K-Means Clustering

Zander Bonnet

September 4, 2024

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style as sty
sty.use('default')
import seaborn as sns
from sklearn.preprocessing import MultiLabelBinarizer, StandardScaler
from yellowbrick.cluster import SilhouetteVisualizer, KElbowVisualizer
from sklearn.cluster import KMeans
```

In the k-means process, we select k-random points to make the center of our clusters and then assign all surrounding points to the closest 'centroid'. After doing this we take the clusters of points and find the average of them. This average becomes the centroid, and the process repeats until the model converges. This means that there are no, or very little, data points that jump between clusters in between runs.

The major questions that we hope to awnser when using kmeans clustering are:

1. Can we detect any patterns within the data?
2. If we do detect patterns what insite can we gain from it?

In [2]:
```
#Creates a Kmeans method
from sklearn.metrics.pairwise import pairwise_distances
class MyKMeans:
    def __init__(self, n_clusters, max_iters=100, seed = None):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        if seed != None:
            np.random.seed(seed)

    def fit(self, X):
        # Pick random centroids
        self.centroids = X.iloc[np.random.choice(X.shape[0], self.n_clusters, re

        for i in range(self.max_iters):
            # Assign each data point to the nearest centroid
            labels = self.make_labels(X)

            # Update centroids
            new_centroids = self.better_centroids(X, labels)

            # Check for convergence
            if np.all(self.centroids == new_centroids):
                break

            self.centroids = new_centroids
```

```
    def make_labels(self, X):
        # Compute distances from each data point to centroids
        distances = pairwise_distances(X,self.centroids)

        # Assign labels based on the nearest centroid
        return np.argmin(distances, axis=1)

    def better_centroids(self, X, labels):
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(sel
        return new_centroids
```

The dataset is explained below. Using clustering we hope to uncover trends within the data that are not easily seen at first glance of the data.

In [3]:

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
online_shoppers_purchasing_intention_dataset = fetch_ucirepo(id=468)

# data (as pandas dataframes)
X = online_shoppers_purchasing_intention_dataset.data.features
y = online_shoppers_purchasing_intention_dataset.data.targets

# metadata
print(online_shoppers_purchasing_intention_dataset.metadata)

# variable information
print(online_shoppers_purchasing_intention_dataset.variables)
```


```
{'uci_id': 468, 'name': 'Online Shoppers Purchasing Intention Dataset', 'reposit
ory_url': 'https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+in
tention+dataset', 'data_url': 'https://archive.ics.uci.edu/static/public/468/dat
a.csv', 'abstract': 'Of the 12,330 sessions in the dataset,\n84.5% (10,422) were
negative class samples that did not\nend with shopping, and the rest (1908) were
positive class\nsamples ending with shopping.', 'area': 'Business', 'tasks': ['C
lassification', 'Clustering'], 'characteristics': ['Multivariate'], 'num_instanc
es': 12330, 'num_features': 17, 'feature_types': ['Integer', 'Real'], 'demograph
ics': [], 'target_col': ['Revenue'], 'index_col': None, 'has_missing_values': 'n
o', 'missing_values_symbol': None, 'year_of_dataset_creation': 2018, 'last_updat
ed': 'Thu Jan 11 2024', 'dataset_doi': '10.24432/C5F88Q', 'creators': ['C. Saka
r', 'Yomi Kastro'], 'intro_paper': {'title': 'Real-time prediction of online sho
ppers' purchasing intention using multilayer perceptron and LSTM recurrent neura
l networks', 'authors': 'C. O. Sakar, S. Polat, Mete Katircioglu, Yomi Kastro',
'published_in': 'Neural computing & applications (Print)', 'year': 2019, 'url':
'https://www.semanticscholar.org/paper/747e098f85ca2d20afd6313b11242c0c427e6fb
3', 'doi': '10.1007/s00521-018-3523-0'}, 'additional_info': {'summary': 'The dat
aset consists of feature vectors belonging to 12,330 sessions. \r\nThe dataset w
as formed so that each session\r\nwould belong to a different user in a 1-year p
eriod to avoid\r\nany tendency to a specific campaign, special day, user\r\nprof
ile, or period. ', 'purpose': None, 'funded_by': None, 'instances_represent': No
ne, 'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_desc
ription': None, 'variable_info': 'The dataset consists of 10 numerical and 8 cat
egorical attributes.\r\nThe \'Revenue\' attribute can be used as the class labe
l.\r\n\r\n"Administrative", "Administrative Duration", "Informational", "Informa
tional Duration", "Product Related" and "Product Related Duration" represent the
number of different types of pages visited by the visitor in that session and to
tal time spent in each of these page categories. The values of these features ar
e derived from the URL information of the pages visited by the user and updated
```

```
in real time when a user takes an action, e.g. moving from one page to another.
The "Bounce Rate", "Exit Rate" and "Page Value" features represent the metrics m
easured by "Google Analytics" for each page in the e-commerce site. The value of
"Bounce Rate" feature for a web page refers to the percentage of visitors who en
ter the site from that page and then leave ("bounce") without triggering any oth
er requests to the analytics server during that session. The value of "Exit Rat
e" feature for a specific web page is calculated as for all pageviews to the pag
e, the percentage that were the last in the session. The "Page Value" feature re
presents the average value for a web page that a user visited before completing
an e-commerce transaction. The "Special Day" feature indicates the closeness of
the site visiting time to a specific special day (e.g. Mother’s Day, Valentine
\'s Day) in which the sessions are more likely to be finalized with transaction.
The value of this attribute is determined by considering the dynamics of e-comme
rce such as the duration between the order date and delivery date. For example,
for Valentina’s day, this value takes a nonzero value between February 2 and Feb
ruary 12, zero before and after this date unless it is close to another special
day, and its maximum value of 1 on February 8. The dataset also includes operati
ng system, browser, region, traffic type, visitor type as returning or new visit
or, a Boolean value indicating whether the date of the visit is weekend, and mon
th of the year.', 'citation': None}}
```

```
                       name     role         type demographic description  \
0            Administrative  Feature      Integer        None        None
1   Administrative_Duration  Feature      Integer        None        None
2             Informational  Feature      Integer        None        None
3    Informational_Duration  Feature      Integer        None        None
4            ProductRelated  Feature      Integer        None        None
5   ProductRelated_Duration  Feature   Continuous        None        None
6               BounceRates  Feature   Continuous        None        None
7                 ExitRates  Feature   Continuous        None        None
8                PageValues  Feature      Integer        None        None
9                SpecialDay  Feature      Integer        None        None
10                    Month  Feature  Categorical        None        None
11         OperatingSystems  Feature      Integer        None        None
12                  Browser  Feature      Integer        None        None
13                   Region  Feature      Integer        None        None
14              TrafficType  Feature      Integer        None        None
15              VisitorType  Feature  Categorical        None        None
16                  Weekend  Feature       Binary        None        None
17                  Revenue   Target       Binary        None        None

   units missing_values
0   None             no
1   None             no
2   None             no
3   None             no
4   None             no
5   None             no
6   None             no
7   None             no
8   None             no
9   None             no
10  None             no
11  None             no
12  None             no
13  None             no
14  None             no
15  None             no
16  None             no
17  None             no
```

In [4]:
```python
X.head()
```

Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated
0	0	0.0	0	0.0	1
1	0	0.0	0	0.0	2
2	0	0.0	0	0.0	1
3	0	0.0	0	0.0	2
4	0	0.0	0	0.0	10

In [5]:
```python
#Get dummy vars and standardize the data
dum1 = pd.get_dummies(X)
scaler = StandardScaler()
dum1 = pd.DataFrame(scaler.fit_transform(dum1))
dum1
```

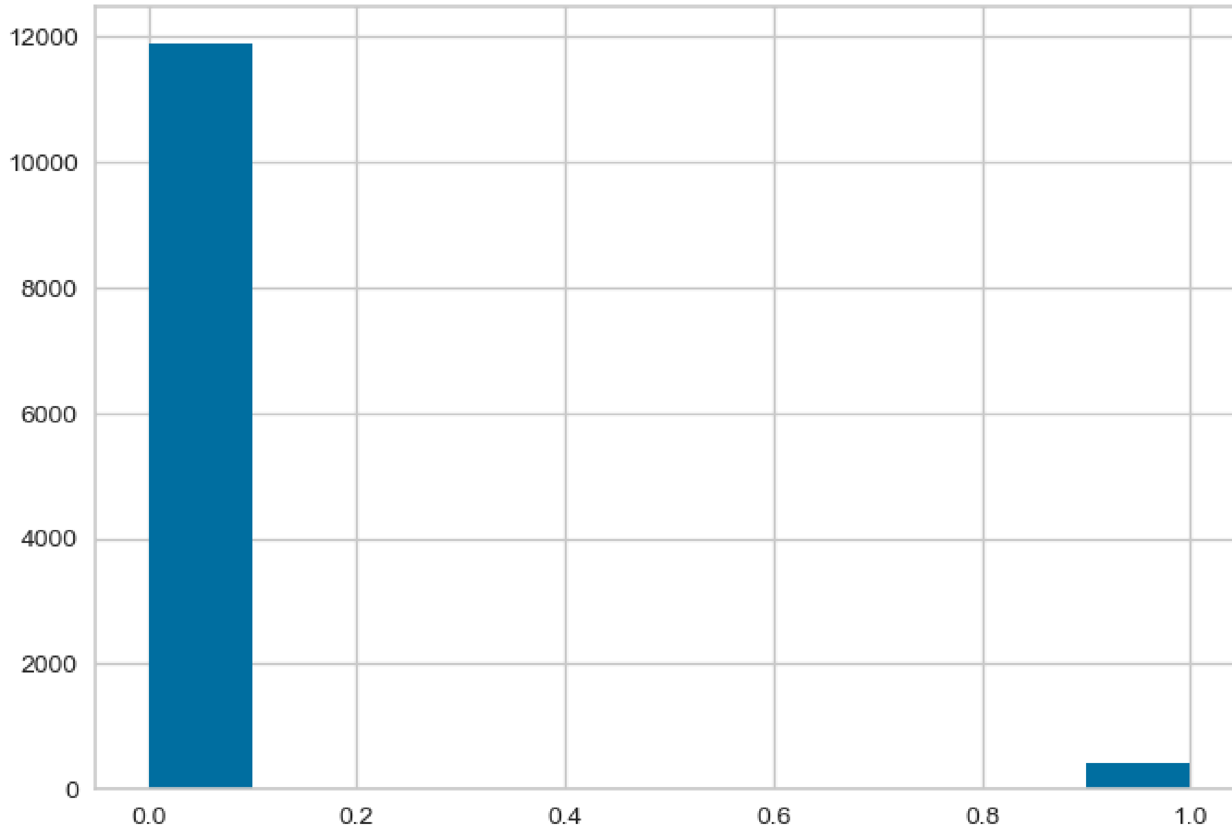
Out[5]:

	0	1	2	3	4	5	6	7
0	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	3.667189	3.229316
1	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.590903	-0.457683	1.171473
2	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	3.667189	3.229316
3	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.622954	0.573535	1.994610
4	-0.696993	-0.457191	-0.396478	-0.244931	-0.488636	-0.296430	-0.045196	0.142551
...	...	...	...	...	...	...	...	...
12325	0.206173	0.363075	-0.396478	-0.244931	0.478227	0.307822	-0.310366	-0.288966
12326	-0.696993	-0.457191	-0.396478	-0.244931	-0.601062	-0.380957	-0.457683	-0.447364
12327	-0.696993	-0.457191	-0.396478	-0.244931	-0.578577	-0.528063	1.261014	0.897093
12328	0.507228	-0.032916	-0.396478	-0.244931	-0.376210	-0.443536	-0.457683	-0.453140
12329	-0.696993	-0.457191	-0.396478	-0.244931	-0.646033	-0.613243	-0.457683	0.485525

12330 rows × 28 columns

In [6]:
```python
#do a mock Kmeans using my method
km = MyKMeans(n_clusters = 2, seed =300)
km.fit(dum1)
pred1 = km.make_labels(dum1)
```

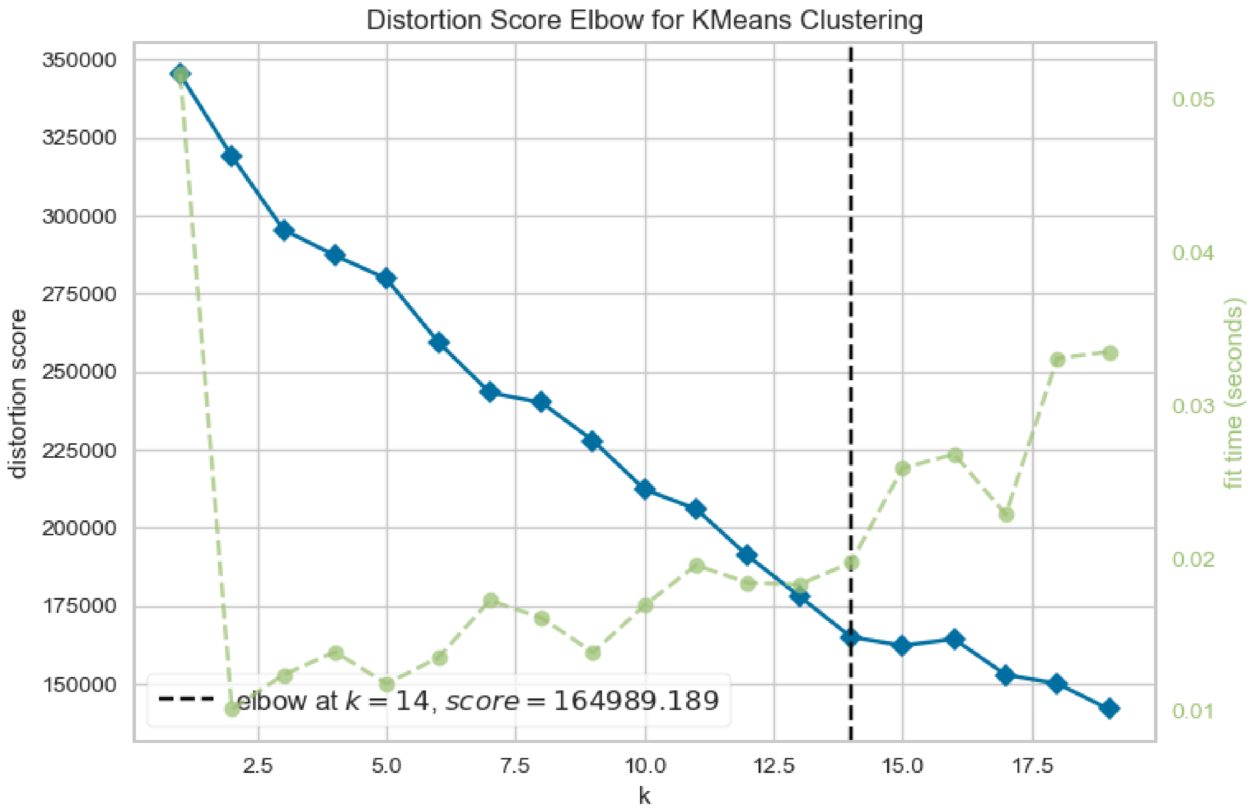
In [7]:
```python
plt.hist(pred1)
plt.show()
```



In [8]:

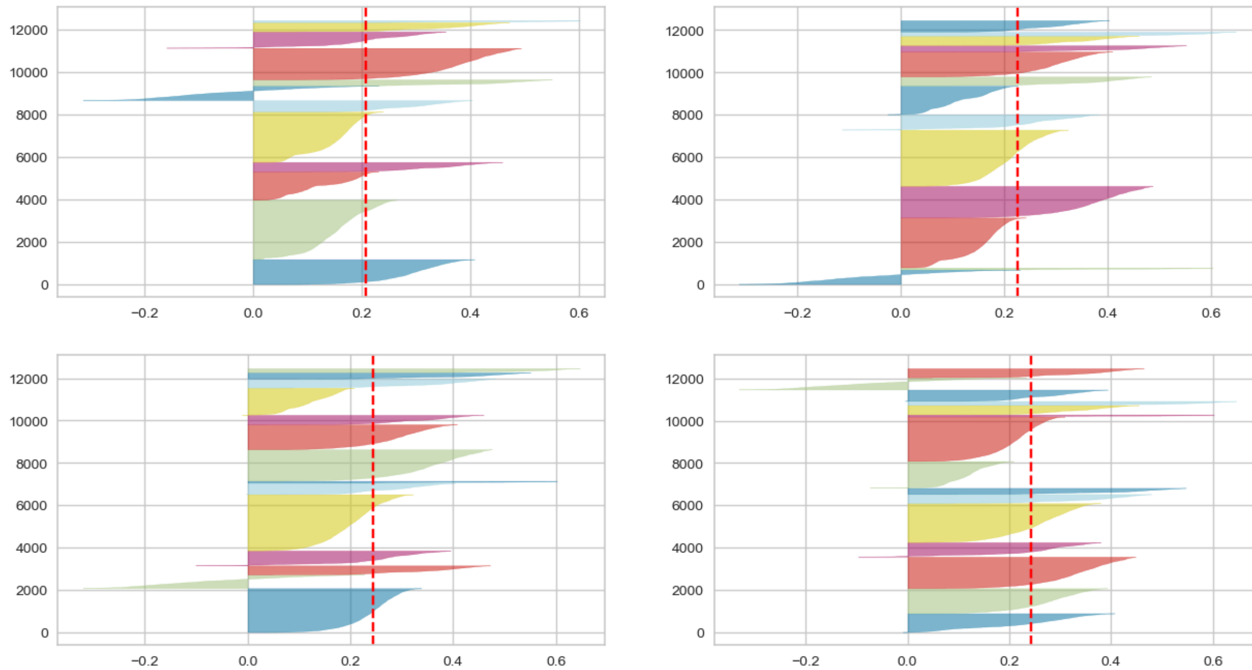
```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,20))

visualizer.fit(dum1)
visualizer.show()
plt.show()
```



In [9]:

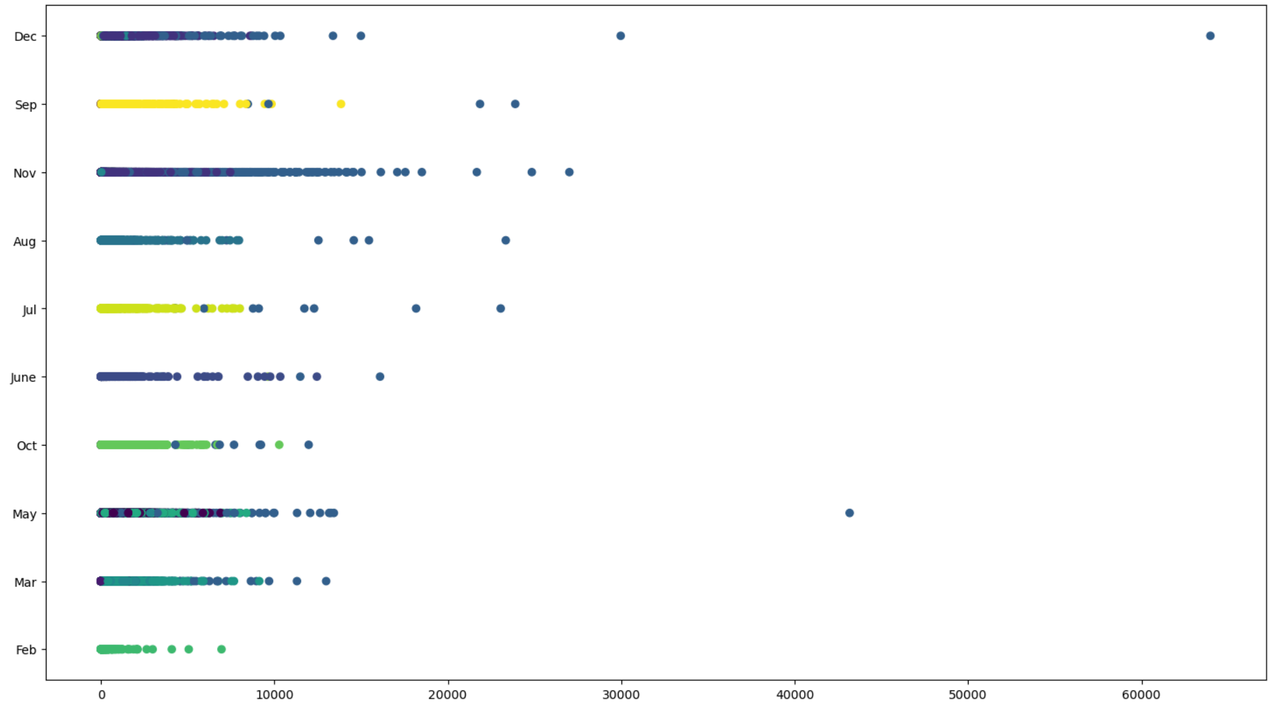
```
fig, ax = plt.subplots(2, 2, figsize=(15,8))
for i in [12, 13, 14, 15]:
    '''
    Create KMeans instance for different number of clusters
    '''
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_
    q, mod = divmod(i, 2)
    '''
    Create SilhouetteVisualizer instance with KMeans instance
    Fit the visualizer
    '''
    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-6][mod])
    visualizer.fit(dum1)
```



Using the elbow plot and the silhouette plot we can see that 14 clusters is most likely the most optimal number of clusters. These plots do also show that there are very large scores associated with them. This could show that the clusters are not very unique, and the model is struggling to find real distinction between the clusters.

In [10]:
```
#fit the model
kmeans = KMeans(n_clusters=14)
pred1 = kmeans.fit(dum1)
```

In [11]:
```
plt.figure(figsize = (18,10))
sty.use('default')
plt.scatter(X['ProductRelated_Duration'], X['Month'], c = pred1.labels_)
plt.show()
```



When looking at the product-related duration vs the month we can see that there is some separation between the clusters. We can see that September seems to be its own cluster, and February seems to also be isolated. This could lead us to see that our data is potentially seasonal. With this model we were able to see that the data might have a seasonal pattern, as it appears that the clusters are formed around the months of the year. So we were able to answer both of our questions with this analysis.

The next dataset is described below.

In [12]:
```
# fetch dataset
apartment_for_rent_classified = fetch_ucirepo(id=555)

# data (as pandas dataframes)
X = apartment_for_rent_classified.data.features
y = apartment_for_rent_classified.data.targets

# metadata
print(apartment_for_rent_classified.metadata)

# variable information
print(apartment_for_rent_classified.variables)
```


```
{'uci_id': 555, 'name': 'Apartment for Rent Classified', 'repository_url': 'http
s://archive.ics.uci.edu/dataset/555/apartment+for+rent+classified', 'data_url':
'https://archive.ics.uci.edu/static/public/555/data.csv', 'abstract': 'This is a
dataset of classified for apartments for rent in USA.\n', 'area': 'Business', 't
asks': ['Classification', 'Regression', 'Clustering'], 'characteristics': ['Mult
ivariate'], 'num_instances': 10000, 'num_features': 21, 'feature_types': ['Categ
orical', 'Integer'], 'demographics': [], 'target_col': None, 'index_col': ['i
d'], 'has_missing_values': 'no', 'missing_values_symbol': None, 'year_of_dataset
_creation': 2019, 'last_updated': 'Mon Feb 26 2024', 'dataset_doi': '10.24432/C5
X623', 'creators': [], 'intro_paper': None, 'additional_info': {'summary': "The
dataset contains of 10'000 or 100'000 rows and of 22 columns The data has been c
leaned in the way that \r\ncolumn price and square_feet never is empty but the d
```

ataset is saved as it was created.\r\n\r\nCan be used for different machine lear
ning tasks such as clustering, classification and also regression for the square
s feet column", 'purpose': None, 'funded_by': None, 'instances_represent': None,
'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_descript
ion': None, 'variable_info': 'id = unique identifier of apartment\ncategory = ca
tegory of classified\ntitle = title text of apartment\nbody = body text of apart
ment\namenities = like AC, basketball,cable, gym, internet access, pool, refrige
rator etc.\nbathrooms = number of bathrooms\nbedrooms = number of bedrooms\ncurr
ency = price in current\nfee = fee\nhas_photo = photo of apartment\npets_allowed
= what pets are allowed dogs/cats etc.\nprice = rental price of apartment\nprice
_display = price converted into display for reader\nprice_type = price in USD\ns
quare_feet = size of the apartment\naddress = where the apartment is located\nc
ityname = where the apartment is located\nstate = where the apartment is locat
ed\nlatitude = where the apartment is located\nlongitude = where the apartment i
s located\nsource = origin of classified\ntime = when classified was created',
'citation': None}}
```
             name     role         type demographic description units  \
0              id       ID      Integer        None        None  None
1        category  Feature  Categorical        None        None  None
2           title  Feature  Categorical        None        None  None
3            body  Feature  Categorical        None        None  None
4       amenities  Feature  Categorical        None        None  None
5       bathrooms  Feature      Integer        None        None  None
6        bedrooms  Feature  Categorical        None        None  None
7        currency  Feature  Categorical        None        None  None
8             fee  Feature  Categorical        None        None  None
9       has_photo  Feature  Categorical        None        None  None
10   pets_allowed  Feature  Categorical        None        None  None
11          price  Feature      Integer        None        None  None
12  price_display  Feature      Integer        None        None  None
13     price_type  Feature  Categorical        None        None  None
14    square_feet  Feature  Categorical        None        None  None
15        address  Feature  Categorical        None        None  None
16       cityname  Feature  Categorical        None        None  None
17          state  Feature      Integer        None        None  None
18       latitude  Feature      Integer        None        None  None
19      longitude  Feature  Categorical        None        None  None
20         source  Feature      Integer        None        None  None
21           time  Feature  Categorical        None        None  None

   missing_values
0              no
1              no
2              no
3              no
4              no
5              no
6              no
7              no
8              no
9              no
10             no
11             no
12             no
13             no
14             no
15             no
16             no
17             no
18             no
```

```
19            no
20            no
21            no
/var/folders/fc/97_w0wn53dd0skbf332p6dd00000gn/T/ipykernel_39279/2935559136.py:
2: DtypeWarning: Columns (0,5,6,12,14,15) have mixed types.Specify dtype option
on import or set low_memory=False.
  apartment_for_rent_classified = fetch_ucirepo(id=555)
```

In [13]:

```
#pick the most important variables  and elminate redundent varaibles
keep = ['amenities', 'bathrooms', 'bedrooms', 'currency', 'fee', 'has_photo', 'p
        'price', 'price_type', 'square_feet', 'latitude', 'longitude']
clean = X[keep]
#Take a sample of the whole dataset to reduce the computational power required
clean = clean.sample(round(clean.shape[0]*.15), replace = False)
clean = clean.reset_index(drop = True)
print(clean.shape)
clean.head()
```

```
(14974, 12)
```

Out[13]:

	amenities	bathrooms	bedrooms	currency	fee	has_photo	pets_allowe
0	Parking	2	2	USD	No	Yes	Na
1	Gym,Internet Access,Parking,Pool	1	1	USD	No	Yes	Na
2	Gym,Pool	1.5	1	USD	No	Yes	Na
3	NaN	1	1	USD	No	No	Ca
4	AC,Cable or Satellite,Clubhouse,Dishwasher,Fir...	1	1	USD	No	Yes	Cats,Dog

In [14]:

```
clean.dtypes
```

Out[14]:

```
amenities        object
bathrooms        object
bedrooms         object
currency         object
fee              object
has_photo        object
pets_allowed     object
price           float64
price_type       object
square_feet      object
latitude        float64
longitude       float64
dtype: object
```

In [15]:

```
#find the nonfloats in the categories that are expected to be numerical
check = ['bathrooms', 'bedrooms', 'square_feet']
bad = []
index = 0
for i in range(clean[check].shape[0]):
    for c in range(clean[check].shape[1]):
        try:
            float(clean[check].iloc[i,c])
        except:
            bad.append(index)
```

```
    index = index + 1
bad
```

Out[15]:

```
[12860, 12860]
```

In [16]:

```
clean = clean.drop(bad)
clean = clean.reset_index(drop = True)
```

In [17]:

```
clean.head()
```

Out[17]:

	amenities	bathrooms	bedrooms	currency	fee	has_photo	pets_allowe
0	Parking	2	2	USD	No	Yes	Na
1	Gym,Internet Access,Parking,Pool	1	1	USD	No	Yes	Na
2	Gym,Pool	1.5	1	USD	No	Yes	Na
3	NaN	1	1	USD	No	No	Ca
4	AC,Cable or Satellite,Clubhouse,Dishwasher,Fir...	1	1	USD	No	Yes	Cats,Dog

In [18]:

```
#fix the data types
clean['square_feet'] = clean['square_feet'].astype(float)
clean['bedrooms'] = clean['bedrooms'].astype(float)
clean['bathrooms'] = clean['bathrooms'].astype(float)
```

In [19]:

```
#Fix the nan values in the categorical varaibles
clean.loc[:,'amenities'] = clean.loc[:,'amenities'].replace(np.nan, 'No Amenitie
clean.loc[:,'pets_allowed'] = clean.loc[:,'pets_allowed'].replace(np.nan, 'No Pe
```

In [20]:

```
clean.isna().apply(lambda x: sum(x))
```

Out[20]:

```
amenities        0
bathrooms        7
bedrooms        17
currency         0
fee              0
has_photo        0
pets_allowed     0
price            0
price_type       0
square_feet      0
latitude         4
longitude        4
dtype: int64
```

In [21]:

```
#make the lists true python lists
clean.loc[:,'amenities'] = clean.loc[:,'amenities'].apply(lambda x: x.split(','))
clean.loc[:,'pets_allowed'] = clean.loc[:,'pets_allowed'].apply(lambda x: x.spli
```

In [22]:

```
clean = clean.dropna()
clean = clean.reset_index(drop = True)
```

In [23]:

```
clean.head()
```

Out[23]:

	amenities	bathrooms	bedrooms	currency	fee	has_photo	pets_allowed	price	price_typ
0	[Parking]	2.0	2.0	USD	No	Yes	[No Pets]	1468.0	Month
1	[Gym, Internet Access, Parking, Pool]	1.0	1.0	USD	No	Yes	[No Pets]	1456.0	Month
2	[Gym, Pool]	1.5	1.0	USD	No	Yes	[No Pets]	4050.0	Month
3	[No Amenities]	1.0	1.0	USD	No	No	[Cats]	1715.0	Month
4	[AC, Cable or Satellite, Clubhouse, Dishwasher...	1.0	1.0	USD	No	Yes	[Cats, Dogs]	810.0	Month

In [24]:

```
#preprocess to make dummy variables for amenities and animals
mlb = MultiLabelBinarizer()
amens = pd.DataFrame(mlb.fit_transform(clean['amenities']),columns=mlb.classes_,
pets = pd.DataFrame(mlb.fit_transform(clean['pets_allowed']),columns=mlb.classes
```

In [25]:

```
dum2 = pd.concat([clean,amens,pets],axis = 1)
dum2 = dum2.drop(['amenities', 'pets_allowed'], axis = 1)
```

In [26]:

```
#standardize the data after assigning all the dummy variables
dum2 = pd.get_dummies(dum2)
scaler = StandardScaler()
dum2 = pd.DataFrame(scaler.fit_transform(dum2))
dum2.head()
```

Out[26]:

	0	1	2	3	4	5	6	7	
0	1.026959	0.358141	-0.073817	0.406299	-2.099860	0.581439	-0.436740	-0.062418	-0.20
1	-0.823970	-0.973637	-0.088618	-0.398183	1.063966	0.223069	-0.436740	-0.062418	-0.20
2	0.101495	-0.973637	3.110976	0.273630	-0.656227	-1.694704	-0.436740	-0.062418	-0.20
3	-0.823970	-0.973637	0.230848	-0.121554	0.849783	1.101143	-0.436740	-0.062418	-0.20
4	-0.823970	-0.973637	-0.885433	-0.906277	-0.378316	0.688473	2.289692	-0.062418	-0.20

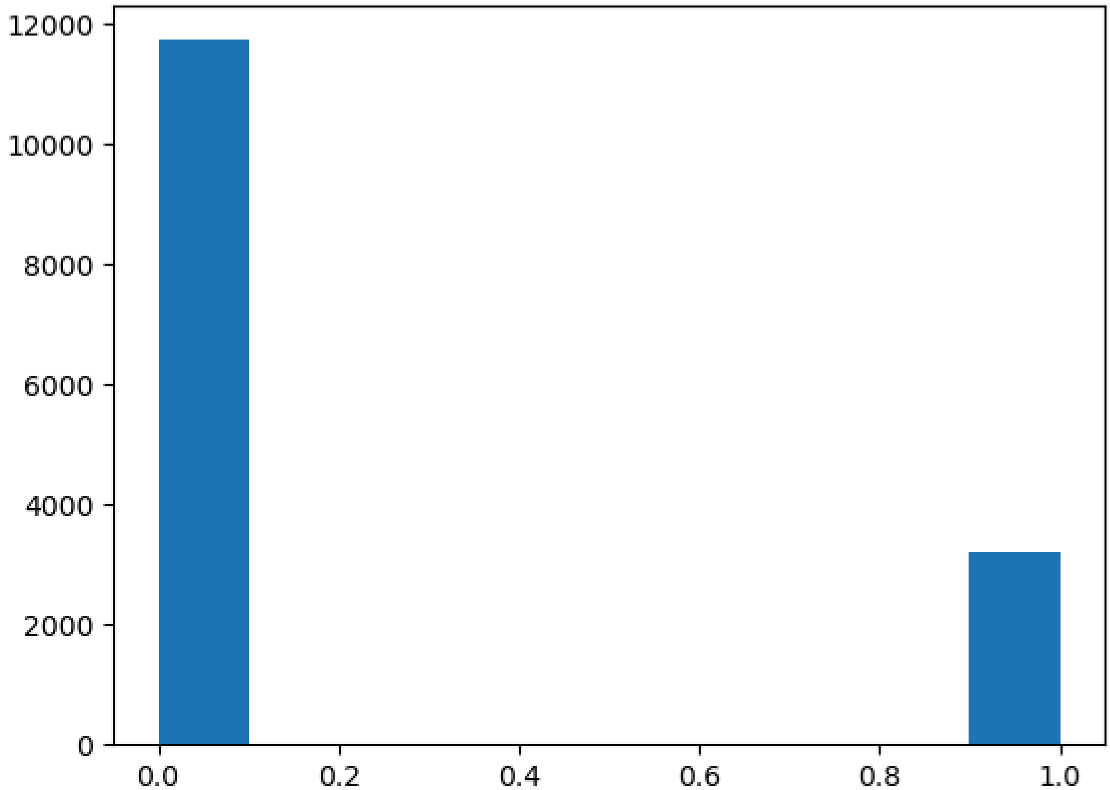
5 rows × 45 columns

In [27]:

```
km = MyKMeans(n_clusters = 2, seed =300)
km.fit(dum2)
pred2 = km.make_labels(dum2)
```

In [28]:

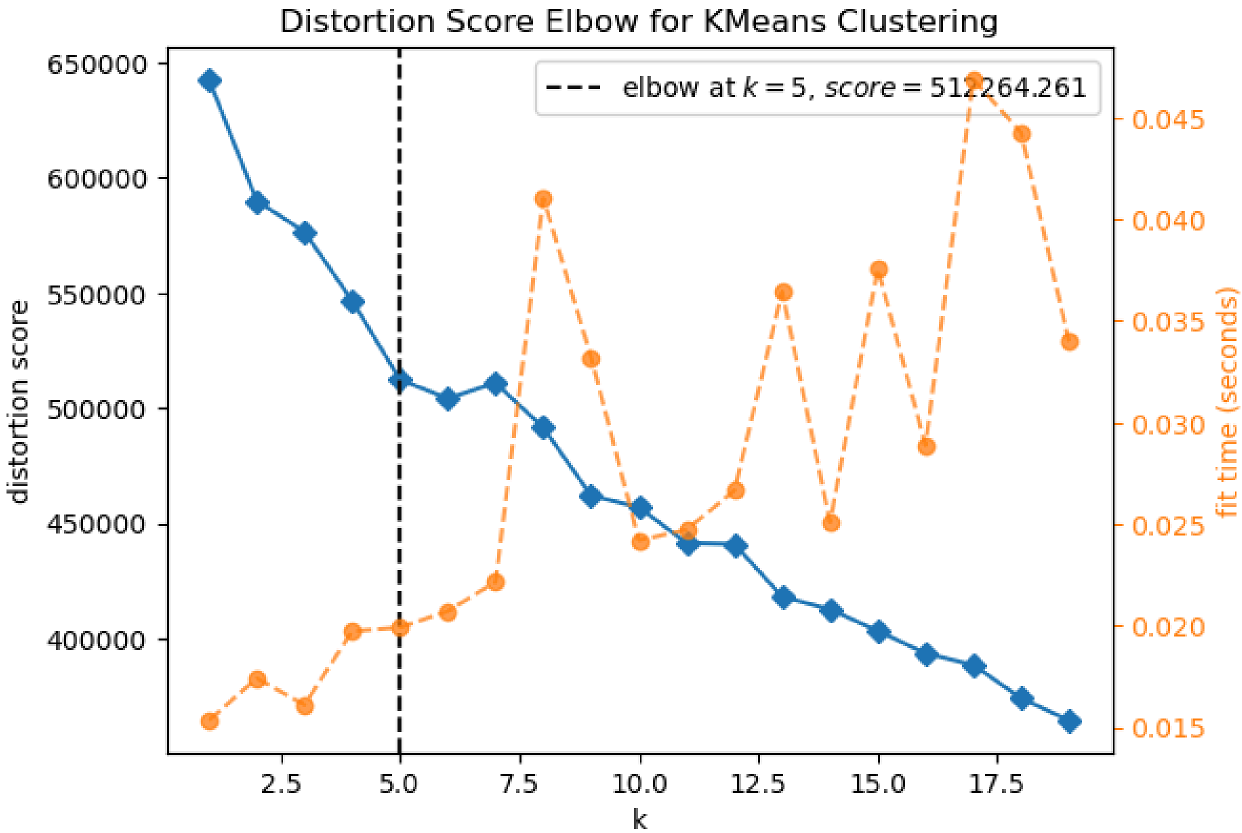
```
plt.hist(pred2)
plt.show()
```



In [29]:

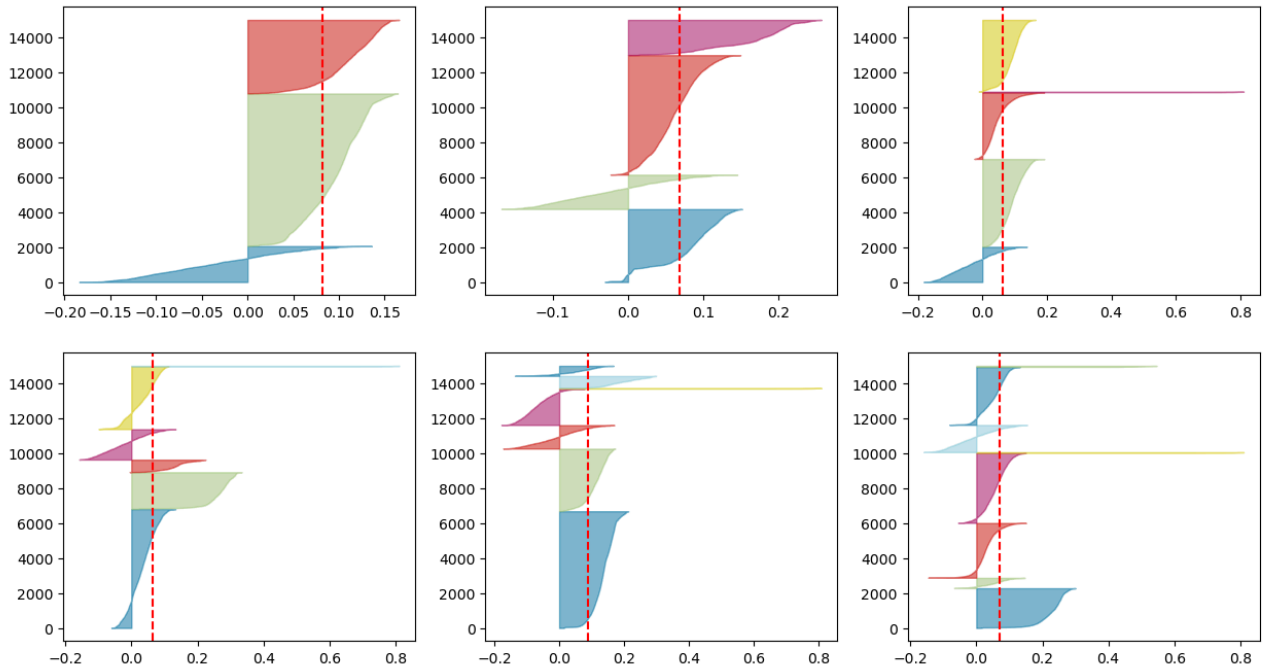
```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,20))

visualizer.fit(dum2)
visualizer.show()
plt.show()
```



In [30]:

```
fig, ax = plt.subplots(2, 3, figsize=(15,8))
for i in [3, 4, 5, 6, 7, 8]:
    '''
    Create KMeans instance for different number of clusters
    '''
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_
    q, mod = divmod(i, 3)
    '''
    Create SilhouetteVisualizer instance with KMeans instance
    Fit the visualizer
    '''
    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod])
    visualizer.fit(dum2)
```



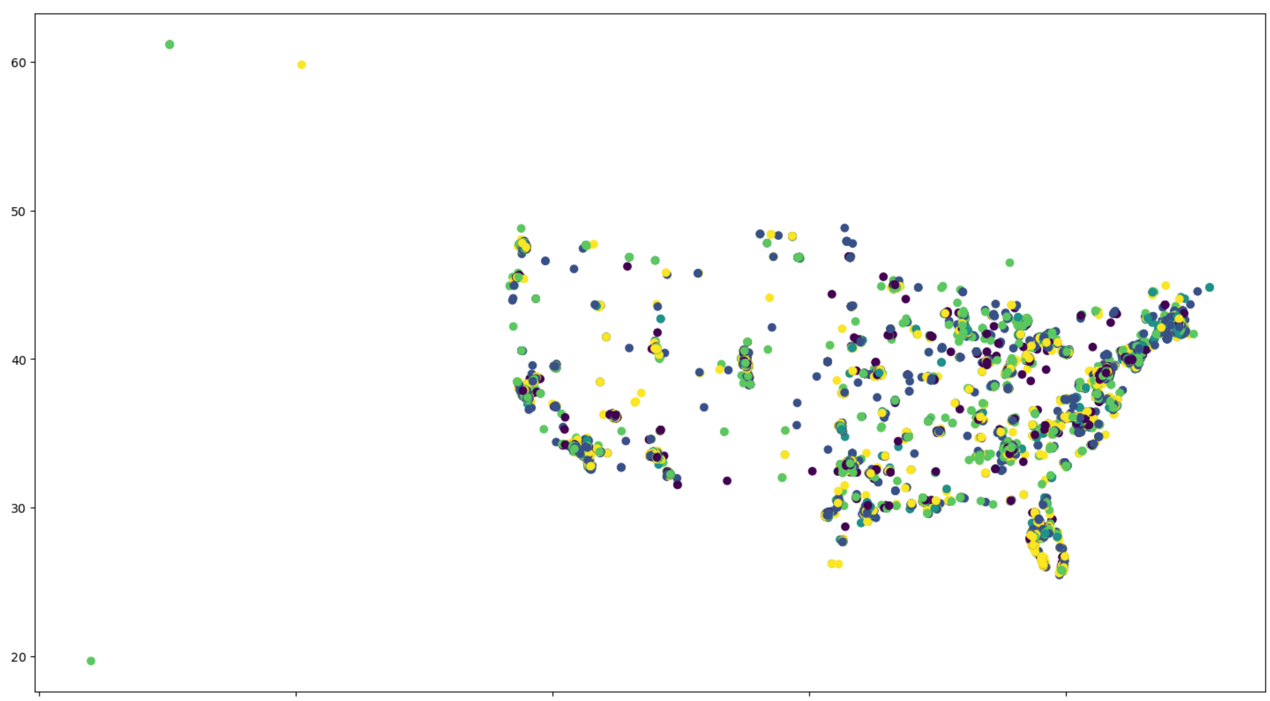
By looking at both the silhouette plot and the elbow plot we can see that the most likely optimal number of clusters is about 5. There is a rough elblow at that point, and the silhouette plot shows almost all 5 clusters are past thje average line, and are all roughly the same size.

In [31]:

```
kmeans = KMeans(n_clusters=5)
pred2 = kmeans.fit(dum2)
```

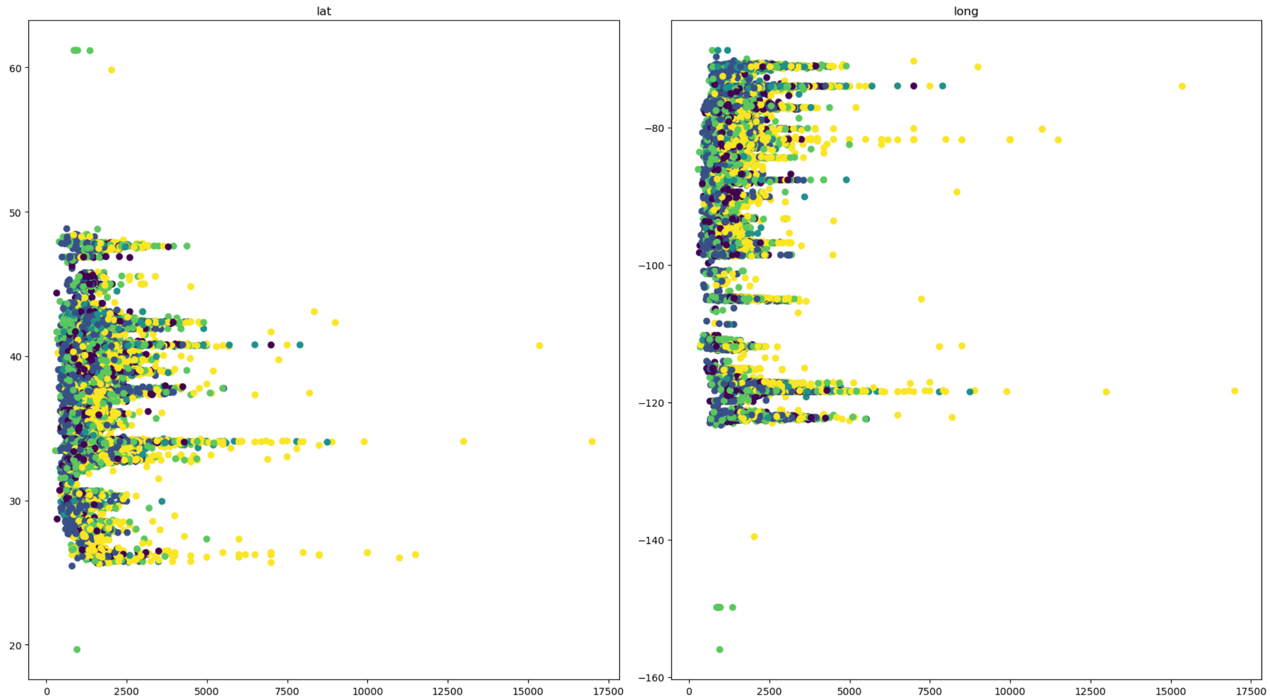
In [32]:

```
plt.figure(figsize = (18,10))
sty.use('default')
plt.scatter(clean['longitude'], clean['latitude'], c = pred2.labels_)
plt.show()
```



In [33]:

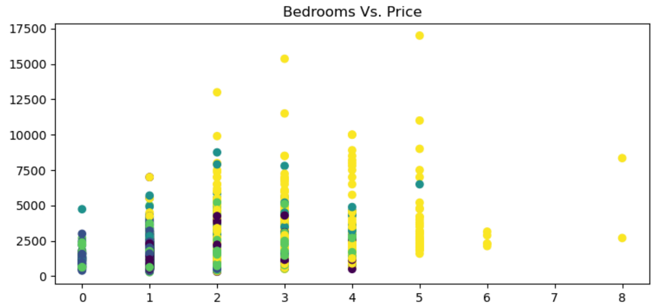
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(18, 10))
axes[0].scatter(clean['price'], clean['latitude'], c = pred2.labels_)
axes[0].set_title('lat')
axes[1].scatter(clean['price'], clean['longitude'], c = pred2.labels_)
axes[1].set_title('long')
fig.tight_layout()
plt.show()
```

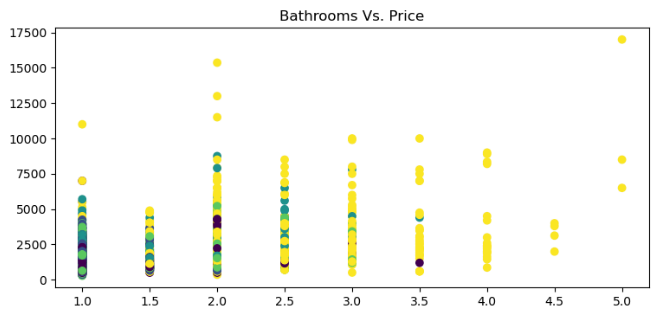


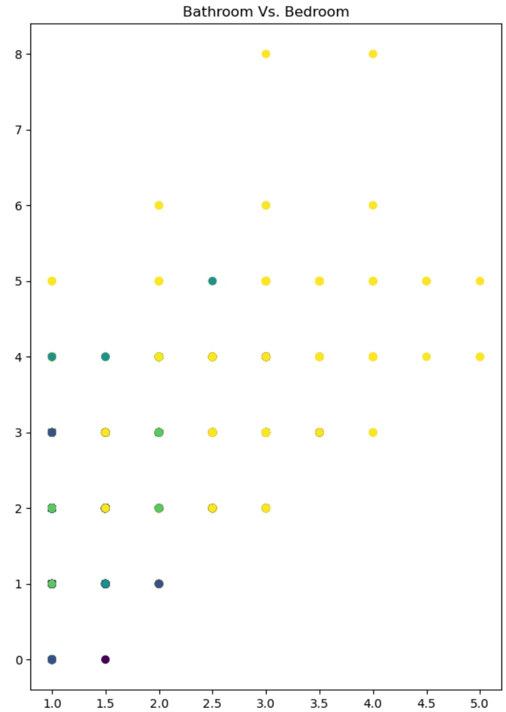
In [34]:

```
plt.figure(figsize = (18,10))
plot1 = plt.subplot2grid((10, 10), (0,0), rowspan = 4,colspan=5)
plot2 = plt.subplot2grid((10, 10), (0, 6), rowspan=10, colspan=10)
plot3 = plt.subplot2grid((10, 10), (6, 0), rowspan = 5,colspan=5)

plot2.scatter(clean['bathrooms'], clean['bedrooms'], c = pred2.labels_)
plot2.set_title('Bathroom Vs. Bedroom')
plot1.scatter(clean['bedrooms'], clean['price'], c = pred2.labels_)
plot1.set_title('Bedrooms Vs. Price')
plot3.scatter(clean['bathrooms'], clean['price'], c = pred2.labels_)
plot3.set_title('Bathrooms Vs. Price')
fig.tight_layout()
plt.show()
```







When looking at these results there is not a ton of separation between the groups, but we can see that it does seem to roughly separate the cost. In the bedroom and bathroom plots, we can see the yellow cluster starts to take over as the price increases. We can also see that there is a correlation between price and the number of bedrooms and bathrooms. Then when looking at the geographic data we can see that the prices tend to be higher around the coastal areas. We can see this as there is much more of the yellow is heavily populated areas and coastal areas. This answers both of our research questions. We were able to find a handful of distinct groups, and then find a pattern within the data.

Reference


```
Apartment for Rent Classified. (2019). UCI Machine Learning
Repository.

https://archive.ics.uci.edu/dataset/555/apartment+for+rent+classified

Online Shoppers Purchasing Intention Dataset. (2018). UCI Machine
Learning Repository.

https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+intent
```