Data Cleaning and EDA using python

By: Smith Kwabena Agyei

Case Study

Perform EDA and initial Cleaning on the dataset provided. Justify all choices and produce a document to be handed over to the modeling team that includes all plots, charts and deletions.

Steps Taken to Clean Data

After reading the csv file into a dataframe.
Using `df = pd.read_csv('/content/raw_house_data.csv')`
`I had 5000 rows and 16 columns`

➔ **Checked for duplicates**
`df.duplicated().sum() "returned 0"`

➔ **Viewed the Datatypes of the Dataset**
df.dtypes

Data Types in the Dataset

	0
MLS	int64
sold_price	float64
zipcode	int64
longitude	float64
latitude	float64
lot_acres	float64
taxes	float64
year_built	int64
bedrooms	int64
bathrooms	float64
sqrt_ft	float64
garage	float64
kitchen_features	object
fireplaces	object
floor_covering	object
HOA	object

dtype: object

Nulls in Dataset

I checked for the count of null values in all columns.

➔ **df.isnull().sum()**

This code gave me this output which show the count of records/rows with null values based on the columns in the dataset.

Nulls in Dataset (3. cont'd)

	0
MLS	0
sold_price	0
zipcode	0
longitude	0
latitude	0
lot_acres	10
taxes	0
year_built	0
bedrooms	0
bathrooms	6
sqrt_ft	56
garage	7
kitchen_features	33
fireplaces	0
floor_covering	1
HOA	562

Remove rows with Lot_acres as null

- ➔ **Return indexes of Lot_acres columns with null / NaN values**

```
vals = [np.NaN]
mask = df["lot_acres"].isin(vals)
df[mask].index
```

- ➔ **this removes all records/rows with null value in the Lot_acres column. (This is because the Lot_Acres column is not consistent with the Data) 10 records affected (4990)**

```
df = df.drop(df.index[df[mask].index],
axis=0)
```

Update all columns having null values

➔ **Updating some few Columns[Garage,fireplaces,HOA,Kitchen_Features,Floor_Covering] with 0 / "Nothing" where the value is null or empty**

```
df["garage"] = df["garage"].fillna(0)
df["HOA"] = df["HOA"].fillna(0)
df["fireplaces"] =
df["fireplaces"].replace(" ", 0)
```

➔ **updating Kitchen_features and Floor_covering to Nothing where value is null**

```
df["kitchen_features"] =
df["kitchen_features"].replace(np.NaN,
"Nothing")
df["floor_covering"] =
df["floor_covering"].replace(np.NaN,
"Nothing")
df
```

Updating bathroom column with values

➔ **Fixing null / empty values for the bathroom column**

```
zeroBathrooms =
df[df["bathrooms"].isnull()]
#The immediate code snippet assigns
the columns with null values for
bathrooms to variable zeroBathrooms
zeroBathrooms
zBathrmsIndex = zeroBathrooms.index
#The index of the columns is stored in
a variable so that the records can be
accessible using their indexes
zBathrmsIndex
```

Updating bathroom column with values (Cont'd)

➔ **Run a loop to fill the null values of the bathroom columns**

```
for i in zBathrmsIndex:
bathVals = df[(df["bedrooms"] ==
df.loc[i,
"bedrooms"])][["bathrooms"]].mode()
# for every record the mode for
bathrooms that have the same number of
bedrooms is assigned to a variable
bathVals = bathVals["bathrooms"]
# The bathroom column of the bathVals
dataframe is assigned to the variable
df.loc[i, "bathrooms"] = bathVals[0]
# The value of bathVals is assigned to
the respective bathroom columns of the
indexes
print(df.loc[i, "bathrooms"]) # The
update values are printed out.
```

Updating Sqrt_ft column with values

➔ **Fixing null / empty values for the Sqrt_ft column**

```
nSqrft = df[df["sqrt_ft"].isnull()]
#The immediate code snippet assigns
the columns with null values for
bathrooms to variable nSqrft
nSqrft
nSqrftIndex = nSqrft.index
#The index of the columns is stored in
a variable so that the records can be
accessible using their indexes
nSqrftIndex
```

Updating Sqrt_Ft column with values (Cont'd)

➔ **Run a loop to fill the null values of the Sqrt_ft columns**

```
for i in nSqrftIndex:
      sqrtftVals = df[(df["bedrooms"]
== df.loc[i,
"bedrooms"])][["sqrt_ft"]].mean() // 1
# for every record the mean for
sqrt_ft that have the same number of
bedrooms is assigned to a variable
sqrtftVals = sqrtftVals["sqrt_ft"]
# The sqrt_ft column of the sqrtftVals
dataframe is assigned to the variable
df.loc[i, "sqrt_ft"] = sqrtftVals #
The value of sqrtftVals is assigned to
the respective sqrt_ft columns of the
indexes
print(sqrtftVals) # The value of
sqrtftVals is assigned to the
respective sqrt_ft columns of the
indexes
```

Sum of Nulls after Cleaning

	0
MLS	0
sold_price	0
zipcode	0
longitude	0
latitude	0
lot_acres	0
taxes	0
year_built	0
bedrooms	0
bathrooms	0
sqrt_ft	0
garage	0
kitchen_features	0
fireplaces	0
floor_covering	0
HOA	0

dtype: int64

Aggregation and Renaming Columns

Run aggregations on the dataframe

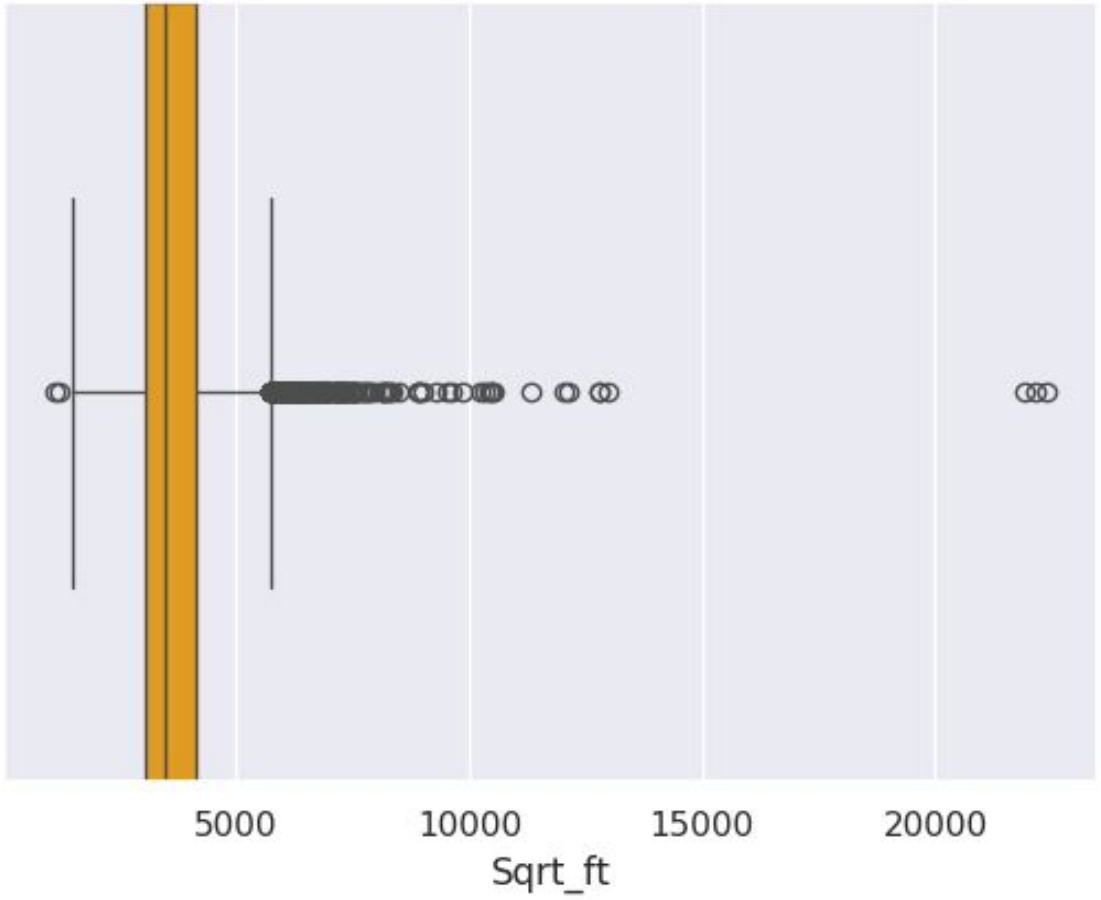
```
#Renaming columns

df = df.rename(columns={'sold_price' : 'Sold_Price', 'zipcode' : 'Zipcode', 'longitutde' :
'Longitutde', 'latitude' : 'Latitude', 'lot_acres' : 'Lot_Acres','taxes' : 'Taxes',
'year_built':'Year_Built', 'bedrooms' : 'Bedrooms', 'bathrooms' : 'Bathrooms', 'sqrt_ft' :
'Sqrt_ft', 'garage' : 'Garage',          'kitchen_features' : 'Kitchen_Features',
'fireplaces' : 'Fireplaces', 'floor_covering' : 'Floor_Covering'})

df.describe()
```

	MLS	Sold_Price	Zipcode	Longitude	Latitude	Lot_Acres	Taxes	Year_Built	Bedrooms	Bathrooms	Sqrt_ft	Garage	Fireplaces	HOA
count	4.990000e+03	4.990000e+03	4990.000000	4990.000000	4990.000000	4990.000000	4.990000e+03	4990.000000	4990.000000	4990.000000	4990.000000	4990.000000	4990.000000	4972.000000
mean	2.130720e+07	7.749513e+05	85723.223447	-110.911893	32.309526	4.661317	9.412291e+03	1992.316433	3.935471	3.829359	3716.715351	2.812024	1.879158	73.480376
std	2.257876e+06	3.187799e+05	37.838772	0.120623	0.176727	51.685230	1.731116e+05	65.542978	1.245817	1.387561	1147.044442	1.197368	1.140038	90.927379
min	3.042851e+06	1.690000e+05	85118.000000	-112.520168	31.356362	0.000000	0.000000e+00	0.000000	1.000000	1.000000	1100.000000	0.000000	0.000000	0.000000
25%	2.140750e+07	5.850000e+05	85718.000000	-110.979109	32.277974	0.580000	4.807892e+03	1987.000000	3.000000	3.000000	3049.000000	2.000000	1.000000	0.000000
50%	2.161501e+07	6.750000e+05	85737.000000	-110.923309	32.318570	0.990000	6.227810e+03	1999.000000	4.000000	4.000000	3506.000000	3.000000	2.000000	44.000000
75%	2.180494e+07	8.367500e+05	85749.000000	-110.859025	32.394625	1.757500	8.091920e+03	2006.000000	4.000000	4.000000	4125.000000	3.000000	3.000000	122.000000
max	2.192856e+07	5.300000e+06	86323.000000	-109.454637	34.927884	2154.000000	1.221508e+07	2019.000000	36.000000	36.000000	22408.000000	30.000000	9.000000	925.000000

FINDING OUTLIERS

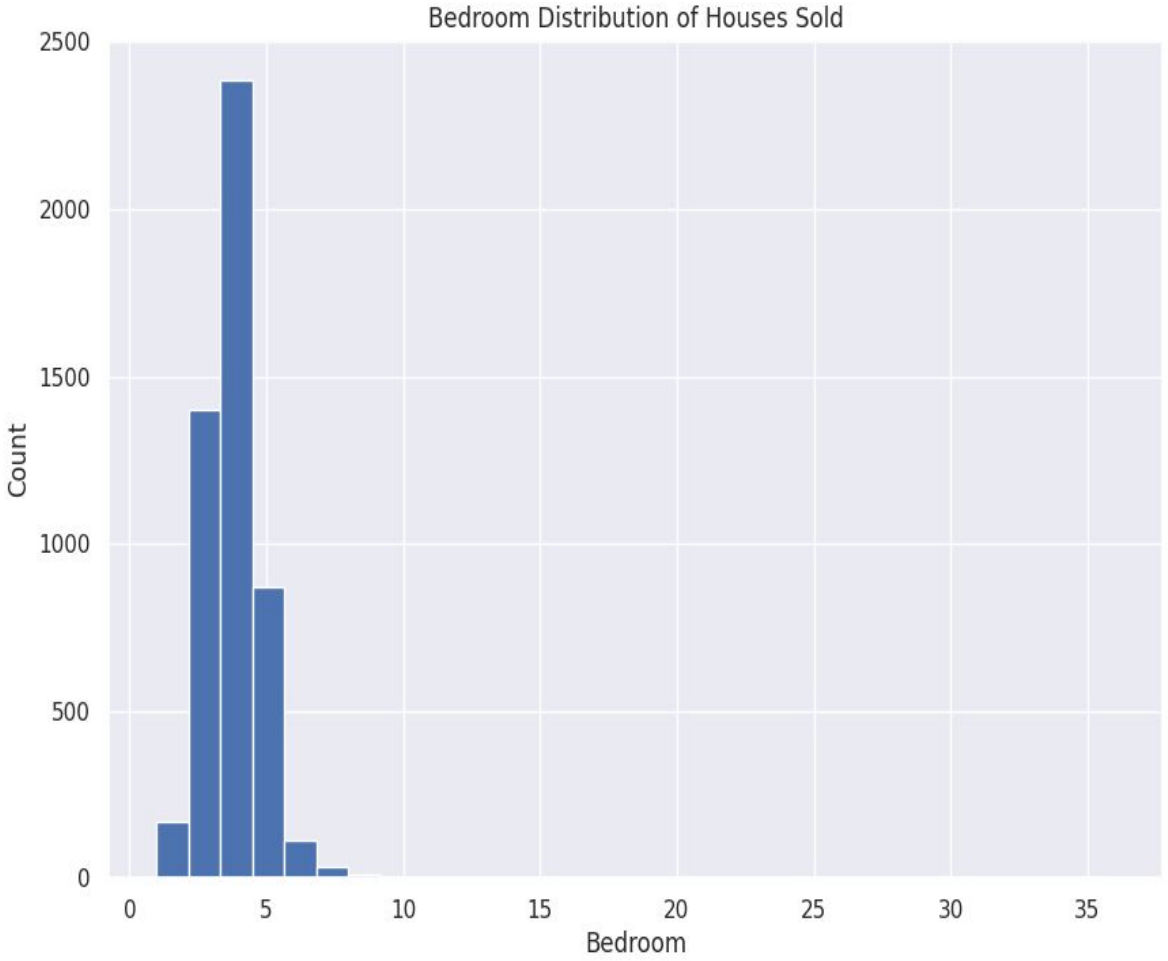
This Box plot is based on the Square Feet of houses sold.

```
sns.boxplot(df, x='Sqrt_ft',
color='orange', width=1)
```

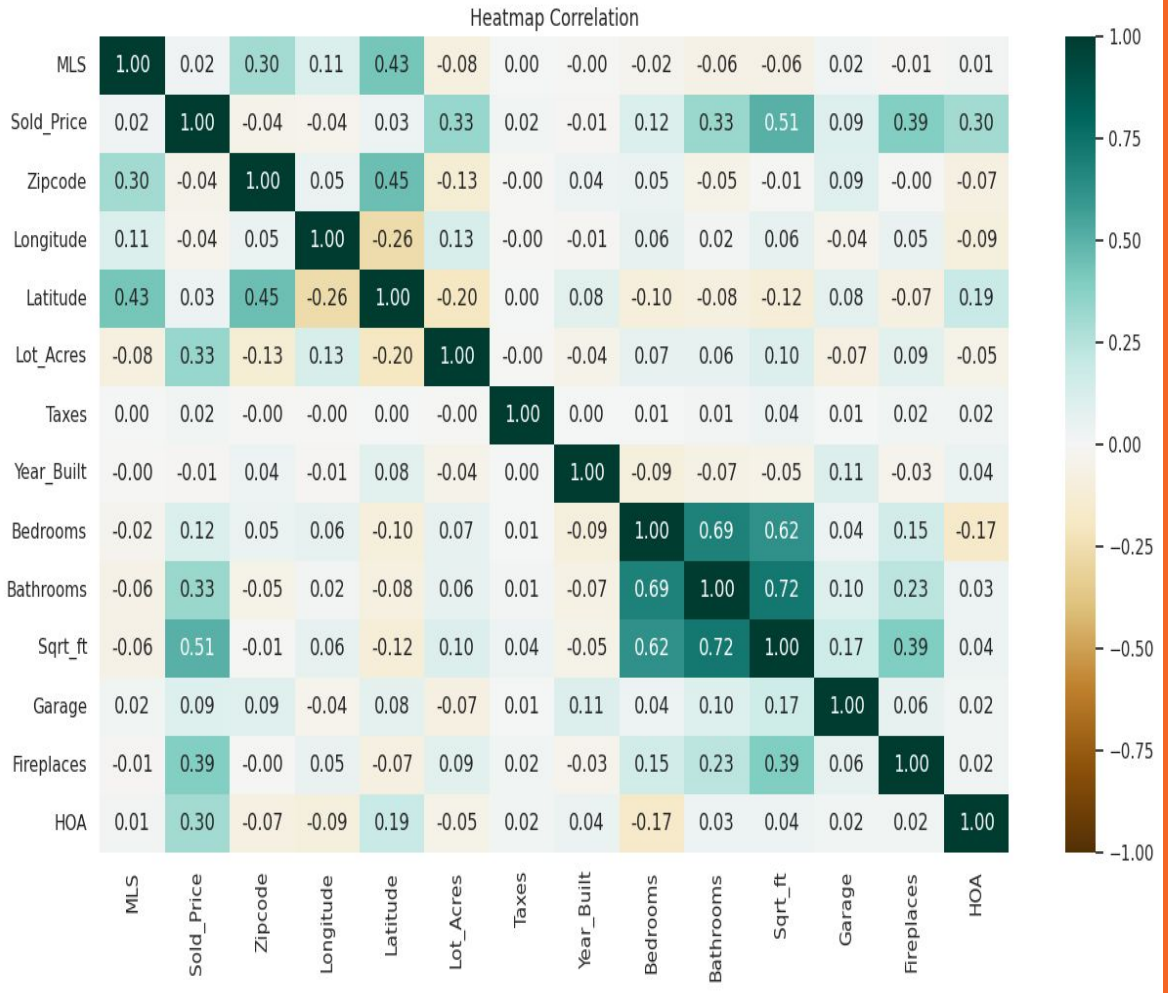


This Shows the count distribution of house solde based on the number of bedrooms

```
plt.figure(figsize=(10,7))
plt.hist(df['Bedrooms'], bins=30)
plt.title('Bedroom
Distribution of Houses Sold')
plt.xlabel('Bedroom')
plt.ylabel('Count')
```

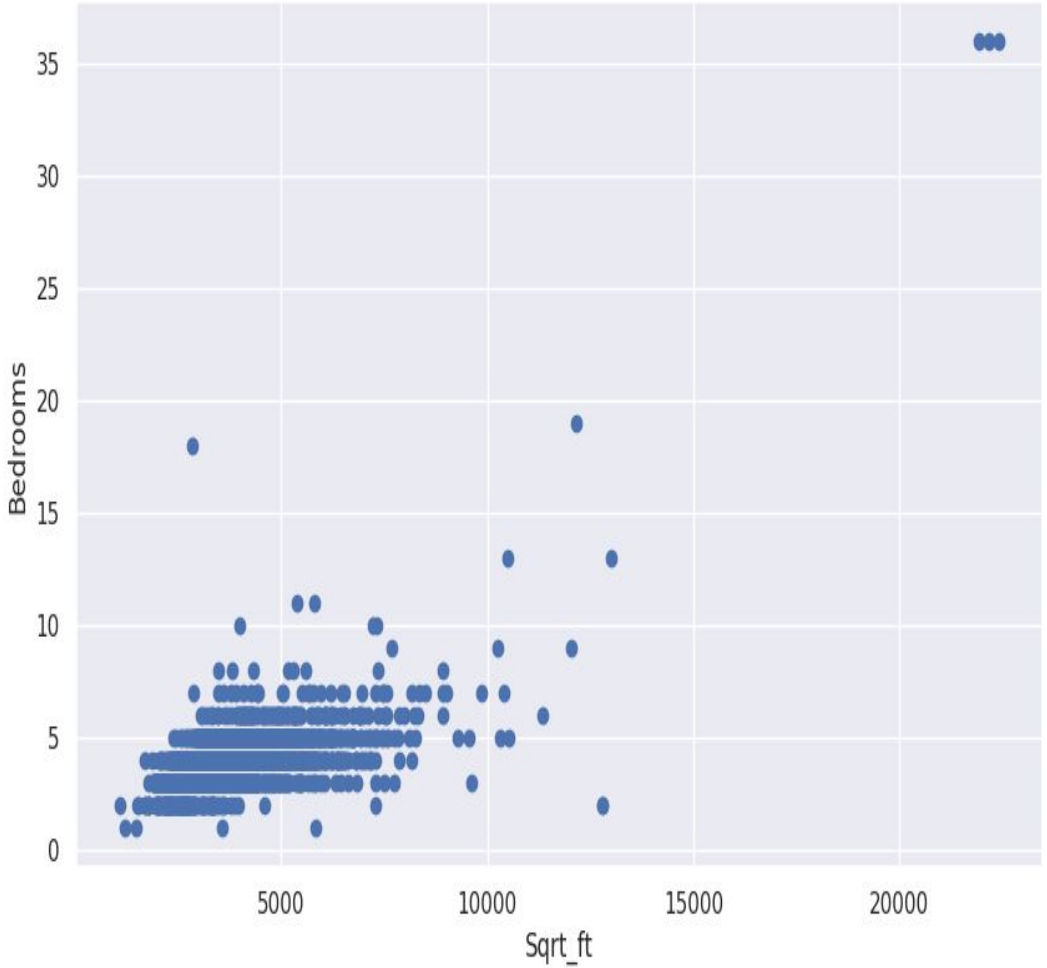


```
# Columns to generate the HeatMap on.
df =
df[['MLS','Sold_Price','Zipcode','Long
itude','Latitude','Lot_Acres','Taxes',
'Year_Built','Bedrooms','Bathrooms','S
qrt_ft','Garage','Fireplaces','HOA']]

plt.figure(figsize=(15,8))
cor = df.corr()
sns.heatmap(cor, cmap="BrBG",
annot=True, fmt=".2f", vmin=-1,
vmax=1)
plt.title('Heatmap Correlation')
```



Created a scatter plot using Matplotlib to visualize the relationship between the two variables `Sqrt_ft` (presumably the square footage of properties) and `Bedrooms` (the number of bedrooms).

```
fig, ax = plt.subplots(figsize=(10, 6))# figure with dimensions and subplots
ax.scatter(df['Sqrt_ft'], df['Bedrooms'])
#scatter plot is created
plt.title('Bedroom Distribution of Houses Sold')
ax.set_xlabel('Sqrt_ft') label set for x-axis
ax.set_ylabel('Bedrooms') label set for y-axis
```



Project Summary

Reading the Data: Use `pd.read_csv()` to read the data (5000 rows * 16 columns)
Handling Missing Values: Use `df.isnull()` or `df.info()` to detect missing data, then apply strategies like removing rows (`df.dropna()`), filling with mean/median/mode (`df.fillna()`).
Removing Duplicates: Identify and remove duplicate rows with `df.duplicated()` and `df.drop_duplicates()`, ensuring data consistency.
Data Type Conversion: Checked data types and no converted two columns to float (HOA, Fireplaces]
Outlier Detection: Detect outliers using statistical methods like Z-scores or IQR, and handle them by either removing or capping the values within a range.
Normalizing/Scaling Data: Scaled numeric columns using libraries like Pandas techniques to ensure features have similar ranges, improving model performance.
Clean Data : `df.to_csv('Cleaned_House_dataset.csv', index=True)` to save the cleaned data in csv ignoring the indexes (4990 rows * 16 columns)

THANK YOU