Housing Prediction Analysis

September 21, 2024

[2]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import neighbors
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree
import plotly.express as px
```

[3]:
```python
df = pd.read_csv("house.csv")

df.head(5)
```

[3]:
```
   bedroom_count    net_sqm  center_distance  metro_distance  floor  age  \
0              1  26.184098          1286.68      204.003817     22   67
1              1  34.866901          1855.25      186.980360      8   30
2              1  36.980709           692.09      111.224999     24   24
3              1  17.445723          1399.49      237.998760      1   66
4              1  52.587646            84.65      100.996400     20    3

          price
0  96004.804557
1  92473.722568
```

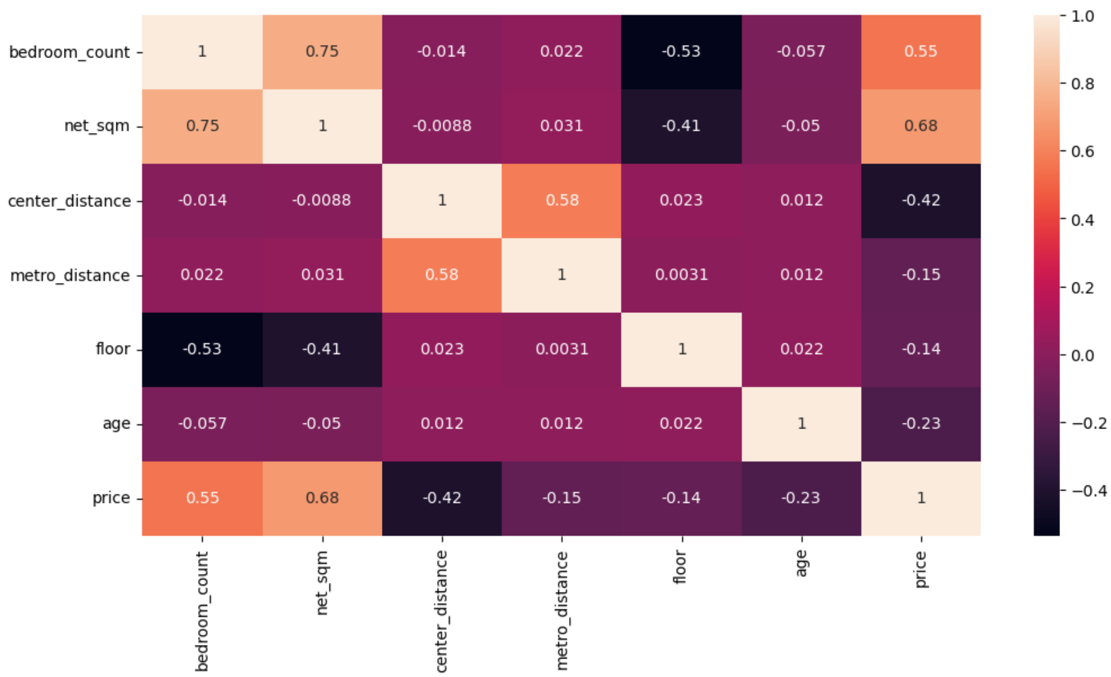
```
2   98112.519942
3   92118.326874
4   98976.653176
```

[4]:
```
df.isna().sum()
```

[4]:
```
bedroom_count      0
net_sqm            0
center_distance    0
metro_distance     0
floor              0
age                0
price              0
dtype: int64
```

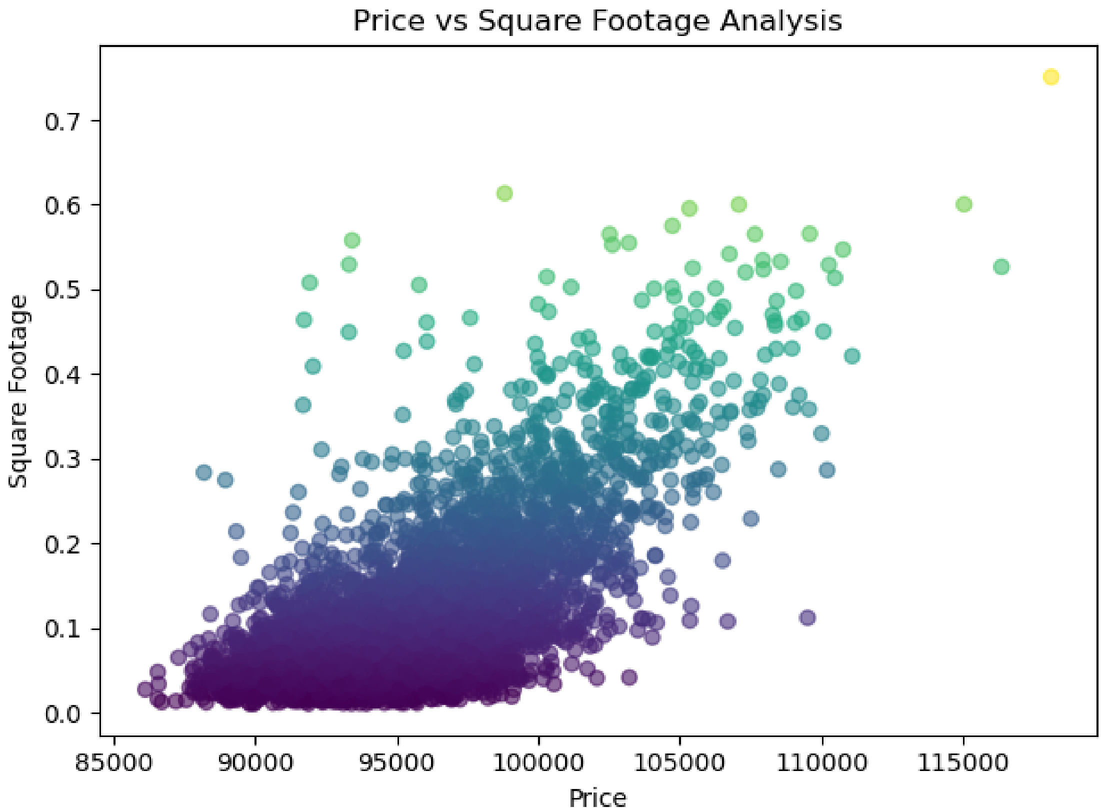
[5]:
```
plt.figure(figsize = (12,6))
sns.heatmap(df.corr(), annot = True)
```

[5]:
```
<Axes: >
```

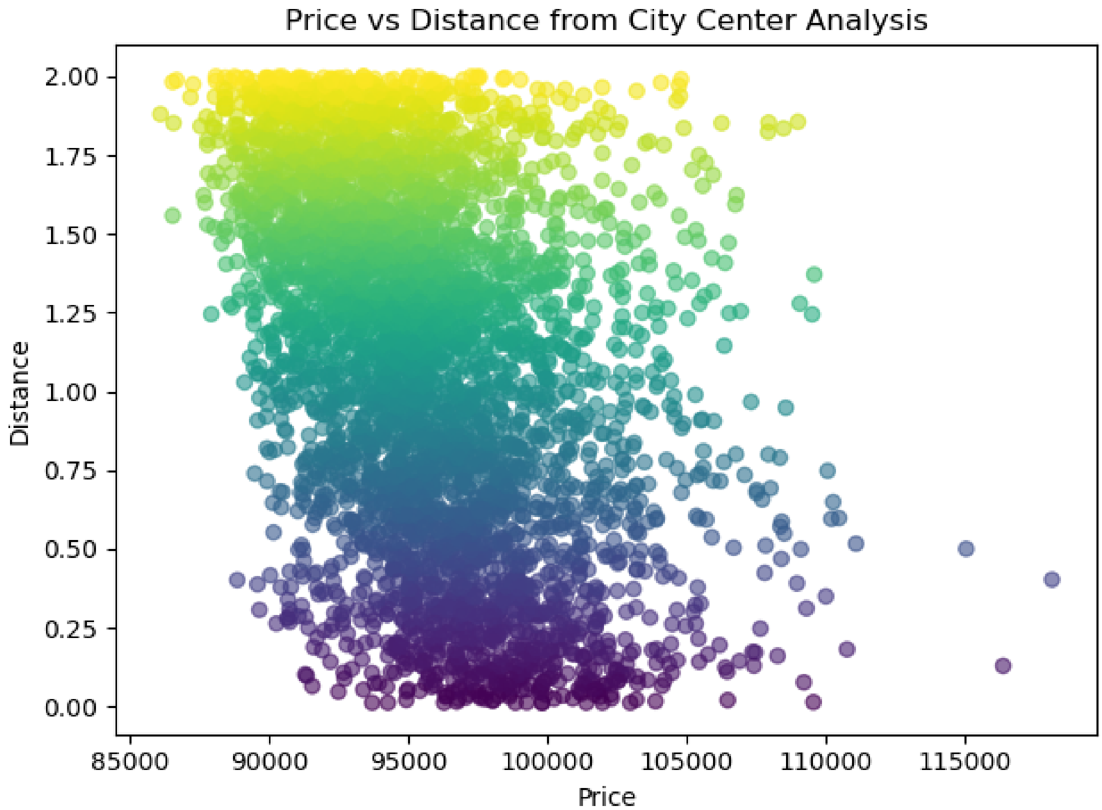


[6]:
```
plt.scatter(df["price"], df["net_sqm"]/1000, s=35, alpha=0.6,
            c=df["net_sqm"]/1000)
plt.title("Price vs Square Footage Analysis")
plt.xlabel("Price")
plt.ylabel("Square Footage")
```

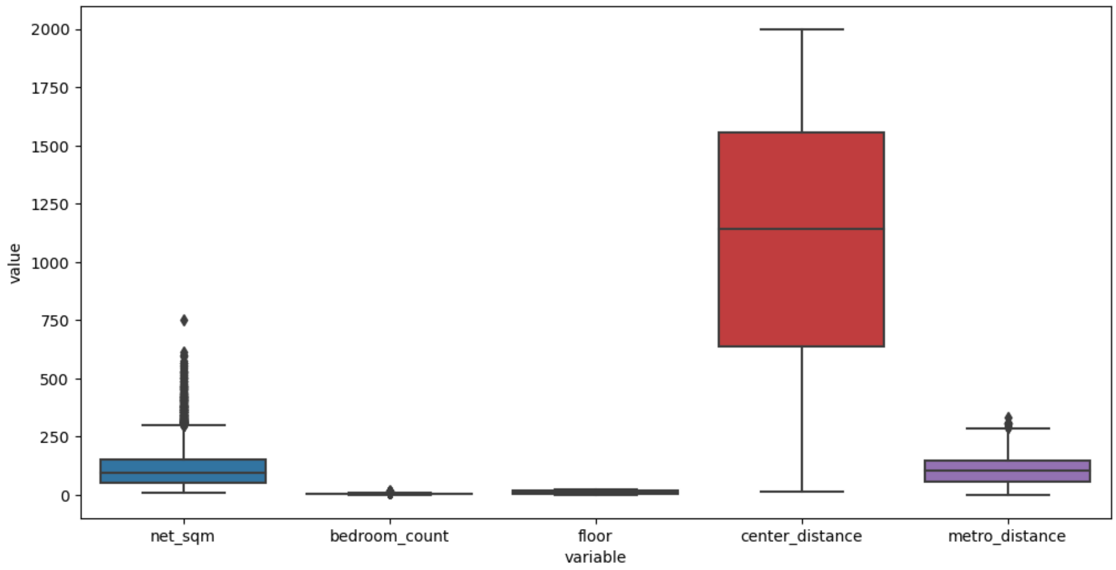
```
plt.tight_layout()
plt.show()
```



[7]:
```
plt.scatter(df["price"], df["center_distance"]/1000, s=35, alpha=0.6,
            c=df["center_distance"]/1000)
plt.title("Price vs Distance from City Center Analysis")
plt.xlabel("Price")
plt.ylabel("Distance")
plt.tight_layout()
plt.show()
```



[8]:
```
plt.figure(figsize = (12,6))
box_df = pd.DataFrame(data = df, columns = ["net_sqm", "bedroom_count", "floor",
                                            "center_distance", "metro_distance"])
sns.boxplot(x = "variable", y = "value", data = pd.melt(box_df))
plt.show()
```



[9]:
```
X = df.drop('price', axis = 1)
y = df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

[10]:
```
target_model = LinearRegression()
target_model.fit(X_train, y_train)
```

[10]: LinearRegression()

[11]:
```
pd.DataFrame(target_model.coef_, X.columns, columns = ['Coef'])
```

[11]:

	Coef
bedroom_count	291.739147
net_sqm	25.502881
center_distance	-3.355592
metro_distance	6.424707
floor	119.469723
age	-25.931095

[12]:
```
test_predictions = target_model.predict(X_test)
train_predictions = target_model.predict(X_train)

print('Test Metrics:')
print('R2', metrics.r2_score(y_test, test_predictions))
print('RMSE', metrics.mean_squared_error(y_test, test_predictions, squared =␣
 ↪False))
print('MAE', metrics.mean_absolute_error(y_test, test_predictions))
```

```
print('\nTrain Metrics:')
print('R2', metrics.r2_score(y_train, train_predictions))
print('RMSE', metrics.mean_squared_error(y_train, train_predictions, squared =
 ↪False))
print('MAE', metrics.mean_absolute_error(y_train, train_predictions))
```

```
Test Metrics:
R2 0.7388983935638092
RMSE 1997.6695914810248
MAE 1422.5810848136125

Train Metrics:
R2 0.7135721895863436
RMSE 2097.6717838827267
MAE 1488.8936592324578
```

```
/home/65c9f9d3-081c-46ec-823e-52cd3305b641/.local/lib/python3.11/site-
packages/sklearn/metrics/_regression.py:483: FutureWarning: 'squared' is
deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean
squared error, use the function'root_mean_squared_error'.
  warnings.warn(
/home/65c9f9d3-081c-46ec-823e-52cd3305b641/.local/lib/python3.11/site-
packages/sklearn/metrics/_regression.py:483: FutureWarning: 'squared' is
deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean
squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

[13]:
```
X = sm.add_constant(df[["age", "net_sqm", "bedroom_count", "floor",
 ↪"center_distance", "metro_distance"]])
y = df['price']
model = sm.OLS(y, X).fit()
predictions = model.predict(X)
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.719
Model:                            OLS   Adj. R-squared:                  0.719
Method:                 Least Squares   F-statistic:                     1835.
Date:                Sat, 21 Sep 2024   Prob (F-statistic):               0.00
Time:                        22:56:10   Log-Likelihood:                -39021.
No. Observations:                4308   AIC:                         7.806e+04
Df Residuals:                    4301   BIC:                         7.810e+04
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
===
```

```
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
const              9.46e+04    129.991    727.773      0.000    9.43e+04
9.49e+04
age                -26.1286      1.149    -22.739      0.000     -28.381
-23.876
net_sqm             25.1994      0.505     49.909      0.000      24.209
26.189
bedroom_count      308.5671     21.016     14.682      0.000     267.364
349.770
floor              121.6607      4.989     24.385      0.000     111.880
131.442
center_distance     -3.3608      0.070    -47.908      0.000      -3.498
-3.223
metro_distance       6.9984      0.642     10.909      0.000       5.741
8.256
==============================================================================
Omnibus:                      641.697   Durbin-Watson:                   1.956
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7305.816
Skew:                          -0.321   Prob(JB):                         0.00
Kurtosis:                       9.347   Cond. No.                     5.08e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.08e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[14]:
```
real_df = pd.read_csv("real_estate.csv")

real_df.head(1)
```

```
/tmp/ipykernel_1771/3674078920.py:1: DtypeWarning: Columns (7,8) have mixed
types. Specify dtype option on import or set low_memory=False.
  real_df = pd.read_csv("real_estate.csv")
```

[14]:
```
  Date Recorded  List Year     Town        Address  Assessed Value  \
0    2021-04-14       2020  Ansonia  323 BEAVER ST        133000.0

   Sale Amount  Sales Ratio Property Type Residential Type  Longitude  \
0     248400.0       0.5354   Residential    Single Family -73.06822

   Latitude
0  41.35014
```

[15]:
```python
real_df.columns
```

[15]:
```
Index(['Date Recorded', 'List Year', 'Town', 'Address', 'Assessed Value',
       'Sale Amount', 'Sales Ratio', 'Property Type', 'Residential Type',
       'Longitude', 'Latitude'],
      dtype='object')
```

[16]:
```python
subset = real_df[["List Year", "Assessed Value", "Sale Amount", "Property␣
 ↪Type"]]

subset.head(3)
```

[16]:

	List Year	Assessed Value	Sale Amount	Property Type
0	2020	133000.0	248400.0	Residential
1	2020	110500.0	239900.0	Residential
2	2020	150500.0	325000.0	Commercial

[17]:
```python
print(subset['Property Type'].value_counts())
```

```
Property Type
Single Family     401612
Residential       112099
Condo             105420
Two Family         26408
Three Family       12586
Vacant Land         5746
Commercial          4208
Four Family         2150
Apartments           943
Industrial           533
Public Utility         8
Name: count, dtype: int64
```

[18]:
```python
subset.rename(columns={"List Year": "List_Year", "Assessed Value":␣
 ↪"Assessed_Value",
                        "Sale Amount": "Sale_Amount", "Property Type":␣
 ↪"Property_Type"}, inplace = True)

subset.head(1)
```

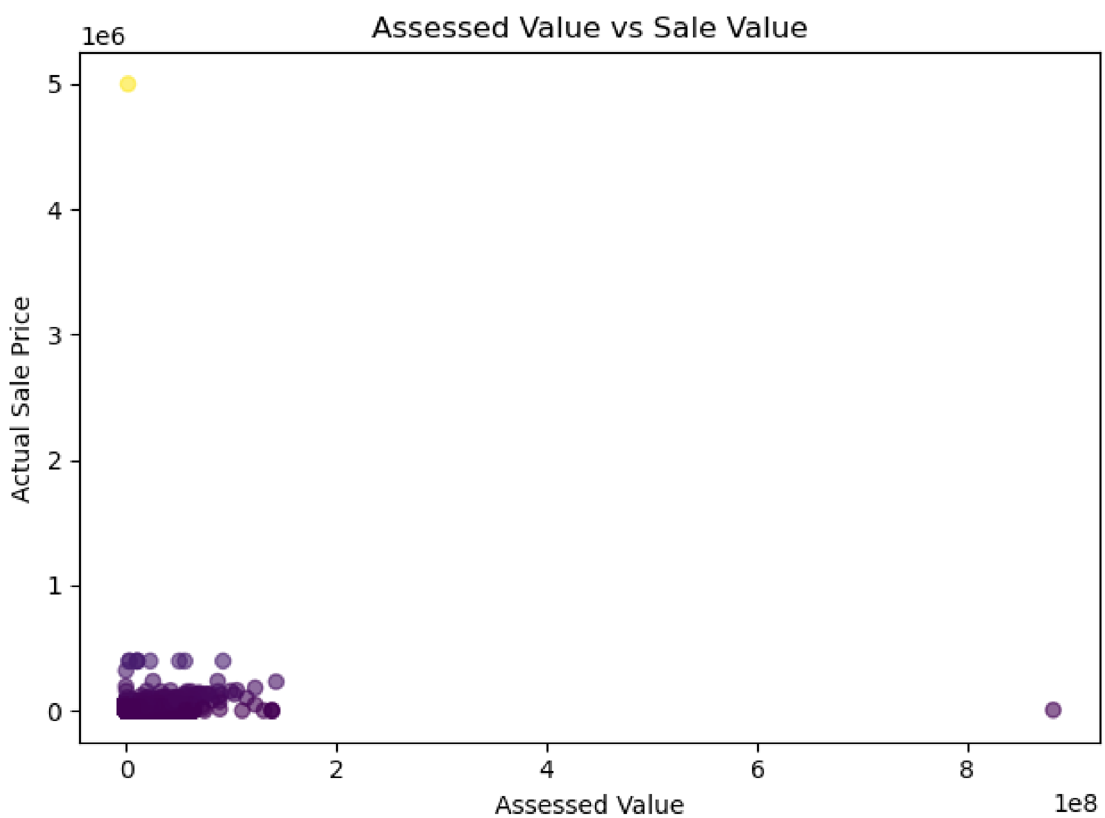
```
/tmp/ipykernel_1771/3300491291.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset.rename(columns={"List Year": "List_Year", "Assessed Value":
"Assessed_Value",
```

[18]:
```
   List_Year  Assessed_Value  Sale_Amount Property_Type
0       2020        133000.0     248400.0   Residential
```

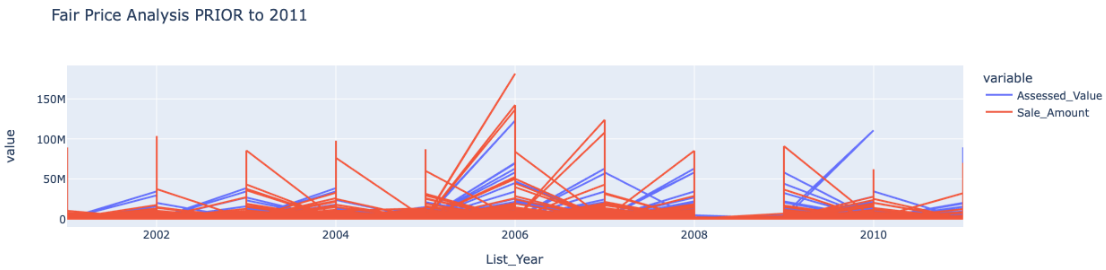
[19]:
```
subset.List_Year.min(), subset.List_Year.max()
```

[19]:
```
(2001, 2021)
```

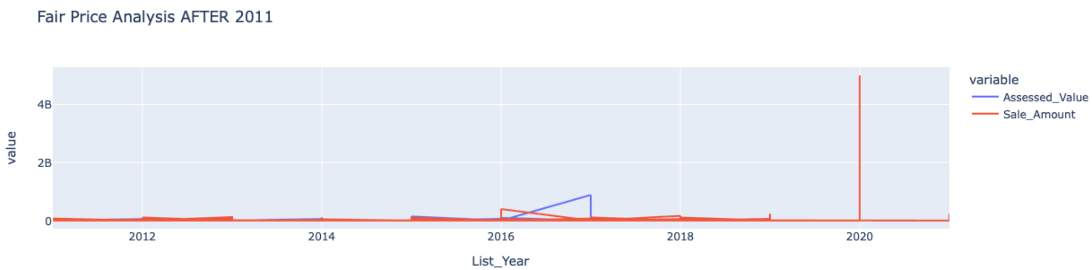
[20]:
```
plt.scatter(subset["Assessed_Value"], subset["Sale_Amount"]/1000, s=35, alpha=0.
 ↪6,
            c=subset["Sale_Amount"]/1000)
plt.title("Assessed Value vs Sale Value")
plt.xlabel("Assessed Value")
plt.ylabel("Actual Sale Price")
plt.tight_layout()
plt.show()
```



[21]:
```
fig = px.line(subset[subset.List_Year <=2011], x='List_Year',
              y = ['Assessed_Value', 'Sale_Amount'],
              title = "Fair Price Analysis PRIOR to 2011")

fig.show()
```



[22]:
```
fig = px.line(subset[subset.List_Year >=2011], x='List_Year',
              y = ['Assessed_Value', 'Sale_Amount'],
              title = "Fair Price Analysis AFTER 2011")

fig.show()
```



[23]:
```
assessed_year = subset.groupby(['List_Year']).mean(numeric_only =␣
↪True)['Assessed_Value'].sort_values().reset_index()

assessed_year
```

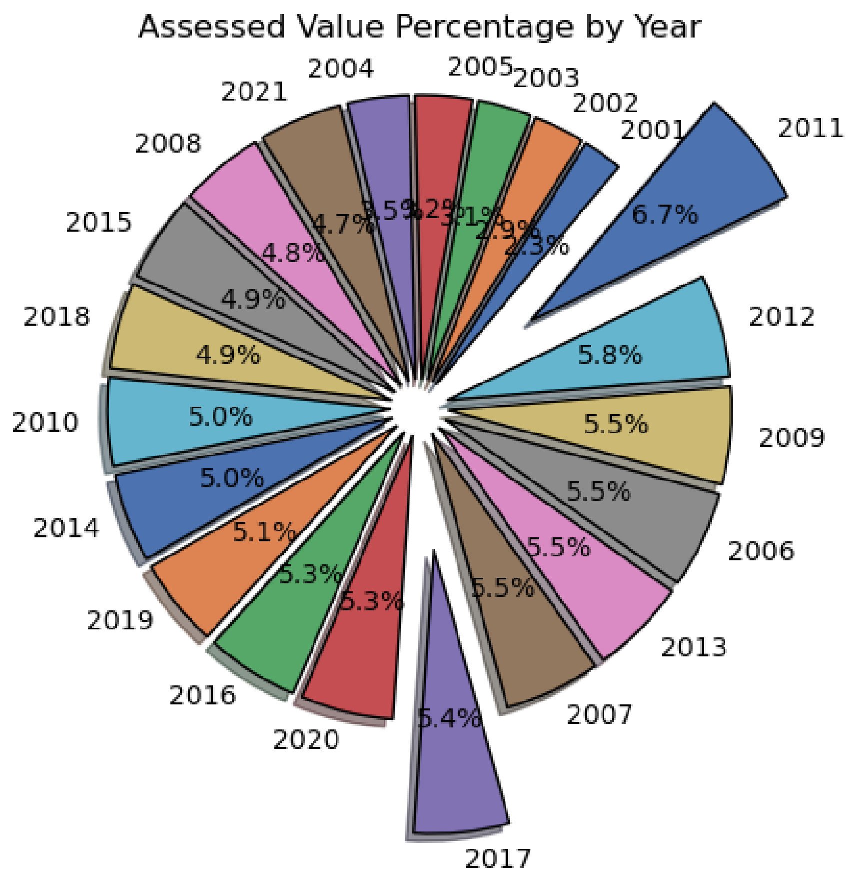
[23]:

	List_Year	Assessed_Value
0	2001	138961.045482
1	2002	177434.080594
2	2003	188067.638755
3	2005	198067.146424
4	2004	214266.720175
5	2021	290211.723317
6	2008	292110.829388
7	2015	300223.060792
8	2018	300339.526218
9	2010	307099.145621
10	2014	307782.171116

```
11        2019    309314.686637
12        2016    323196.232958
13        2020    323368.449258
14        2017    331720.028999
15        2007    334548.775079
16        2013    336146.203891
17        2006    338248.086912
18        2009    338422.314482
19        2012    356468.186279
20        2011    412067.636279
```

[24]:

```
explode = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
           0.5, 0.1, 0.1, 0.1, 0.1, 0.1, 0.5]

plt.pie(assessed_year['Assessed_Value'], labels = assessed_year['List_Year'],␣
 ↪explode = explode,
        shadow = True,
        startangle = 50,
        autopct = '%1.1f%%',
        colors = sns.color_palette('deep'),
        wedgeprops = {'edgecolor':'black'})

plt.title("Assessed Value Percentage by Year")
plt.figure(figsize = (200, 200))
plt.show()
```




```
<Figure size 20000x20000 with 0 Axes>
```

[25]:

```
sale_year = subset.groupby(['List_Year']).mean(numeric_only =
    True)['Sale_Amount'].sort_values().reset_index()

sale_year
```

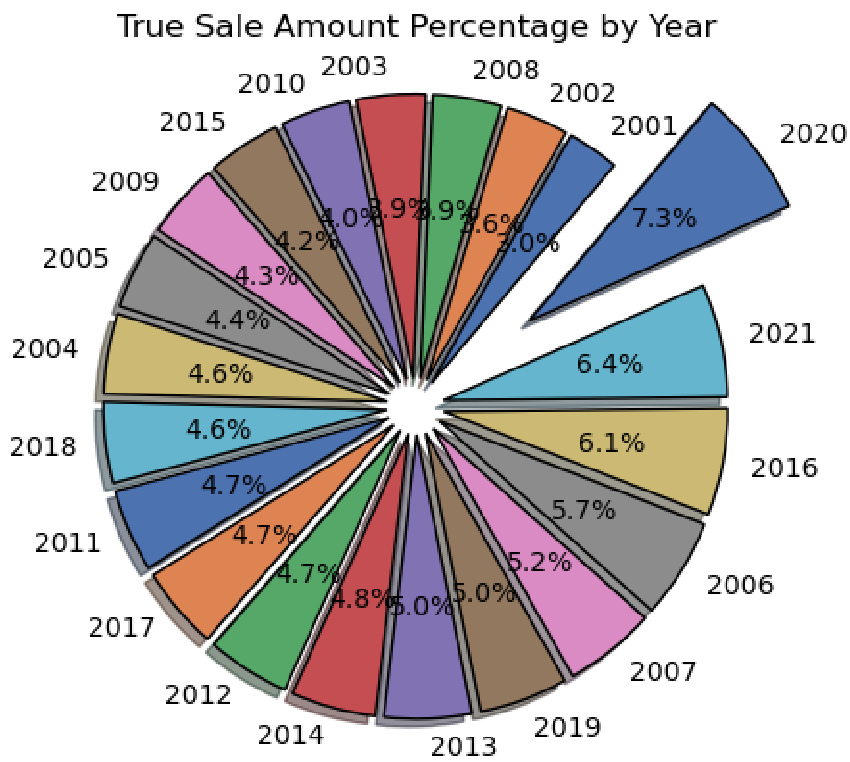
[25]:

	List_Year	Sale_Amount
0	2001	246235.035160
1	2002	296357.123706
2	2008	325831.792393
3	2003	327217.932922
4	2010	331657.472575
5	2015	345883.763949
6	2009	355250.327162
7	2005	364030.126084
8	2004	380297.014169
9	2018	383727.664935
10	2011	391684.320747

```
11        2017  393251.314693
12        2012  395477.676013
13        2014  401421.941220
14        2013  413516.239641
15        2019  420296.971308
16        2007  435713.379734
17        2006  475379.225385
18        2016  507761.249272
19        2021  536975.197072
20        2020  604963.871051
```

[26]:

```
explode = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
           0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.5]

plt.pie(sale_year['Sale_Amount'], labels = sale_year['List_Year'], explode =␣
 ↪explode,
        shadow = True,
        startangle = 50,
        autopct = '%1.1f%%',
        colors = sns.color_palette('deep'),
        wedgeprops = {'edgecolor':'black'})

plt.title("True Sale Amount Percentage by Year")
plt.figure(figsize = (200, 200))
plt.show()
```




```
<Figure size 20000x20000 with 0 Axes>
```

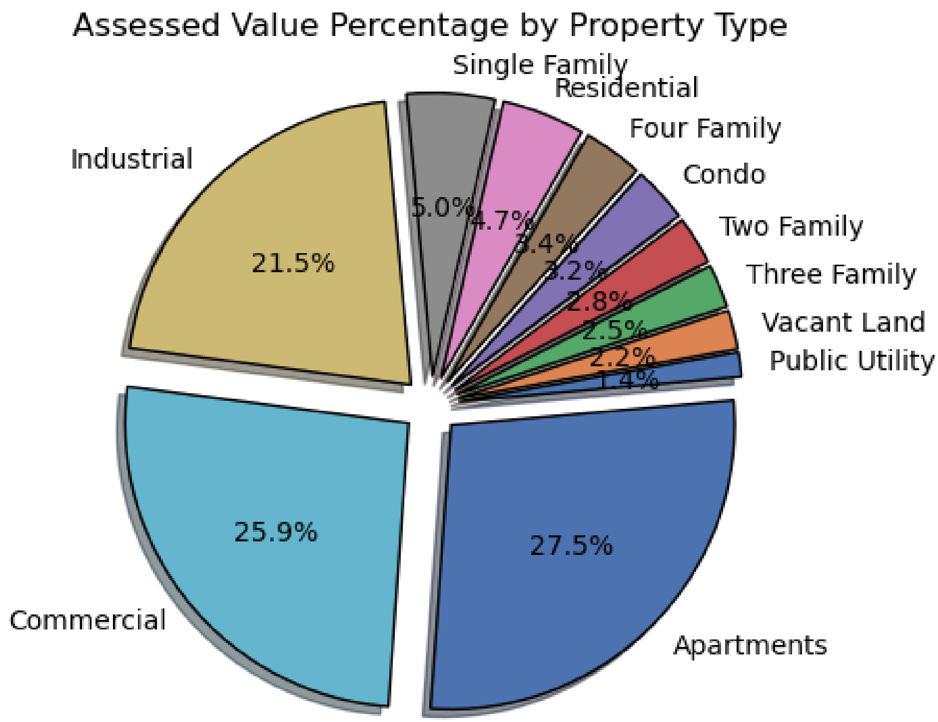
[27]:
```
property_value = subset.groupby(['Property_Type']).mean(numeric_only =
↪True)['Assessed_Value'].sort_values().reset_index()

property_value
```

[27]:

	Property_Type	Assessed_Value
0	Public Utility	7.686125e+04
1	Vacant Land	1.223628e+05
2	Three Family	1.367156e+05
3	Two Family	1.525845e+05
4	Condo	1.745589e+05
5	Four Family	1.861389e+05
6	Residential	2.609691e+05
7	Single Family	2.771346e+05
8	Industrial	1.187925e+06
9	Commercial	1.433600e+06
10	Apartments	1.523121e+06

[28]:
```
explode = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
```

```
plt.pie(property_value['Assessed_Value'], labels =
    property_value['Property_Type'], explode = explode,
        shadow = True,
        startangle = 5,
        autopct = '%1.1f%%',
        colors = sns.color_palette('deep'),
        wedgeprops = {'edgecolor':'black'})

plt.title("Assessed Value Percentage by Property Type")
plt.figure(figsize = (200, 200))
plt.show()
```




```
<Figure size 20000x20000 with 0 Axes>
```

[29]:
```
property_sale = subset.groupby(['Property_Type']).mean(numeric_only =
    True)['Sale_Amount'].sort_values().reset_index()

property_sale
```

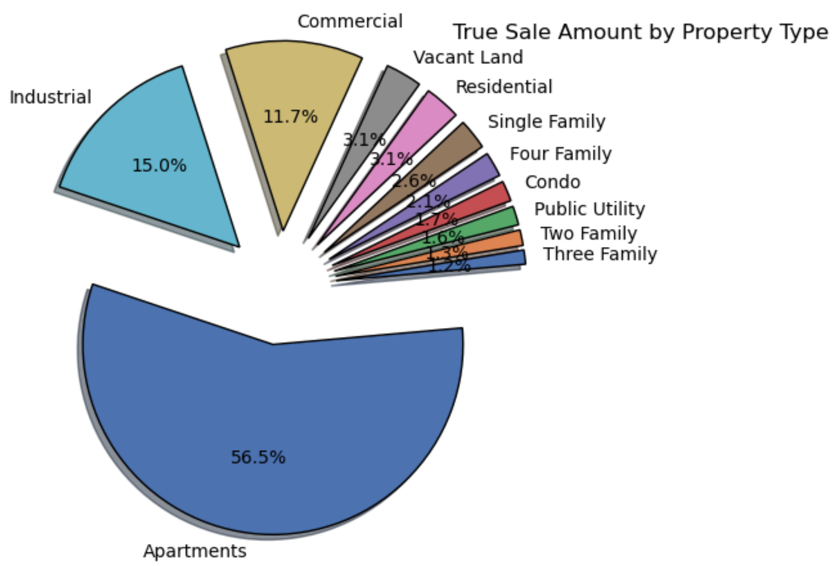
[29]:

	Property_Type	Sale_Amount
0	Three Family	1.798445e+05
1	Two Family	1.990446e+05

```
2    Public Utility  2.407555e+05
3             Condo  2.602110e+05
4       Four Family  3.142910e+05
5     Single Family  3.885143e+05
6       Residential  4.609935e+05
7       Vacant Land  4.631553e+05
8        Commercial  1.750861e+06
9        Industrial  2.236719e+06
10       Apartments  8.445812e+06
```

[30]:

```
explode = [0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3]

plt.pie(property_sale['Sale_Amount'], labels = property_sale['Property_Type'],␣
 ↪explode = explode,
        shadow = True,
        startangle = 5,
        autopct = '%1.1f%%',
        colors = sns.color_palette('deep'),
        wedgeprops = {'edgecolor':'black'})

plt.title("                                                                           ␣
 ↪                                         True Sale Amount by Property Type")
plt.figure(figsize = (200, 200))
plt.show()
```




```
<Figure size 20000x20000 with 0 Axes>
```

[ ]: