Data exploration

Spyros Samothrakis
Research Fellow, IADS
University of Essex

February 21, 2017

About

Clustering

Dimensionality reduction

Outlier detection

Conclusion

About

- We will be discussing ways of understanding the data
  - Without assuming there is something to predict
- For example, you might want to split your customers into different groups
  - But you have no idea which groups are out there
- You just have a description of each sample
  - For example each $< sales, location.... >$

How often do people do it?

- Labelled data is often scarce
  - . . . and often requires human intervention
- We tend to group things all the time
  - Short / tall
  - Fast / slow
  - Certain types of clothing
- . . . etc

Exploring data

"If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake."

– Yann LeCun

Types of learning (again)

- Supervised Learning
  - Predictions
- Reinforcement Learning
  - (very close to bandits)
- Unsupervised Learning
  - Learning about the data without any signal
  - We are not doing that well (but this might be about to change)
  - Alternatively you can try to predict all your features!

Clustering

- We would like to group our data into different groups
- Are there "natural classes" in the data?[1]

[1] `http://www.telegram.com/assets/microsites/weddings_site/weddings/0004.html?`

Our data - US 1990 Census

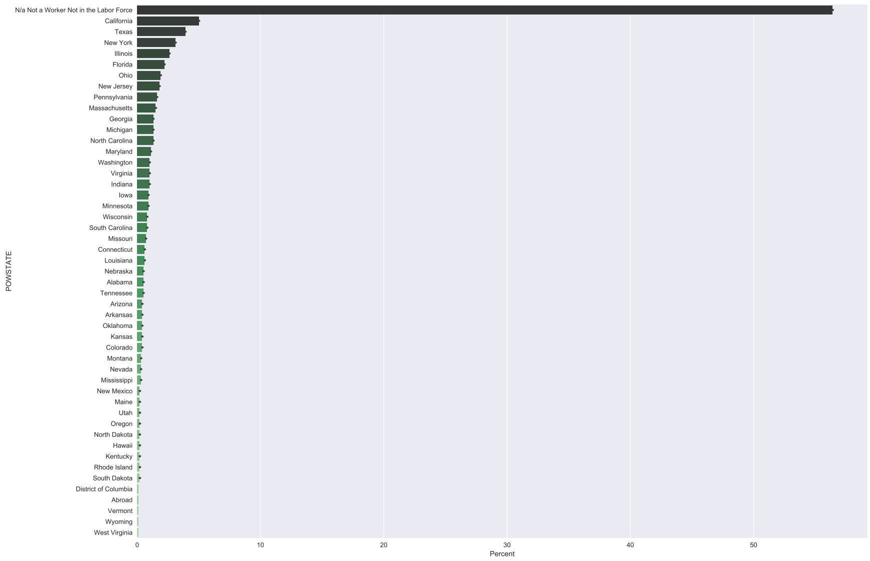
- 2,458,286 people
- 125 different features
- Age, ethnicity, state, income, education etc

Loading only part of the data[2]

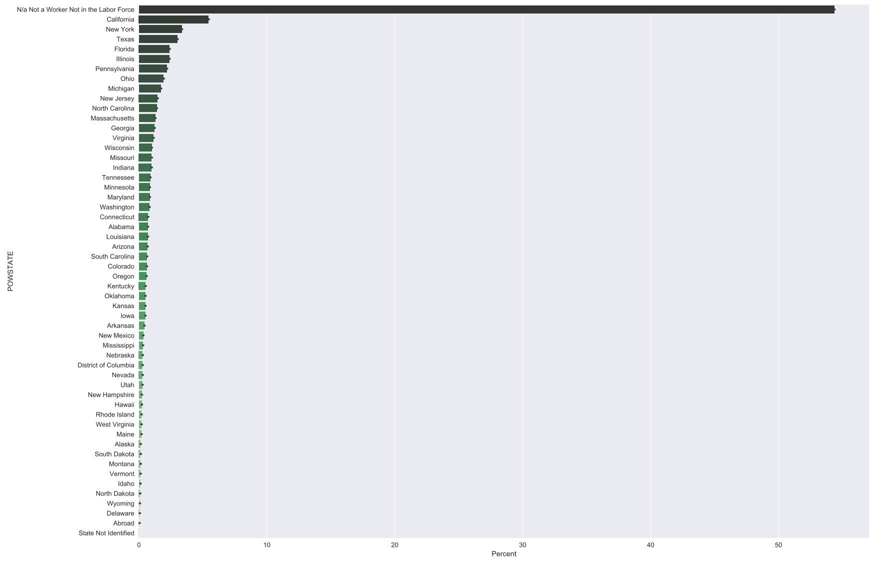
```
n = 2458286 # Number of rows in file
s = 1000 # desired sample size
filename = "hUSCensus1990raw.data.zip"
skip = sorted(np.random.choice(range(1,n), n-s-1, replace=False))
df = pandas.read_csv(filename,compression = "zip", header=0, sep='\t', skiprows=skip)
```

[2] http://stackoverflow.com/questions/22258491/read-a-small-random-sample-from-a-big-csv-file-into-a-python-data-frame

Labour (1000 samples)



Labour (20,000 samples)



Creating the plot

```
state_codes = {
0:"N/a Not a Worker Not in the Labor Force",
1:"Alabama",
2:"Alaska",
4:"Arizona",
5:"Arkansas",
6:"California",
8:"Colorado",
9:"Connecticut"
...
}

x = df[["POWSTATE"]]

x = x.replace(state_codes)
counts = x["POWSTATE"].value_counts()
sort = counts.index.tolist()

#print df.index
plt.figure(figsize=(20, 15))

ax = sns.barplot(x="POWSTATE", y="POWSTATE", data=x,
                    estimator =lambda x: (float(len(x)) / float(len(df))* 100.0) ,
                    palette="Greens_d", order = sort, orient="h")

ax.set(xlabel="Percent")
```


Clustering data

```
income_sum = df[["INCOME" + str(i) for i in range(1,8)]].sum(axis = 1)

df_age_income = df[["AGE"]].copy()
df_age_income["INCOME"] = income_sum

df_age_income.head()
```

	AGE	INCOME
0	14	0
1	74	3984
2	48	10500
3	41	0
4	24	7300

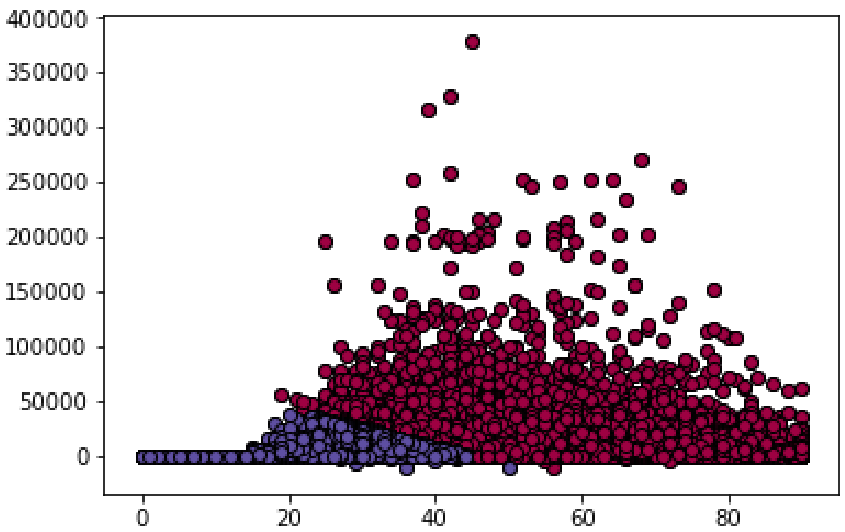
KMeans

- Possibly the most popular algorithm for clustering
- Initialise with “*n_clusters*” random “centroids”
- Iterates over two steps
  - Assign each point to one of the centroids it is closer to using euclidean distance
  - Create new centroids by defining each centroid as the average of each dimension
- Repeat
- Algorithms is unstable, different starting positions will result in different clusters

Let’s run it

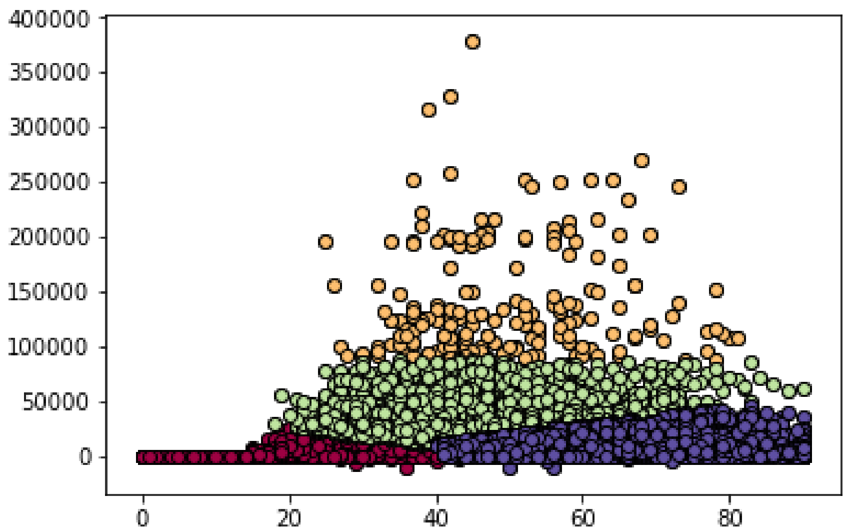
```
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_db = sc.fit_transform(X)
n_clusters = 2

clusterer = KMeans(n_clusters = n_clusters).fit(X_db)
labels = clusterer.predict(X_db)
```


Output



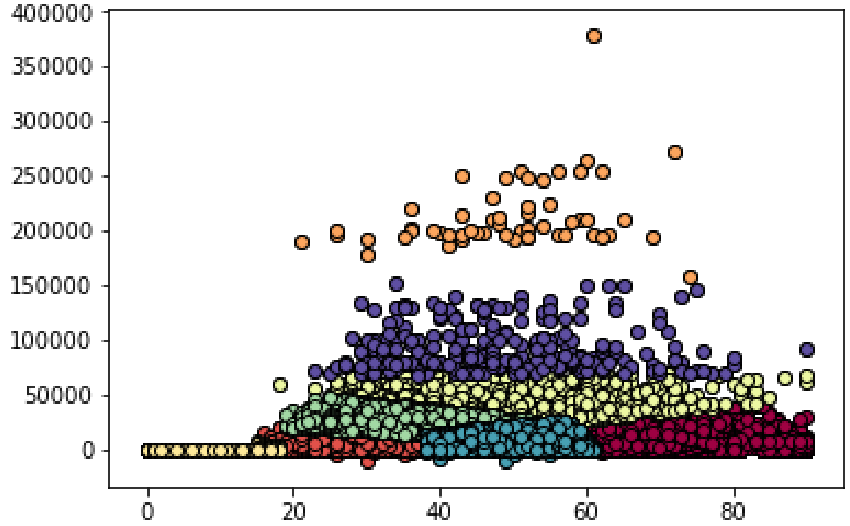
Number of clusters: 2 Silhouette Coefficient: 0.458

We can do the same for more clusters (4)



Number of clusters: 4 Silhouette Coefficient: 0.509

We can do the same for more clusters (8)



Number of clusters: 8 Silhouette Coefficient: 0.436

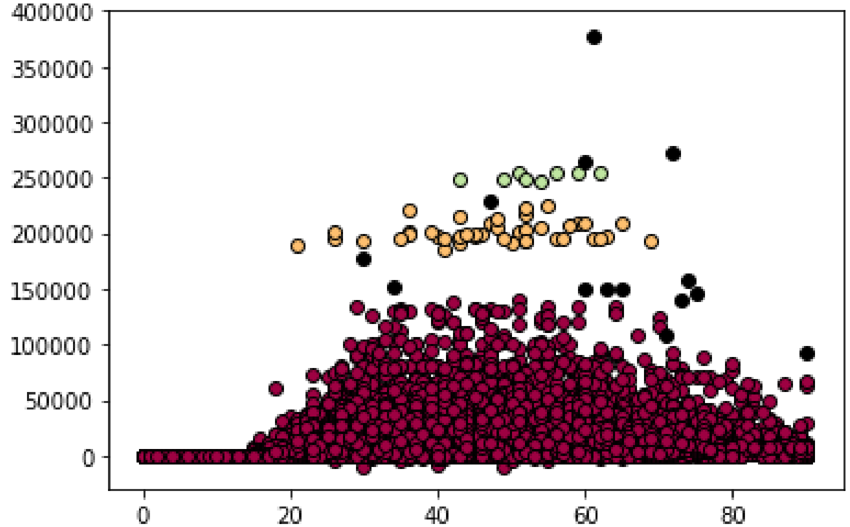
Inverse transforms

```
sc.inverse_transform(clusterer.cluster_centers_)
```

Number of clusters: 4 Silhouette Coefficient: 0.509

```
array([[     63.39942604,     9381.90607879],
       [     41.20647543,    34883.62681057],
       [     49.44502618,   142403.79581152],
       [     17.09429913,     3370.73360161]])
}
```


DBScan



Estimated number of clusters: 3 Silhouette Coefficient: 0.800

Silhouette coefficient

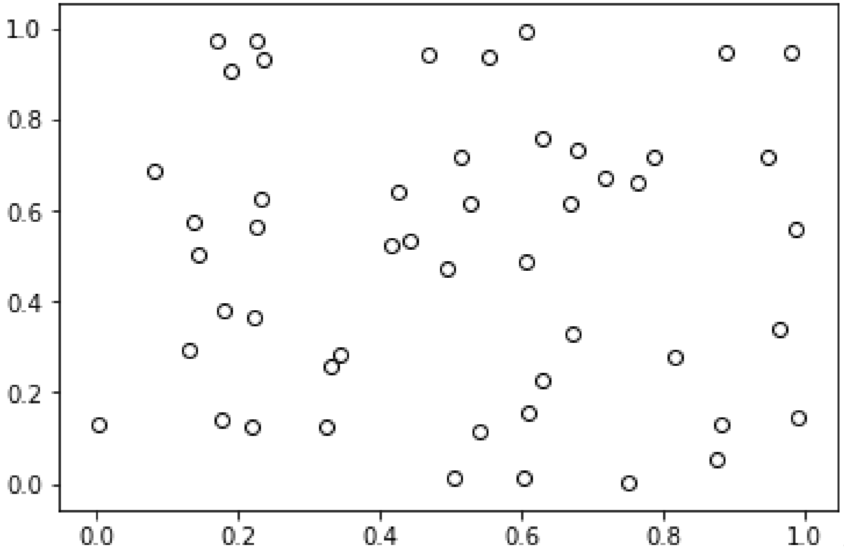
- One of the few clustering metrics that does not require the ground truth
- ... which if you had, you probably wouldn't be clustering anything
- (What would you do?)
- For each object find the distance a_i in its own cluster
- Calculate the average distance b_{ij} of object i to every other cluster

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

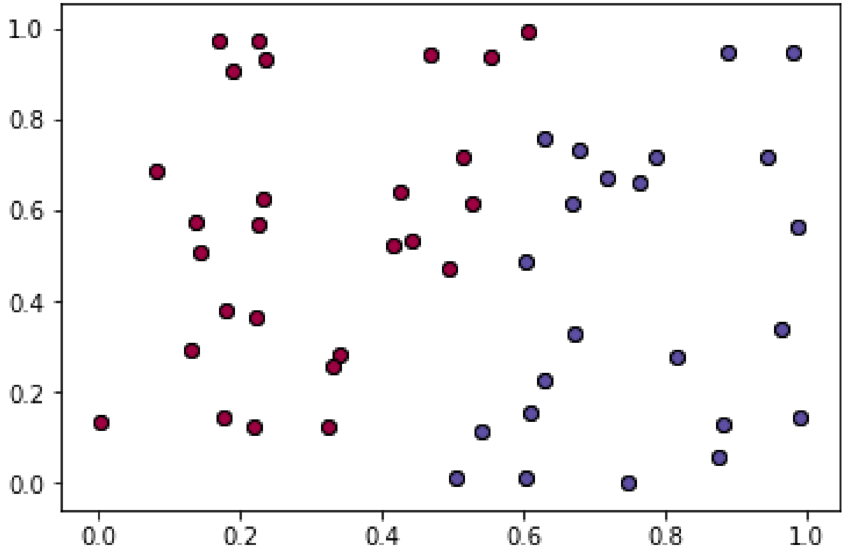
- The average s over all points is the coefficient

Clustering Tendency

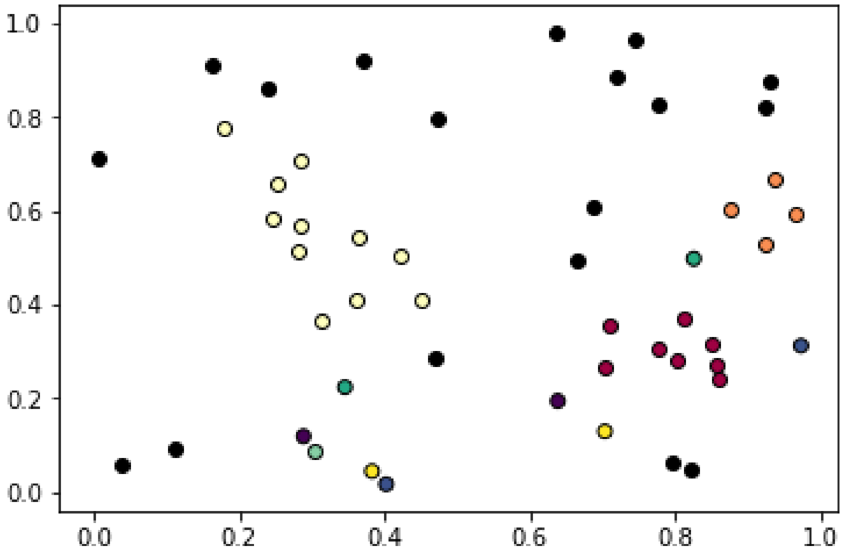
Can you see the clusters here?



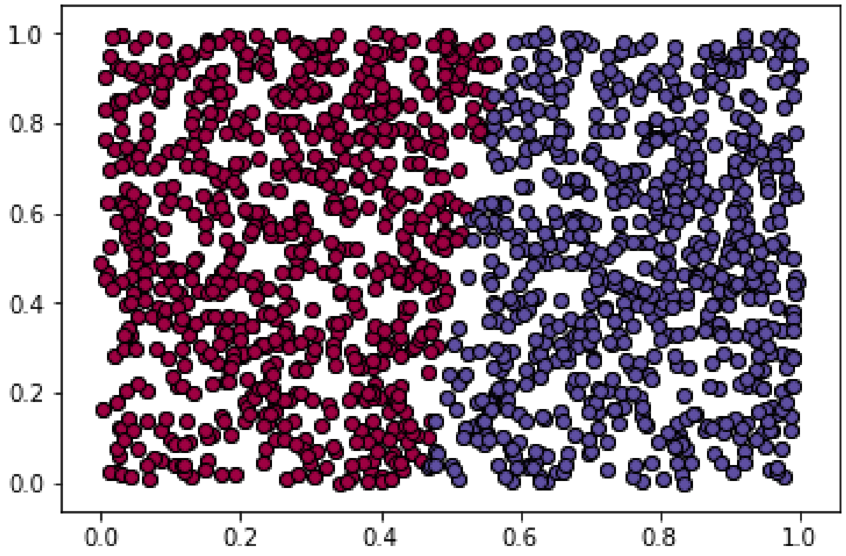
KMeans on random data



DBSCAN on random data



KMeans on 1500 random data points



Measurements

- Hopkins coefficient will tell you if your data come from a random uniform distribution
- Checks the clustering tendency of the data
- q is a distance from an observation to each nearest neighbour
- w is the distance from a generate random observation to the nearest real neighbour

$$H = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i}$$

- If close to 0.5, data is random, close to 1.0 data is real

Projections and dimensionality reduction

- The data we clustered was in very low dimensional space (2 dimensions)
- How about data that has a massive number of dimensions?
  - Or just higher than two for our exhibition purposes
- Maybe there is a also a way of transforming to lower dimensions to at least visualise it

PCA

- Principle Component Analysis
- One of the most popular methods of dimensionality reduction
- Finds the components of the data where the highest variance (change) takes place
- You select as many of them as you think are relevant for your tasks
- Quite often followed up by predictions

Let's play a bit with the data

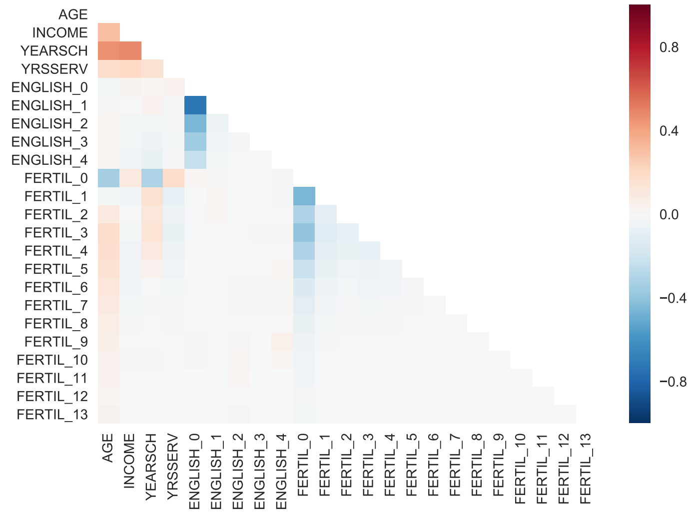
```
income_sum = df[["INCOME" + str(i) for i in range(1,8)]].sum(axis = 1)

df_demo = pd.DataFrame()


df_demo["AGE"] = df[["AGE"]].copy()
df_demo["INCOME"] = income_sum

df_demo["YEARSCH"] = df[["YEARSCH"]].copy()
df_demo["ENGLISH"] = df[["ENGLISH"]].copy()
df_demo["FERTIL"] = df[["FERTIL"]].copy()
df_demo["YRSSERV"] = df[["YRSSERV"]].copy()


df_demo = pd.get_dummies(df_demo, columns = ["ENGLISH", "FERTIL" ] )
```

Correlation

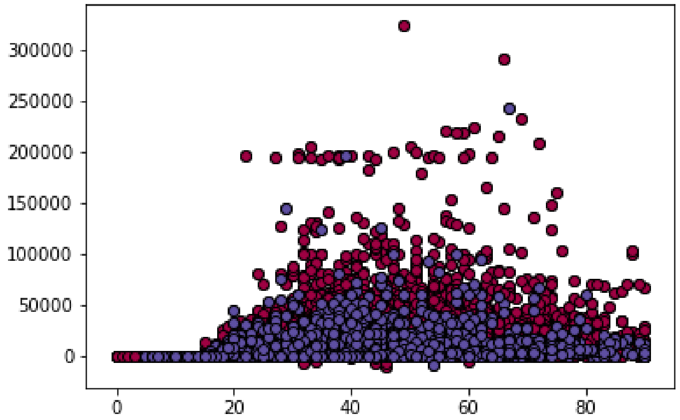
- We can see how much each input feature is “correlated” with each other
- Pearson correlation coefficient ranges from -1 to 1
- 1 means “features change together”, -1 means “features change the opposite direction”
- Uncorrelated features have correlation values close to 0

```
import seaborn as sns
sns.set(style="white")
mask = np.zeros_like(df_demo.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(df_demo.corr(), mask = mask)
```


Correlation plot



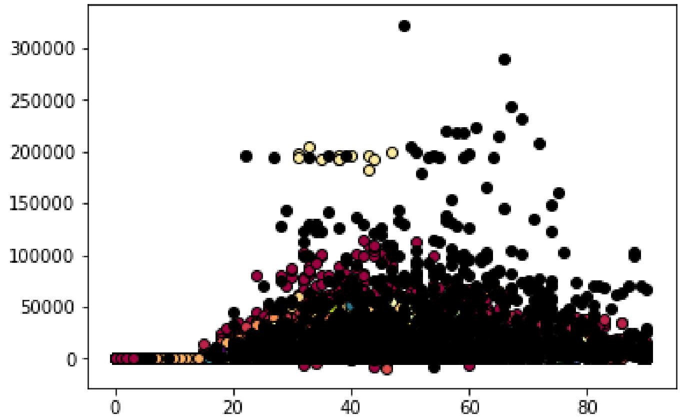
Results KMeans



Number of clusters: 2

Silhouette Coefficient: 0.396

Results DBSCAN



Estimated number of clusters: 52

Silhouette Coefficient: 0.049

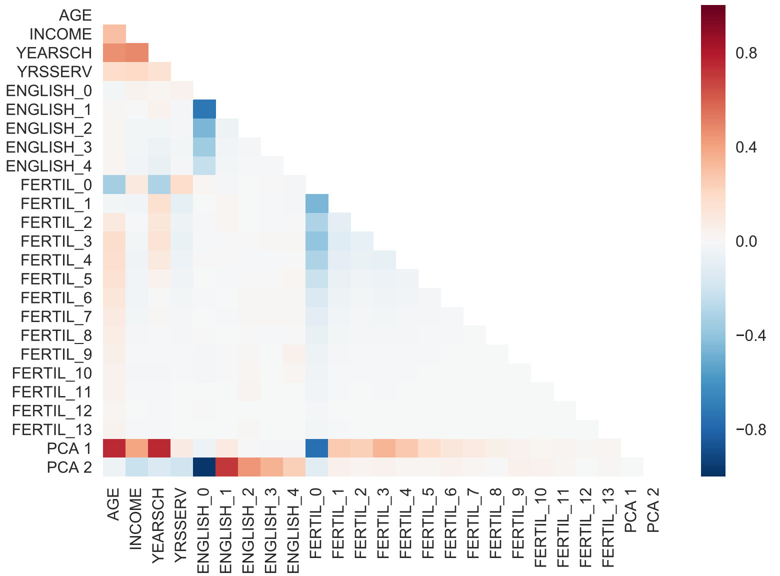
Running PCA

```
from sklearn.decomposition import PCA, KernelPCA
from sklearn.manifold import TSNE, MDS

model = PCA(n_components = 2)

X_r = model.fit_transform(StandardScaler().fit_transform(X))
```

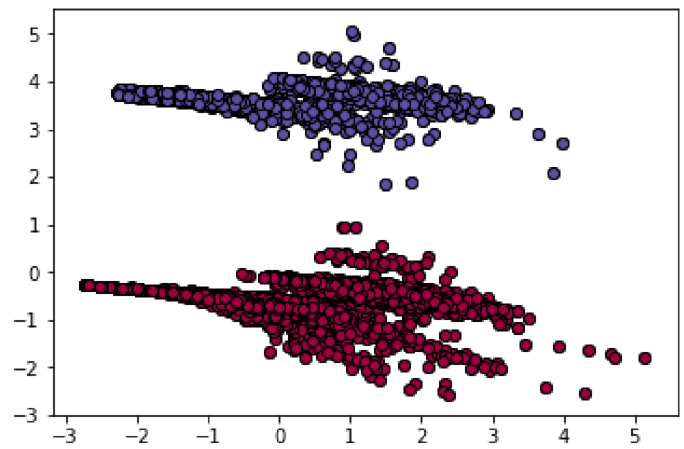

Using PCA

- We know have the two components with the highest variance
- PCA is not the only algorithm to do this
- You can also use PCA for doing predictions
- Each PCA component is a linear combination of the input features
- So we can plot a correlation plot and see how they are related

Correlation plot



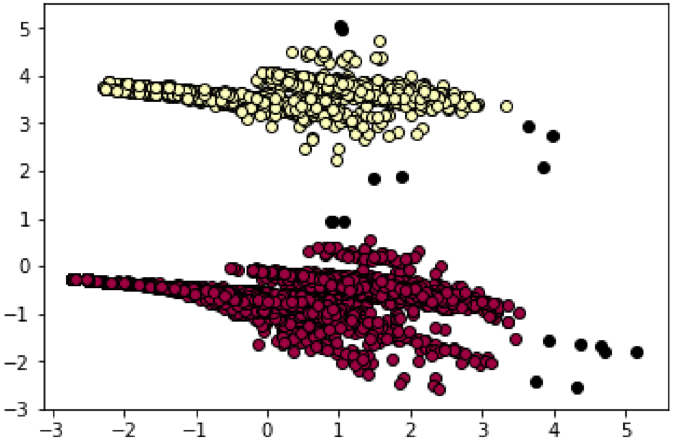
Results KMeans PCA



Number of clusters: 2

Silhouette Coefficient: 0.617

Results DBSCAN PCA



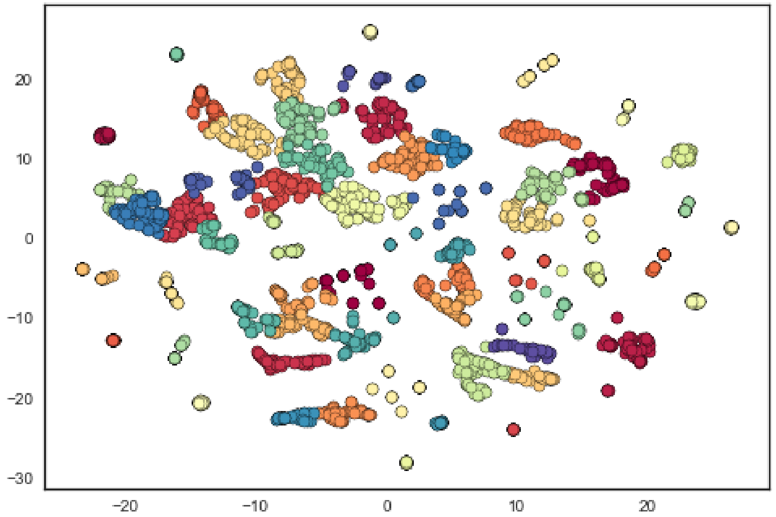
Estimated number of clusters: 2

Silhouette Coefficient: 0.394

T-SNE

- Another option (but more linked to visualisation) is T-SNE (t-Distributed Stochastic Neighbour Embedding)
- Think of having the globe (thee dimensions) and trying to unfold it to two dimensions
- Use it whenever you have to visualise data
- What is being produced is not obvious, but if you have

T-SNE results



Outlier detection

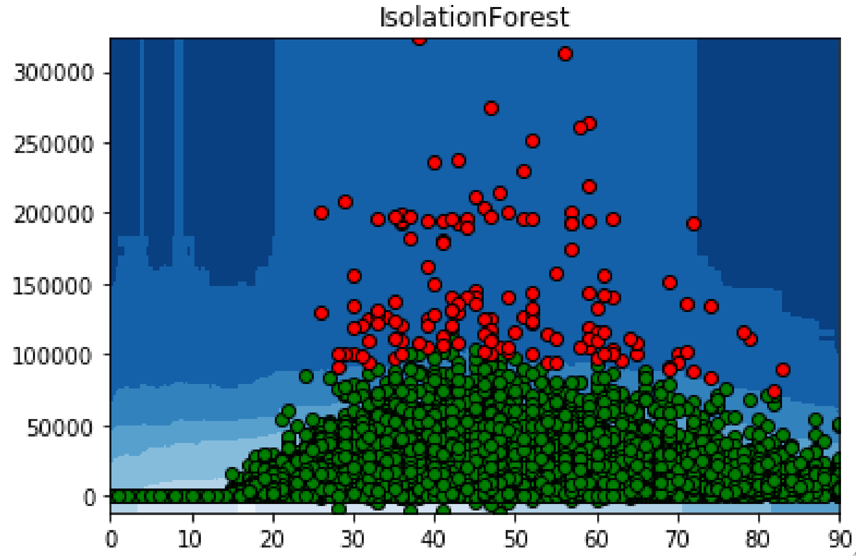
- You want to check which observations do not fit in with the rest
- Isolations forests
  - Keep splitting features until at random until every observation is in it's own tree leaf
  - Observations with short paths are treated

Isolation forests code

```
from sklearn.ensemble import IsolationForest

clf = IsolationForest(max_samples=100, contamination = 0.01)
clf.fit(X)
y_pred_train = clf.predict(X)

pos = y_pred_train > 0
neg = y_pred_train < 0

# plot the line, the samples, and the nearest vectors to the plane
xx, yy = np.meshgrid(np.linspace(min((X[:, 0])), max((X[:, 0])), 500), np.linspace(min((X[:, 1])), max((X[:,
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)


plt.title("IsolationForest")
plt.contourf(xx, yy, Z, cmap=plt.cm.Blues_r)

b1 = plt.scatter(X[pos][:, 0], X[pos][:, 1], c='green', edgecolor='k')
b2 = plt.scatter(X[neg][:, 0], X[neg][:, 1], c='red', edgecolor='k')

plt.axis('tight')

plt.xlim((xx.min(), xx.max()))
plt.ylim((yy.min(), yy.max()))

print pos.sum()
print neg.sum()
```


Result

14850 Normal predictions, 150 Outliers



Conclusion

- We have seen a set of methods for understanding the data without an outcome signal to guide us
- There is a revolution currently going on in this field, we will cover it after neural networks
- Some of the methods useful even if you have a signal, e.g. do PCA/KMeans on the data and then feed them into a classifier