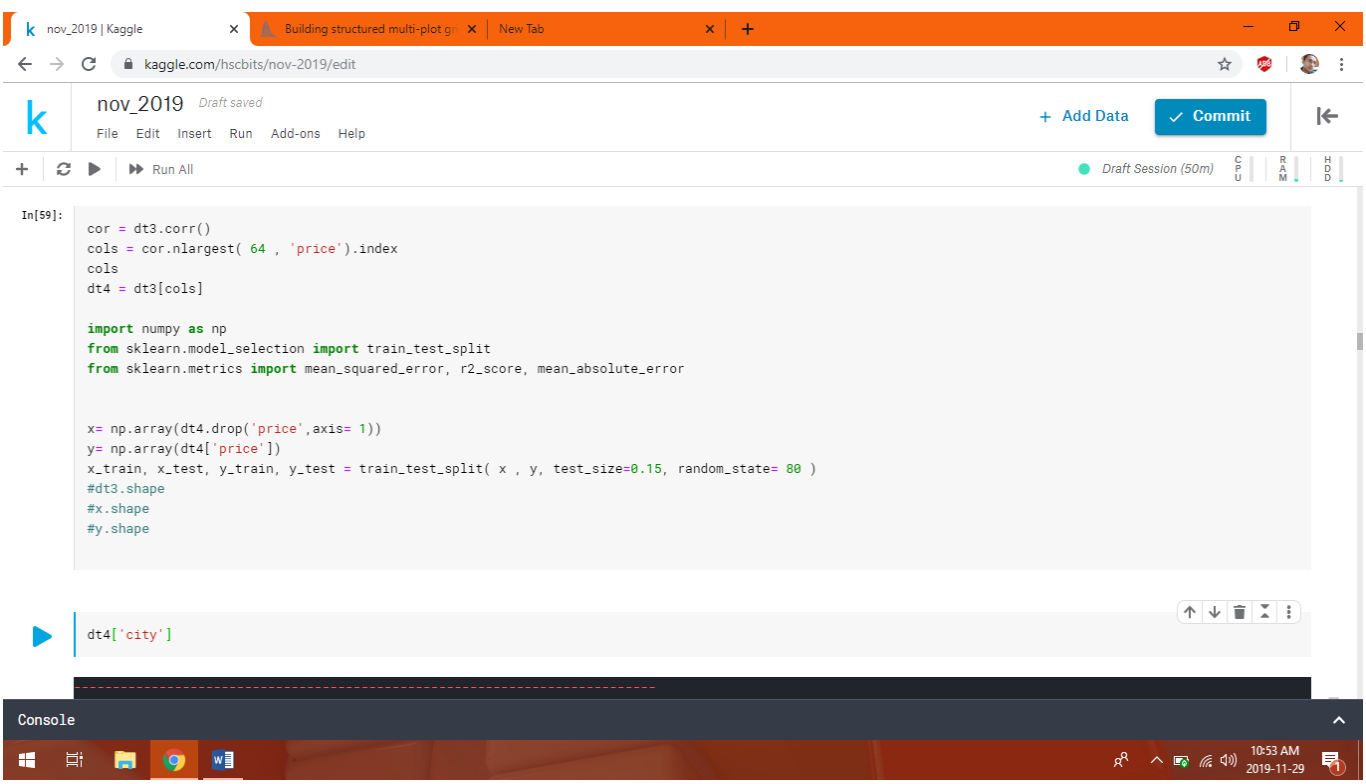
+ Code + Markdown

Console

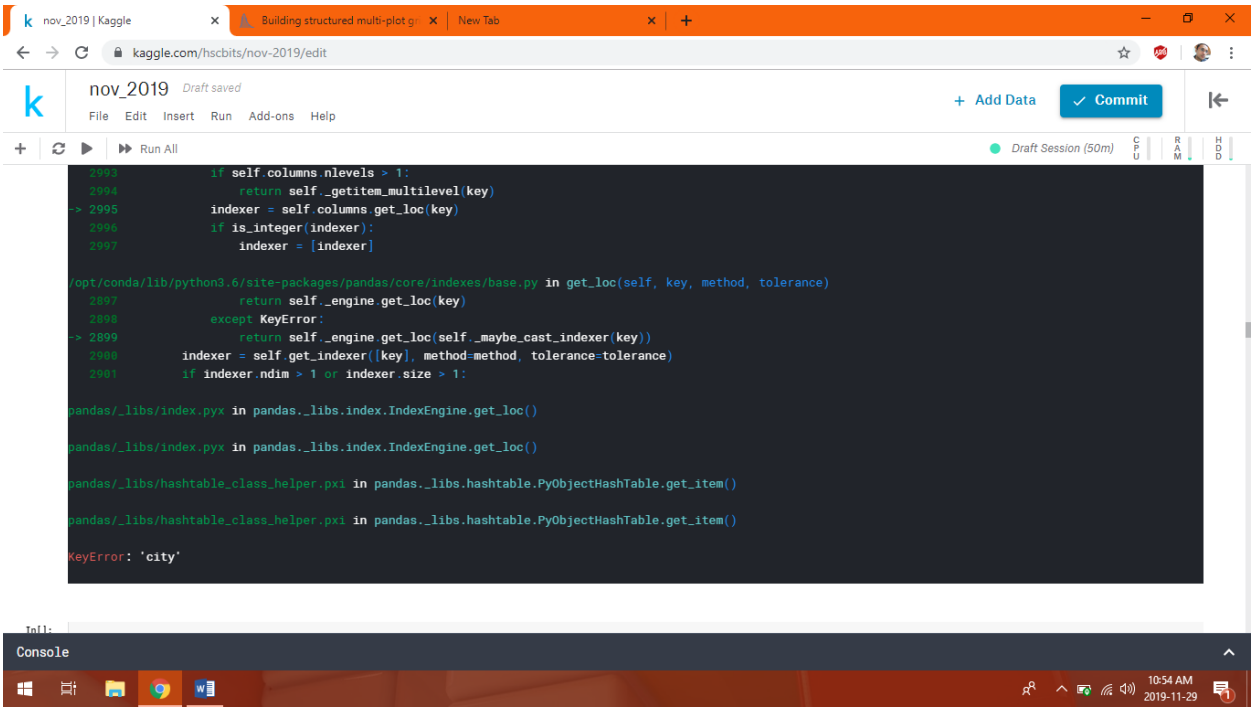
10:51 AM
2019-11-29



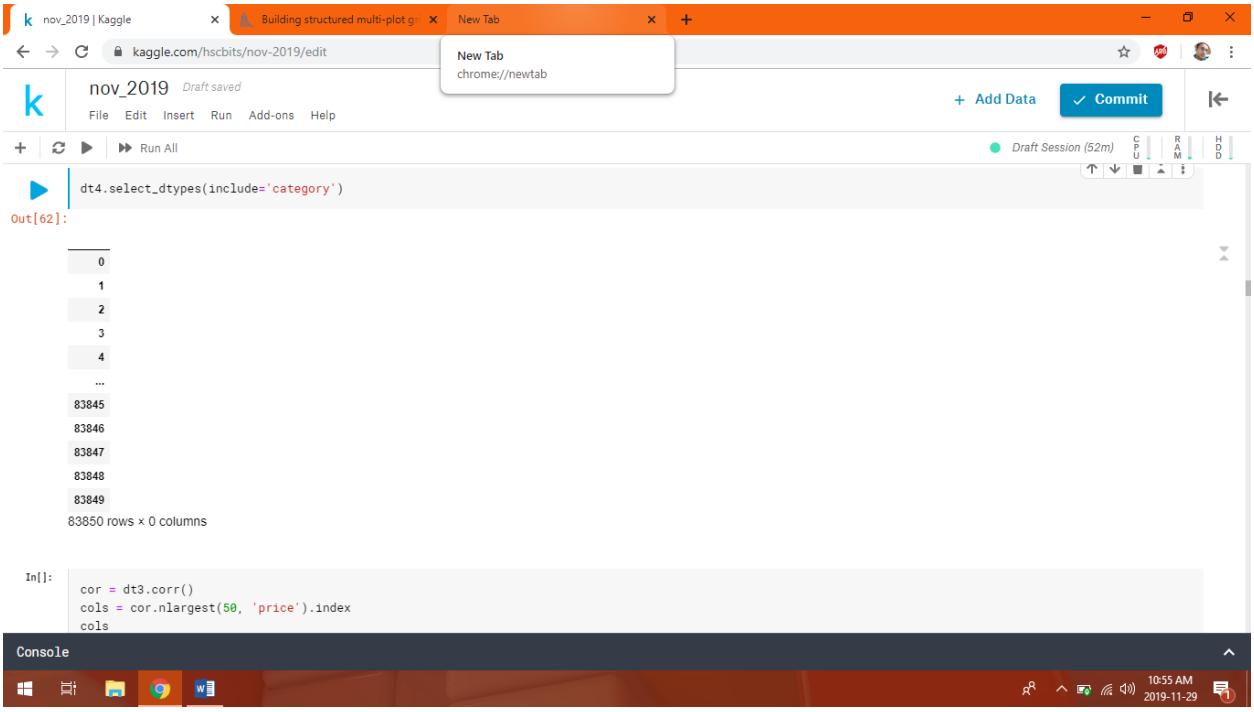
Selecting top price influencing features as shown below.



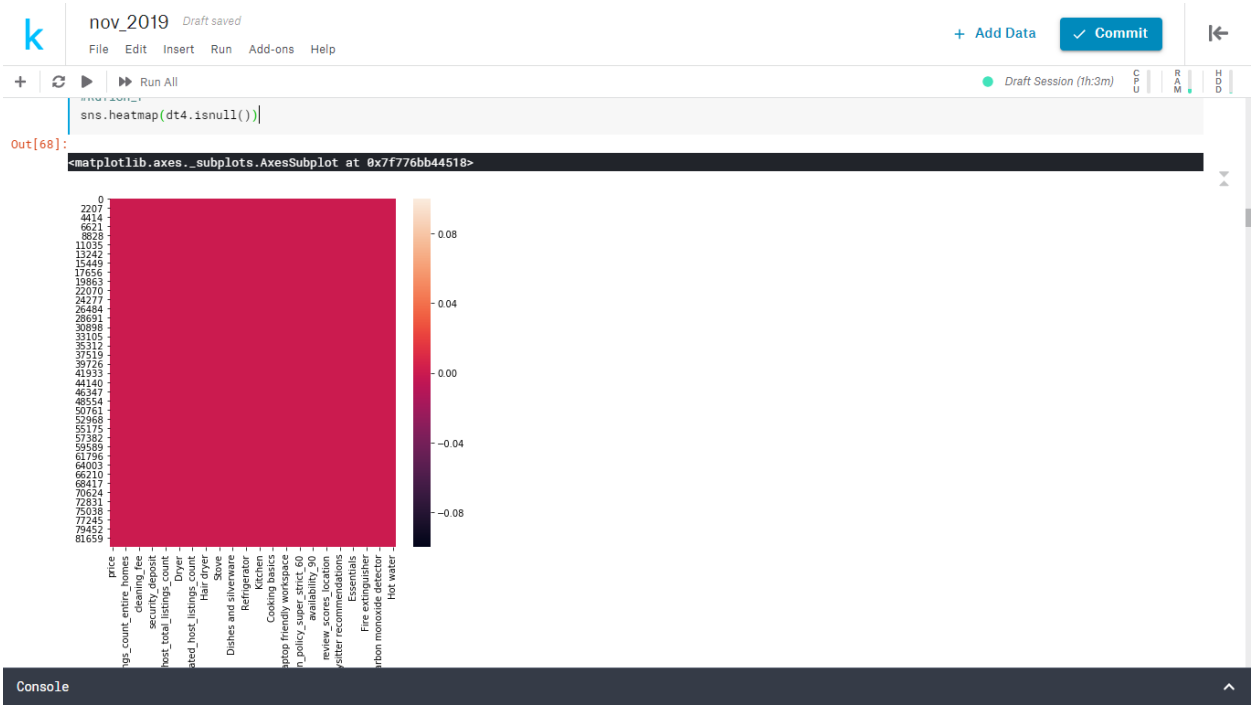
Checking whether “city” column is selected in the final dataset having only selected top features.



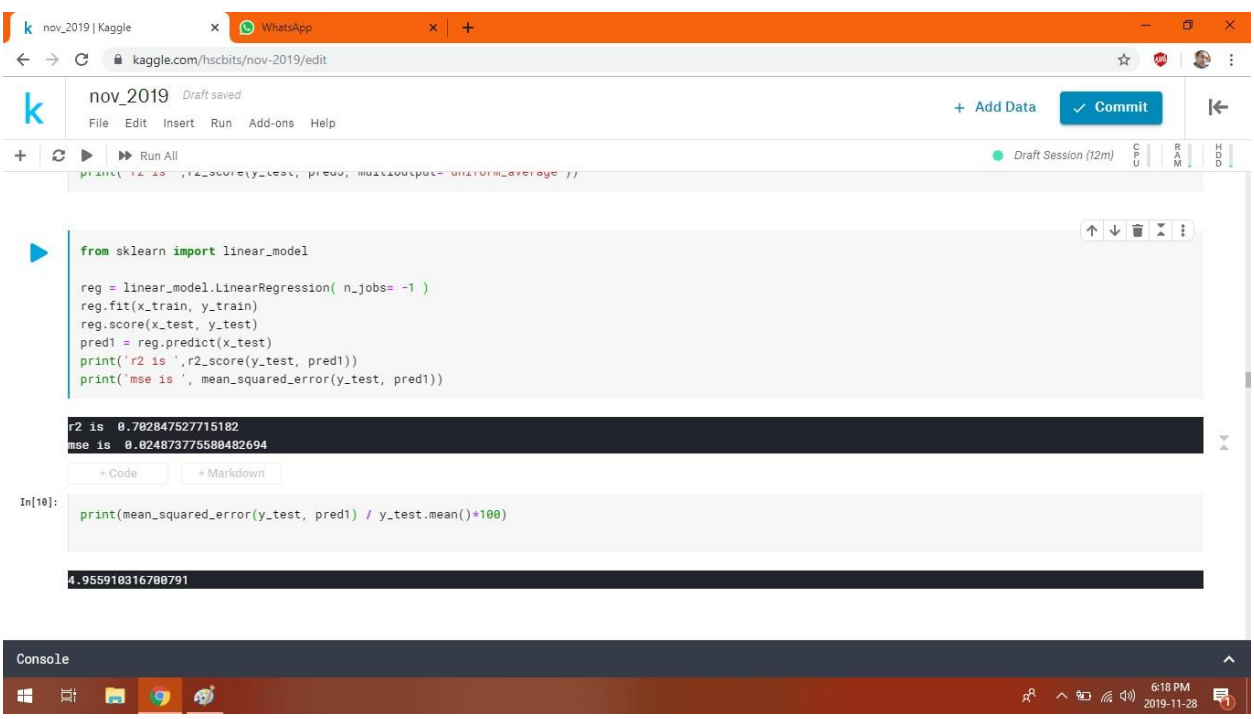
Checking whether any categorical column is still left in the dataset as shown below.



Checking for null values in the final dataset moving forward.



Implementation of sklearn LinearRegression with associated results can be observed in the screenshot as follows.



Implementation of sklearn ElasticNet with associated results can be observed in the screenshot as follows.

```
#x.shape
#y.shape
```

In[]:

```
dt4
```

```
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score

eln = ElasticNet(max_iter= 15000 ,alpha= 0.0001, l1_ratio= 0.001, random_state= 80)
eln.fit(x_train, y_train)
eln.score(x_test, y_test)
pred = eln.predict(x_test)
print('r2 is ' , r2_score(y_test, pred))
print('mse is ', mean_squared_error(y_test, pred))
```

```
r2 is  0.702827333012118
mse is  0.024875466020774402
```

Implementation of sklearn RandomForestRegressor with associated results can be observed in the screenshot as follows.

In[22]:

```
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score

eln = ElasticNet(max_iter= 15000 ,alpha= 0.0001, l1_ratio= 0.001, random_state= 80)
```

In[25]:

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(max_depth= 128, random_state=80,n_estimators= 1200, max_features = 0.4 , n_jobs = -1)
rf.fit(x_train, y_train)
pred3 = rf.predict(x_test)
print('r2 is ',r2_score(y_test, pred3))
print('mse is ', mean_squared_error(y_test, pred3))
print('mae is ',mean_absolute_error(y_test, pred3))
```

```
r2 is  0.7805280401259538
mse is  0.0183713641489855 54
```

Implementation of XGBoost model and the outcomes can be observed in the following screenshot.

```
In[22]:
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score

eln = ElasticNet(max_iter= 15000 ,alpha= 0.0001, l1_ratio= 0.001, random_state= 80)
```

```
import xgboost as xgb
xg = xgb.XGBRegressor(colsample_bytree= 1, subsample= 1,learning_rate=0.1, max_depth=80, min_child_weight= 1, n_estimators= 800,
                      reg_alpha=0.1, reg_lambda= 1, gamma=0.01, silent=0 , random_state = 80, nthread = -1)
xg.fit(x_train, y_train)
pred5 = xg.predict(x_test)
print('r2 is ',r2_score(y_test, pred5))
print('mse is ', mean_squared_error(y_test, pred5))
print('mae is ',mean_absolute_error(y_test, pred5))
```

```
[13:06:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
r2 is  0.7716290620211286
mse is  0.019116271914931673
```

Columnwise comparison of the test data with the predicted data using XGBoost model .

```
comp = pd.DataFrame( {'prediction':pred5*10000 , 'test': y_test*10000})
comp.head(10)
```

Out[36]:

	prediction	test
0	35.739540	29.197080
1	248.986786	149.635036
2	60.139297	29.197080
3	32.382011	29.197080
4	127.347710	145.985401
5	78.171791	36.496350
6	60.549973	65.693431
7	92.200340	113.868613
8	37.724674	30.656934
9	325.548645	324.087591

```
In[]:
( ( (mean_squared_error(y_test, pred5))) / abs( y_test.mean() )  )*100
```

nov_2019 | Kaggle

kaggle.com/hscbits/nov-2019/edit

nov_2019 Draft saved

File Edit Insert Run Add-ons Help

+ Add Data ✓ Commit

Run All

Draft Session (49m)

```
comp = ps.DataFrame( {'prediction':pred5*10000 , 'test': y_test*10000})
comp
```

Out[22]:

	prediction	test
0	34.297108	29.197080
1	227.777069	149.635036
2	59.753059	29.197080
3	32.237770	29.197080
4	132.389069	145.985401
...	...	...
12573	773.950806	619.708029
12574	78.151230	69.343066
12575	89.292824	109.489051
12576	81.281067	109.489051
12577	69.420036	47.445255

12578 rows × 2 columns

+ Code + Markdown

In[]:

```
( ( (mean_squared_error(y_test, pred5))) / abs( y_test.mean() ) )*100
```

Console

4:20 PM 2019-11-28

Columnwise comparison of the test data with the predicted data using RandomForestRegressor model .

nov_2019 | Kaggle

kaggle.com/hscbits/nov-2019/edit

nov_2019 Draft saved

File Edit Insert Run Add-ons Help

+ Add Data ✓ Commit

Run All

Draft Session (1h:12m)

```
comp = ps.DataFrame( {'prediction':pred3*10000 , 'test': y_test*10000})
comp
```

Out[31]:

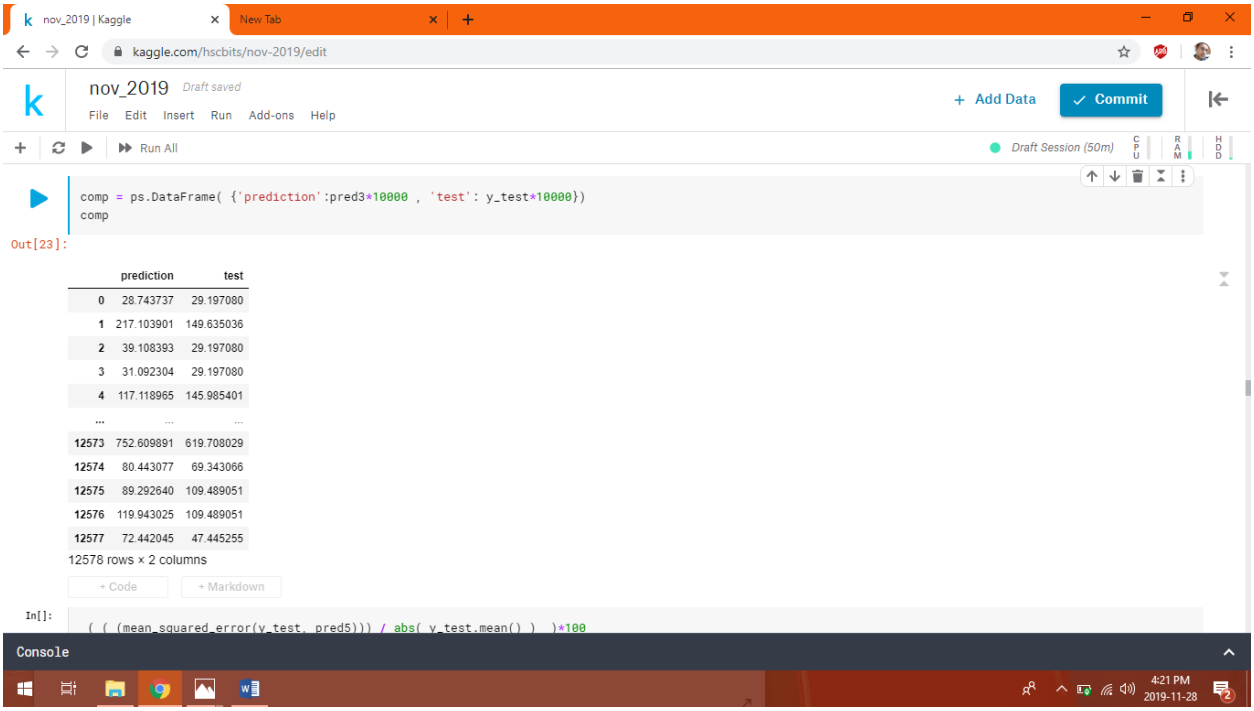
	prediction	test
0	30.144288	29.197080
1	198.849754	149.635036
2	47.272868	29.197080
3	31.684408	29.197080
4	117.699358	145.985401
...	...	...
12573	753.015187	619.708029
12574	78.280102	69.343066
12575	92.865058	109.489051
12576	111.476518	109.489051
12577	72.646617	47.445255

12578 rows × 2 columns

+ Code + Markdown

Console

4:42 PM 2019-11-28



Intended Refinements / Extensions

- Further use of text data available in the dataset for more features' creation
- Implementation of location bins as per geographical and associated financial parameters
- Consequently, creation of areas or segments of houses as per cities and prices
- More feature engineering based on columns and following the first point
- Consideration of dropping some entries based on accuracy improvements
- Optimisation, tuning of the models to enhance the results to get better prediction
- Experimenting to go with fewer feature to focus on performance enhancements
- Checking for the accuracy after implementing neural network
- Collecting related reviews affecting the prices and decisions

Thank You !