MACS 33002: PSet 4

Ezra Max

March 2, 2020

[1]:
```
#Import packages
import pandas as pd
import math
import pyclustertend
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from sklearn import cluster
from sklearn import mixture
from sklearn import datasets
from sklearn import metrics
from sklearn.preprocessing import scale
from pyclustertend import hopkins
from pyclustertend import vat
from pyclustertend import assess_tendency_by_metrics
from scipy.cluster.hierarchy import dendrogram

#Set seed
random.seed(30)
```

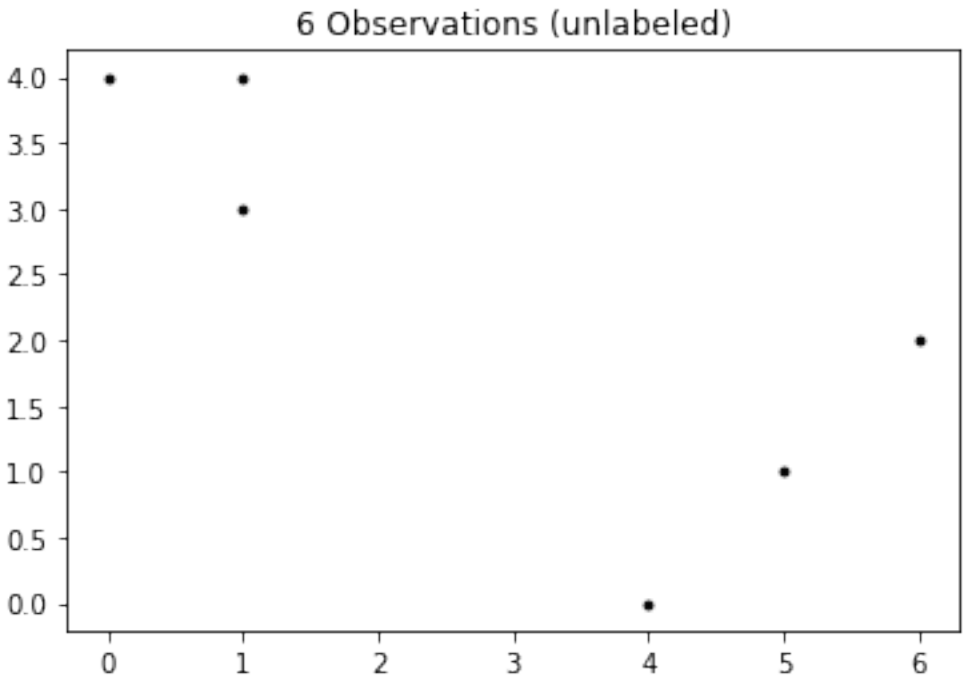
[2]:
```
#Set up helper functions
def l2dist(t1,t2):
    '''
    Function for Euclidean distance between points t1,t2 in R2
    '''
    return math.sqrt((t1[0]-t2[0])**2+(t1[1]-t2[1])**2)

class Observation:
    '''
    Class for an observation with cluster assignment.
    '''
    def __init__(self,coordinates):
        '''
        Constructor for class
        '''
        self.coordinates=coordinates
```

```
        #Initialize cluster assignment as 0
        self.label=0
    def update_label(self,centroid_1,centroid_2):
        '''
        Update cluster assignment (cluster 1 or cluster 2)
        '''
        if l2dist(self.coordinates, centroid_2)<l2dist(self.coordinates,␣
↪centroid_1):
            self.label=2
        else:
            self.label=1

def get_centroid(cluster_1,cluster_2):
    '''
    Function to get centroids for cluster 1 and cluster 2. If one of the␣
↪clutsers has no points, we choose
    a cluster far from other points (cheap solution to this case.)
    '''
    if len(cluster_1)!=0:
        c1=(sum(observation.coordinates[0] for observation in cluster_1)/
↪len(cluster_1),
            sum(observation.coordinates[1] for observation in cluster_1)/
↪len(cluster_1))
    else:
        c1=(15,15)
    if len(cluster_2)!=0:
        c2=(sum(observation.coordinates[0] for observation in cluster_2)/
↪len(cluster_2),
            sum(observation.coordinates[1] for observation in cluster_2)/
↪len(cluster_2))
    else:
        c2=(15,15)
    return c1, c2

def label(observations,centroid_1,centroid_2):
    '''
    Function to assign data to nearest centroid
    '''
    new_list=observations
    for observation in new_list:
        observation.update_label(centroid_1, centroid_2)
    return(new_list)
def scatter_plot(observations,color,size):
    '''
    Map observations
    '''
```

```
    plt.scatter([observation.coordinates[0] for observation in observations],
                [observation.coordinates[1] for observation in observations],
                c=color, s=size)
```

[3]:
```
#Store simulated feature data
coordinate_data=list(zip([1, 1, 0, 5, 6, 4],[4, 3, 4, 1, 2, 0]))
observations=[Observation(coordinates) for coordinates in coordinate_data]

#Plot observations
scatter_plot(observations,"black",8)
plt.title('6 Observations (unlabeled)')
```

[3]:
```
Text(0.5, 1.0, '6 Observations (unlabeled)')
```

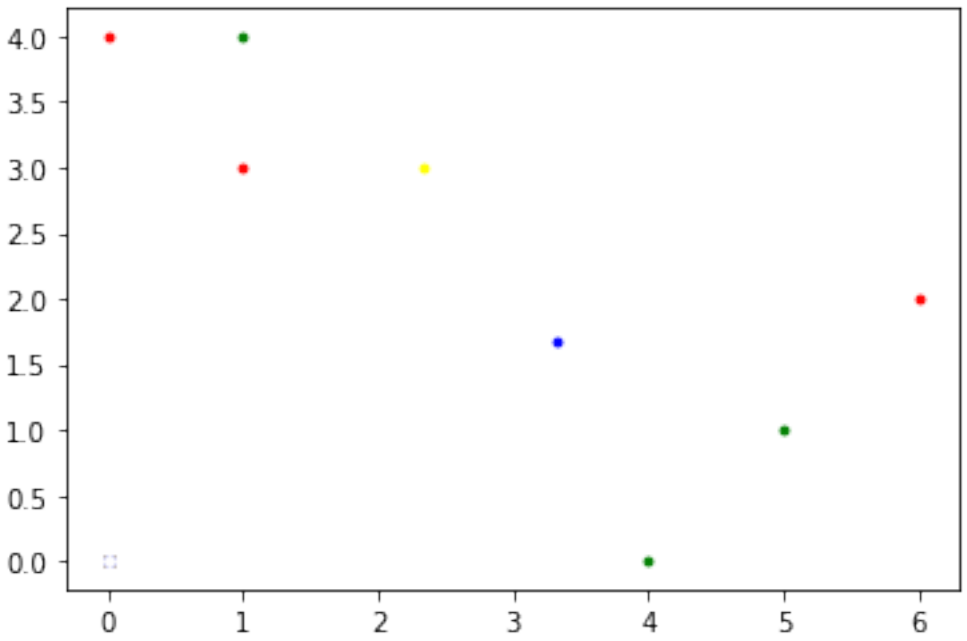


[4]:
```
#Randomly assign observations to clusters
for observation in observations:
    observation.label=random.randint(1,2)

cluster_1=[observation for observation in observations if observation.label==1]
cluster_2=[observation for observation in observations if observation.label==2]

#Report cluster labels
pd.DataFrame({'Observation': [observation.coordinates for observation in␣
 ↪observations],
              'Cluster label': [observation.label for observation in␣
 ↪observations]})
```

[4]:
```
  Observation  Cluster label
0      (1, 4)              2
1      (1, 3)              1
2      (0, 4)              1
3      (5, 1)              2
4      (6, 2)              1
5      (4, 0)              2
```

[5]:
```
#Compute centroids for clusters
c1,c2=get_centroid(cluster_1,cluster_2)

#Plot
scatter_plot(cluster_1, "red", 8)
scatter_plot(cluster_2, "green",8)
plt.scatter(c1[0],c1[1], c="yellow", s=8)
plt.scatter(c2[0],c2[1], c="blue", s=8)
red_dot, = plt.plot(0, "ro", markersize=4)
green_dot, = plt.plot(0, "go", markersize=4)
yellow_dot, = plt.plot(0, "yo", markersize=4)
blue_dot, = plt.plot(0, "bo", markersize=4)
white_dot, =plt.plot(0, "wo", markersize=4)
custom_lines = [red_dot,
                green_dot,
                purple_dot,
                blue_dot
               ]


plt.legend(custom_lines, ['Cluster 1','Cluster 2', 'Centroid 1', 'Centroid 2'])
plt.title('Random clusters and computed centroids')
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-5-450e6fa1643c> in <module>
 14 custom_lines = [red_dot,
 15                 green_dot,
---> 16                 purple_dot,
 17                 blue_dot
 18                ]


NameError: name 'purple_dot' is not defined
```



[6]:
```
#Relabel by assigning to nearest centroid
old_labels=[observation.label for observation in observations]
label(observations, c1, c2)

#Check whether label has changed
pd.DataFrame({'Point':[observation.coordinates for observation in observations],
                    'Cluster before reassignment':old_labels,
                    'Cluster after reassignment':[observation.label for␣
 ↪observation in observations]})
```

[6]:

	Point	Cluster before reassignment	Cluster after reassignment
0	(1, 4)	2	1
1	(1, 3)	1	1
2	(0, 4)	1	1
3	(5, 1)	2	2
4	(6, 2)	1	2
5	(4, 0)	2	2

[7]:
```
def k_means(observations):
    '''
    Function to recursively repeat steps above until cluster labels are stable.
    '''
    old_observations=observations
    c1,c2=get_centroid([observation for observation in observations if␣
 ↪observation.label==1],
                       [observation for observation in observations if␣
 ↪observation.label==2])
```

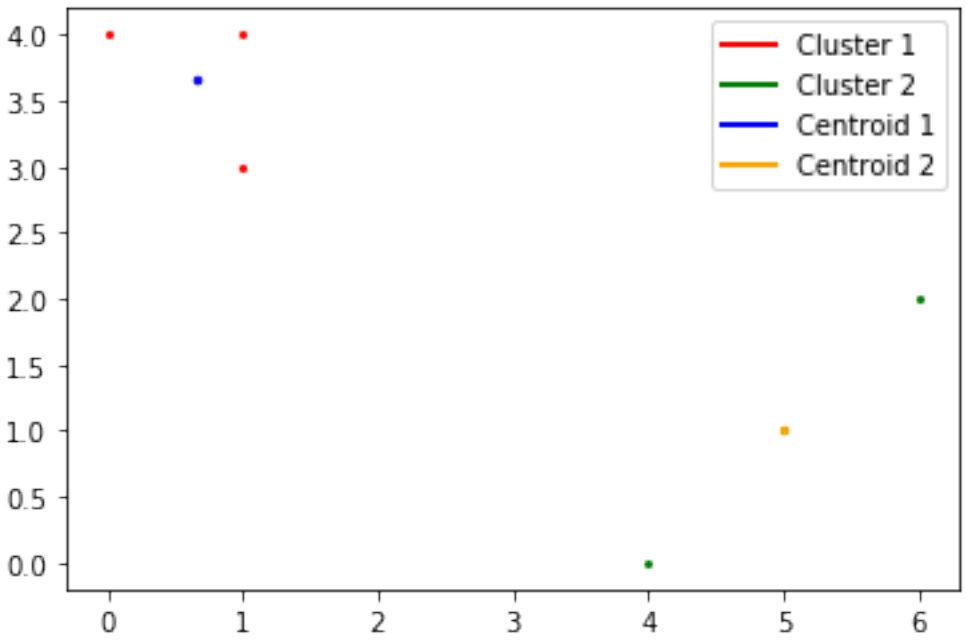
```
    label(observations, c1,c2)
    if old_observations==observations:
        return(c1,c2, observations)
    else:
        return(k_means(new_observations))

c1,c2,observations=k_means(observations)

cluster_1=[observation for observation in observations if observation.label==1]
cluster_2=[observation for observation in observations if observation.label==2]

#Report final cluster data for each observation
pd.DataFrame({'Point':[observation.coordinates for observation in observations],
              'Original Cluster': old_labels,
                    'Final Cluster':[observation.label for observation in␣
 ↪observations]})
```

[7]:

	Point	Original Cluster	Final Cluster
0	(1, 4)	2	1
1	(1, 3)	1	1
2	(0, 4)	1	1
3	(5, 1)	2	2
4	(6, 2)	1	2
5	(4, 0)	2	2

[8]:
```
#Plot final observations
scatter_plot(cluster_1, "red", 4)
scatter_plot(cluster_2, "green",4)
plt.scatter(c1[0],c1[1], c="blue", s=6)
plt.scatter(c2[0],c2[1], c="orange", s=6)

custom_lines = [Line2D([0],[0], color='red',lw=2),
                Line2D([0],[0], color='green',lw=2),
                Line2D([0],[0], color='blue',lw=2),
               Line2D([0],[0], color='orange',lw=2)]


plt.legend(custom_lines, ['Cluster 1','Cluster 2', 'Centroid 1', 'Centroid 2'])
```

[8]:
```
<matplotlib.legend.Legend at 0x1a17c712e8>
```

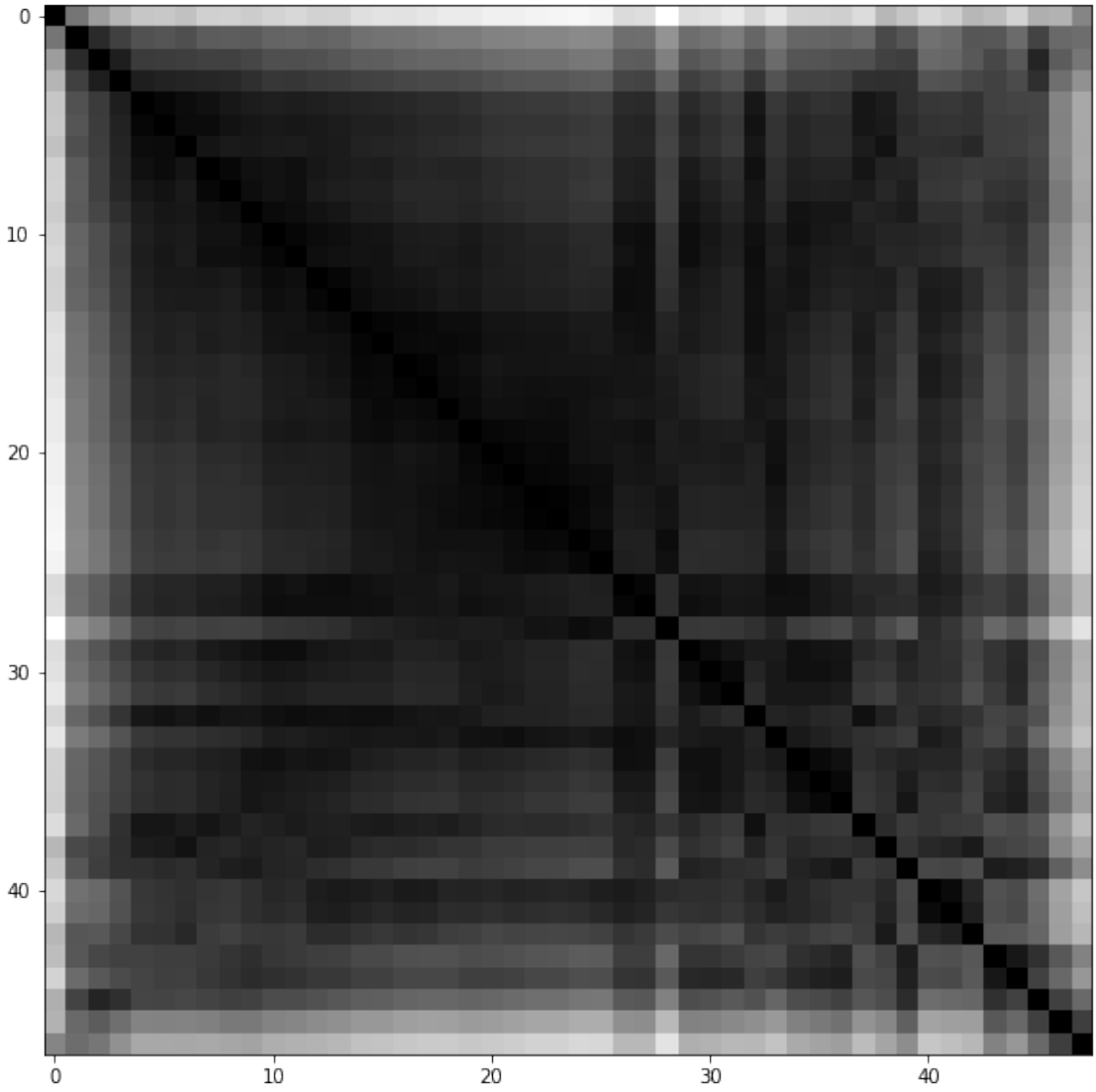


[9]:
```
#I got the data as a CSV from the source. It still is v1.0 as specified in the
#assignment.
ref_df=pd.read_csv("/Users/emax/Downloads/legprof-components.v1.0.tab", sep="\t")

#Clean the data
df=ref_df[['stateabv','t_slength','slength','salary_real','expend','year']][ref_df.
 ↪year>=2009][ref_df.year<=2010]
df.dropna(inplace=True)
df[['t_slength','slength','salary_real','expend']]=(df[['t_slength','slength','salary_real','ex
 ↪mean())/df[['t_slength','slength','salary_real','expend']].std()
```

[10]:
```
#Get Hopkins statistic
hopkins(df[['t_slength','slength','salary_real','expend']],7)
```

[10]:
```
0.2604372446159299
```

[11]:
```
#Display VAT
vat(df[['t_slength','slength','salary_real','expend']])
```



[12]:
```
#Compute optimal number of clusters from kmeans algorithm
assess_tendency_by_metrics(df[['t_slength','slength','salary_real','expend']])
```

[12]:
```
7.333333333333333
```

We get somewhat underwhelming results when it comes to clusterability. The Hopkins statistic, H, is a sparse sampling technique to evaluate how clustered our observations are. (Info above and below adapted from Wikipedia entry.)

To compute H, we randomly sample $m = 7$ data points from our n observations. We then generate m uniform, random data points over this range and compute u_i, the distance of each of our m generated observations from its nearest neighbour in our m sampled observations, and w_i, the distance of each of our m sampled observations from its nearest neighbor in the sampled

observations. Therefore, for p features, the ratio

$$H = \frac{\sum_{i=1}^{m} u_i^p}{\sum_{i=1}^{m} u_i^p + w_i^p}$$

should evaluate how close our data is to being clustered: for a near-uniform spread of observations, we should have a low value near 0, since w_i will be large on average; random values generated by a Poisson point process should result a Hopkins statistic around .5, since then roughly $u_i \sim w_i$; and very well-clustered data will have a Hopkins statistic near 1, since $w_i << u_i$ on average.

In our case, with $H \sim .26$, we find that our data appear to be fairly spread out, and are near uniform in their distribution. This is a sign that we will not get great results when trying to cluster these data. In particular, the statistic tells us that the observations we have are not clustered into groups as we might hope, and that the distribution of the observations is similar to the results of generating our data from a memoryless Poisson point proess. This will make it hard to tease out any strong, underlying probability densities in our data.

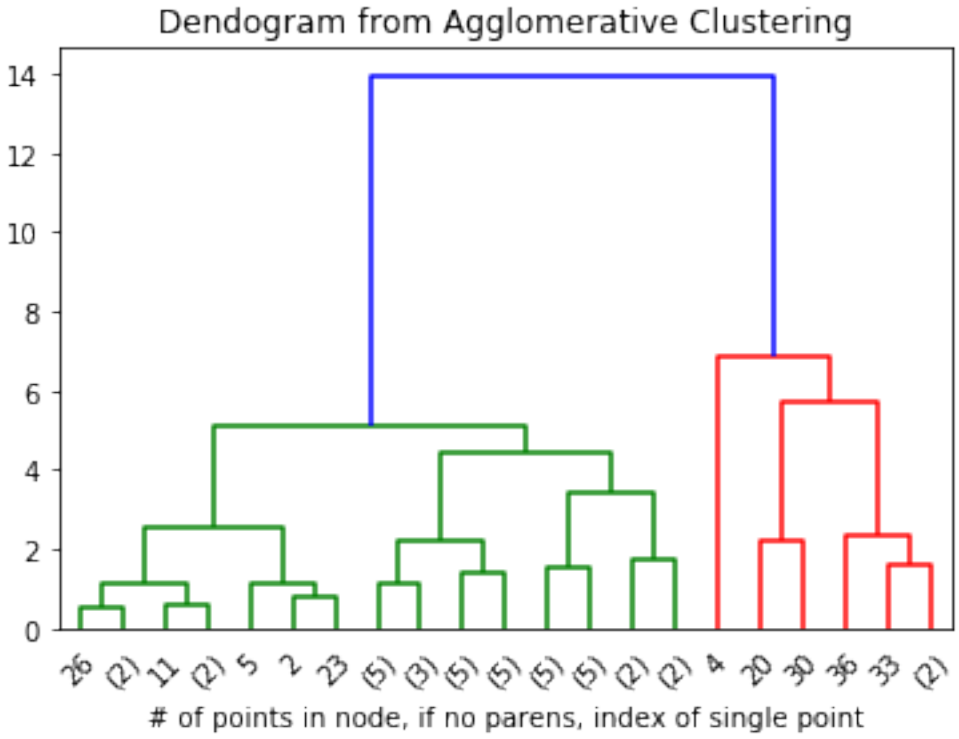
Another bad sign in this direction is the VAT above. Highly clustered data should result in distinct dark and light regions. In the graphic above, while there is enough variation that we see our data is not uniformly distributed, it seems heuristically the case that our data, as suggested by the Hopkins statistic above, has (a) no immediately clear structure underlying it or (b) is too noisy to identify that structure. Indeed after perusing the VATs searchable online, the VAT above seems to suggest our data is on the low end of clusterability.

However, we should again note that neither our Hopkins statistic or our VAT is truly dire (i.e., our data is somewhat clusterable,) and that there is some hope that we can perform a useful, or at least not totally useless, cluster analysis on our osbervations.

[13]:
```
#Set up helper function for dendogram
def get_dendogram(model):
    '''
    A helper function to get the dendogram from a clustering mode. Adapted from:
    https://scikit-learn.org/stable/auto_examples/cluster/
    plot_agglomerative_dendrogram.
 ↪html#sphx-glr-auto-examples-cluster-plot-agglomerative-dendrogram-py
    '''
    #Counts for each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count = current_count +1
            else:
                current_count = current_count+counts[child_idx - n_samples]
        counts[i] = current_count

    #Get linkage matrix
    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)
```

```
    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, p=4, truncate_mode='level')
```

[14]:
```
#Agglomerative hierarchical cluster
agg_cluster=cluster.AgglomerativeClustering(distance_threshold=0,␣
 ↪n_clusters=None)
agg_cluster.fit(df[['t_slength','slength','salary_real','expend']])
type(agg_cluster)
get_dendogram(agg_cluster)
plt.xlabel("# of points in node, if no parens, index of single point")
plt.title("Dendogram from Agglomerative Clustering")

#Due to a technical difficulty, had to rerun to use fit predict with same num␣
 ↪clusters specified as given by
#dendogram
agg_cluster=cluster.AgglomerativeClustering(distance_threshold=None,␣
 ↪n_clusters=22)
df['agg_label']=agg_cluster.
 ↪fit_predict(df[['t_slength','slength','salary_real','expend']])
```



Some general observations about the dendogram above from the agglomerative hierarchical cluster technique: because we did not specify clusters, and we have so few observations, the agglomerative hierarchical clustering algorithm produced many smaller clusters in the (2-5) range

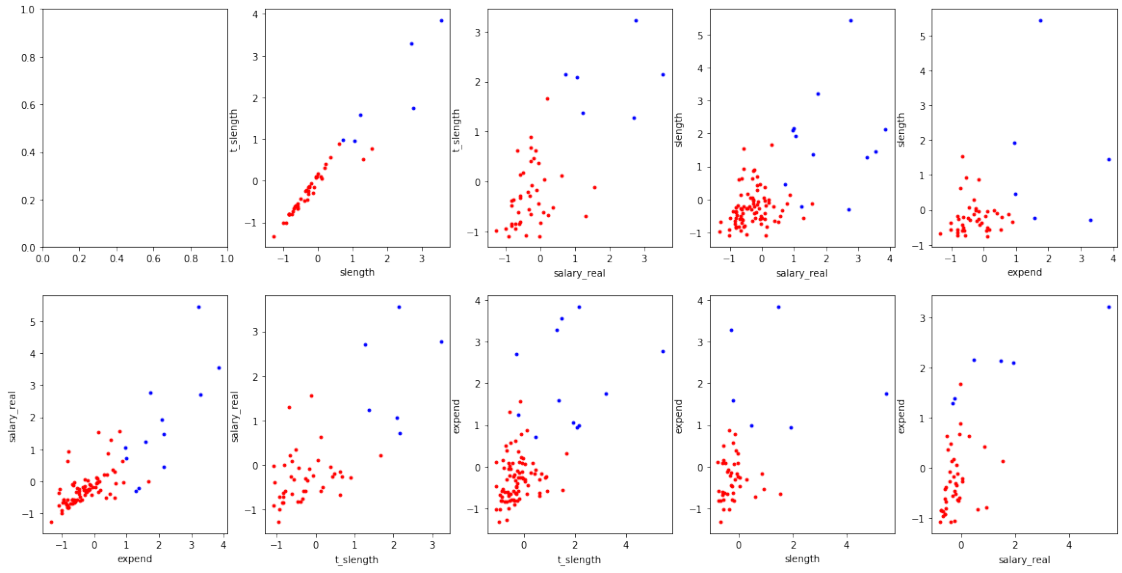
that are close together, and several one-point clusters whose indexes to outliers. This would suggest we have quite a few outliers, which we will see below, and which–given that we only have ∼ 50 observations–may explain our data's poor clusterability overall.

[15]:
```
k_mean_cluster=cluster.KMeans(2)
k_mean_cluster.fit(df[['t_slength','slength','salary_real','expend']])
df['k_means_label']=k_mean_cluster.
 ↪predict(df[['t_slength','slength','salary_real','expend']])
```

[16]:
```
gauss_em=mixture.GaussianMixture(2)
gauss_em.fit(df[['t_slength','slength','salary_real','expend']])
df['gauss_em_label']=gauss_em.
 ↪predict(df[['t_slength','slength','salary_real','expend']])
```

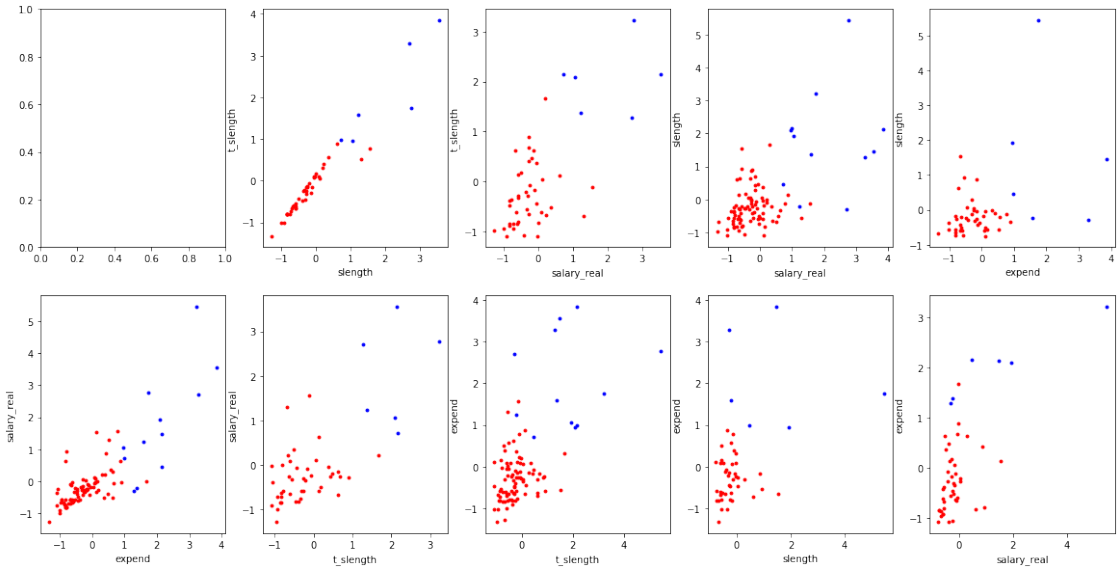
[17]:
```
#1-Feature graphs for k_means cluster
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
for i in range(0,4):
    for j in range(0,4):
        if i!=j:
            plt.subplot(2,5,(int(i>j))*4+(i+j+1))
            name0, name1=['t_slength','slength','salary_real','expend'][i],␣
 ↪['t_slength','slength','salary_real','expend'][j]
            plt.scatter(df[[name0]][df.k_means_label==0],df[[name1]][df.
 ↪k_means_label==0], c="red", s=8)
            plt.scatter(df[[name0]][df.k_means_label==1],df[[name1]][df.
 ↪k_means_label==1], c="blue", s=8)
            plt.ylabel(name0)
            plt.xlabel(name1)

print("Two-Feature Clustering for K-Means Model")
```

```
Two-Feature Clustering for K-Means Model
```



[28]:
```
#2-Feature graphs for gauss_em cluster
plt.clf()
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
for i in range(0,4):
    for j in range(0,4):
        if i!=j:
            plt.subplot(2,5,(int(i>j))*4+(i+j+1))
            name0, name1=['t_slength','slength','salary_real','expend'][i],␣
↪['t_slength','slength','salary_real','expend'][j]
            plt.scatter(df[[name0]][df.gauss_em_label==0],df[[name1]][df.
↪gauss_em_label==0], c="red", s=8)
            plt.scatter(df[[name0]][df.gauss_em_label==1],df[[name1]][df.
↪gauss_em_label==1], c="blue", s=8)
            plt.ylabel(name0)
            plt.xlabel(name1)
print("Two-Feature Clustering for Gaussian Mixture Models")
```

```
Two-Feature Clustering for Gaussian Mixture Models

<Figure size 432x288 with 0 Axes>
```



First, let us note the obvious: the gaussian mixture and models above agree seemingly on every observation, as is easily verrified below:

[19]:
```
len(df[['gauss_em_label','k_means_label']][df.gauss_em_label!=df.k_means_label])
```

[19]: 0

This fact tells us that there is only one obvious way to cluster the given data into two groups: one cluster will contain the majority of all observations (in the charts above, this is the blue cluster) and the other will contain the outliers. This in turn suggests that $k = 2$ is not a very good choice for how to label our data: the result is a tight, well-formed blue cluster and then a spread red cluster.

By contrast, the dendogram with agglomerative clustering techniques above produces many more small clusters, many of them in the size approximately 2-5 of close clusters, close together (i.e., a further division of the blue group above) and then several outliers in one-point clusters (further divisions in the red group above.) This would suggest that forcing $k = 2$ is a poor choice for clustering in this case: because of the the noisy data (i.e., the outlying points,) forcing $k = 2$ will simply make one group for the noisy data and then another, undistinguished and nonuseful group of all the non-noisy data, which that solely due to the magnitude of these outliers our useful data (the blue points) is amalgamated into one cluster without useful distinction.

To further confirm that $k = 2$ is a poor choice for our dataset, we can easily compute the k value which produces the optimal Calinski Harabaz score using k-means clustering using the pyclustertend package.

[20]:
```
print("Optimal # clusters: "+str(pyclustertend.metric.
 ↪assess_tendency_calinski_harabaz(df[['t_slength','slength','salary_real','expend']],␣
 ↪max_nb_cluster=10)[0]))
print("Optimal # clusters: "+str(pyclustertend.metric.
 ↪assess_tendency_calinski_harabaz(df[['t_slength','slength','salary_real','expend']],␣
 ↪max_nb_cluster=40)[0]))
```

```
Optimal # clusters: 10

Optimal # clusters: 40
```

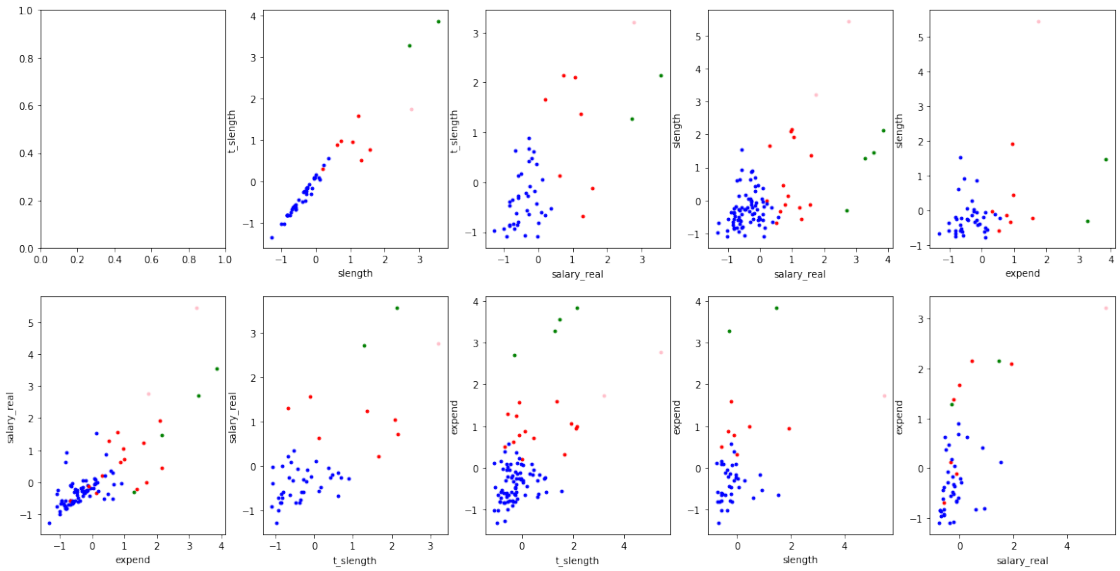
So the optimal number of clusters k for $0 \leq k \leq 10$ is 10, and for $0 \leq k \leq 40$ it is 40. Since we have only 48 observations in our dataset, we can take this to mean that higher k produce more accurate models, although obviously this comes at the cost of making a useful cluster model (i.e. $k = 48$ will be most accurate for a k-means clustering approach, and this makes sense since we do not have that much data that seems to be easily clusterable, per the Hopkins statistic and VAT above. However, $k = 48$ is an absolutely useless approach! It gives us no new information.) The bottom line is that $k = 2$ is probably too low a parameter for this model, and we hope and suspect that, at least in the case of k-means and most likely in the case of the Gaussian model, we might get better results trying, say, $k = 4$. Let's see.

[21]:
```
k_mean_cluster=cluster.KMeans(4)
k_mean_cluster.fit(df[['t_slength','slength','salary_real','expend']])
df['k_means_label_4']=k_mean_cluster.
 ↪predict(df[['t_slength','slength','salary_real','expend']])

gauss_em=mixture.GaussianMixture(4)
gauss_em.fit(df[['t_slength','slength','salary_real','expend']])
df['gauss_em_label_4']=gauss_em.
 ↪predict(df[['t_slength','slength','salary_real','expend']])
```

[22]:
```
#1-Feature graphs for k_means cluster
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
```

```
for i in range(0,4):
    for j in range(0,4):
        if i!=j:
            plt.subplot(2,5,(int(i>j))*4+(i+j+1))
            name0, name1=['t_slength','slength','salary_real','expend'][i],␣
 ↪['t_slength','slength','salary_real','expend'][j]
            plt.scatter(df[[name0]][df.k_means_label_4==0],df[[name1]][df.
 ↪k_means_label_4==0], c="red", s=8)
            plt.scatter(df[[name0]][df.k_means_label_4==1],df[[name1]][df.
 ↪k_means_label_4==1], c="blue", s=8)
            plt.scatter(df[[name0]][df.k_means_label_4==2],df[[name1]][df.
 ↪k_means_label_4==2], c="pink", s=8)
            plt.scatter(df[[name0]][df.k_means_label_4==3],df[[name1]][df.
 ↪k_means_label_4==3], c="green", s=8)
            plt.ylabel(name0)
            plt.xlabel(name1)

print("Two-Feature Clustering for K-Means Model")
```

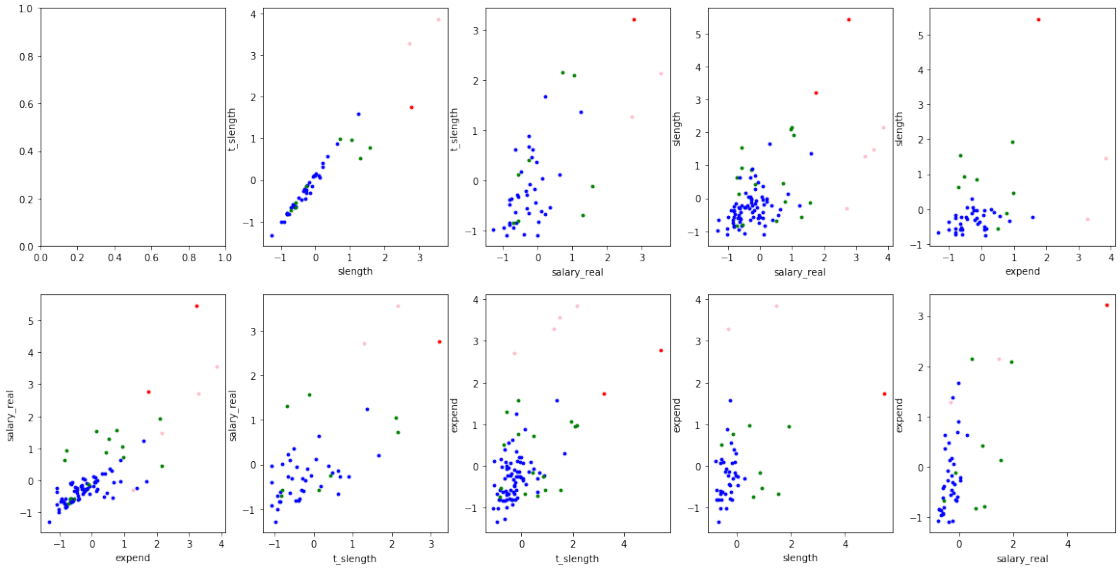
```
Two-Feature Clustering for K-Means Model
```



[23]:
```
#2-Feature graphs for gauss_em cluster
plt.clf()
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
for i in range(0,4):
    for j in range(0,4):
        if i!=j:
            plt.subplot(2,5,(int(i>j))*4+(i+j+1))
```

```
            name0, name1=['t_slength','slength','salary_real','expend'][i],␣
↪['t_slength','slength','salary_real','expend'][j]
            plt.scatter(df[[name0]][df.gauss_em_label_4==0],df[[name1]][df.
↪gauss_em_label_4==0], c="red", s=8)
            plt.scatter(df[[name0]][df.gauss_em_label_4==1],df[[name1]][df.
↪gauss_em_label_4==1], c="blue", s=8)
            plt.scatter(df[[name0]][df.gauss_em_label_4==2],df[[name1]][df.
↪gauss_em_label_4==2], c="pink", s=8)
            plt.scatter(df[[name0]][df.gauss_em_label_4==3],df[[name1]][df.
↪gauss_em_label_4==3], c="green", s=8)
            plt.ylabel(name0)
            plt.xlabel(name1)
print("Two-Feature Clustering for Gaussian Mixture Models")
```

```
Two-Feature Clustering for Gaussian Mixture Models

<Figure size 432x288 with 0 Axes>
```



Comparing our charts above, we see that, at the very least, with higher k we can actually at least compare the predictive power of our two different models (the predictions from the Gaussian and k-means model are not identical with $k = 4$, though with $k = 2$ they were.) In particular we can observe that the Gaussian Mixture models tend to put more weight on grouping outliers together in the $k = 4$ case than do K-Means models. This is because, perhaps, there is less robustness in Gaussian mixture models when it comes to local conditions, and thus the Gaussian Mixture models, even with higher k, tends to do a bad job distinguishing points that are closer together and to put too much effort into completely distinguishing outliers, as compared to the k-means model when k=4.

While this might tell us that the K-means model does a slightly better job producing distinct clusters, we can still note that both of these models are pretty mediocre when applied to the observations we have fed them. Likely, these methods are not good for our dataset because, as we had already guessed far above looking at the VAT, Hopkins Statistic, etc, there is not very much underlying structure for these models to go off. Hence the suggestion by the Calinksi Habaraz score above of using very high k: it is hard to make only a few clusters for the data and to have these clusters be reasonably accurate, and the consistency of this result across techniques suggests that this is due not simply to particular clustering alghorithms but also to the underlying data itself. To confirm this, let us look at the silhouette score of the five different models we have tried.

[24]:
```
#Agglomerative silhouette score
agg_score=metrics.
 ↪silhouette_score(df[['t_slength','slength','salary_real','expend']],␣
 ↪df['agg_label'])

#KMeans silhouette score with k=2, 4
k_means_score=metrics.
 ↪silhouette_score(df[['t_slength','slength','salary_real','expend']],␣
 ↪df['k_means_label'])
k_means_score_4=metrics.
 ↪silhouette_score(df[['t_slength','slength','salary_real','expend']],␣
 ↪df['k_means_label_4'])

#Gaussian EM silhouette score with k=2, 4
gauss_em_score=metrics.
 ↪silhouette_score(df[['t_slength','slength','salary_real','expend']],␣
 ↪df['gauss_em_label'])
gauss_em_score_4=metrics.
 ↪silhouette_score(df[['t_slength','slength','salary_real','expend']],␣
 ↪df['gauss_em_label_4'])

pd.DataFrame({'Method':['Agglomerative (k=22)', 'K Means (k=2)', 'K means␣
 ↪(k=4)', 'Gaussian EM (k=2)', 'Gaussian EM (k=4)'],
              'Silhouette score':[agg_score, k_means_score, k_means_score_4,␣
 ↪gauss_em_score, gauss_em_score_4]})
```

[24]:

	Method	Silhouette score
0	Agglomerative (k=22)	0.294930
1	K Means (k=2)	0.638989
2	K means (k=4)	0.474790
3	Gaussian EM (k=2)	0.638989
4	Gaussian EM (k=4)	0.337099

Recall that the Silhouette score in the range $(-1,1)$, with higher values representing a better fit for our data. As might be predicted, the two approaches with low k (for both K means and Gaussian EM) outperform the rest of the group by a decent margin. This is because, as stated above, it is quite easy to put observations into two groups: the natural choice is to make one large group with most of the data, and a smaller group for the outliers. Hence, by the technical scoring of the silhouette score, k means approaches with $k = 2$ and Gaussian EM approaches with $k = 2$

are more robust than the other models.

But as we have seen above, this is really not true: $k = 48$ may minimize our Calinksi Habaraz score, but it is in fact the most useless possible clustering result (i.e., it tells us lieterally nothing we didn't know by looking at the data); similarly, $k = 2$, though producing a superior silhouette score, tells us next-to-nothing about our data, only that it can be neatly broken down into outliers and non-outliers.

This is a perfet example of the sort of situation in which we might want to ignore what the direct technical silhouette scores are and try to examine some models that are, by this particular metric, worse, such as K means with k=4, Gaussian EM with k=4, or the Agglomerative approach, all three of which–despite being according to our scores above less robust than the GEM and KM appraoches with $k = 2$–have the benefit of actually giving us some non-trivial insight/information about the structure underlying our data.

Of these, the agglomerative model with $k = 22$ seems to be the least promising, which is obvious in part because $k = 22$ is simply too high. As discussed above, in this approach many of the data points are given their own clusters which is not particularly useful. This is reflected in a much lower Silhouette score since points that ought to be in the same cluster, or at least are not obviously deserving of their own cluster, are forced into different clusters due to the agglomerative algorithm.

This leaves the Gaussian EM and K Means cases for k=4. As we said, the Gaussian case tends to do a comparatively bad job with outliers, and this sensitivity is bad news in our case since our data seems to be fairly noisy (or at least to have a few observations that are disperse from the main cluster and one another, as shown above.) This explains, in part, why the $k = 4$ Gaussian EM model has the worst Silhouette score of any above.

So the best choice seemingly, when we balance the technical and heuristic information we have gathered, would be K means, with k=4. Though this is the best choice of those models tested above–offering a reasonably high Silhouette score and a reasonably robust Calinksi Habaraz score–we should try to see if we cannot tune this model slightly to better balance the two validation statistics above.

We present the Calinksi Habaraz scores and Silhouette scores for the k-means model for k, $0 <= k <= 10$ below.

[25]:
```
ch_scores=pyclustertend.metric.
 ↪assess_tendency_calinski_harabaz(df[['t_slength','slength','salary_real','expend']],␣
 ↪max_nb_cluster=10)[1]
silhouette_scores=[
    metrics.silhouette_score(df[['t_slength','slength','salary_real','expend']],␣
 ↪cluster.KMeans(k).
 ↪fit_predict(df[['t_slength','slength','salary_real','expend']])) for k in␣
 ↪range(2,11)]
```

[26]:
```
pd.DataFrame({'k':[k for k in range(2,11)],'Calinksi Habaraz score':ch_scores,␣
 ↪'Silhouette score': silhouette_scores}).set_index('k')
```

[26]:

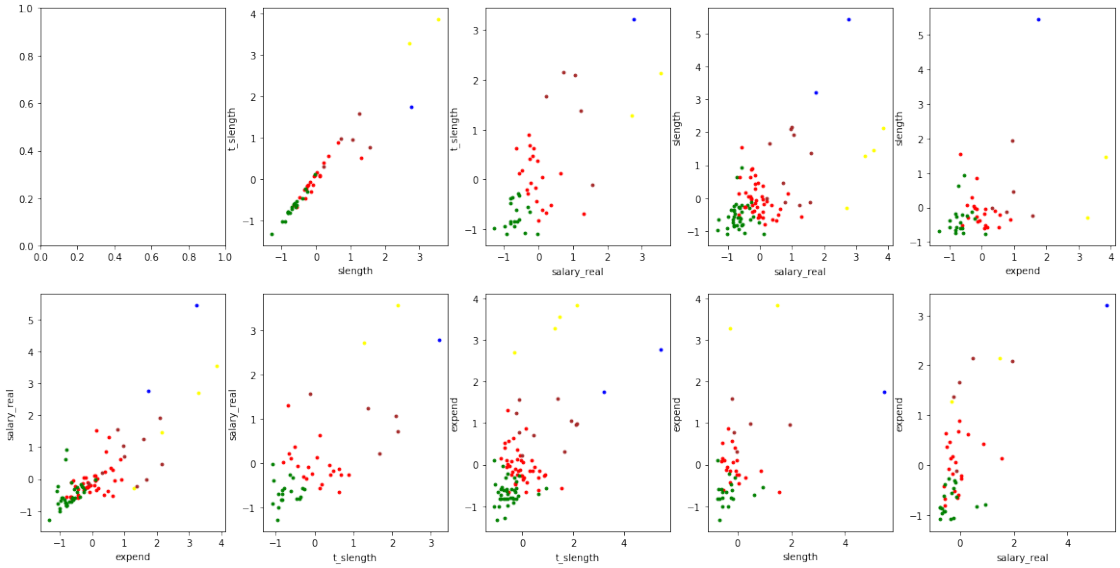
k	Calinksi Habaraz score	Silhouette score
2	52.506374	0.638989
3	41.288484	0.562044
4	40.954540	0.280838
5	46.112164	0.300493

```
6                51.639822          0.319717
7                52.438361          0.291916
8                53.573321          0.350444
9                53.353181          0.345857
10               56.664789          0.307202
```

Per the above, we find that the Calinksi Habaraz score tends to improve in k, while the Silhouette score hovers around .30 for $k \geq 5$. We might pick then as an optimal value of $k = 5$, since this, like others in the range $k = 5, \ldots, 10$ offer a good, similar balance between our Calinski-Habaraz and Silhouette scores and, all else being (approximately) equal, it is probably better to aim for fewer clusters when dealing with such a small dataset of ~50 observations. Just to round out this analysis, we present the 2D cluster models for a k-means analysis with our chosen value of $k = 5$ below.

[27]:
```
k_mean_cluster=cluster.KMeans(5)
k_mean_cluster.fit(df[['t_slength','slength','salary_real','expend']])
df['k_means_label_5']=k_mean_cluster.
 ↪predict(df[['t_slength','slength','salary_real','expend']])

fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
for i in range(0,4):
    for j in range(0,4):
        if i!=j:
            plt.subplot(2,5,(int(i>j))*4+(i+j+1))
            name0, name1=['t_slength','slength','salary_real','expend'][i],␣
 ↪['t_slength','slength','salary_real','expend'][j]
            plt.scatter(df[[name0]][df.k_means_label_5==0],df[[name1]][df.
 ↪k_means_label_5==0], c="red", s=8)
            plt.scatter(df[[name0]][df.k_means_label_5==1],df[[name1]][df.
 ↪k_means_label_5==1], c="blue", s=8)
            plt.scatter(df[[name0]][df.k_means_label_5==2],df[[name1]][df.
 ↪k_means_label_5==2], c="green", s=8)
            plt.scatter(df[[name0]][df.k_means_label_5==3],df[[name1]][df.
 ↪k_means_label_5==3], c="yellow", s=8)
            plt.scatter(df[[name0]][df.k_means_label_5==4],df[[name1]][df.
 ↪k_means_label_5==4], c="brown", s=8)
            plt.ylabel(name0)
            plt.xlabel(name1)

print("Two-Feature Clustering for K-Means Model")
```

```
Two-Feature Clustering for K-Means Model
```



While the clustering above is not so good (as we have said many times above, this is not so surprising, since we are working with a small and somewhat hard-to-predict dataset) it does seem like after our analysis the k-tuned model with $k = 5$ offers comparatively robust, reliable, and meaningful clustering results. This reflects our best efforts at creating a tuned, accurate model by balancing out technical specs (the Silhouette and CH scores above) with heuristics (the fact we should keep the ratio k/n where n =number of obs relatively low, and by judging a reasonable tradeoff between our scores). We hope that, as is often the case, a healthy mix of heuristics and technical optimization–with our best instincts checking and giving depth to our technical results, and vis versa–has produced a comparatively useful model.

[ ]: