APPENDIX: SOURCE CODE

In [119]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import geopandas as gpd
from descartes import PolygonPatch
from scipy.stats import skew
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
sns.set(style="darkgrid")
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

A.1 SAMPLE DATA

In [120]:
```
print('')
print('')
data = pd.read_csv('forAnalysis/data.csv')
data.sample(5)
```

Out[120]:

	street	city	province	neighbourhood	sale_price	list_price	bedrooms	bat
1930	<removed for anonymity>	Toronto	ON	High Park-swansea	2225000	2298000	4	4
986	<removed for anonymity>	Toronto	ON	Niagara	851000	859999	3	2
289	<removed for anonymity>	Toronto	ON	Dovercourt-wallace Emerson-junction	900000	899900	3	2
2462	<removed for anonymity>	Toronto	ON	Leaside	406000	399000	1	1
2429	<removed for anonymity>	Toronto	ON	West Hill	448800	448800	3	2

A.2 ROW COUNT

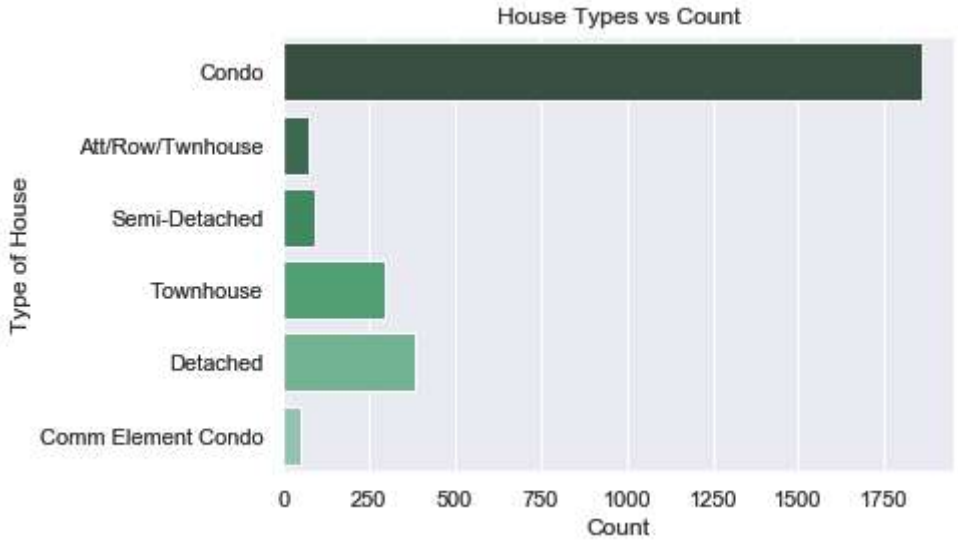
In [121]:
```
data.shape[0]
```

Out[121]: 2779

2779 rows were obtained by scraping the site.

A.3 DATA DISTRIBUTION

In [122]:
```
print('')
print('')
houseTypes = data.groupby("house_type").filter(lambda x: len(x) > 50)
fig = sns.countplot(y='house_type', data=houseTypes, palette='BuGn_d')
fig.set(xlabel='Count', ylabel='Type of House', title='House Types vs Co
unt')
plt.show(fig)
```



In [123]:
```
nb = 'forAnalysis/Neighbourhoods/Neighbourhoods.shp'
regions = gpd.read_file(nb)
regions['neighbourhood'] = regions['FIELD_7'].str.replace(' \(.+\)', '')
.str.lower()
```

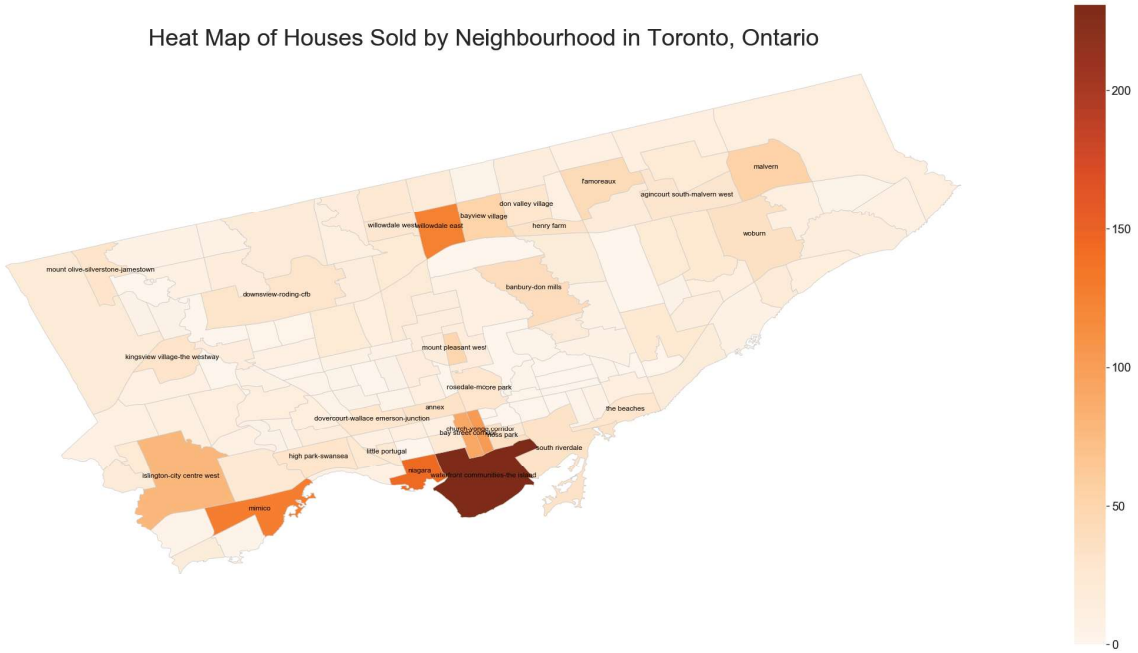
In [124]:
```
print('')
print('')
dataByNeighbourhood = data.groupby('neighbourhood').count()[['street']].
reset_index().replace('Waterfront Communities C1', 'Waterfront Communiti
es-The Island').replace('Rouge E11', 'Rouge')
dataByNeighbourhood.reset_index()
dataByNeighbourhood['neighbourhood'] = dataByNeighbourhood['neighbourhoo
d'].str.lower()
dataByNeighbourhood.sort_values('street', ascending=False).head(10)
```

Out[124]:

	neighbourhood	street
114	waterfront communities-the island	231
86	niagara	144
76	mimico	128
121	willowdale east	125
22	church-yonge corridor	102
6	bay street corridor	91
57	islington-city centre west	78
72	malvern	54
7	bayview village	53
82	mount pleasant west	49

A.4 GEO-MAPPING/HEAT MAP

In [125]:
```
# Join the dataset with the available mapping files.
# We will fill in the neighbourhoods which were not present in the datas
et with 0.
# this way we will still see the neighbourhood in the map, but no housin
g will be shown on it.
merged = regions.set_index('neighbourhood').join(dataByNeighbourhood.set
_index('neighbourhood'))
merged = merged.reset_index()
merged = merged.fillna(0)


# we are using the maximum and minimum house_type count values from the
 previous cell.
# setting additionally properties for the plot such as titles, turning o
f the axis for better visibility
# and setting the color scheme to look like a heat map.
vmin, vmax = 0, 231
fig, ax = plt.subplots(1, figsize=(40, 20))
ax.axis('off')
ax.set_title('Heat Map of Houses Sold by Neighbourhood in Toronto, Ontar
io', fontdict={'fontsize': '40', 'fontweight' : '3'})
color = 'Oranges'


# Create colorbar as a legend
# empty array for the data range
# add the colorbar to the figure
# set the color bar label text size
sm = plt.cm.ScalarMappable(cmap=color, norm=plt.Normalize(vmin=vmin, vma
x=vmax))
sm._A = []
cbar = fig.colorbar(sm)
cbar.ax.tick_params(labelsize=20)


# actually plot the map
# we will only annotate the plot for neighbourhoods with more than 25 ho
uses sold
merged.plot('street', cmap=color, linewidth=0.8, ax=ax, edgecolor='0.8',
figsize=(40,20))
for idx, row in merged.iterrows():
    if(row['street'] > 25):
        plt.annotate(s=row['neighbourhood'], xy=(row['FIELD_11'], row['F
IELD_12']),
                 horizontalalignment='center', fontsize='large', color=
'black', wrap=True)
plt.show()
```



A.5 CONVERTING AND FILTERING CONDOMINIUMS

In [126]:
```
print('')
print('')
data['house_type'].value_counts()
```

Out[126]:
```
Condo                1860
Detached              385
Townhouse             298
Semi-Detached          94
Att/Row/Twnhouse       75
Comm Element Condo     51
Co-Ownership Apt        5
Det Condo               4
Triplex                 2
Link                    2
Co-Op Apt               1
Duplex                  1
Multiplex               1
Name: house_type, dtype: int64
```

In [127]:
```
print('')
print('')
modified_data = data.replace('Comm Element Condo', 'Condo').replace('Det
Condo', 'Condo')
modified_data['house_type'].value_counts()
```

Out[127]:
```
Condo                1915
Detached              385
Townhouse             298
Semi-Detached          94
Att/Row/Twnhouse       75
Co-Ownership Apt        5
Triplex                 2
Link                    2
Co-Op Apt               1
Duplex                  1
Multiplex               1
Name: house_type, dtype: int64
```

In [128]:
```
print('')
print('')
condos_all_columns = modified_data[modified_data['house_type'] == 'Cond
o']
condos_all_columns.sample(5)
```

Out[128]:

	street	city	province	neighbourhood	sale_price	list_price	bedrooms	bat
2322	<removed for anonymity>	Toronto	ON	Church-yonge Corridor	600000	598000	1	1
213	<removed for anonymity>	Toronto	ON	Waterfront Communities C1	1175000	1149900	2	2
2079	<removed for anonymity>	Toronto	ON	Malvern	337000	299800	2	2
1908	<removed for anonymity>	Toronto	ON	Wychwood	570000	599000	1	1
587	<removed for anonymity>	Toronto	ON	Waterfront Communities C1	690000	699000	2	2

In [129]:
```
print('')
print('')
condos = condos_all_columns.drop(['street', 'city', 'province', 'neighbo
urhood', 'house_type','url'], axis=1)
condos.sample(5)
```

Out[129]:

	sale_price	list_price	bedrooms	bathrooms	square_feet	days_on_market	parking_
80	450000	450000	1	1	499	2	0
490	513000	524900	1	1	599	16	1
134	420000	429999	1	1	499	19	1
52	427000	439900	1	1	599	26	1
661	470000	475000	1	1	699	37	1

In [130]:
```
condos.shape
```

Out[130]: (1915, 25)

In [131]:
```
condos.isnull().values.any()
```

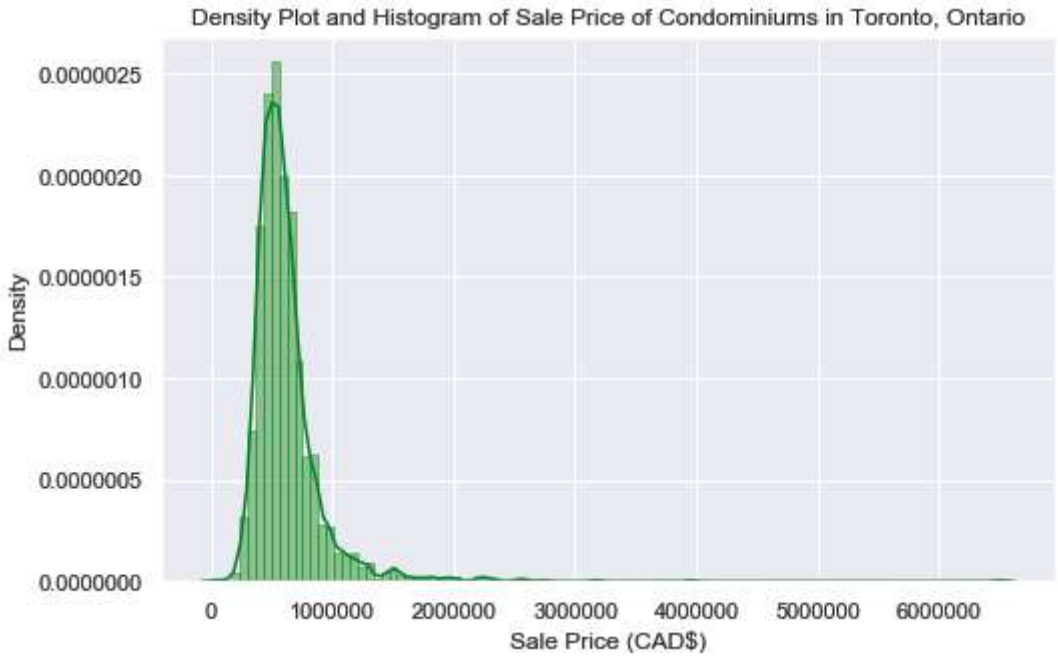
Out[131]: False

In [132]:
```
print('')
print('')
print(condos['sale_price'].describe().apply(lambda x: format(x, 'f')))
```

```
count       1915.000000
mean      627137.378068
std       318574.865134
min        47500.000000
25%       460000.000000
50%       560000.000000
75%       690000.000000
max      6500000.000000
Name: sale_price, dtype: object
```

A.6 DENSITY PLOT OF SALE PRICE

In [133]:
```
print('')
print('')
plt.figure(figsize=(8, 5))
fig = sns.distplot(condos['sale_price'], color='green', bins=100, hist_k
ws={'edgecolor':'green'});
plt.title('Density Plot and Histogram of Sale Price of Condominiums in T
oronto, Ontario', fontsize='large')
plt.ylabel('Density')
plt.xlabel('Sale Price (CAD$)')
plt.show(fig)
```

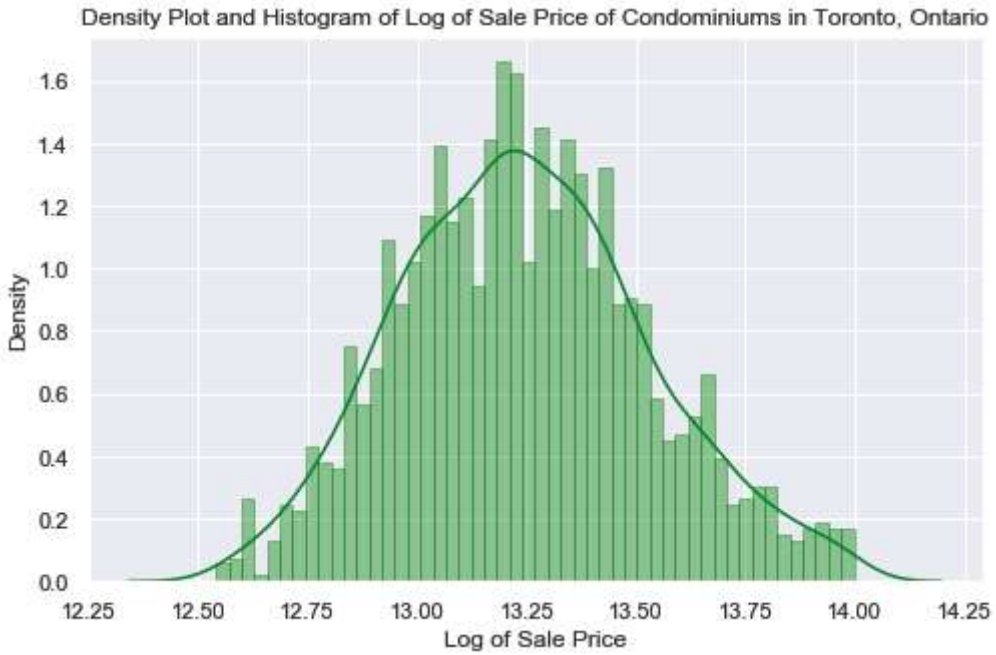


In [134]:
```
condos['sale_price'].skew()
```

Out[134]: 5.906281071445114

A.7 DENSITY PLOT OF SALE PRICE (NORMALIZED)

In [135]:
```
condos['log_sale_price'] = np.log(condos['sale_price'])
mean = condos['log_sale_price'].mean()
sd = condos['log_sale_price'].std()
condos_normalized = condos[np.abs(condos['log_sale_price'] - mean) < 2*s
d]

print('')
print('')
plt.figure(figsize=(8, 5))
fig = sns.distplot(condos_normalized['log_sale_price'], color='green', b
ins=50, hist_kws={'edgecolor':'green'});
plt.title('Density Plot and Histogram of Log of Sale Price of Condominiu
ms in Toronto, Ontario', fontsize='large')
plt.ylabel('Density')
plt.xlabel('Log of Sale Price')
plt.show(fig)
```



In [136]:
```
condos_normalized['log_sale_price'].skew()
```

Out[136]: 0.2151143258994082

In [137]:
```
condos_normalized.shape
```

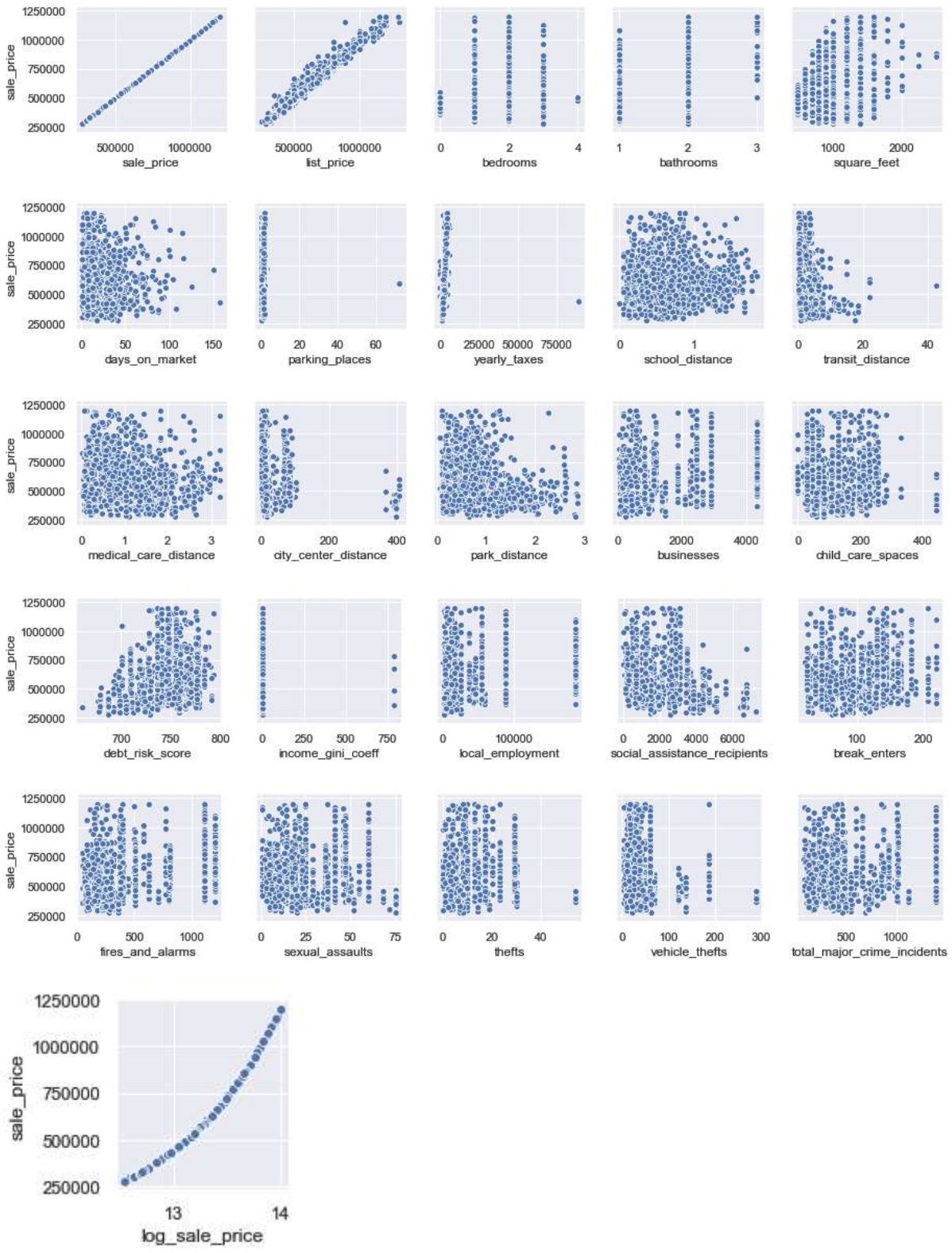
Out[137]: (1820, 26)

In [138]:
```
condos_normalized['sale_price'].describe().apply(lambda x: format(x,'f'
))
```

Out[138]:
```
count       1820.000000
mean      588375.977473
std       173871.162415
min       279500.000000
25%       460000.000000
50%       557250.000000
75%       676000.000000
max      1200000.000000
Name: sale_price, dtype: object
```

A.8 PAIR-WISE SCATTER PLOTS OF ALL FEATURES

In [139]:
```
print('')
print('')

for i in range(0, len(condos_normalized.columns), 5):
    sns.pairplot(data=condos_normalized,
                 x_vars=condos_normalized.columns[i:i+5],
                 y_vars=['sale_price'])
```

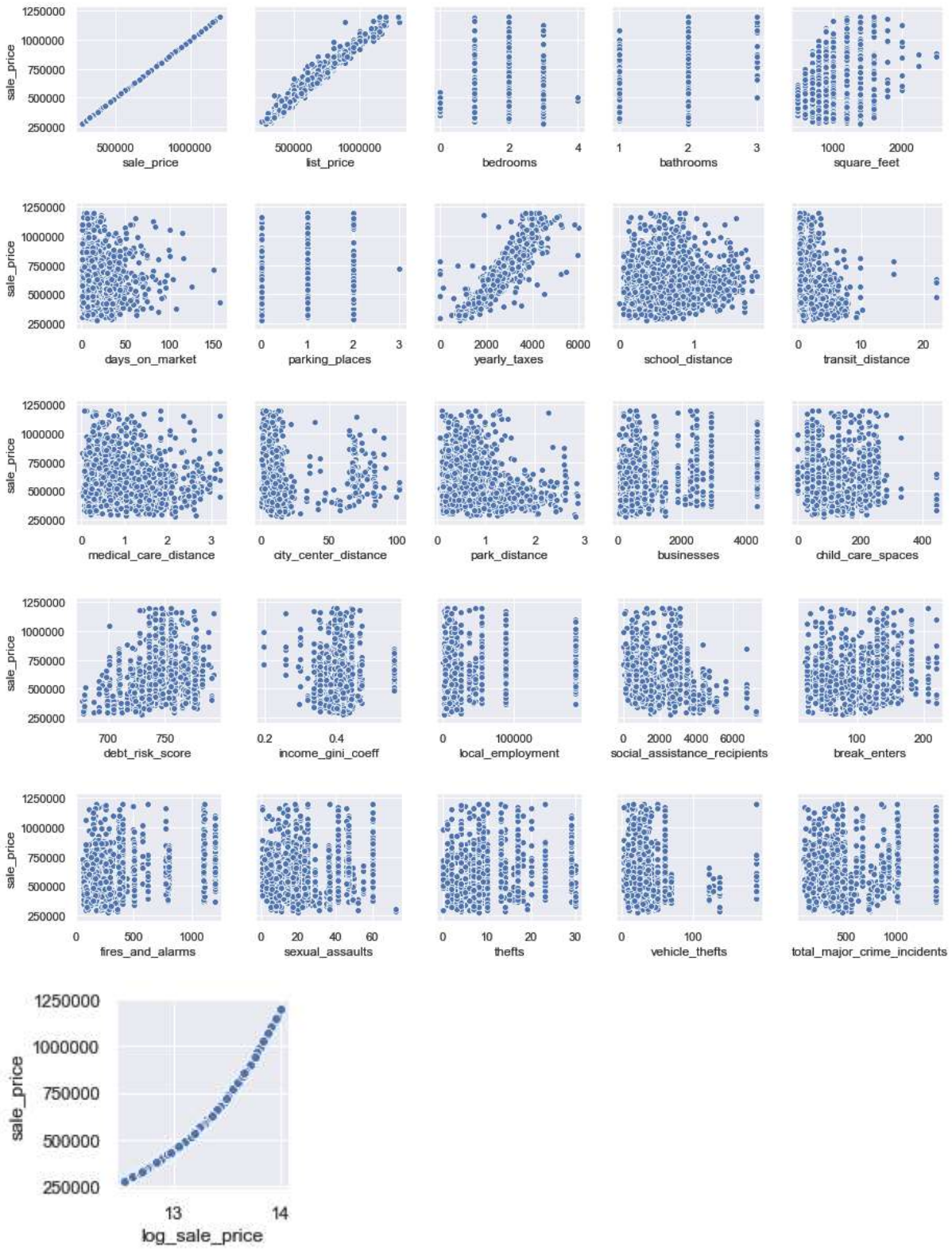


A.9 REMOVING OUTLIERS IN OTHER FEATURES AND REPLOTTING SCATTER PLOTS

In [140]:
```
condos_cleaned = condos_normalized[(condos_normalized['parking_places'] < 20 )
                                   & (condos_normalized['yearly_taxes'] < 25000)
                                   & (condos_normalized['transit_distance'] < 30)
                                   & (condos_normalized['city_center_distance'] < 200)
                                   & (condos_normalized['thefts'] < 40)
                                   & (condos_normalized['income_gini_coeff'] < 40 )
                                                                            ]

condos_cleaned.shape
```

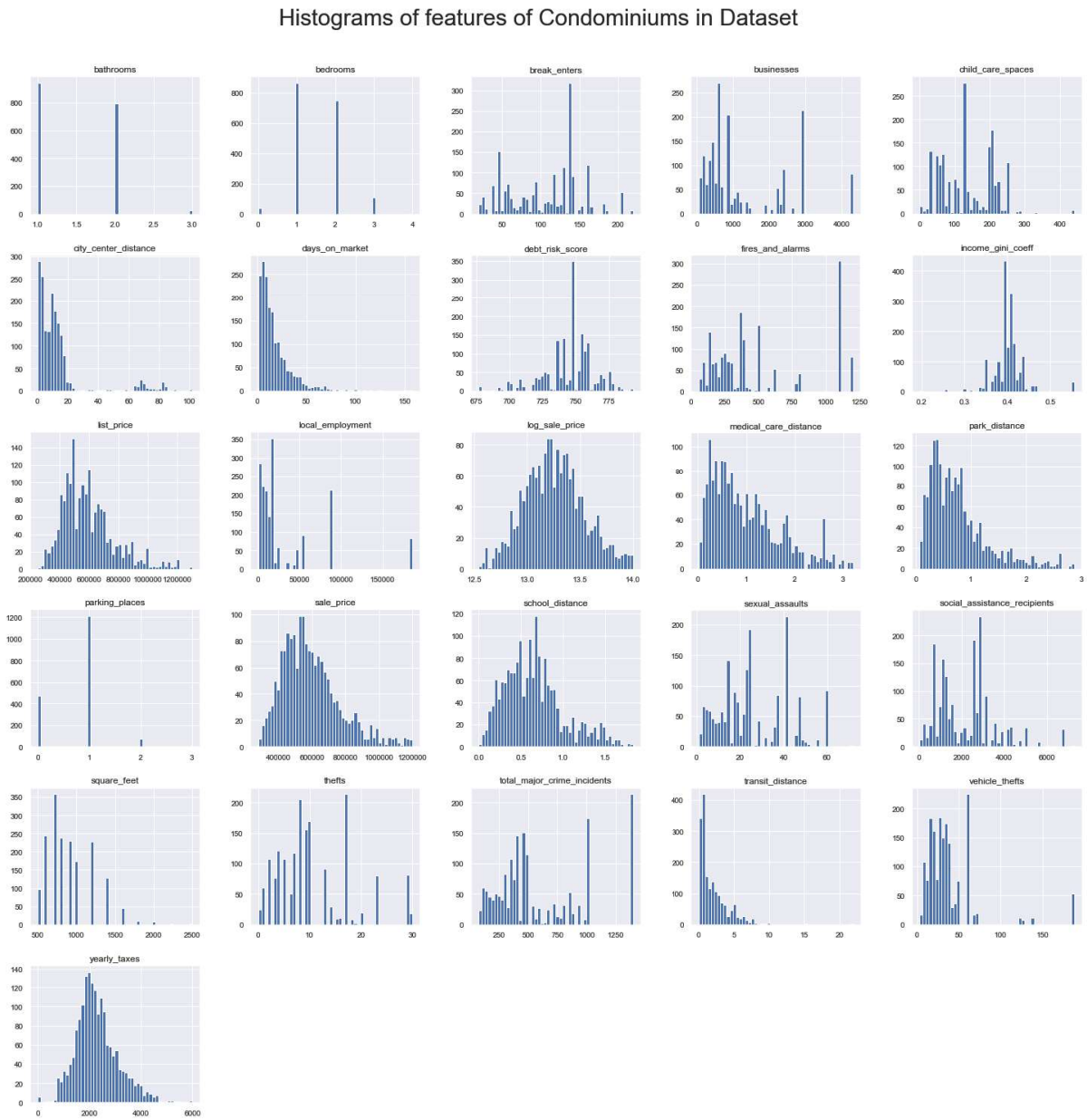
Out[140]: (1760, 26)

In [141]:
```
print('')
print('')

for i in range(0, len(condos_cleaned.columns), 5):
    sns.pairplot(data=condos_cleaned,
                 x_vars=condos_cleaned.columns[i:i+5],
                 y_vars=['sale_price'])
```



A.10 HISTOGRAM OF ALL FEATURES

In [142]:
```
print('')
print('')
condos_cleaned.hist(figsize=(25,25),bins=50, xlabelsize=10, ylabelsize=10)
plt.suptitle('Histograms of features of Condominiums in Dataset', y=0.93, fontsize=30)
plt.show()
```



In [143]:
```
condos_cleaned['sale_price'].describe().apply(lambda x: format(x,'f'))
```

Out[143]:
```
count       1760.000000
mean      593363.885227
std       173765.526225
min       280000.000000
25%       467125.000000
50%       560000.000000
75%       680000.000000
max      1200000.000000
Name: sale_price, dtype: object
```

In [144]:
```
condos_cleaned['log_sale_price'].skew()
```

Out[144]:
```
0.19544742161038794
```

A.11 PEARSON CORRELATION OF ALL FEATURES WITH OUTCOME

In [145]:
```
print('')
print('')
condos_corr_cleaned = condos_cleaned.corr()['log_sale_price'][1:]
golden_features_cleaned_list = condos_corr_cleaned[abs(condos_corr_clean
ed) > 0.1].sort_values(ascending=False)
print("There are {} strongly correlated values with Log Sale Price:\n{}"
.format(len(golden_features_cleaned_list), golden_features_cleaned_list
))
```

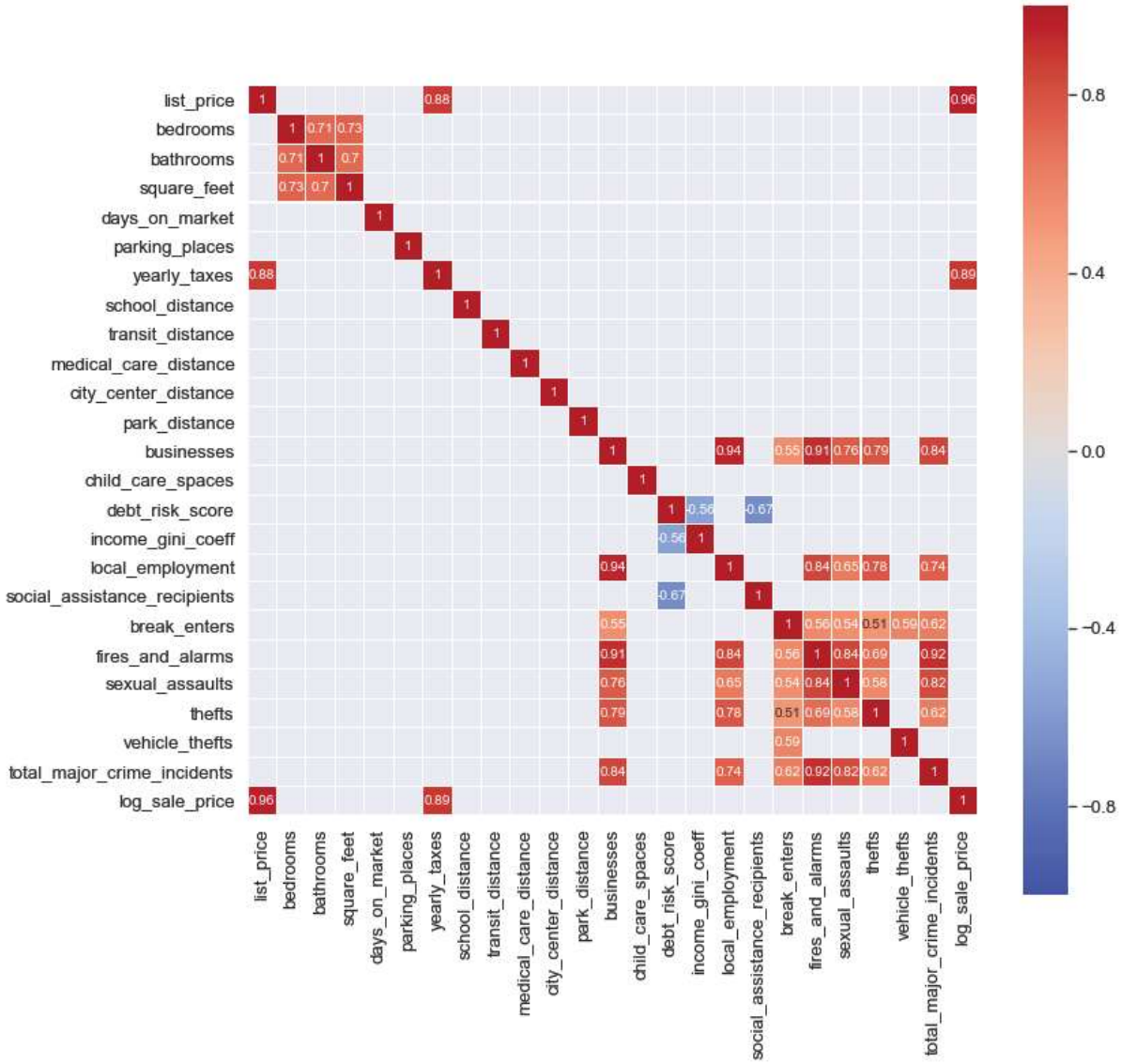
```
There are 18 strongly correlated values with Log Sale Price:
log_sale_price                  1.000000
list_price                      0.961250
yearly_taxes                    0.886556
bathrooms                       0.373000
fires_and_alarms                0.308304
debt_risk_score                 0.271255
local_employment                0.265035
businesses                      0.264677
square_feet                     0.251013
total_major_crime_incidents     0.198109
bedrooms                        0.170696
thefts                          0.169478
sexual_assaults                 0.131436
break_enters                    0.125633
parking_places                  0.109175
park_distance                  -0.250178
social_assistance_recipients   -0.305269
transit_distance               -0.318992
Name: log_sale_price, dtype: float64
```

In [146]:
```
golden_features_cleaned_list.index
```

Out[146]:
```
Index(['log_sale_price', 'list_price', 'yearly_taxes', 'bathrooms',
       'fires_and_alarms', 'debt_risk_score', 'local_employment', 'busi
nesses',
       'square_feet', 'total_major_crime_incidents', 'bedrooms', 'theft
s',
       'sexual_assaults', 'break_enters', 'parking_places', 'park_dista
nce',
       'social_assistance_recipients', 'transit_distance'],
      dtype='object')
```

A.12 PAIR-WISE PEARSON CORRELATION

In [147]:
```
cleaned_corr = condos_cleaned.drop('sale_price', axis = 1).corr()
plt.figure(figsize=(12, 12))
sns.set(font_scale=1.2)
sns.heatmap(cleaned_corr[(np.abs(cleaned_corr) >= 0.5) | (cleaned_corr ==
= 1)],
            cmap='coolwarm', vmax=1, vmin=-1, linewidths=0.1,
            annot=True, annot_kws={"size": 10}, square=True);
```



A.13 FEATURE SELECTION

In [148]:
```
golden_features_cleaned_list.index
```

Out[148]:
```
Index(['log_sale_price', 'list_price', 'yearly_taxes', 'bathrooms',
       'fires_and_alarms', 'debt_risk_score', 'local_employment', 'busi
nesses',
       'square_feet', 'total_major_crime_incidents', 'bedrooms', 'theft
s',
       'sexual_assaults', 'break_enters', 'parking_places', 'park_dista
nce',
       'social_assistance_recipients', 'transit_distance'],
      dtype='object')
```

In [149]:
```
golden_unstacked_corr = cleaned_corr.loc[golden_features_cleaned_list.in
dex, golden_features_cleaned_list.index].unstack()
golden_unstacked_corr[(golden_unstacked_corr.abs() > 0.5) & (golden_unst
acked_corr != 1)]
```

Out[149]:

log_sale_price	list_price	0.961250
	yearly_taxes	0.886556
list_price	log_sale_price	0.961250
	yearly_taxes	0.879152
yearly_taxes	log_sale_price	0.886556
	list_price	0.879152
bathrooms	square_feet	0.703789
	bedrooms	0.711302
fires_and_alarms	local_employment	0.840883
	businesses	0.914650
	total_major_crime_incidents	0.916897
	thefts	0.692334
	sexual_assaults	0.844394
	break_enters	0.563208
debt_risk_score	social_assistance_recipients	-0.667067
local_employment	fires_and_alarms	0.840883
	businesses	0.941023
	total_major_crime_incidents	0.737111
	thefts	0.779708
	sexual_assaults	0.645528
businesses	fires_and_alarms	0.914650
	local_employment	0.941023
	total_major_crime_incidents	0.842298
	thefts	0.787174
	sexual_assaults	0.761872
	break_enters	0.546692
square_feet	bathrooms	0.703789
	bedrooms	0.729675
total_major_crime_incidents	fires_and_alarms	0.916897
	local_employment	0.737111
	businesses	0.842298
	thefts	0.622084
	sexual_assaults	0.820557
	break_enters	0.621937
bedrooms	bathrooms	0.711302
	square_feet	0.729675
thefts	fires_and_alarms	0.692334
	local_employment	0.779708
	businesses	0.787174
	total_major_crime_incidents	0.622084
	sexual_assaults	0.577087
	break_enters	0.513636
sexual_assaults	fires_and_alarms	0.844394
	local_employment	0.645528
	businesses	0.761872
	total_major_crime_incidents	0.820557
	thefts	0.577087
	break_enters	0.539629
break_enters	fires_and_alarms	0.563208
	businesses	0.546692
	total_major_crime_incidents	0.621937
	thefts	0.513636
	sexual_assaults	0.539629
social_assistance_recipients	debt_risk_score	-0.667067

dtype: float64

A.14 REGRESSION ANALYSIS

In [150]:
```
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
```

In [151]:
```
##########################################
# First Model - selected features
##########################################


sc = StandardScaler()

X_original = condos_cleaned[['yearly_taxes', 'businesses',
       'square_feet', 'break_enters', 'parking_places', 'park_distance',
       'social_assistance_recipients', 'transit_distance']]

y = condos_cleaned['log_sale_price']
X = sc.fit_transform(X_original)
linearModel = linear_model.LinearRegression()
linear_score = cross_val_score(linearModel, X, y, cv = 10)

print('Linear model 1 accuracy: ', np.mean(linear_score))

model = linear_reg.fit(X,y)
print('R-Squared: \n', model.score(X,y))
print('Intercept: \n', model.intercept_)
print('Coefficients: \n', model.coef_)
scores = cross_val_score(linear_reg, X, y, cv=10)
print('Cross Validation Scores: \n', scores)
print('\nAverage Accuracy: {}'.format(np.mean(scores)))
x = sm.add_constant(X)
model = sm.OLS(y, x).fit()
predictions = model.predict(x)

# Print out the statistics
model.summary()
```

```
Linear model 1 accuracy:  0.7951379918974435
R-Squared:
 0.7997820048346587
Intercept:
 13.253426767235963
Coefficients:
 [ 0.22344751  0.01931482  0.02761301  0.00245046  0.007251   -0.021403
54
 -0.01866655 -0.00764285]
Cross Validation Scores:
 [0.86489429 0.75807808 0.84995199 0.75876826 0.72799924 0.80958705
 0.75601648 0.75083514 0.86875572 0.80649366]

Average Accuracy: 0.7951379918974435
```

Out[151]:

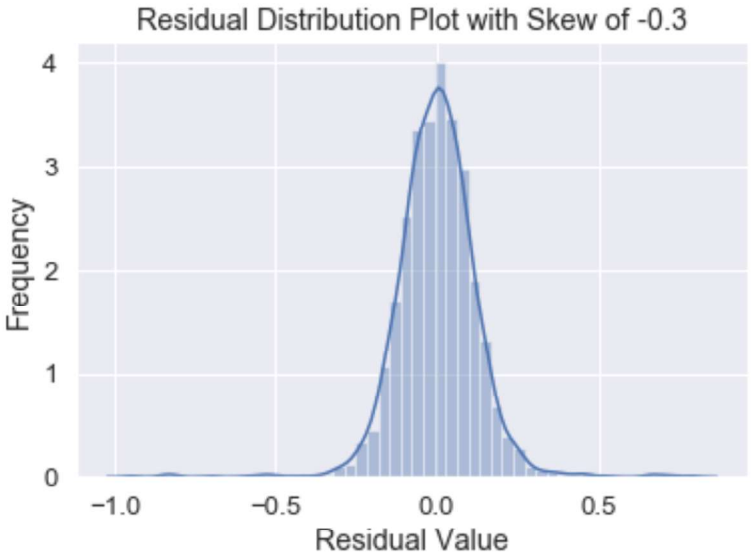
OLS Regression Results

Dep. Variable:	log_sale_price	**R-squared:**	0.800
Model:	OLS	**Adj. R-squared:**	0.799
Method:	Least Squares	**F-statistic:**	874.3
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	0.00
Time:	12:32:39	**Log-Likelihood:**	1152.2
No. Observations:	1760	**AIC:**	-2286.
Df Residuals:	1751	**BIC:**	-2237.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	13.2534	0.003	4410.880	0.000	13.248	13.259
x1	0.2234	0.004	56.533	0.000	0.216	0.231
x2	0.0193	0.004	4.789	0.000	0.011	0.027
x3	0.0276	0.004	7.363	0.000	0.020	0.035
x4	0.0025	0.004	0.630	0.529	-0.005	0.010
x5	0.0073	0.003	2.131	0.033	0.001	0.014
x6	-0.0214	0.003	-6.682	0.000	-0.028	-0.015
x7	-0.0187	0.004	-5.202	0.000	-0.026	-0.012
x8	-0.0076	0.004	-2.148	0.032	-0.015	-0.001

Omnibus:	320.656	**Durbin-Watson:**	1.966
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5489.182
Skew:	0.320	**Prob(JB):**	0.00
Kurtosis:	11.628	**Cond. No.**	2.60

In [152]:
```
residuals = predictions - y
sns.distplot(residuals)
plt.xlabel('Residual Value')
plt.ylabel('Frequency')
plt.title('Residual Distribution Plot with Skew of -0.3')
plt.show()
```



In [153]:
```
#######################################################################
### SEcond model - only socio economic variables
#######################################################################

X2_original = condos_cleaned.iloc[:,13:25]

X2 = sc.fit_transform(X2_original)

y2 = condos_cleaned['log_sale_price']

linearModel2 = linear_model.LinearRegression()
linear_score2 = cross_val_score(linearModel2, X2, y2, cv = 10)

print('Linear model 2 accuracy: ', np.mean(linear_score2))

model2 = linearModel2.fit(X2,y2)

print('R-Squared: \n', model2.score(X2,y2))
print('Intercept: \n', model2.intercept_)
print('Coefficients: \n', model2.coef_)
scores2 = cross_val_score(linear_reg, X2, y2, cv=10)
print('Cross Validation Scores: \n', scores2)
print('\nAverage Accuracy: {}'.format(np.mean(scores2)))
x2 = sm.add_constant(X2)
model2 = sm.OLS(y2, x2).fit()
predictions = model2.predict(x2)

model2.summary()
```

```
Linear model 2 accuracy:  0.278496567410697
R-Squared:
 0.2946912606777127
Intercept:
 13.253426767235963
Coefficients:
 [-0.0500798   0.03599302 -0.00487565  0.01299521 -0.03214592 -0.139282
31
  0.01485984  0.16811977 -0.0305812  -0.01419821 -0.03483877  0.0656379
2]
Cross Validation Scores:
 [0.29636009 0.18923254 0.35174084 0.29403391 0.30454744 0.07308851
 0.39438747 0.30448477 0.31671943 0.26037069]

Average Accuracy: 0.278496567410697
```

Out[153]:

OLS Regression Results

Dep. Variable:	log_sale_price	**R-squared:**	0.295
Model:	OLS	**Adj. R-squared:**	0.290
Method:	Least Squares	**F-statistic:**	60.83
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	3.52e-123
Time:	12:32:40	**Log-Likelihood:**	44.087
No. Observations:	1760	**AIC:**	-62.17
Df Residuals:	1747	**BIC:**	8.977
Df Model:	12		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	13.2534	0.006	2347.419	0.000	13.242	13.265
x1	-0.0501	0.029	-1.756	0.079	-0.106	0.006
x2	0.0360	0.008	4.724	0.000	0.021	0.051
x3	-0.0049	0.012	-0.405	0.685	-0.028	0.019
x4	0.0130	0.008	1.537	0.124	-0.004	0.030
x5	-0.0321	0.023	-1.387	0.166	-0.078	0.013
x6	-0.1393	0.011	-12.644	0.000	-0.161	-0.118
x7	0.0149	0.010	1.536	0.125	-0.004	0.034
x8	0.1681	0.027	6.282	0.000	0.116	0.221
x9	-0.0306	0.014	-2.265	0.024	-0.057	-0.004
x10	-0.0142	0.011	-1.321	0.187	-0.035	0.007
x11	-0.0348	0.009	-3.946	0.000	-0.052	-0.018
x12	0.0656	0.021	3.137	0.002	0.025	0.107

Omnibus:	81.353	**Durbin-Watson:**	2.000
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	92.213
Skew:	0.532	**Prob(JB):**	9.47e-21
Kurtosis:	3.353	**Cond. No.**	15.1

In [154]:
```
##############################################
# Third model all House Values
##############################################

X3_original = condos_cleaned.iloc[:,2:13]

y3 = condos_cleaned['log_sale_price']

X3 = sc.fit_transform(X3_original)

lasso3 = linear_model.Lasso()
linearModel3 = linear_model.LinearRegression()
linear_score3 = cross_val_score(linearModel3, X3, y3, cv = 10)

print('Linear model 3 accuracy: ', np.mean(linear_score3))

model3 = linearModel3.fit(X3,y3)

print('R-Squared: \n', model3.score(X3,y3))
print('Intercept: \n', model3.intercept_)
print('Coefficients: \n', model3.coef_)
scores3 = cross_val_score(linear_reg, X3, y3, cv=10)
print('Cross Validation Scores: \n', scores3)
print('\nAverage Accuracy: {}'.format(np.mean(scores3)))

x3 = sm.add_constant(X3)
model3 = sm.OLS(y3, x3).fit()
predictions = model3.predict(x3)

model3.summary()
```

```
Linear model 3 accuracy:  0.7995831791442237
R-Squared:
 0.8047944594949075
Intercept:
 13.253426767235963
Coefficients:
 [ 0.00178804  0.02592625  0.00616648 -0.01202886  0.00034881  0.229960
29
  0.01299522 -0.01622896  0.00292425 -0.01471657 -0.01973935]
Cross Validation Scores:
 [0.85690838 0.76175959 0.85996563 0.76518193 0.7338329  0.82443518
 0.76424512 0.74062718 0.8714891  0.81738678]

Average Accuracy: 0.7995831791442237
```

Out[154]:

OLS Regression Results

Dep. Variable:	log_sale_price	**R-squared:**	0.805
Model:	OLS	**Adj. R-squared:**	0.804
Method:	Least Squares	**F-statistic:**	655.2
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	0.00
Time:	12:32:40	**Log-Likelihood:**	1174.5
No. Observations:	1760	**AIC:**	-2325.
Df Residuals:	1748	**BIC:**	-2259.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	13.2534	0.003	4463.324	0.000	13.248	13.259
x1	0.0018	0.005	0.368	0.713	-0.008	0.011
x2	0.0259	0.005	5.328	0.000	0.016	0.035
x3	0.0062	0.005	1.236	0.217	-0.004	0.016
x4	-0.0120	0.003	-3.992	0.000	-0.018	-0.006
x5	0.0003	0.003	0.105	0.917	-0.006	0.007
x6	0.2300	0.004	62.521	0.000	0.223	0.237
x7	0.0130	0.003	4.006	0.000	0.007	0.019
x8	-0.0162	0.004	-4.467	0.000	-0.023	-0.009
x9	0.0029	0.003	0.897	0.370	-0.003	0.009
x10	-0.0147	0.003	-4.919	0.000	-0.021	-0.009
x11	-0.0197	0.003	-6.258	0.000	-0.026	-0.014

Omnibus:	346.390	**Durbin-Watson:**	1.983
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6279.784
Skew:	0.394	**Prob(JB):**	0.00
Kurtosis:	12.220	**Cond. No.**	3.53

In [155]:
```
##############################################
# Fourth model all features
##############################################

X4_original = condos_cleaned.iloc[:,2:25]

X4 = sc.fit_transform(X4_original)

y4 = condos_cleaned['log_sale_price']

linearModel4 = linear_model.LinearRegression()
linear_score4 = cross_val_score(linearModel4, X4, y4, cv = 10)

print('Linear model 4 accuracy: ', np.mean(linear_score4))


model4 = linearModel4.fit(X4,y4)

print('R-Squared: \n', model4.score(X4,y4))
print('Intercept: \n', model4.intercept_)
print('Coefficients: \n', model4.coef_)
scores4 = cross_val_score(linear_reg, X4, y4, cv=10)
print('Cross Validation Scores: \n', scores4)
print('\nAverage Accuracy: {}'.format(np.mean(scores4)))


x4 = sm.add_constant(X4)
model4 = sm.OLS(y4, x4).fit()
predictions = model4.predict(x4)

model4.summary()
```

```
Linear model 4 accuracy:  0.8236611625359462
R-Squared:
 0.831871064290193
Intercept:
 13.253426767235963
Coefficients:
 [ 0.00807902  0.02949302  0.0211603  -0.01223962  0.01105085  0.198647
6
  0.00998604 -0.01069755  0.01298931 -0.01113045 -0.01087425 -0.0243100
9
  0.00316682 -0.00901195  0.01476469 -0.02304404 -0.064435   -0.0044493
8
  0.05270837 -0.00349306  0.00278768 -0.01137757  0.05437557]
Cross Validation Scores:
 [0.86806366 0.79480259 0.85887877 0.78812348 0.77436878 0.83636191
 0.81733777 0.77552272 0.89709651 0.82605543]

Average Accuracy: 0.8236611625359462
```

Out[155]:

OLS Regression Results

Dep. Variable:	log_sale_price	**R-squared:**	0.832
Model:	OLS	**Adj. R-squared:**	0.830
Method:	Least Squares	**F-statistic:**	373.5
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	0.00
Time:	12:32:40	**Log-Likelihood:**	1305.9
No. Observations:	1760	**AIC:**	-2564.
Df Residuals:	1736	**BIC:**	-2432.
Df Model:	23		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	13.2534	0.003	4792.778	0.000	13.248	13.259
x1	0.0081	0.005	1.765	0.078	-0.001	0.017
x2	0.0295	0.005	6.455	0.000	0.021	0.038
x3	0.0212	0.005	4.427	0.000	0.012	0.031
x4	-0.0122	0.003	-4.347	0.000	-0.018	-0.007
x5	0.0111	0.003	3.413	0.001	0.005	0.017
x6	0.1986	0.004	49.094	0.000	0.191	0.207
x7	0.0100	0.003	3.096	0.002	0.004	0.016
x8	-0.0107	0.004	-3.014	0.003	-0.018	-0.004
x9	0.0130	0.003	3.992	0.000	0.007	0.019
x10	-0.0111	0.003	-3.952	0.000	-0.017	-0.006
x11	-0.0109	0.003	-3.285	0.001	-0.017	-0.004
x12	-0.0243	0.014	-1.699	0.089	-0.052	0.004
x13	0.0032	0.004	0.829	0.407	-0.004	0.011
x14	-0.0090	0.006	-1.503	0.133	-0.021	0.003
x15	0.0148	0.004	3.480	0.001	0.006	0.023
x16	-0.0230	0.012	-2.000	0.046	-0.046	-0.000
x17	-0.0644	0.006	-10.735	0.000	-0.076	-0.053
x18	-0.0044	0.005	-0.930	0.352	-0.014	0.005
x19	0.0527	0.014	3.852	0.000	0.026	0.080
x20	-0.0035	0.007	-0.514	0.607	-0.017	0.010

x21	0.0028	0.006	0.488	0.626	-0.008	0.014
x22	-0.0114	0.004	-2.552	0.011	-0.020	-0.003
x23	0.0544	0.011	5.177	0.000	0.034	0.075

Omnibus:	286.295	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3884.627
Skew:	0.295	**Prob(JB):**	0.00
Kurtosis:	10.254	**Cond. No.**	16.3

A.15 KMEANS CLUSTERING

In [156]:
```
# condos_cleaned.iloc[:,2:25]
x = condos_cleaned.iloc[:,2:25]
x[1:5]
```

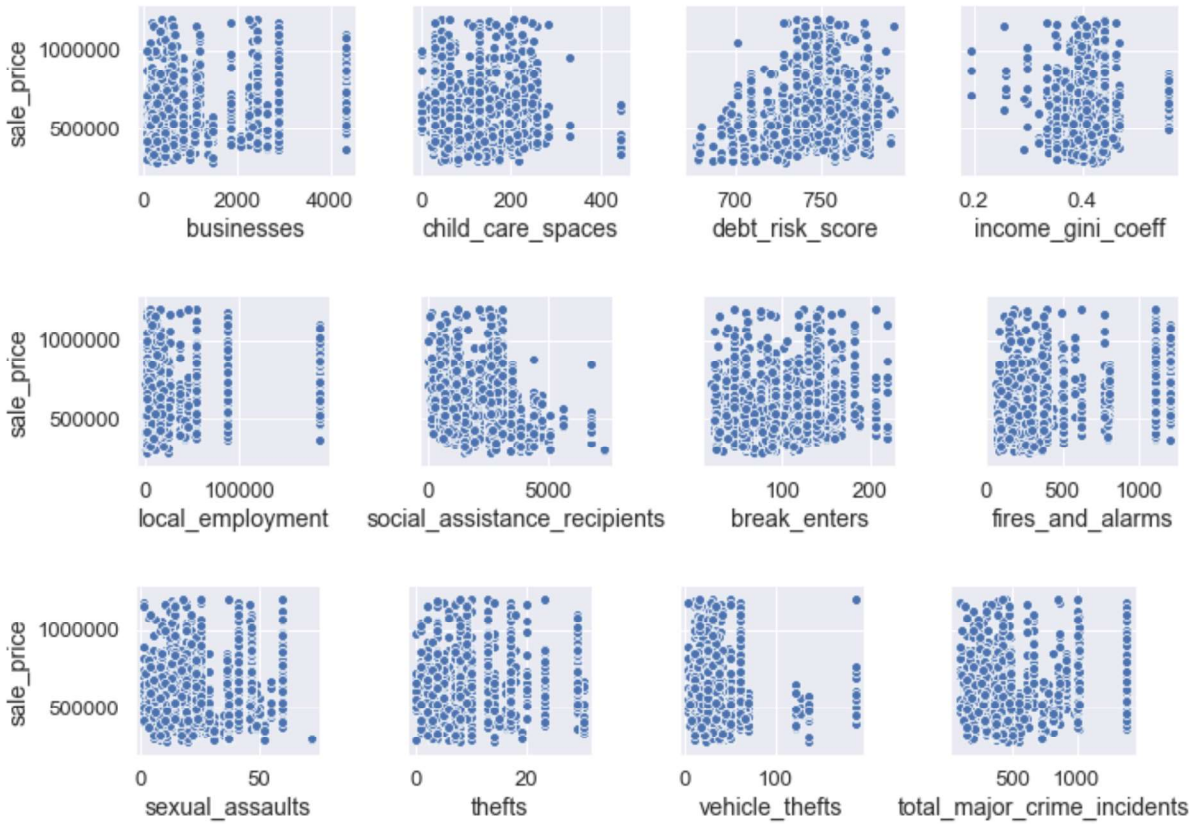
Out[156]:

	bedrooms	bathrooms	square_feet	days_on_market	parking_places	yearly_taxes	s
5	1	1	599	34	0	2586.00	0
6	1	1	599	32	1	1849.30	0
7	2	2	799	32	0	2184.95	0
10	2	2	1199	30	1	3952.84	0

In [157]:
```
x.shape
```

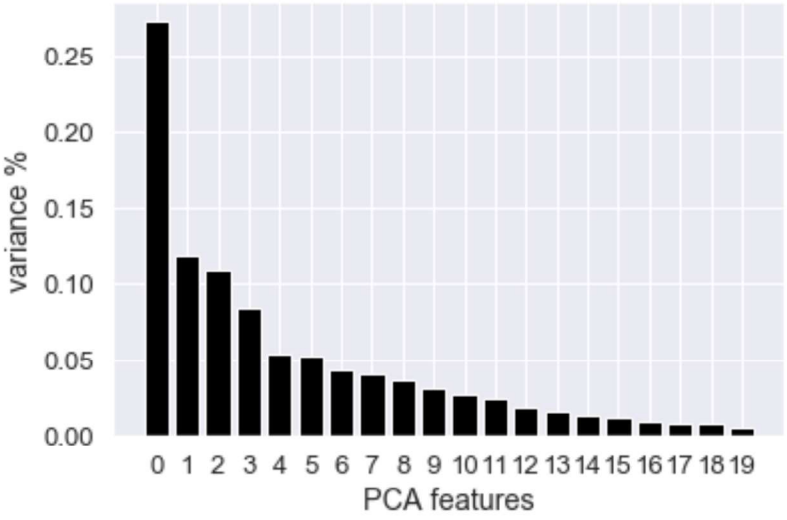
Out[157]: (1760, 23)

In [158]:
```
for i in range(13,25,4):
    sns.pairplot(data=condos_cleaned,
                 x_vars=condos_cleaned.columns[i:i+4],
                 y_vars=['sale_price'])
```



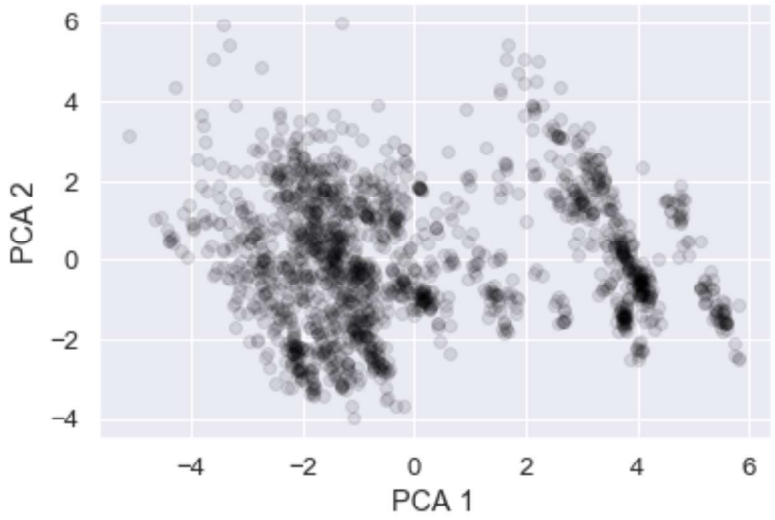
A.16 KMEANS CLUSTERING WITH PCA

In [159]:
```
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_scaled = sc.fit_transform(x)
pca = PCA(n_components=20)
principalComponents = pca.fit_transform(x_scaled)

features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)

PCA_components = pd.DataFrame(principalComponents)
```



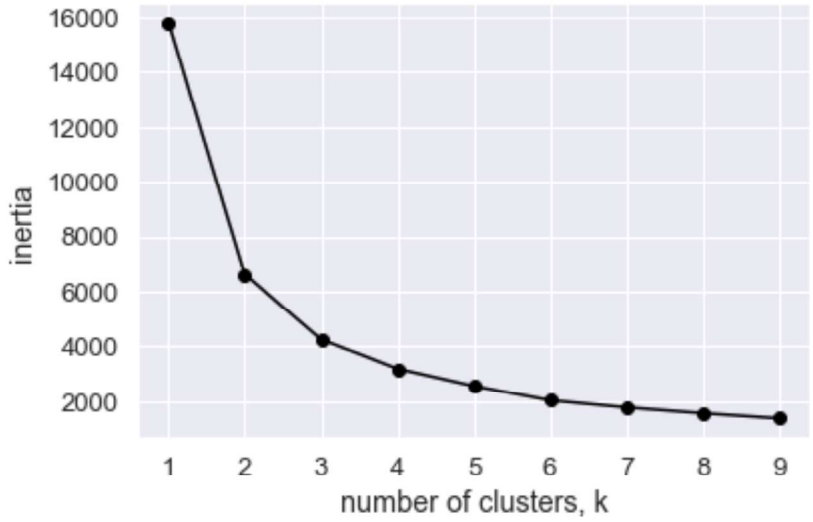
In [160]:
```
plt.scatter(PCA_components[0], PCA_components[1], alpha=.1, color='blac
k')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
```

Out[160]: <matplotlib.text.Text at 0x1a240f2ef0>



In [161]:
```
pca = PCA(n_components=2)
x_pca = pca.fit_transform(x_scaled)
```

In [162]:
```
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:2])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



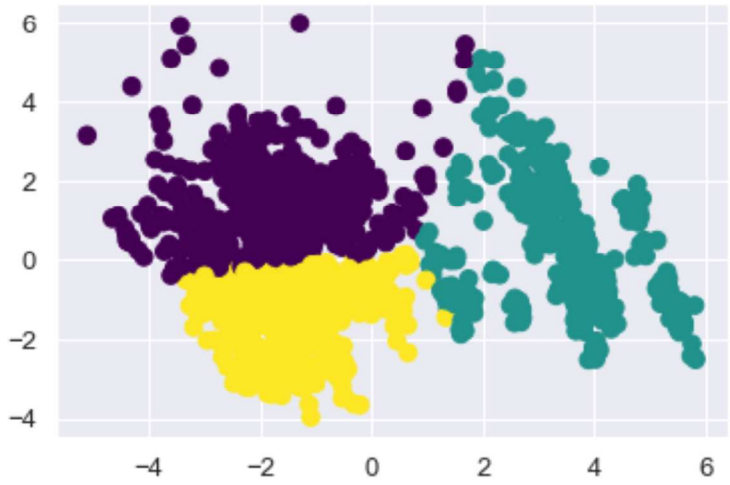
In [163]:
```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=170)
kmeans = kmeans.fit(PCA_components.iloc[:,:2])
kmeans
```

Out[163]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='aut
o',
       random_state=170, tol=0.0001, verbose=0)
```

In [164]:
```
print("The cluster centroids are: \n", kmeans.cluster_centers_)
print("Cluster_label:\n", kmeans.labels_)
```

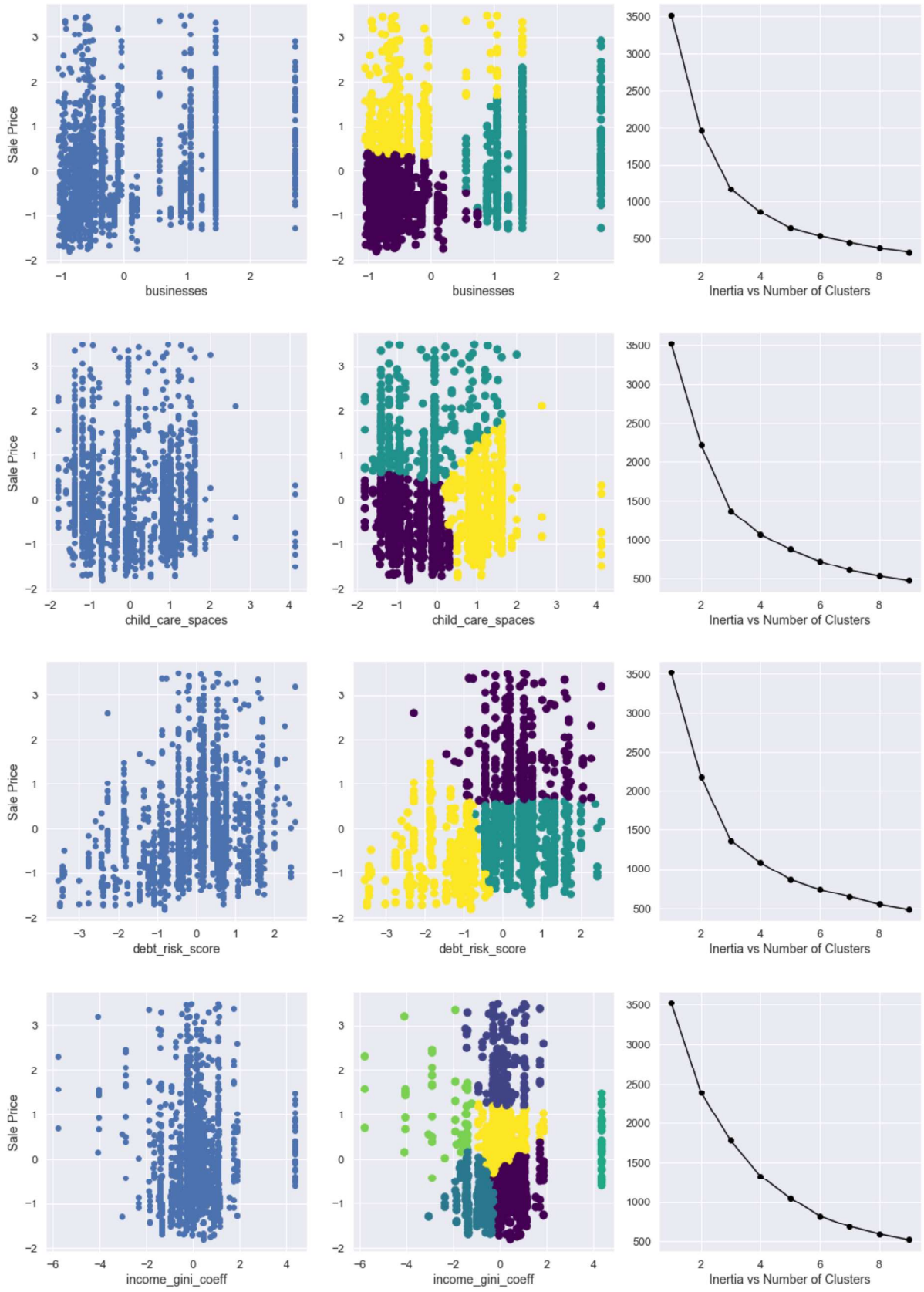
```
The cluster centroids are:
 [[-1.71827824  1.30489472]
 [ 3.50066172  0.21392529]
 [-1.24151665 -1.44093781]]
Cluster_label:
 [1 1 2 ... 1 0 2]
```

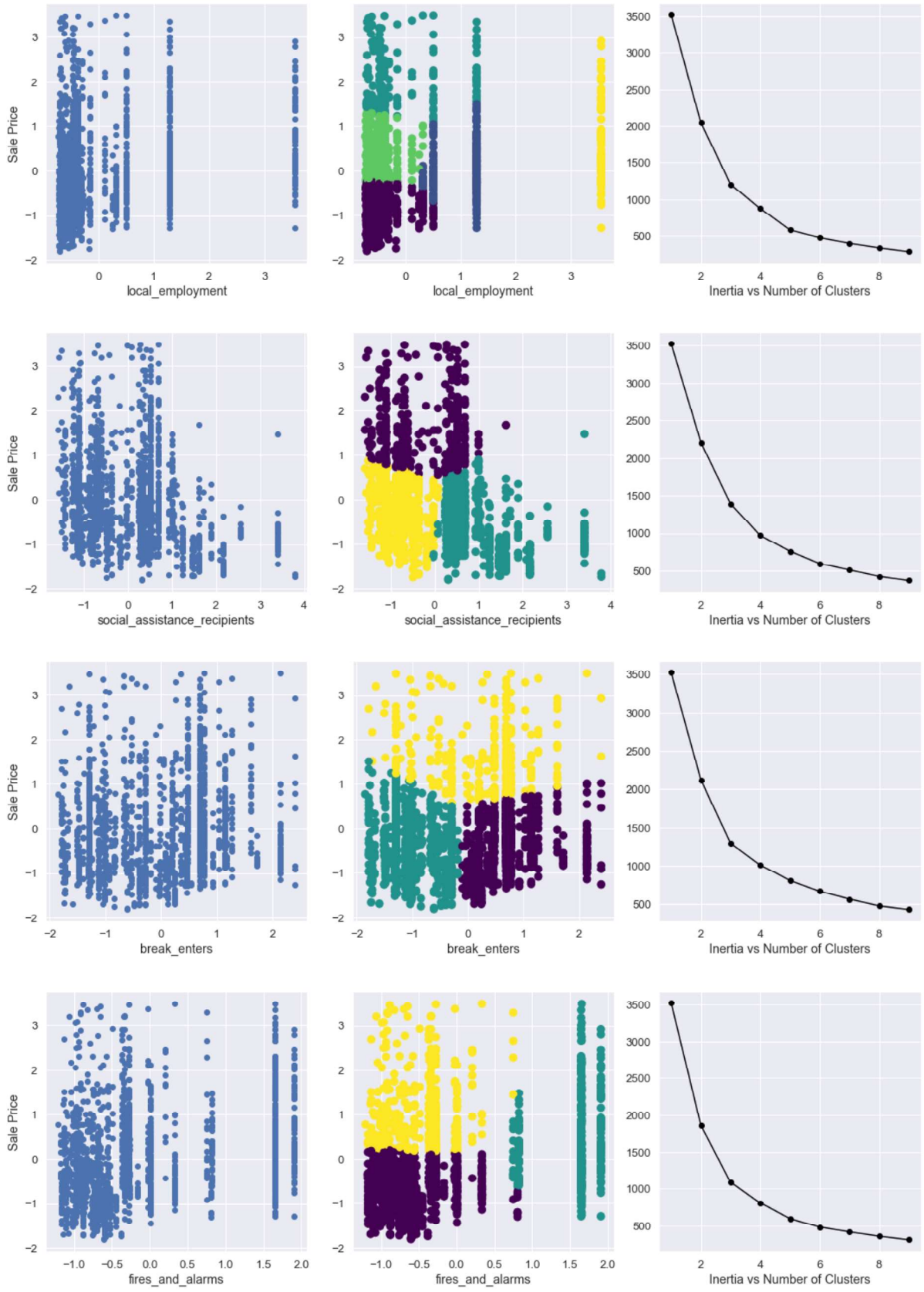
In [165]:
```
colors = ['blue','red']
plt.scatter(PCA_components[0],
            PCA_components[1],
            c=kmeans.labels_,
            cmap='viridis',
            s=75)
plt.show()
```

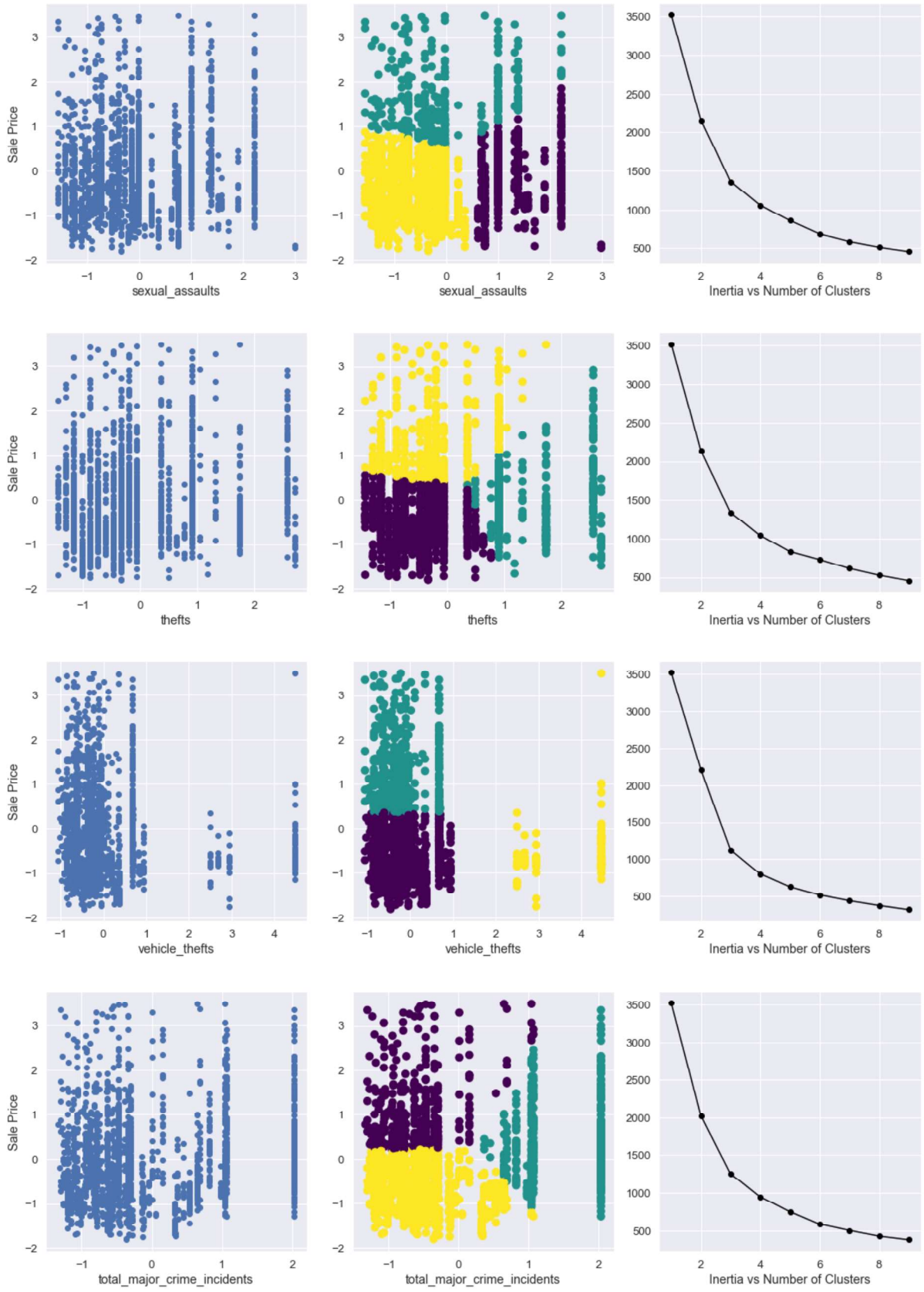


A.17 KMEANS CLUSTERING FOR ALL SOCIO-ECONOMIC VARIABLES VS SALE PRICE

In [166]:
```
sev = condos_cleaned.iloc[:,13:25]
sev['sale_price'] = condos_cleaned['sale_price']

scaled_features = sc.fit_transform(sev)

sev_norm = pd.DataFrame(scaled_features, index=sev.index, columns=sev.co
lumns)

k_values = [3,3,3,6,5,3,3,3,3,3,3,3]
for i in range(0, 12):
    f, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(15, 5))

    f.tight_layout()

    ax1.scatter(sev_norm.iloc[:,i],sev_norm.iloc[:,12])
    ax1.set_ylabel('Sale Price')
    ax1.set_xlabel(sev_norm.columns[i])

    inertias = []
    ks = range(1, 10)
    x_norm = sev_norm.iloc[:,[i,12]]
    for k in ks:
        model = KMeans(n_clusters=k)
        model.fit(x_norm)
        inertias.append(model.inertia_)

    kmeans = KMeans(n_clusters=k_values[i], init='k-means++', random_sta
te=170)
    kmeans = kmeans.fit(x_norm)

    ax2.scatter(sev_norm.iloc[:, i],
            sev_norm.iloc[:, 12],
            c=kmeans.labels_,
            cmap='viridis',
            s=75)
    ax2.set_xlabel(sev_norm.columns[i])

    ax3.plot(ks, inertias, '-o', color='black')
    ax3.set_xlabel('Inertia vs Number of Clusters')

    plt.show()
```

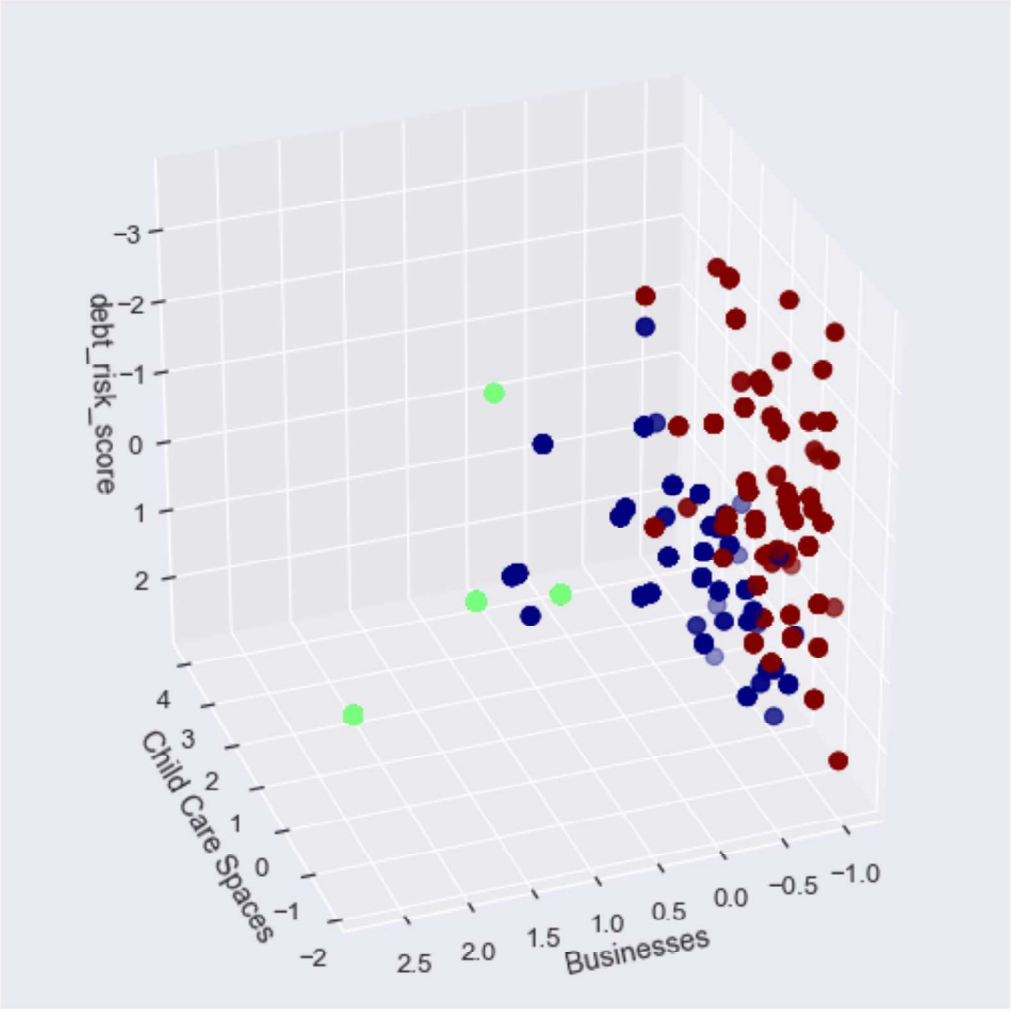

Sale Price
businesses
Inertia vs Number of Clusters
child_care_spaces
debt_risk_score
income_gini_coeff


Sale Price
local_employment
Inertia vs Number of Clusters
social_assistance_recipients
break_enters
fires_and_alarms

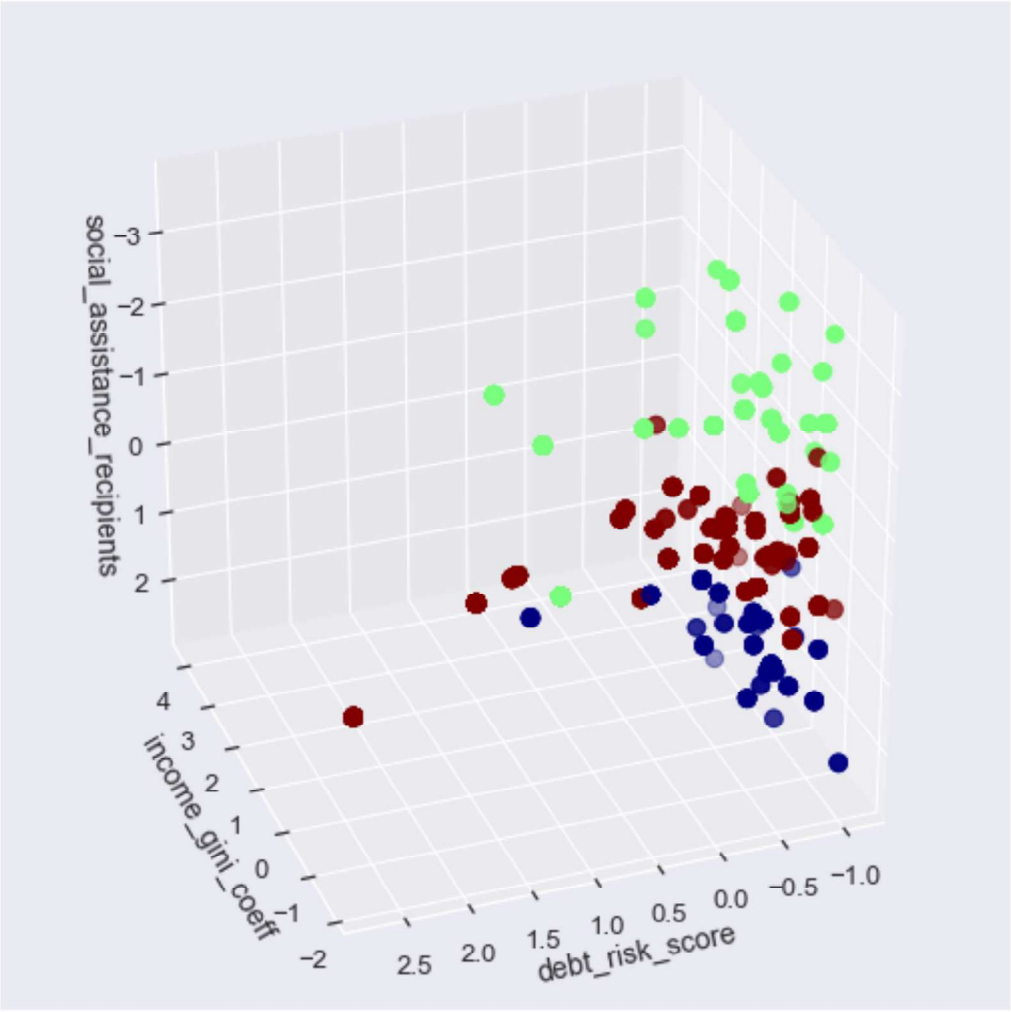

Sale Price
sexual_assaults
Inertia vs Number of Clusters
thefts
vehicle_thefts
total_major_crime_incidents

A.18 KMEANS CLUSTERING 3-D PLOTS FOR ALL SOCIO-ECONOMIC VARIABLES

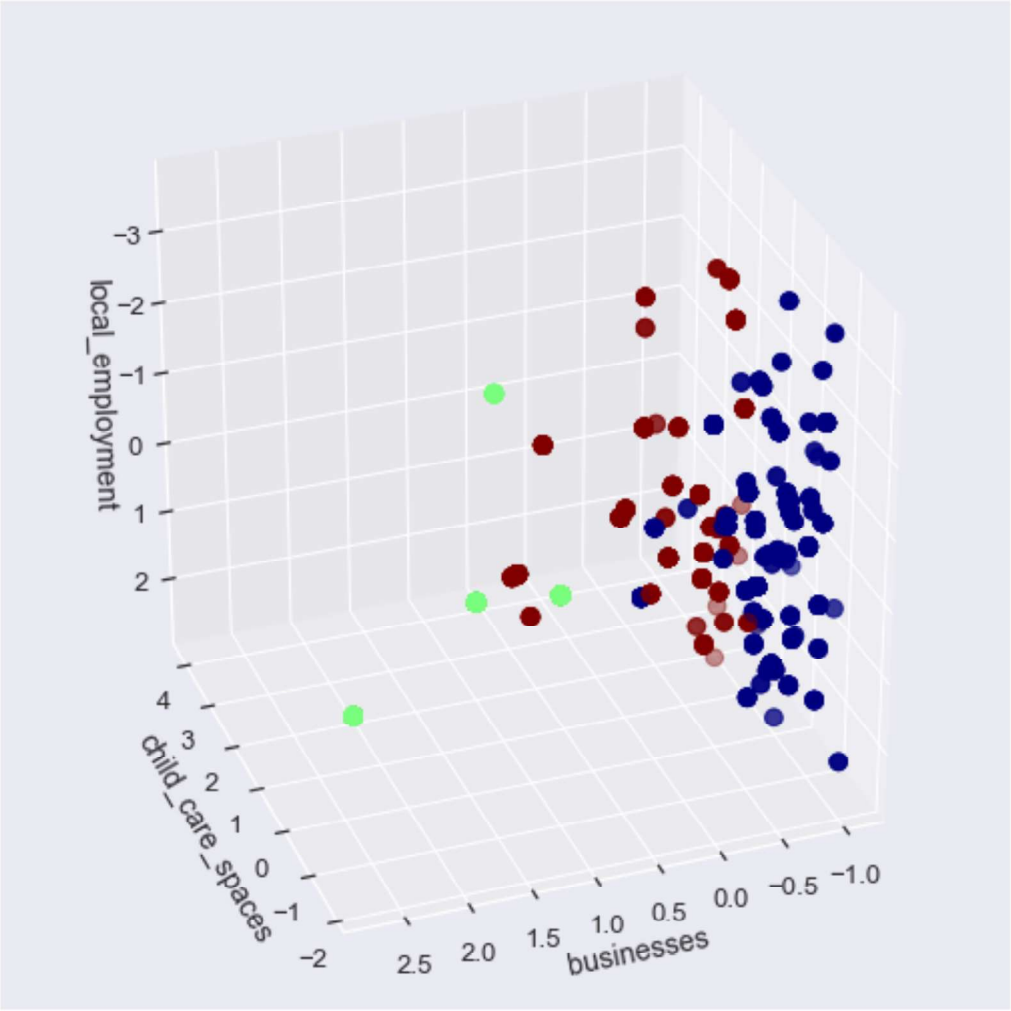
In [167]:
```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=170)
kmeans = kmeans.fit(sev_norm.iloc[:,0:3])


fig = plt.figure(1, figsize=(7, 7))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(sev_norm.iloc[:,0],
           sev_norm.iloc[:,1],
           sev_norm.iloc[:,2],
           c=kmeans.labels_,
           cmap='jet_r',
           s=75)
ax.set_xlabel('Businesses')
ax.set_ylabel('Child Care Spaces')
ax.set_zlabel('debt_risk_score')
plt.show()
```



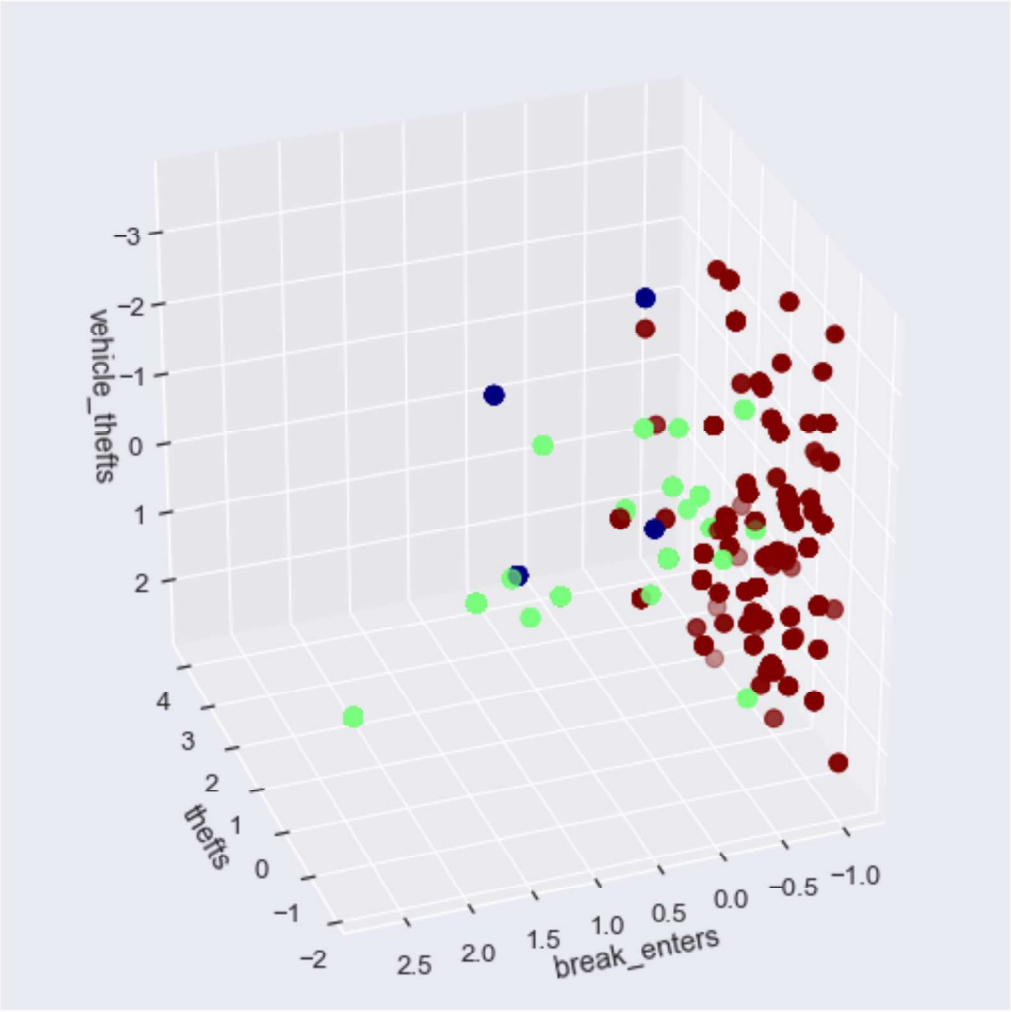
In [168]:
```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=170)
labels = ['debt_risk_score', 'income_gini_coeff', 'social_assistance_rec
ipients']
kmeans = kmeans.fit(sev_norm.loc[:,labels])


fig = plt.figure(1, figsize=(7, 7))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(sev_norm.iloc[:,0],
           sev_norm.iloc[:,1],
           sev_norm.iloc[:,2],
           c=kmeans.labels_,
           cmap='jet_r',
           s=75)
ax.set_xlabel(labels[0])
ax.set_ylabel(labels[1])
ax.set_zlabel(labels[2])
plt.show()
```



In [169]:
```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=170)
labels = ['businesses', 'child_care_spaces', 'local_employment']
kmeans = kmeans.fit(sev_norm.loc[:,labels])


fig = plt.figure(1, figsize=(7, 7))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(sev_norm.iloc[:,0],
           sev_norm.iloc[:,1],
           sev_norm.iloc[:,2],
           c=kmeans.labels_,
           cmap='jet_r',
           s=75)
ax.set_xlabel(labels[0])
ax.set_ylabel(labels[1])
ax.set_zlabel(labels[2])
plt.show()
```

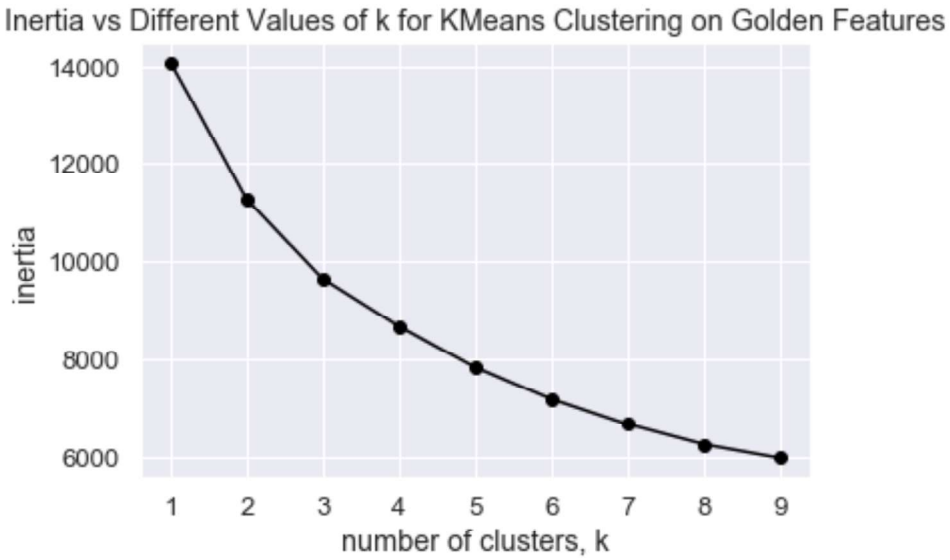


In [170]:
```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=170)
labels = ['break_enters', 'thefts', 'vehicle_thefts']
kmeans = kmeans.fit(sev_norm.loc[:,labels])


fig = plt.figure(1, figsize=(7, 7))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(sev_norm.iloc[:,0],
           sev_norm.iloc[:,1],
           sev_norm.iloc[:,2],
           c=kmeans.labels_,
           cmap='jet_r',
           s=75)
ax.set_xlabel(labels[0])
ax.set_ylabel(labels[1])
ax.set_zlabel(labels[2])
plt.show()
```



A.19 KMEANS FOR SELECTED FEATURES

In [171]:
```
k_data = condos_cleaned[['yearly_taxes', 'businesses',
       'square_feet', 'break_enters', 'parking_places', 'park_distance',
       'social_assistance_recipients', 'transit_distance']]

k_data_norm = sc.fit_transform(k_data)



ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(k_data_norm)

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.title('Inertia vs Different Values of k for KMeans Clustering on Gol
den Features')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



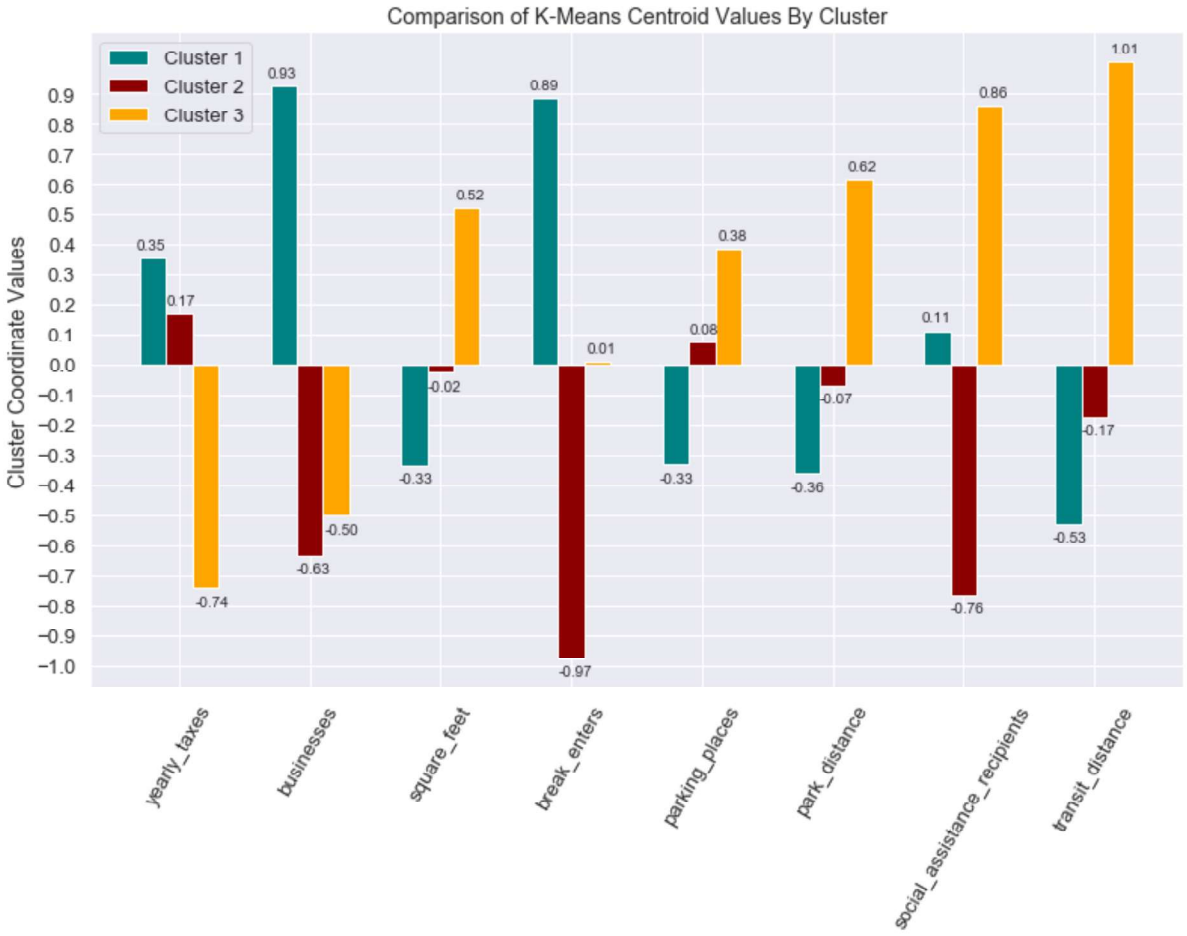
In [172]:
```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=170)
kmeans = kmeans.fit(k_data_norm)

print("The cluster centroids are: \n", kmeans.cluster_centers_)
print("Cluster_label:\n", kmeans.labels_)
```

```
The cluster centroids are:
 [[ 0.35332912  0.92694473 -0.33396801  0.88771086 -0.33224556 -0.35981
584
   0.11233273 -0.52940073]
 [ 0.16972456 -0.63407712 -0.02440088 -0.97458145  0.07524721 -0.068826
93
  -0.76466661 -0.17382368]
 [-0.74351931 -0.5029217   0.52033547  0.0110638   0.38416734  0.617660
97
   0.86160052  1.00608942]]
Cluster_label:
 [0 0 1 ... 0 1 2]
```

A.20 COMPARISON OF CLUSTER CENTROIDS

In [173]:
```
label = ['yearly_taxes', 'businesses',
       'square_feet', 'break_enters', 'parking_places', 'park_distance',
       'social_assistance_recipients', 'transit_distance']


plt.figure(figsize=(13, 8))

ind = np.arange(len(label))
width = 0.2

p1 = plt.bar(ind - 0.3, kmeans.cluster_centers_[0], width, color="teal",
align="edge")
p2 = plt.bar(ind, kmeans.cluster_centers_[1], width, color="darkred")
p3 = plt.bar(ind + 0.2, kmeans.cluster_centers_[2], width, color="orang
e")


plt.ylabel('Cluster Coordinate Values')
plt.xticks(ind, label, rotation=60)
plt.yticks(np.arange(-1, 1, 0.1))
# plt.legend(bbox_to_anchor=(0, 1), loc='upper right', ncol=1)
plt.legend((p1[0], p2[0], p3[0]), ('Cluster 1', 'Cluster 2', 'Cluster 3'
))

for v in ind:
    first = kmeans.cluster_centers_[0][v]
    second = kmeans.cluster_centers_[1][v]
    third = kmeans.cluster_centers_[2][v]
    padding = 0.03
    neg_padding = 0.06
    fs = 10
    plt.text(v-0.32, first - neg_padding if first < 0 else first + paddi
ng , "%.2f" % first, fontsize=fs)
    plt.text(v-0.1, second - neg_padding if second < 0 else second + pad
ding , "%.2f" % second, fontsize=fs)
    plt.text(v+0.12, third - neg_padding if third < 0 else third + paddi
ng , "%.2f" % third, fontsize=fs)

plt.title('Comparison of K-Means Centroid Values By Cluster')

plt.show()


## Alter:
# index = pd.Index(label)

# data = {
#     'Cluster 1': kmeans.cluster_centers_[0],
#     'Cluster 2': kmeans.cluster_centers_[1],
#     'Cluster 3': kmeans.cluster_centers_[2],
# }

# df = pd.DataFrame(data, index=index)
# ax = df.plot(kind='bar', stacked=True, figsize=(13, 8), color=['teal',
'darkred', 'orange'], width=0.3)
# # ax.set_ylabel('foo')
```

```
# # plt.savefig('stacked.png')
# plt.yticks(np.arange(-1, 1, 0.1))

# plt.show()
```



A.21 CLUSTER PRICE VARIATIONS

In [174]:
```
# Apply cluster labels to original dataframe
condos_cleaned['cluster'] = kmeans.labels_
kmeans_condos = condos_cleaned
kmeans_condos['cluster'] = kmeans_condos['cluster'] + 1
```

In [175]:
```
kmeans_condos.groupby('cluster')[['sale_price']].count()
```

Out[175]:

	sale_price
cluster	
1	676
2	622
3	462

In [176]:
```
kmeans_condos.groupby('cluster')[['sale_price']].mean()
```

Out[176]:

	sale_price
cluster	
1	651267.590237
2	619307.041801
3	473711.183983

In [177]:
```
kmeans_condos.groupby('cluster')[['sale_price']].std()
```

Out[177]:

	sale_price
cluster	
1	162389.010164
2	171428.480550
3	129766.139238

In [178]:
```
kmeans_condos['y'] = 0
kmeans_condos['color']='darkred'
kmeans_condos.loc[kmeans_condos.cluster == 1, 'y'] = 1
kmeans_condos.loc[kmeans_condos.cluster == 2, 'y'] = 0
kmeans_condos.loc[kmeans_condos.cluster == 3, 'y'] = -1

kmeans_condos.loc[kmeans_condos.cluster == 1, 'color'] = "teal"
kmeans_condos.loc[kmeans_condos.cluster == 2, 'color'] = "darkred"
kmeans_condos.loc[kmeans_condos.cluster == 3, 'color'] = "orange"

# markers = {1: "s", 2: "X", 3:"o"}
plt.figure(figsize=(10, 5))
arks=['x','d','o']
for i in range(1,4):
    x = kmeans_condos.loc[kmeans_condos.cluster == i,'sale_price']
    y = kmeans_condos.loc[kmeans_condos.cluster == i,'y']
    color = kmeans_condos.loc[kmeans_condos.cluster == i,'color']
    plt.scatter(x, y,  s=30, c=color, marker=marks[i-1])
plt.yticks([1,0,-1], ['Cluster 1','Cluster 2','Cluster 3'])
plt.xlabel('Sale Price (CAD$)')
plt.legend(('Cluster 1', 'Cluster 2', 'Cluster 3'))
plt.title('Variation in Sale Price Across Clusters')
plt.show()
```

A.22 DISTANCES BETWEEN CLUSTER CENTROIDS

In [179]:
```
# calculating distances between centroids

import math

distances = [[0,0,0],[0,0,0],[0,0,0]]
for i in range(0,3):
    for j in range(0,3):
        if i != j:
            sum_values = 0
            for k in range(0,8):
                sum_values += ( kmeans.cluster_centers_[i][k] - kmeans.c
luster_centers_[j][k] )**2
            distance = math.sqrt(sum_values)
            distances[i][j]=distance
distances
```

Out[179]:
```
[[0, 2.6796925273706806, 3.0222143511279795],
 [2.6796925273706806, 0, 2.5928916972929277],
 [3.0222143511279795, 2.5928916972929277, 0]]
```

In [180]:
```
plt.figure(figsize=(5, 5))
sns.set(font_scale=1.2)
sns.heatmap(distances,
            cmap='Blues', vmax=3, vmin=0, linewidths=0.1,
            annot=True, annot_kws={"size": 15}, square=True,
           xticklabels=[1,2,3], yticklabels=[1,2,3] );
plt.title('Heatmap of Distances Between Cluster Centroids', y=1.15)
plt.xlabel('Cluster')
plt.ylabel('Cluster')
```

Out[180]:
```
<matplotlib.text.Text at 0x1a269224e0>
```