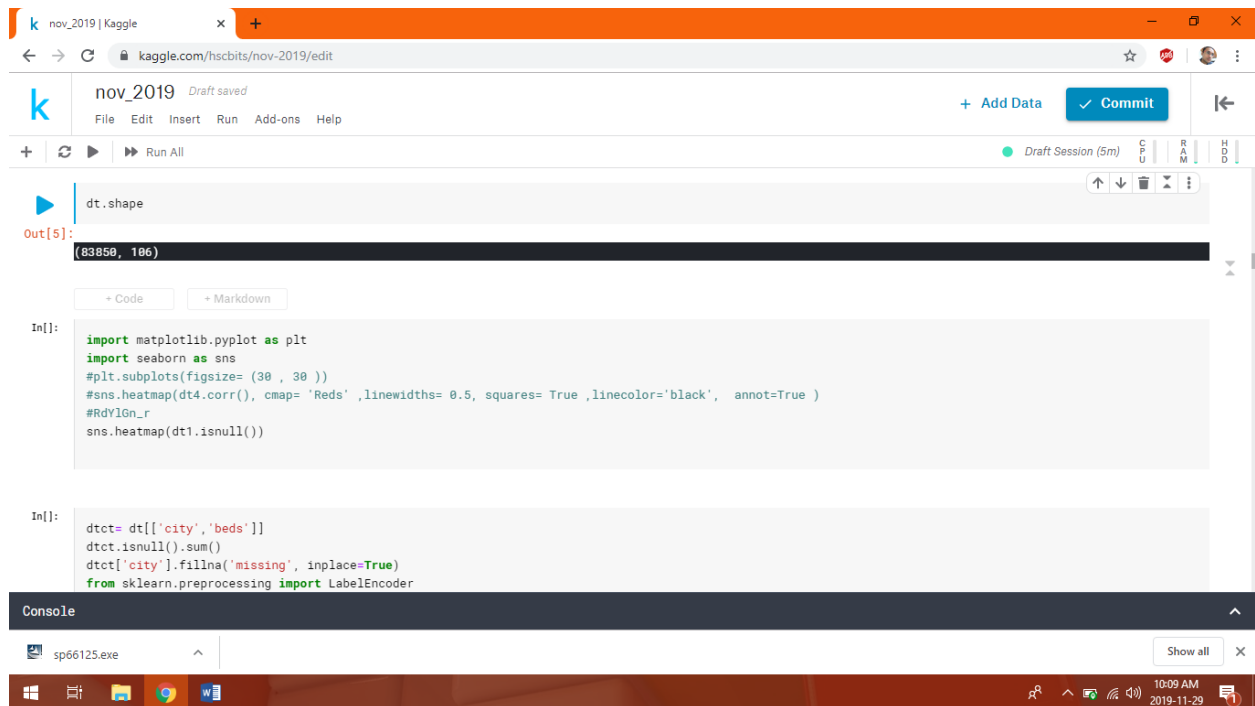


The attached dataset is of 83850 x 106 dimension as can be observed in the following screenshot.



The screenshot shows a Kaggle notebook interface for a dataset named 'nov_2019'. The code cell contains the following Python code:

```
dt.shape
```

The output of the code is:

```
Out[5]: (83850, 106)
```

The code cell also includes the following code:

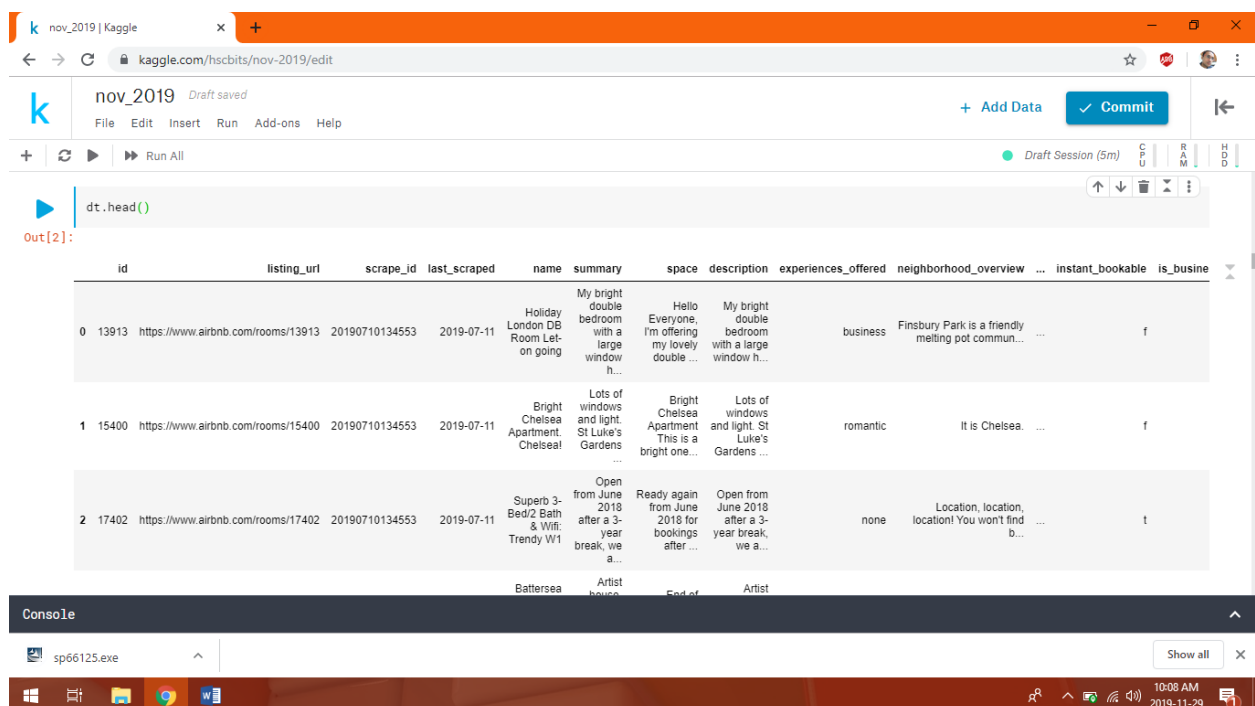
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.subplots(figsize=(30, 30))
sns.heatmap(dt4.corr(), cmap='Reds', linewidths=0.5, squares=True, linecolor='black', annot=True)
#RdYlGn_r
sns.heatmap(dt1.isnull())
```

The code cell also includes the following code:

```
In[ ]: dtct= dt[['city', 'beds']]
dtct.isnull().sum()
dtct['city'].fillna('missing', inplace=True)
from sklearn.preprocessing import LabelEncoder
```

The console shows the command 'sp66125.exe' and the time '10:09 AM 2019-11-29'.

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.



The screenshot shows a Kaggle notebook interface for a dataset named 'nov_2019'. The code cell contains the following Python code:

```
dt.head()
```

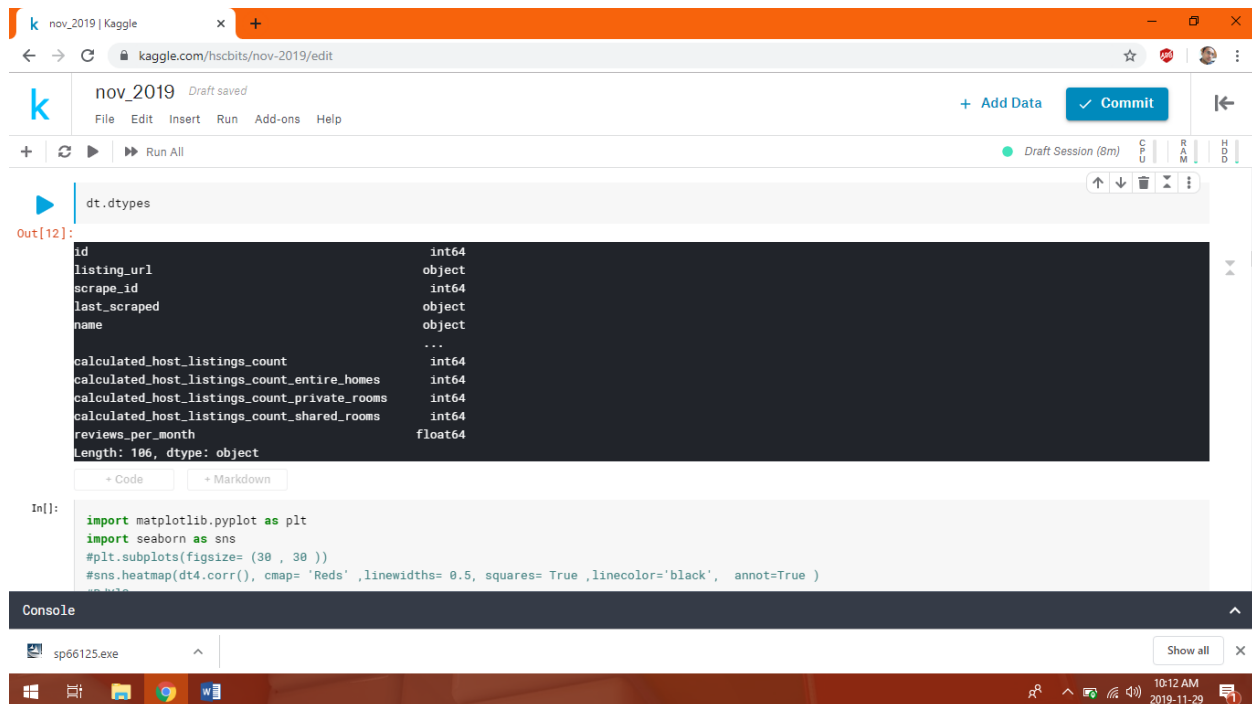
The output of the code is:

```
Out[2]:
```

	id	listing_url	scrape_id	last_scraped	name	summary	space	description	experiences_offered	neighborhood_overview	instant_bookable	is_busine
0	13913	https://www.airbnb.com/rooms/13913	20190710134553	2019-07-11	Holiday London DB Room Let-on going	My bright double bedroom with a large window h...	Hello Everyone, I'm offering my lovely double ...	My bright double bedroom with a large window h...	business	Finsbury Park is a friendly melting pot commun...	f	
1	15400	https://www.airbnb.com/rooms/15400	20190710134553	2019-07-11	Bright Chelsea Apartment, Chelsea!	Lots of windows and light. St Luke's Gardens ...	Bright Chelsea Apartment This is a bright one ...	Lots of windows and light. St Luke's Gardens ...	romantic	It is Chelsea. ...	f	
2	17402	https://www.airbnb.com/rooms/17402	20190710134553	2019-07-11	Open from June 2018 after a 3-year break, we a...	Superb 3-Bed/2 Bath & Wifi: Trendy W1	Ready again from June 2018 for bookings after ...	Open from June 2018 after a 3-year break, we a...	none	Location, location, location! You won't find b...	t	

The console shows the command 'sp66125.exe' and the time '10:08 AM 2019-11-29'.

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.



The screenshot shows a Kaggle notebook interface. The code cell contains `dt.dtypes`. The output, labeled `Out[12]:`, displays the data types for each column in a table format. The columns are: `id` (int64), `listing_url` (object), `scrape_id` (int64), `last_scraped` (object), `name` (object), `calculated_host_listings_count` (int64), `calculated_host_listings_count_entire_homes` (int64), `calculated_host_listings_count_private_rooms` (int64), `calculated_host_listings_count_shared_rooms` (int64), `reviews_per_month` (float64), and `Length: 106, dtype: object`. Below the code cell, there is an input cell with `In[]:` containing imports for `matplotlib.pyplot` and `seaborn`, and a `plt.subplots` call. The console at the bottom shows the file `sp66125.exe` and the system clock at 10:12 AM on 2019-11-29.

```
dt.dtypes
```

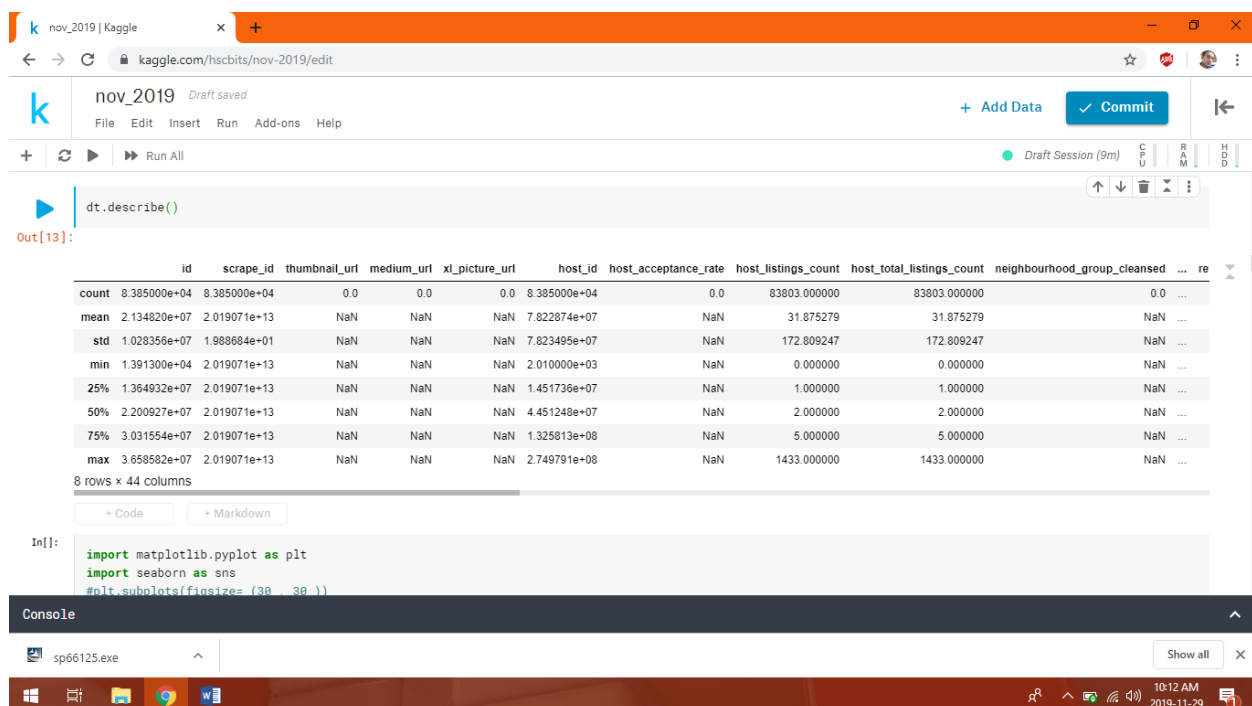
```
Out[12]:
```

id	int64
listing_url	object
scrape_id	int64
last_scraped	object
name	object
...	...
calculated_host_listings_count	int64
calculated_host_listings_count_entire_homes	int64
calculated_host_listings_count_private_rooms	int64
calculated_host_listings_count_shared_rooms	int64
reviews_per_month	float64
Length: 106, dtype: object	

```
In[ ]:
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.subplots(figsize= (30 , 30 ))
sns.heatmap(dt4.corr(), cmap= 'Reds' ,linewidths= 0.5, squares= True ,linecolor='black', annot=True )
```

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



The screenshot shows a Kaggle notebook interface. The code cell contains `dt.describe()`. The output, labeled `Out[13]:`, displays a summary statistics table for the dataset. The columns are: `count`, `id`, `scrape_id`, `thumbnail_url`, `medium_url`, `xl_picture_url`, `host_id`, `host_acceptance_rate`, `host_listings_count`, `host_total_listings_count`, `neighbourhood_group_cleansed`, and `re`. The rows show: `count`, `mean`, `std`, `min`, `25%`, `50%`, `75%`, and `max`. The console at the bottom shows the file `sp66125.exe` and the system clock at 10:12 AM on 2019-11-29.

```
dt.describe()
```

```
Out[13]:
```

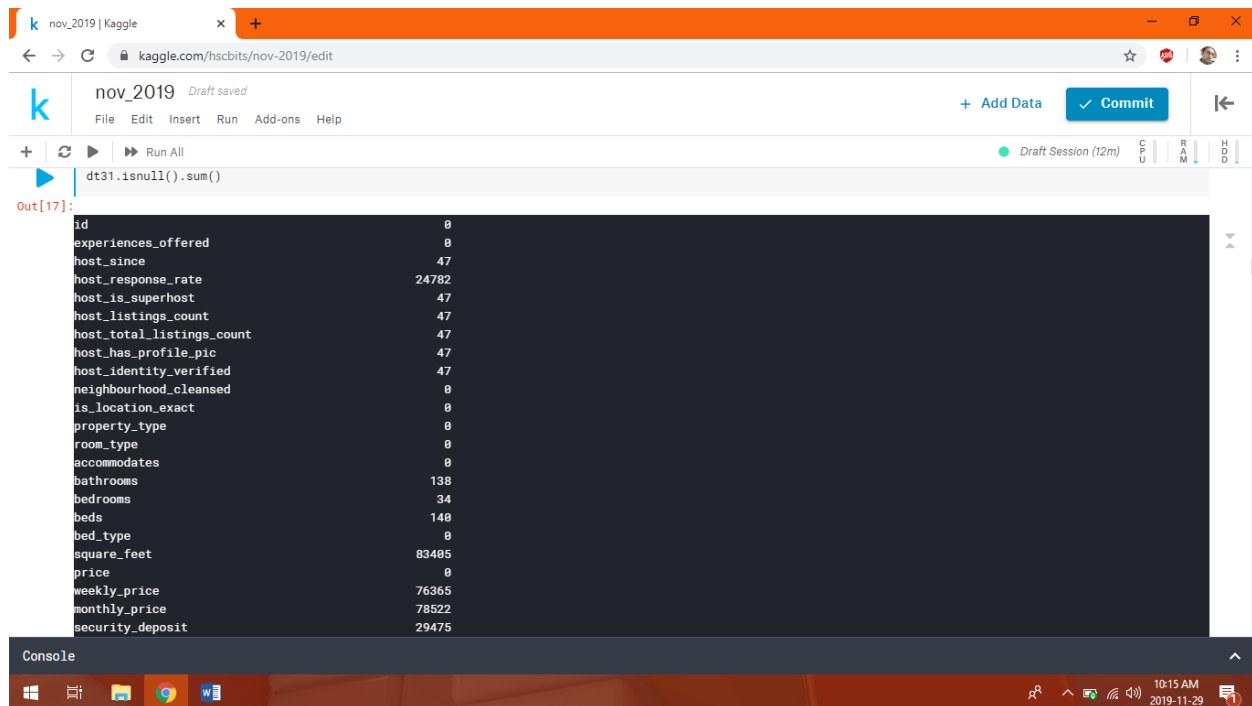
	id	scrape_id	thumbnail_url	medium_url	xl_picture_url	host_id	host_acceptance_rate	host_listings_count	host_total_listings_count	neighbourhood_group_cleansed	re
count	8.385000e+04	8.385000e+04	0.0	0.0	0.0	8.385000e+04	0.0	83803.000000	83803.000000	0.0	...
mean	2.134820e+07	2.019071e+13	NaN	NaN	NaN	7.822874e+07	NaN	31.875279	31.875279	NaN	...
std	1.028356e+07	1.988684e+01	NaN	NaN	NaN	7.823495e+07	NaN	172.809247	172.809247	NaN	...
min	1.391300e+04	2.019071e+13	NaN	NaN	NaN	2.010000e+03	NaN	0.000000	0.000000	NaN	...
25%	1.364932e+07	2.019071e+13	NaN	NaN	NaN	1.451736e+07	NaN	1.000000	1.000000	NaN	...
50%	2.200927e+07	2.019071e+13	NaN	NaN	NaN	4.451248e+07	NaN	2.000000	2.000000	NaN	...
75%	3.031554e+07	2.019071e+13	NaN	NaN	NaN	1.325813e+08	NaN	5.000000	5.000000	NaN	...
max	3.658582e+07	2.019071e+13	NaN	NaN	NaN	2.749791e+08	NaN	1433.000000	1433.000000	NaN	...

```
8 rows x 44 columns
```

```
In[ ]:
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.subplots(figsize= (30 , 30 ))
```

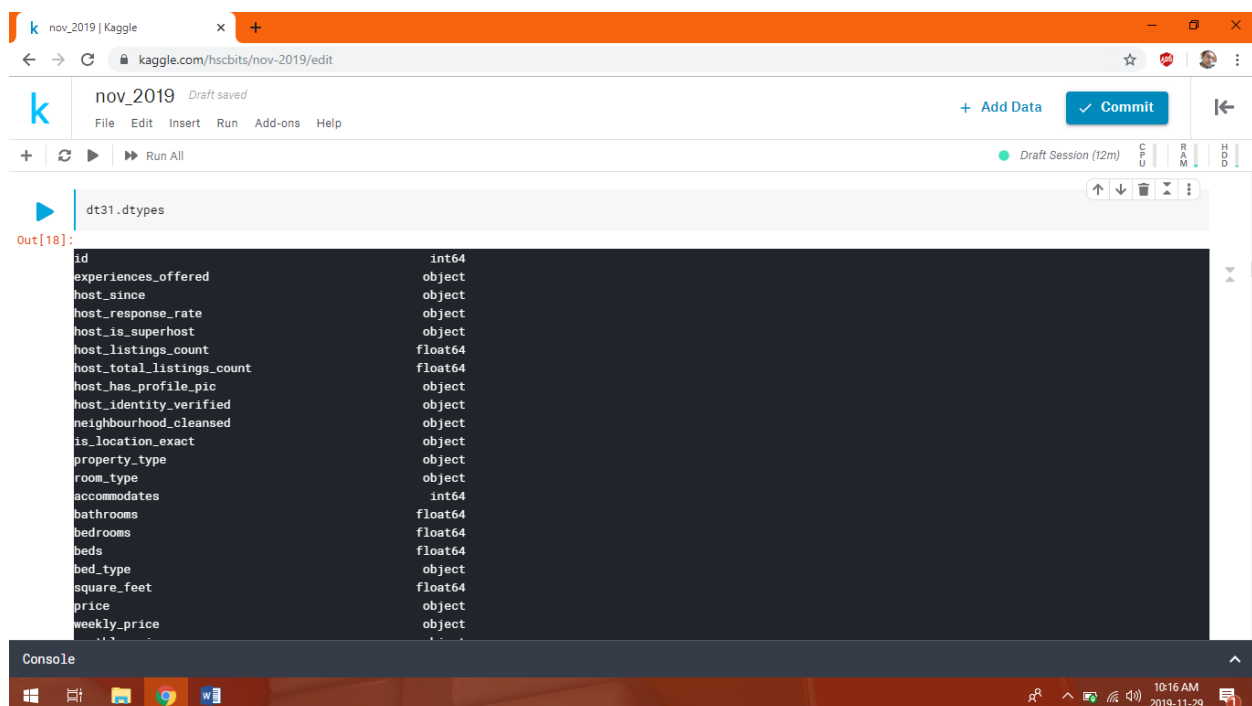
Presence of null values in the dataset checked as below prompting for cleaning and preprocessing.



The screenshot shows a Kaggle notebook interface for the 'nov_2019' dataset. The code cell contains the command `dt31.isnull().sum()`. The output, labeled `Out[17]:`, displays a list of columns and their corresponding null counts. The columns and their values are: `id` (0), `experiences_offered` (0), `host_since` (47), `host_response_rate` (24782), `host_is_superhost` (47), `host_listings_count` (47), `host_total_listings_count` (47), `host_has_profile_pic` (47), `host_identity_verified` (47), `neighbourhood_cleansed` (0), `is_location_exact` (0), `property_type` (0), `room_type` (0), `accommodates` (0), `bathrooms` (138), `bedrooms` (34), `beds` (140), `bed_type` (0), `square_feet` (83485), `price` (0), `weekly_price` (76365), `monthly_price` (78522), and `security_deposit` (29475).

Column	Count
id	0
experiences_offered	0
host_since	47
host_response_rate	24782
host_is_superhost	47
host_listings_count	47
host_total_listings_count	47
host_has_profile_pic	47
host_identity_verified	47
neighbourhood_cleansed	0
is_location_exact	0
property_type	0
room_type	0
accommodates	0
bathrooms	138
bedrooms	34
beds	140
bed_type	0
square_feet	83485
price	0
weekly_price	76365
monthly_price	78522
security_deposit	29475

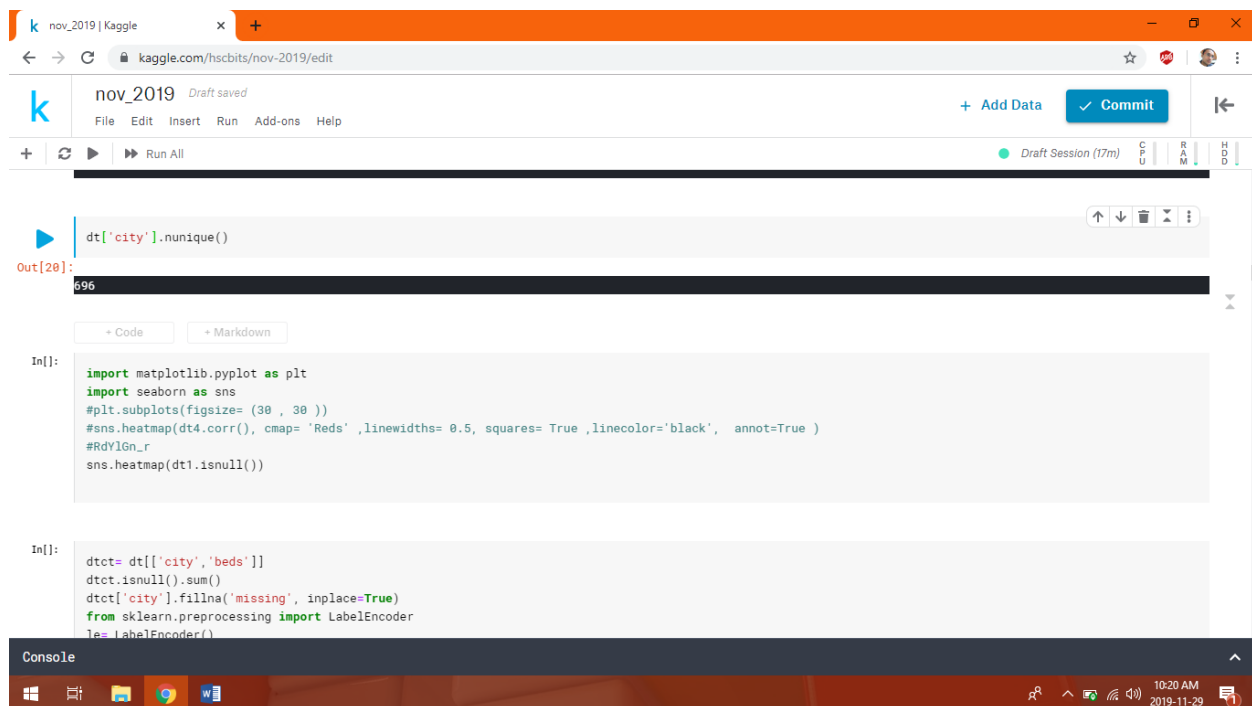
Data types of columns of the dataset we are going further with before cleaning and preprocessing can be observed as follows showing variety of values.



The screenshot shows a Kaggle notebook interface for the 'nov_2019' dataset. The code cell contains the command `dt31.dtypes`. The output, labeled `Out[18]:`, displays a list of columns and their corresponding data types. The columns and their values are: `id` (int64), `experiences_offered` (object), `host_since` (object), `host_response_rate` (object), `host_is_superhost` (object), `host_listings_count` (float64), `host_total_listings_count` (float64), `host_has_profile_pic` (object), `host_identity_verified` (object), `neighbourhood_cleansed` (object), `is_location_exact` (object), `property_type` (object), `room_type` (object), `accommodates` (int64), `bathrooms` (float64), `bedrooms` (float64), `beds` (float64), `bed_type` (object), `square_feet` (float64), `price` (object), and `weekly_price` (object).

Column	Data Type
id	int64
experiences_offered	object
host_since	object
host_response_rate	object
host_is_superhost	object
host_listings_count	float64
host_total_listings_count	float64
host_has_profile_pic	object
host_identity_verified	object
neighbourhood_cleansed	object
is_location_exact	object
property_type	object
room_type	object
accommodates	int64
bathrooms	float64
bedrooms	float64
beds	float64
bed_type	object
square_feet	float64
price	object
weekly_price	object

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



This screenshot shows a Kaggle notebook for the 'nov_2019' dataset. The first code cell runs `dt['city'].nunique()`, which returns 696, indicating a high number of unique categorical values in the 'city' column. The second code cell imports `matplotlib.pyplot` as `plt` and `seaborn` as `sns`, then sets up a heatmap with `sns.heatmap(dt4.corr(), cmap='Reds', linewidths=0.5, squares=True, linecolor='black', annot=True)` and `sns.heatmap(dt1.isnull())`. The third code cell creates a new DataFrame `dtct` containing columns 'city' and 'beds', checks for missing values with `dtct.isnull().sum()`, fills missing values with 'missing' using `dtct['city'].fillna('missing', inplace=True)`, and imports `LabelEncoder` from `sklearn.preprocessing`.

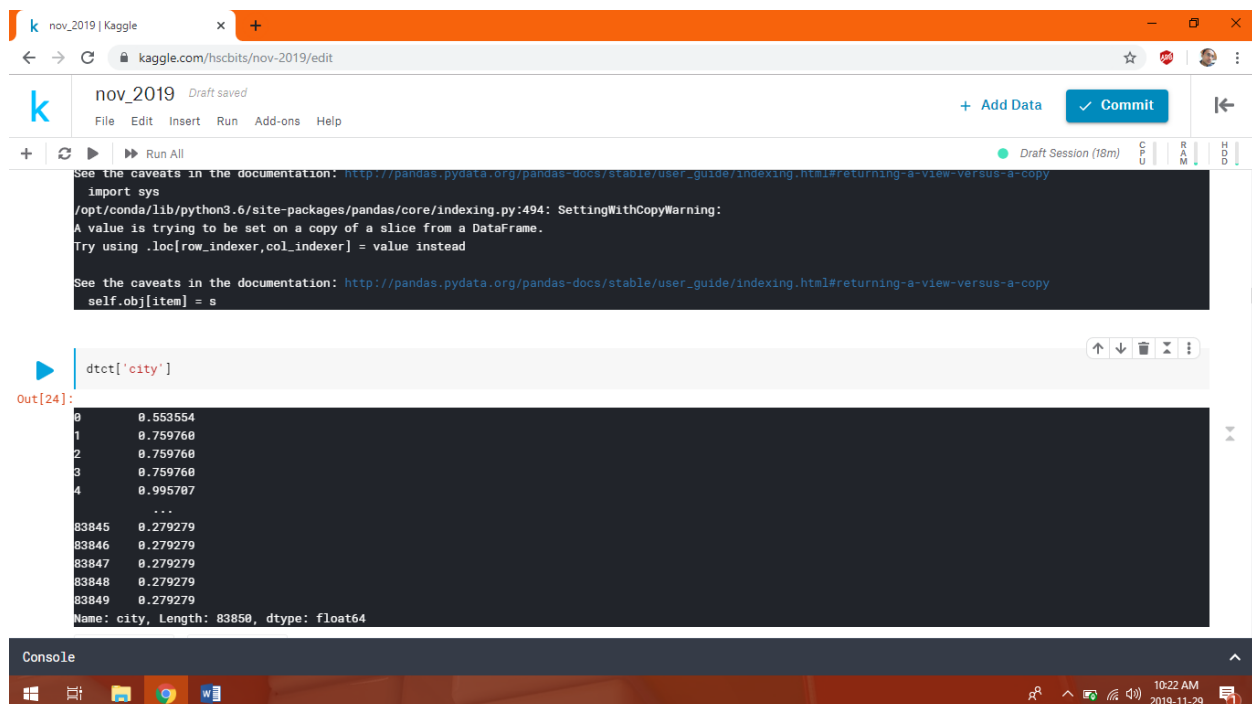
```
dt['city'].nunique()

Out[20]:
696

In[ ]:
import matplotlib.pyplot as plt
import seaborn as sns
#plt.subplots(figsize= (30 , 30 ))
#sns.heatmap(dt4.corr(), cmap= 'Reds' ,linewidths= 0.5, squares= True ,linecolor='black' , annot=True )
#RdYlGn_r
sns.heatmap(dt1.isnull())

In[ ]:
dtct= dt[['city','beds']]
dtct.isnull().sum()
dtct['city'].fillna('missing', inplace=True)
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
```

“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.



This screenshot shows the next step in the notebook. A warning message from pandas is displayed: "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". The code cell then runs `dtct['city']`, which returns a Series of numerical values representing the label-encoded 'city' column. The output shows values ranging from approximately 0.55 to 0.99, with a name of 'city' and a dtype of 'float64'.

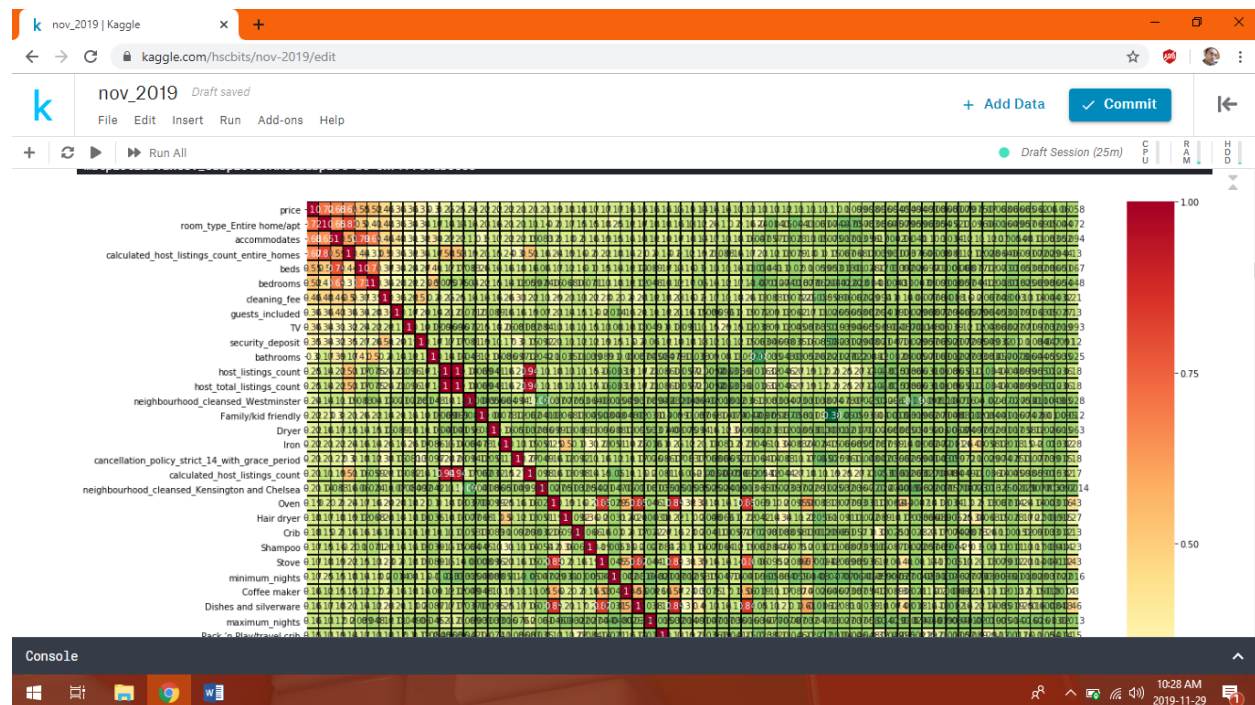
```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
import sys
/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py:494: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
self.obj[item] = s

dtct['city']

Out[24]:
0      0.553554
1      0.759768
2      0.759768
3      0.759768
4      0.995707
...
83845   0.279279
83846   0.279279
83847   0.279279
83848   0.279279
83849   0.279279
Name: city, Length: 83850, dtype: float64
```

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.

```

dt4.isnull().sum()

Out[33]:
price                                0
room_type_Entire home/apt            0
accommodates                         0
calculated_host_listings_count_entire_homes 0
beds                                 0
Paid parking off premises            0
Carbon monoxide detector             0
experiences_offered_family           0
bed_type_Real Bed                    0
Hot water                            0
length: 64, dtype: int64

In[]:
cor = dt3.corr()
cols = cor.nlargest(50, 'price').index
dt4 = dt3[cols]
```

Final data showing top rows can be observed as below.

The screenshot shows a Kaggle notebook interface for the 'nov_2019' dataset. The code cell contains the following Python code:

```
try using .loc[row_indexer,col_indexer] = value instead  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
self.obj[item] = s
```

The output of the code is a DataFrame showing the top 5 rows of the dataset. The columns are: price, room_type_Entire home/apt, accommodates, calculated_host_listings_count_entire_homes, beds, bedrooms, cleaning_fee, guests_included, TV, security_deposit, Essentials, Hangers, and Children. The output is displayed as a table with 5 rows and 13 columns.

	price	room_type_Entire home/apt	accommodates	calculated_host_listings_count_entire_homes	beds	bedrooms	cleaning_fee	guests_included	TV	security_deposit	Essentials	Hangers	Children
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

The output is summarized as 5 rows x 64 columns.

The console shows the following code:

```
In[]:  
cor = dt3.corr()  
cols = cor.nlargest(50, 'price').index
```

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

The screenshot shows a Kaggle notebook interface for the 'nov_2019' dataset. The code cell contains the following Python code:

```
dftemp.tail(10)
```

The output of the code is a DataFrame showing the last 10 rows of the dataset. The columns are: 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, and Roll i showe. The output is displayed as a table with 10 rows and 21 columns.

	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

The output is summarized as 10 rows x 53 columns.

The console shows the following code:

```
In[]:  
dftemp.tail(10)
```

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

The screenshot shows a Kaggle notebook interface. The code cell contains the following Python code:

```
dt['price'] = dt['price'].apply(lambda h: h.replace('$', ''))
dt['security_deposit'] = dt['security_deposit'].apply(lambda h: str(h).replace('$', ''))
dt['cleaning_fee'] = dt['cleaning_fee'].apply(lambda h: str(h).replace('$', ''))
```

A red error message is displayed: `TypeError: 'NoneType' object is not subscriptable`.

The output cell shows the result of `dt['host_since']`, which is a pandas Series of dates:

```
Out[39]:
0      2009-11-16
1      2009-12-05
2      2010-01-04
3      2009-09-28
4      2010-04-03
...
83845   2016-08-19
83846   2015-10-28
83847   2015-10-28
83848   2015-10-28
83849   2015-10-28
Name: host_since, Length: 83850, dtype: object
```

Columns having date values

The screenshot shows a Kaggle notebook interface. The code cell contains the following Python code:

```
dt31[['host_since', 'first_review', 'last_review']]
```

The output cell shows the result of the code, which is a pandas DataFrame with three columns: `host_since`, `first_review`, and `last_review`. The DataFrame has 83850 rows.

	host_since	first_review	last_review
0	2009-11-16	2010-08-18	2019-06-10
1	2009-12-05	2009-12-21	2019-05-05
2	2010-01-04	2011-03-21	2019-06-19
3	2009-09-28	2010-11-15	2019-06-08
4	2010-04-03	2016-03-05	2019-06-22
...
83845	2016-08-19	NaN	NaN
83846	2015-10-28	NaN	NaN
83847	2015-10-28	NaN	NaN
83848	2015-10-28	NaN	NaN
83849	2015-10-28	NaN	NaN

83850 rows x 3 columns

Processed column values making sense for regression

The screenshot shows a Kaggle notebook interface for a file named 'nov_2019'. The code cell contains the following Python code:

```
dt3[['host_since', 'first_review', 'last_review']]
```

The output, labeled 'Out[43]:', displays a DataFrame with three columns: 'host_since', 'first_review', and 'last_review'. The data is as follows:

	host_since	first_review	last_review
0	0.998994	0.999585	0.441441
1	0.998608	1.000000	0.558559
2	0.997998	0.999213	0.374875
3	0.999105	0.999431	0.455455
4	0.997019	0.844845	0.340841
...
83845	0.346680	0.004505	0.015015
83846	0.485485	0.004505	0.015015
83847	0.485485	0.004505	0.015015
83848	0.485485	0.004505	0.015015
83849	0.485485	0.004505	0.015015

The output summary indicates 83850 rows and 3 columns. The console at the bottom shows the system tray with the time 10:35 AM on 2019-11-29.

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

The screenshot shows the same Kaggle notebook interface. The code cell contains the following Python code:

```
dt31[['price', 'security_deposit', 'cleaning_fee']]
```

The output, labeled 'Out[45]:', displays a DataFrame with three columns: 'price', 'security_deposit', and 'cleaning_fee'. The data is as follows:

	price	security_deposit	cleaning_fee
0	\$65.00	\$100.00	\$15.00
1	\$100.00	\$150.00	\$50.00
2	\$300.00	\$350.00	\$65.00
3	\$175.00	\$250.00	\$70.00
4	\$65.00	\$250.00	\$50.00
...
83845	\$100.00	\$0.00	\$50.00
83846	\$82.00	NaN	NaN
83847	\$78.00	NaN	NaN
83848	\$82.00	NaN	NaN
83849	\$72.00	NaN	NaN

The output summary indicates 83850 rows and 3 columns. Below the output, the code cell contains the following Python code:

```
In[32]: import matplotlib.pyplot as plt
import seaborn as sns
plt.subplots(figsize=(20, 20))
```

The console at the bottom shows the system tray with the time 10:38 AM on 2019-11-29.

Processed currency columns

nov_2019 | Kaggle

kaggle.com/hscbits/nov-2019/edit

nov_2019 Draft saved

+ Add Data Commit

File Edit Insert Run Add-ons Help

+ Run All

Draft Session (34m)

CPU RAM HDD

```
dt3[['price', 'security_deposit', 'cleaning_fee']]
```

Out[46]:

	price	security_deposit	cleaning_fee
0	0.391391	0.661662	0.449950
1	0.612613	0.717718	0.774274
2	0.944444	0.889389	0.849850
3	0.835335	0.821321	0.867868
4	0.391391	0.821321	0.774274
...
83845	0.612613	0.000000	0.774274
83846	0.501502	0.000000	0.000000
83847	0.466967	0.000000	0.000000
83848	0.501502	0.000000	0.000000
83849	0.436937	0.000000	0.000000

83850 rows x 3 columns

+ Code + Markdown

```
In[32]: import matplotlib.pyplot as plt
import seaborn as sns
plt.subplots(figsize=(20, 20))
```

Console

10:38 AM 2019-11-29

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

nov_2019 | Kaggle

kaggle.com/hscbits/nov-2019/edit

nov_2019 Draft saved

+ Add Data Commit

File Edit Insert Run Add-ons Help

+ Run All

Draft Session (39m)

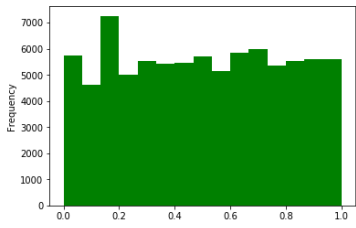
CPU RAM HDD

```
TypeError: 'NoneType' object is not subscriptable
```

```
dt4['price'].plot.hist(bins=15, color='g')
```

Out[54]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f776c6046d8>



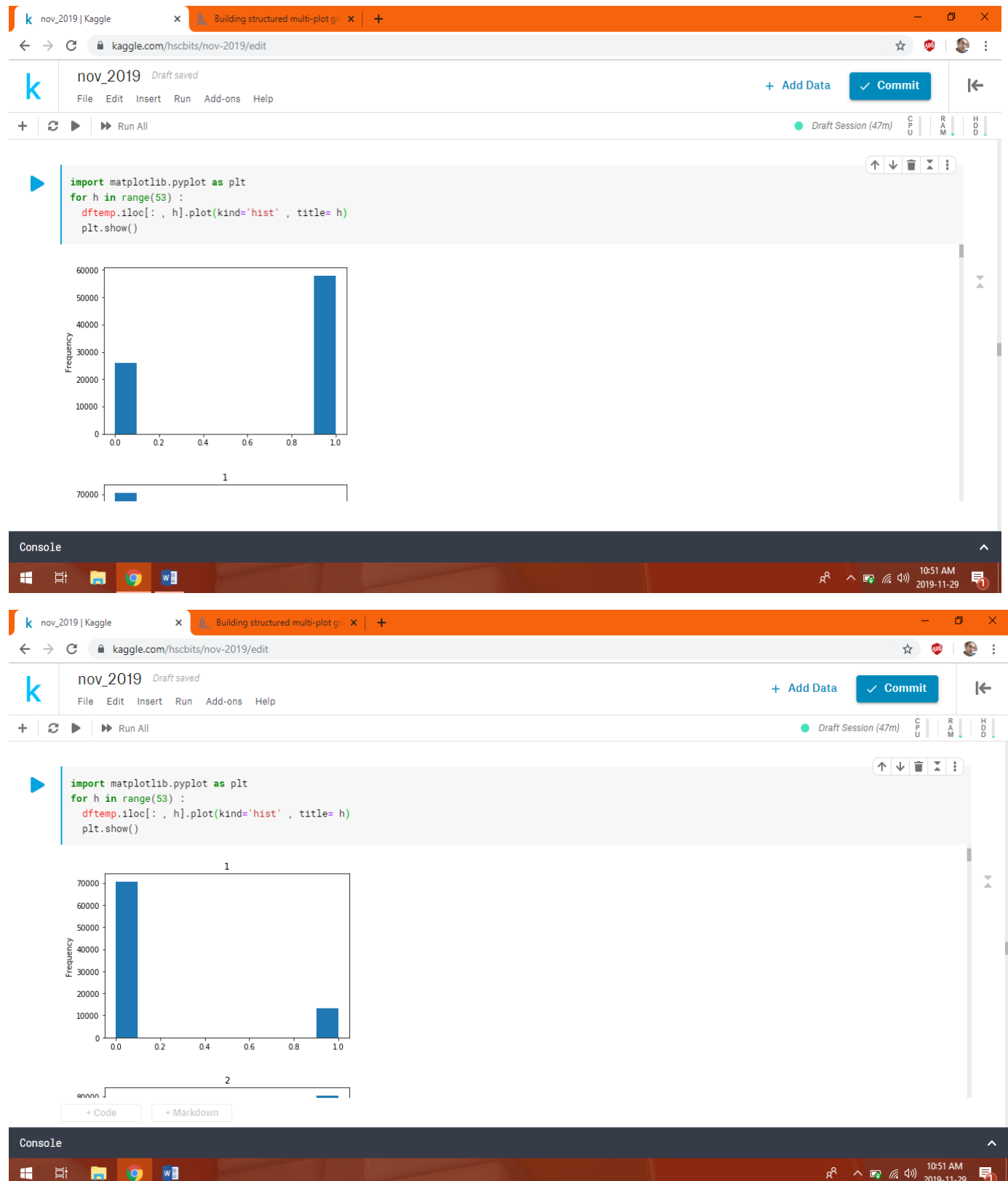
A histogram showing the frequency distribution of prices. The x-axis is labeled from 0.0 to 1.0 in increments of 0.2. The y-axis is labeled 'Frequency' and ranges from 0 to 7000 in increments of 1000. The histogram consists of 15 green bars. The distribution is roughly bell-shaped, centered around 0.2, with a peak frequency of approximately 7000.

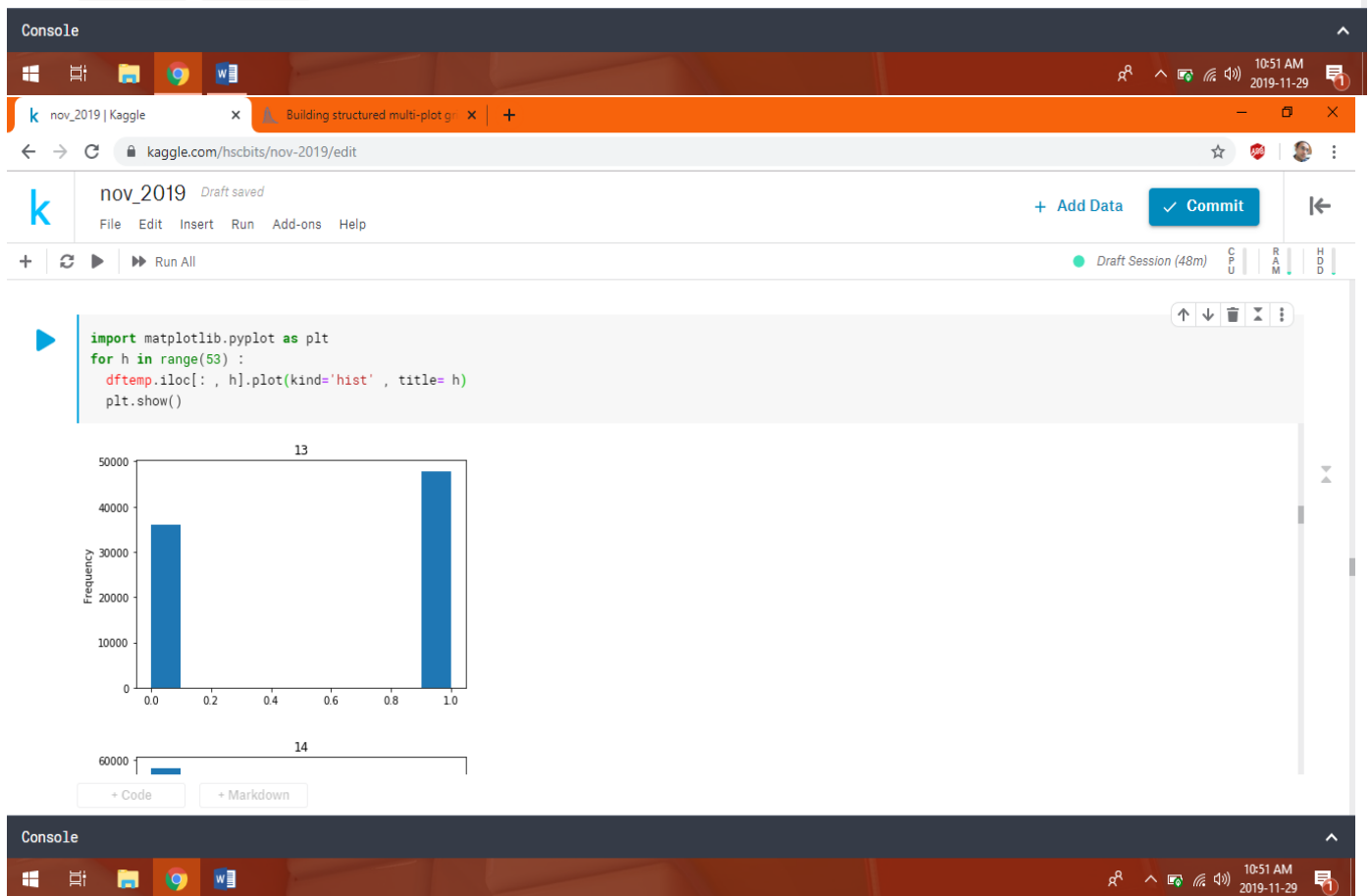
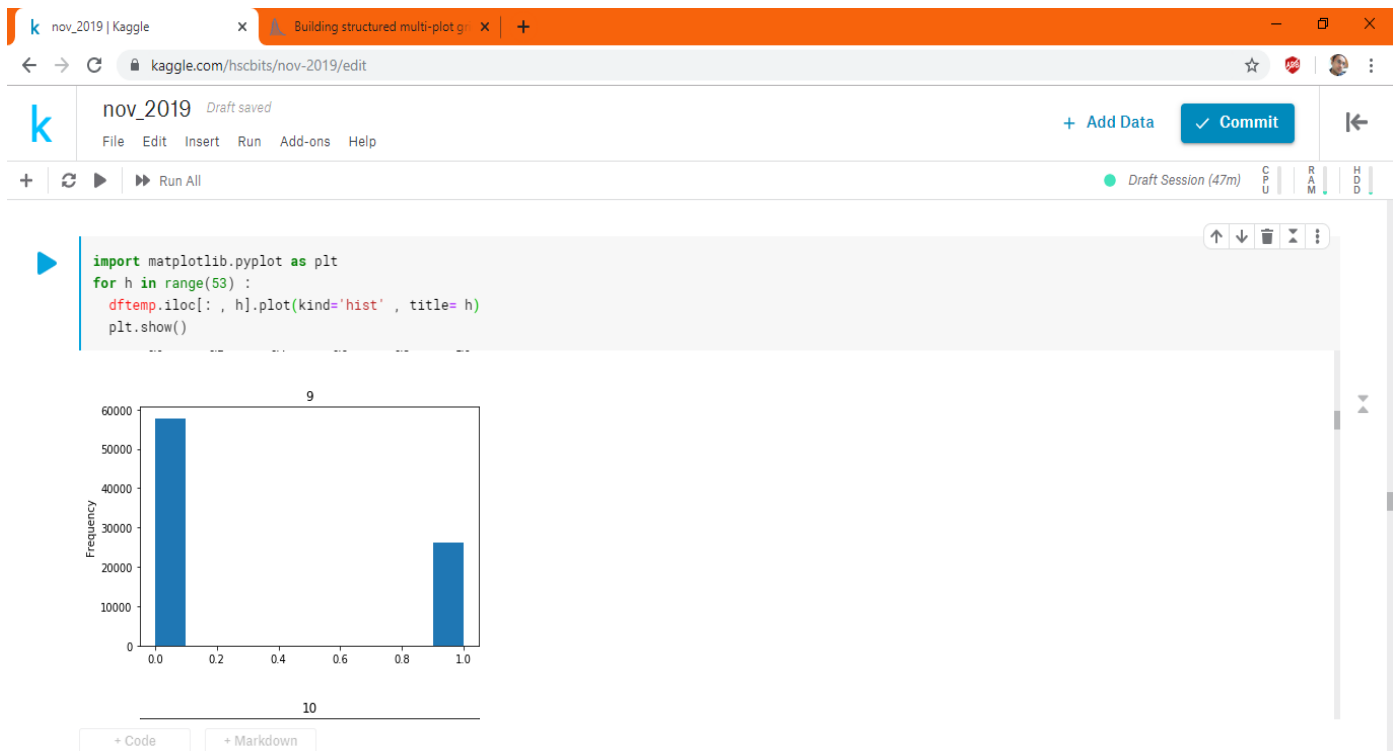
+ Code + Markdown

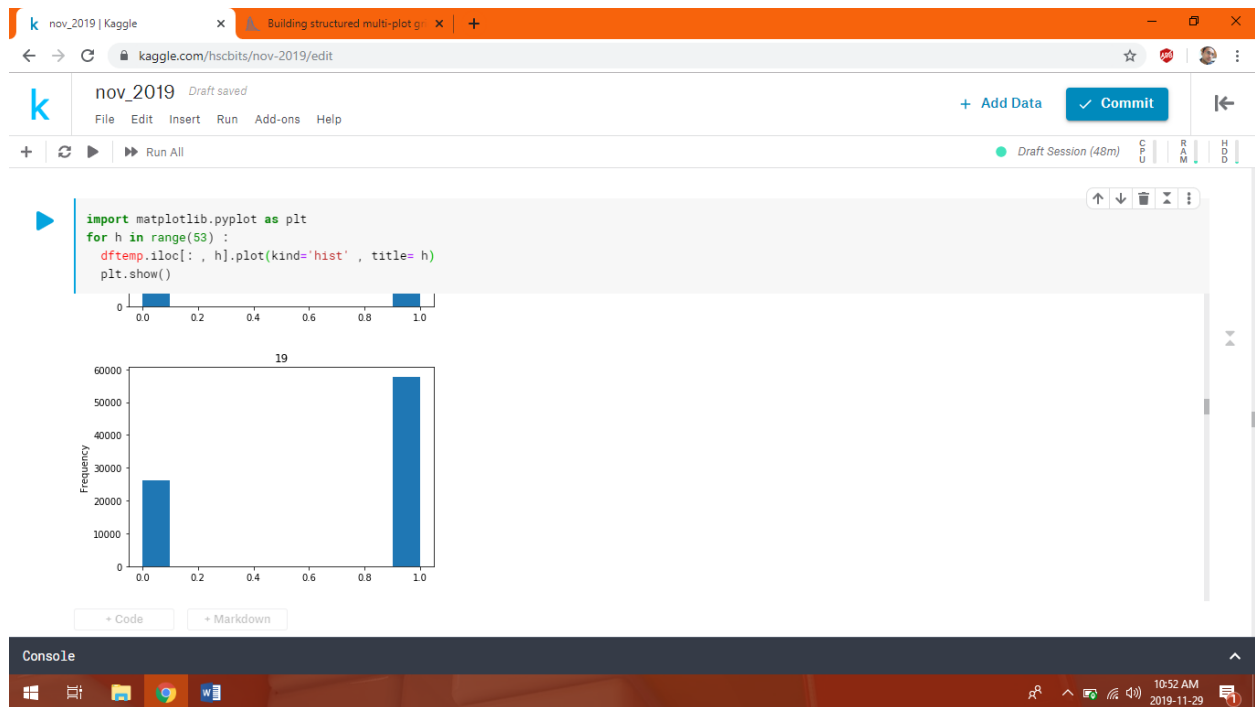
Console

10:42 AM 2019-11-29

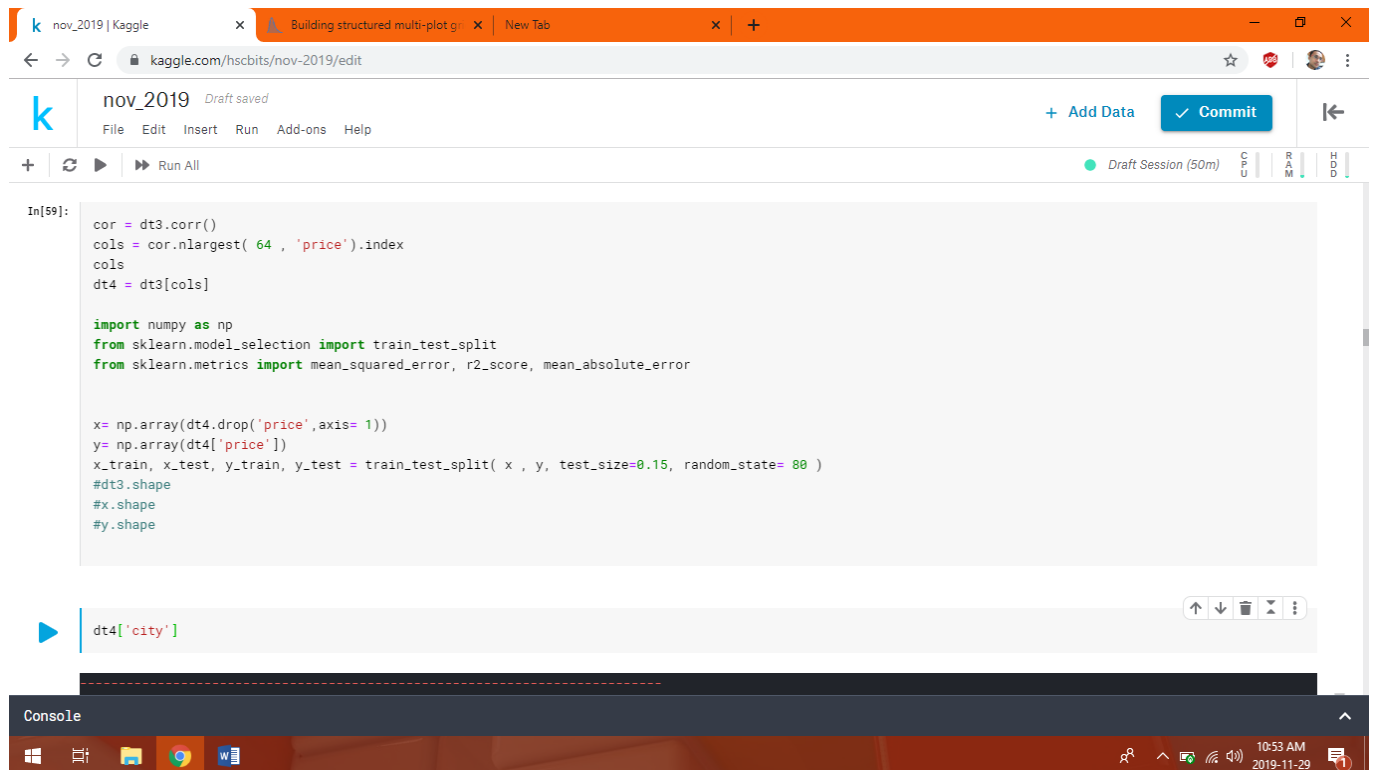
Distribution of the new features created for amenities' availability in the houses is shown graphically below in the attached screenshots.







Selecting top price influencing features as shown below.



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

The screenshot shows a Kaggle notebook interface. The top bar indicates the notebook is named 'nov_2019' and is in a 'Draft saved' state. The main code area displays a Python snippet that attempts to access a column named 'city' using `self.columns.get_loc(key)`. This results in a `KeyError: 'city'` exception. The console at the bottom shows the error message. The interface includes standard notebook controls like 'Run All', 'Add Data', and 'Commit'.

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

The screenshot shows the same Kaggle notebook. A new code cell has been added, executing `dt4.select_dtypes(include='category')`. The output shows a list of categorical column indices: 0, 1, 2, 3, 4, followed by an ellipsis, and then 83845, 83846, 83847, 83848, and 83849. Below this, it states '83850 rows x 0 columns'. A second code cell is shown with the input `In[]:` and the code `cor = dt3.corr()`, `cols = cor.nlargest(50, 'price').index`, and `cols`. The console at the bottom is empty. The notebook interface remains the same with 'Draft Session (52m)' and various control buttons.

```
nov_2019 Draft saved
File Edit Insert Run Add-ons Help

+ ↺ ▶▶ Run All

sns.heatmap(dt4.isnull())

Out[68]:
<matplotlib.axes._subplots.AxesSubplot at 0x7f776bb44518>

0
2207
4414
6621
8828
11035
13242
15449
17656
19863
22070
24277
26484
28691
30898
33105
35312
37519
39726
41933
44140
46347
48554
50761
52968
55175
57382
59589
61796
64003
66210
68417
70624
72831
75038
77245
79452
81659

price
hgs_count_entire_homes
cleaning_fees
security_deposit
host_total_listings_count
Dryer
host_listings_count
Hosted host listings count
Washer
Dishes and silverware
Refrigerator
Kitchen
Cooking basics
spotless_friendly_workspace
policy_super_strict_hosts
average_review_scores_90_days
review_scores_location
reviewer_recommendations
Essentials
Fire extinguisher
carbon_monoxide_detector
hot_water
```

The screenshot shows a Kaggle notebook interface. The notebook is titled 'nov_2019' and is in a 'Draft saved' state. The code in the notebook is as follows:

```
from sklearn import linear_model

reg = linear_model.LinearRegression( n_jobs= -1 )
reg.fit(x_train, y_train)
reg.score(x_test, y_test)
pred1 = reg.predict(x_test)
print('r2 is ', r2_score(y_test, pred1))
print('mse is ', mean_squared_error(y_test, pred1))
```

The output of the code is displayed in the console:

```
r2 is 0.702847527715182
mse is 0.024873775580482694
```

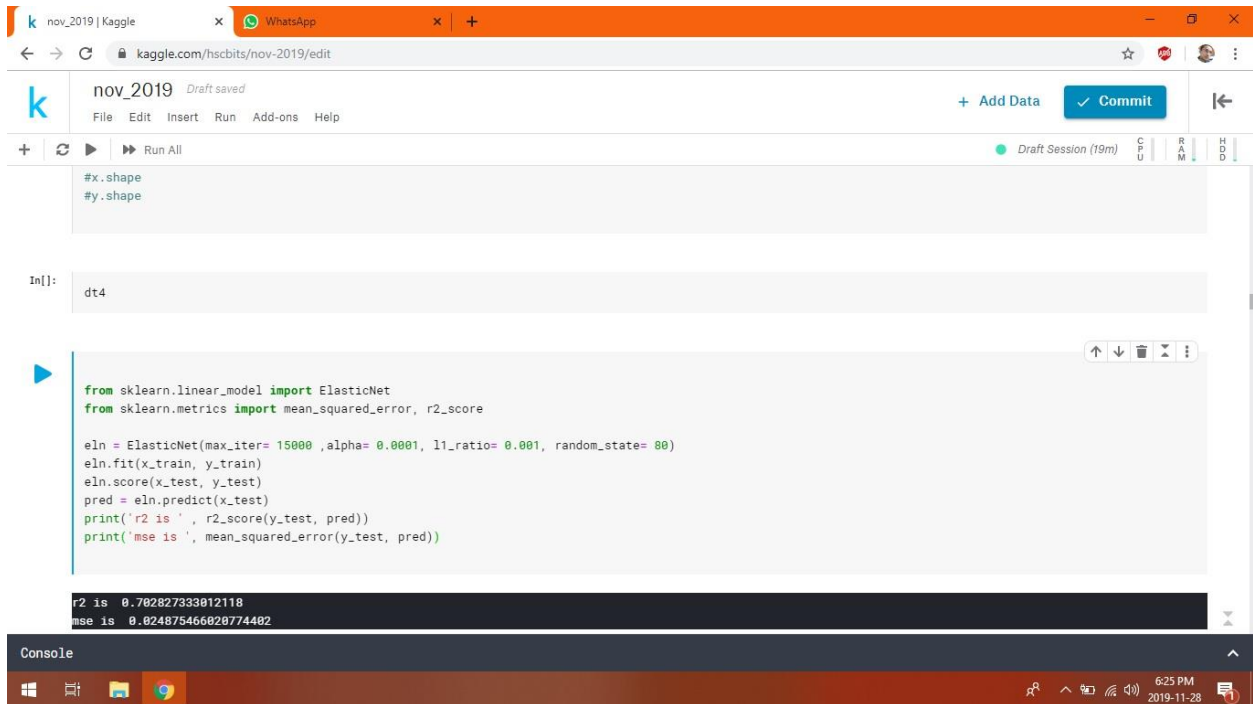
The console also shows the input for the next cell:

```
In[10]: print(mean_squared_error(y_test, pred1) / y_test.mean()*100)
```

The output of this input is:

```
4.955918316700791
```

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



The screenshot shows a Kaggle notebook titled 'nov_2019' with a 'Draft saved' status. The interface includes a top bar with 'Add Data' and 'Commit' buttons, and a 'Run All' button. The code cell 'In[]:' contains the following Python code:

```
#x.shape
#y.shape

In[ ]:
dt4

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

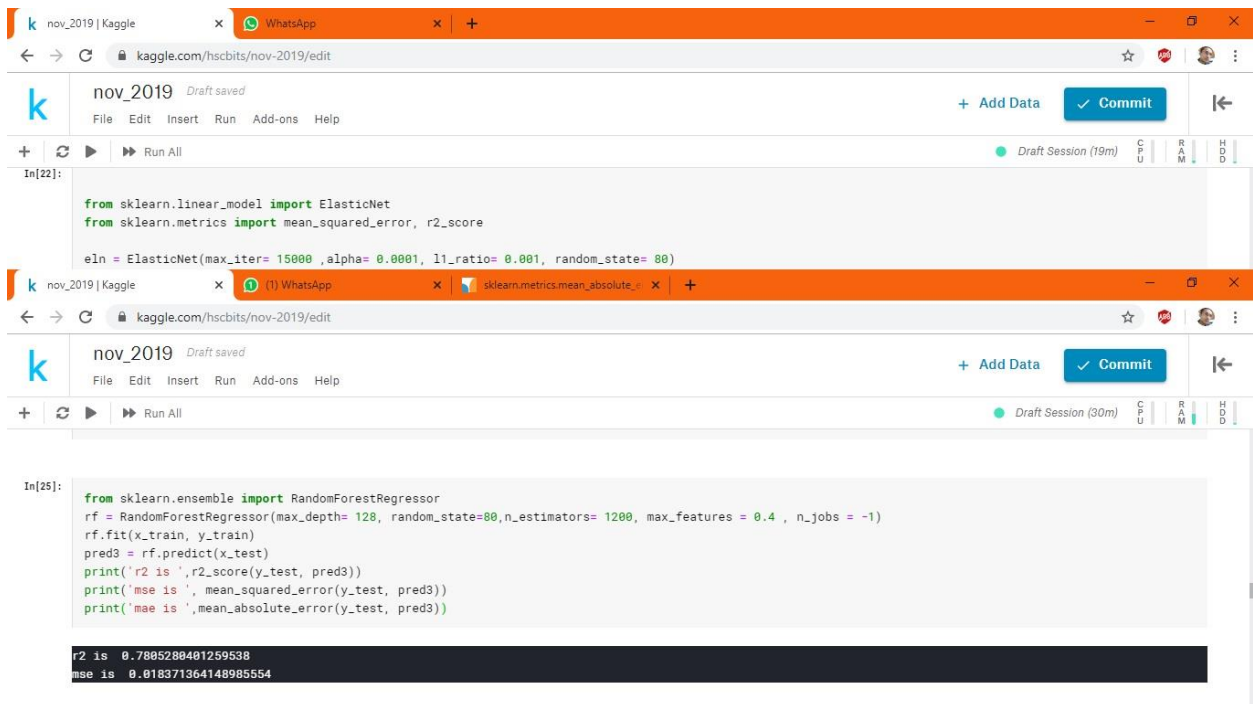
eln = ElasticNet(max_iter= 15000 ,alpha= 0.0001, l1_ratio= 0.001, random_state= 80)
eln.fit(x_train, y_train)
eln.score(x_test, y_test)
pred = eln.predict(x_test)
print(' r2 is ', r2_score(y_test, pred))
print(' mse is ', mean_squared_error(y_test, pred))
```

The output of the code is displayed in a dark box:

```
r2 is 0.70282733012118
mse is 0.024875466020774402
```

The console at the bottom shows the system tray with icons for Windows, taskbar, and system clock (6:25 PM, 2019-11-28).

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



The screenshot shows a Kaggle notebook titled 'nov_2019' with a 'Draft saved' status. The interface includes a top bar with 'Add Data' and 'Commit' buttons, and a 'Run All' button. The code cell 'In[22]:' contains the following Python code:

```
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)
```

The code cell 'In[25]:' contains the following Python code:

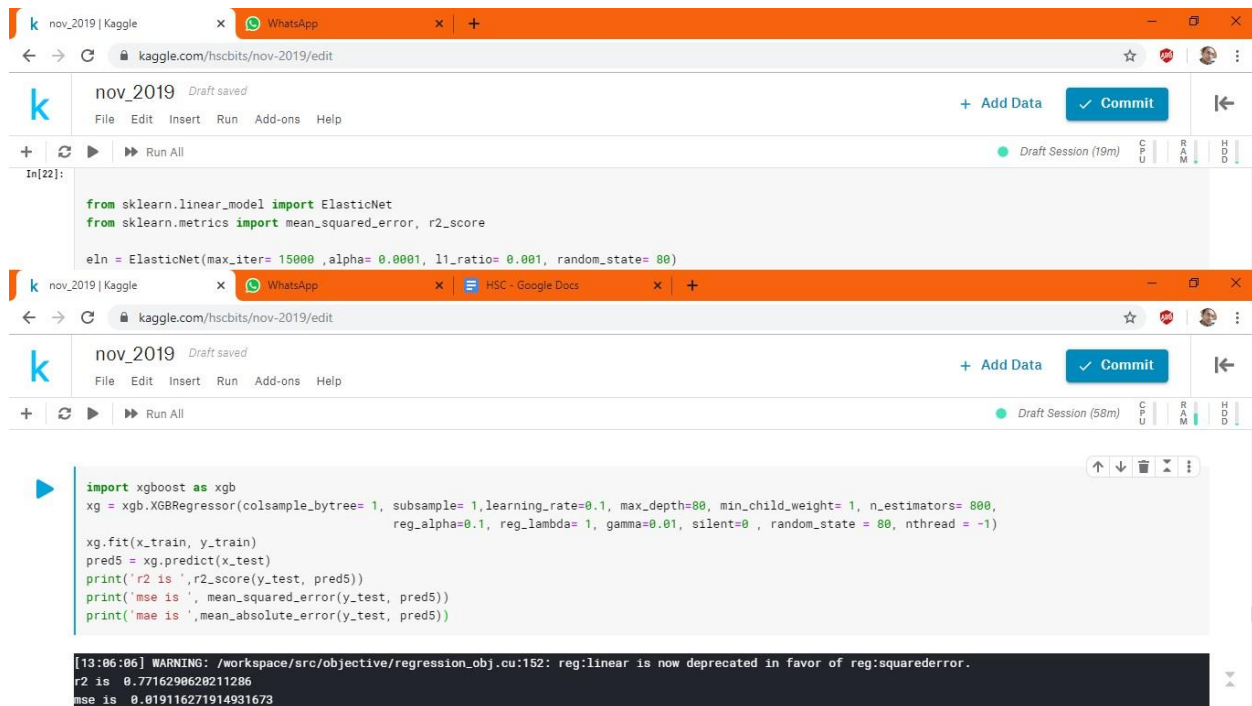
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(max_depth= 128, random_state=80,n_estimators= 1200, max_features = 0.4 , n_jobs = -1)
rf.fit(x_train, y_train)
pred3 = rf.predict(x_test)
print(' r2 is ',r2_score(y_test, pred3))
print(' mse is ', mean_squared_error(y_test,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

The output of the code is displayed in a dark box:

```
r2 is 0.7805280401259538
mse is 0.018371364148985554
```

The console at the bottom shows the system tray with icons for Windows, taskbar, and system clock (6:25 PM, 2019-11-28).

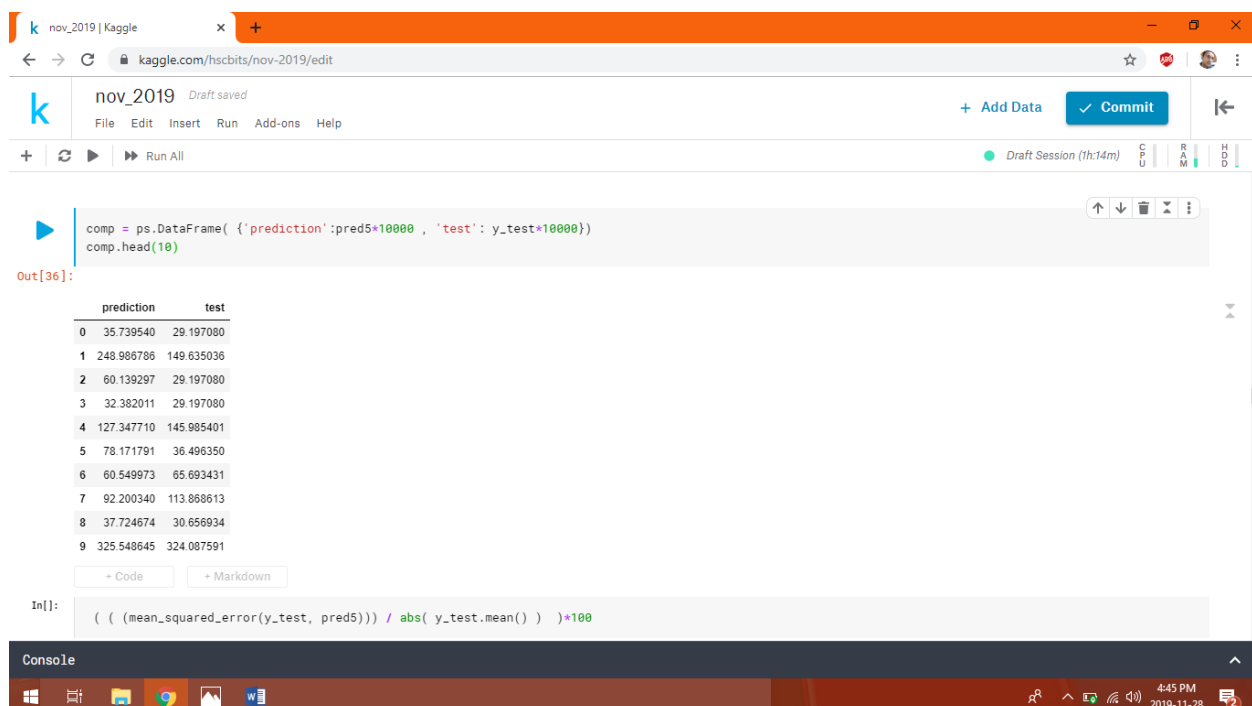
Implementation of XGBoost model and the outcomes can be observed in the following screenshot.



The screenshot shows a Kaggle notebook interface with two code cells. The first cell imports necessary libraries and initializes an ElasticNet model. The second cell imports XGBoost, fits the model, and prints performance metrics. The output shows an R-squared score of approximately 0.77 and an MSE of approximately 0.019.

```
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[23]:  
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  0.7716298628211286  
mse is  0.019116271914931673
```

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



The screenshot shows a Kaggle notebook interface with a code cell that creates a DataFrame for comparison and prints its head. The output shows a table with two columns: 'prediction' and 'test', displaying the first 10 rows of data.

```
comp = ps.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[37]:  
( ( 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 x 2 columns

```
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	92.885058	109.489051
12576	111.476518	109.489051
12577	72.646617	47.445255

12578 rows x 2 columns

Console

4:42 PM 2019-11-28

The screenshot shows a Kaggle notebook titled 'nov_2019'. The code cell contains the following Python code:

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

The output of the code cell is a table with two columns: 'prediction' and 'test'. The table shows the first five rows and the last five rows, with an ellipsis in between indicating the middle rows. The first five rows are:

	prediction	test
0	28.743737	29.197080
1	217.103901	149.635036
2	39.108393	29.197080
3	31.092304	29.197080
4	117.118965	145.985401

The last five rows are:

	prediction	test
12573	752.609891	619.708029
12574	80.443077	69.343066
12575	89.292640	109.489051
12576	119.943025	109.489051
12577	72.442045	47.445255

The console output shows the following command and result:

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

The console output is:

```
Out[23]:
```

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 !