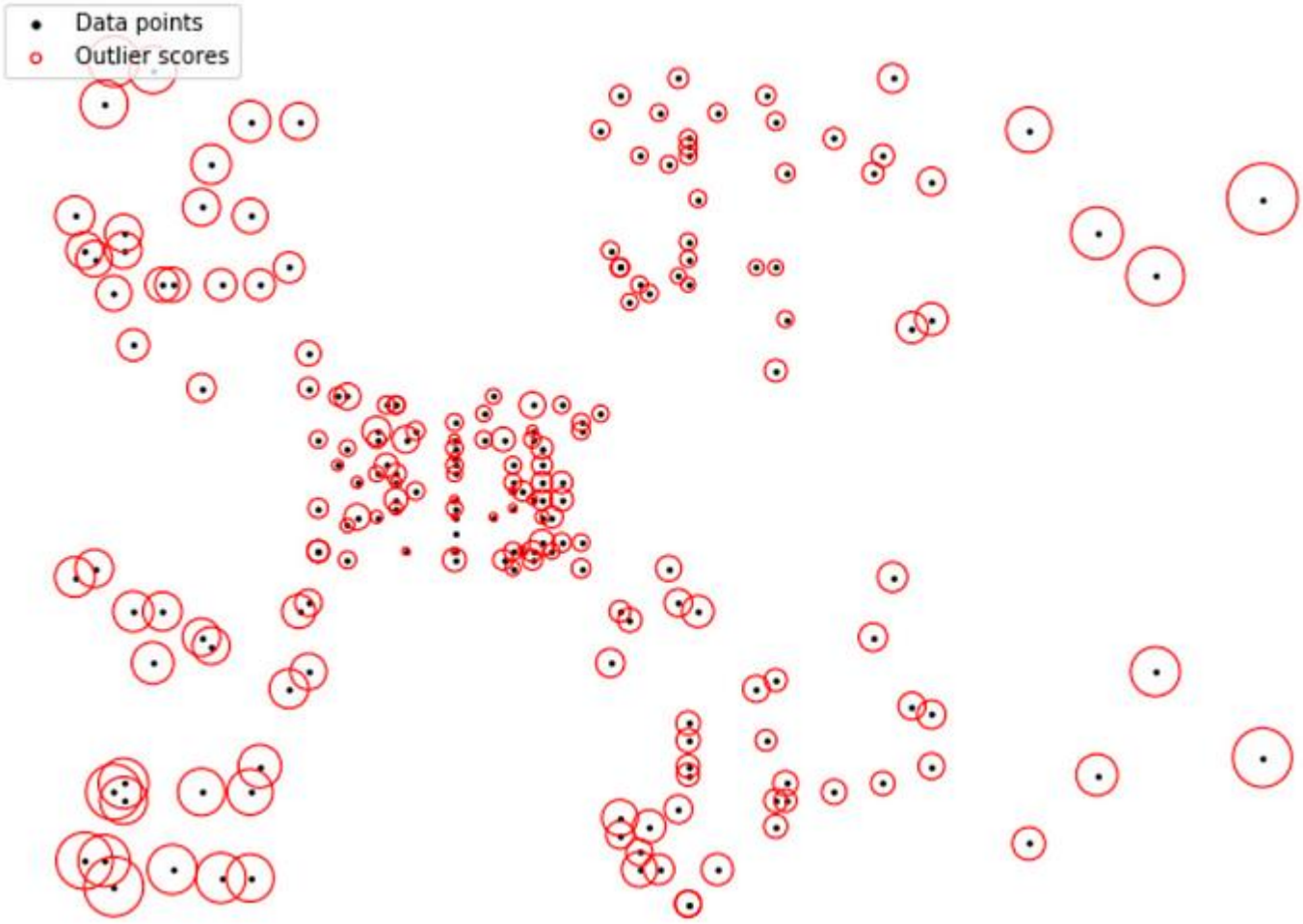
# Anomaly Detection Techniques in Python




Visual Representation of Local Outlier Factor Scores

I recently learned about several anomaly detection techniques in Python. These techniques identify anomalies (outliers) in a more mathematical way than just making a scatterplot or histogram and eyeballing it. If a point is an outlier with respect to its values across 30 features (a multivariate outlier), you can't identify it using the above methods, which is where these techniques come in.

**Format of blog post (most techniques are formatted as follows):**

1. Explanation of technique
2. Basic implementation in sklearn
3. Visualization showing anomalies identified using technique
4. Sources used in the above

**Covered Techniques:**

1. DBSCAN
2. Isolation Forests
3. Local Outlier Factor
4. Elliptic Envelope
5. One-Class Support Vector Machines

**Some Definitions:**

1. An **outlier** is an observation with at least one variable having an unusual value.
2. A **univariate outlier** is an observation with a variable that has an unusual value.
3. A **multivariate outlier** is an observation with at least two variables having unusual values.

These techniques identify outliers, so either univariate or multivariate outliers.

**Several textbooks with discussed techniques:**

1. Hands-On Machine Learning with scikit-learn and Scientific Python Toolkits (Released 7/24/2020)
Discusses DBSCAN, Isolation Forests, LOF, Elliptic Envelope (easy to read)

2. Beginning Anomaly Detection Using Python-Based Deep Learning: With Keras and PyTorch 1st ed. 2019
Discusses Isolation Forests, One-Class SVM, and more (easy to read)

3. Outlier Analysis 2nd ed. 2017 Edition
Discusses Isolation Forests, LOF, One-Class SVM, and more (harder to read)

**Dataset Used in Code Examples:**
https://www.kaggle.com/akram24/mall-customers

**Data Preprocessing:**

```
data=data.drop('CustomerID',axis=1)
data=data.rename(columns={
 'Annual Income (k$)':'Income',
 'Spending Score (1-100)':'Spend_Score'})
df=pd.get_dummies(data)from sklearn.preprocessing import
MinMaxScaler
scaler = MinMaxScaler()
num2 = scaler.fit_transform(num)
num2 = pd.DataFrame(num2, columns = num.columns)
```

## DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

This is a clustering algorithm (an alternative to K-Means) that clusters points together and identifies any points not belonging to a cluster as outliers. It's like K-means, except the number of clusters does not need to be specified in advance.

### The method, step-by-step:

1. Randomly select a point not already assigned to a cluster or designated as an outlier. Determine if it's a core point by seeing if there are at least *min_samples* points around it within *epsilon* distance.
2. Create a cluster of this core point and all points within *epsilon* distance of it (all directly reachable points).
3. Find all points that are within *epsilon* distance of each point in the cluster and add them to the cluster. Find all points that are within *epsilon* distance of all newly added points and add these to the cluster. Rinse and repeat. (i.e. perform "neighborhood jumps" to find all density-reachable points and add them to the cluster).

### Lingo underlying the above:

1. Any point that has at least *min_samples* points within *epsilon* distance of it will form a cluster. This point is called a core point. The core point will itself count towards the *min_samples* requirement.

2. Any point within *epsilon* distance of a core point, but does not have *min_samples* points that are within *epsilon* distance of itself is called a borderline point and does not form its own cluster.

3. A border point that is within *epsilon* distance of multiple core points (multiple *epsilon* balls) will arbitrarily end up in just one of these resultant clusters.

4. Any point that is randomly selected that is not found to be a core point or a borderline point is called a noise point or outlier and is not assigned to any cluster. Thus, it does not contain at least *min_samples* points that are within *epsilon* distance from it or is not within *epsilon* distance of a core point.

5. The epsilon-neighborhood of point p is all points within *epsilon* distance of p, which are said to be directly reachable from p.

6. A point contained in the neighborhood of a point directly reachable from p is not necessarily directly reachable from p, but is density-reachable.

7. Any point that can be reached by jumping from neighborhood to neighborhood from the original core point is density-reachable.

**Implementation Considerations:**

1. You may need to standardize / scale / normalize your data first.
2. Be mindful of data type and the distance measure. I've read that the gower distance metric can be used for mixed data types. I've implemented Euclidean, here, which needs continuous variables, so

I removed gender.

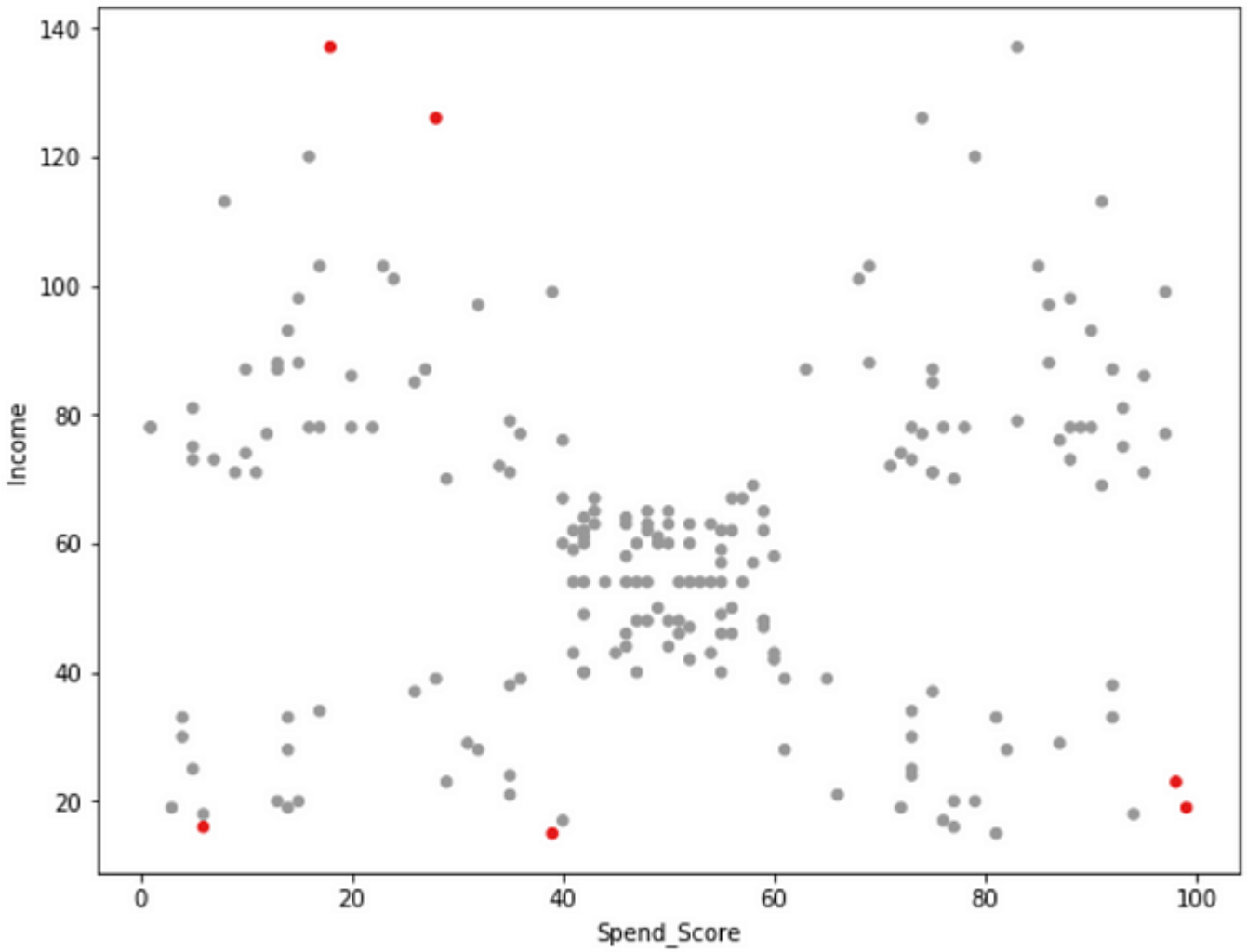
3. You will want to optimize *epsilon* and *min_samples*.

### Sklearn Implementation of DBSCAN:

```
from sklearn.cluster import DBSCAN
outlier_detection = DBSCAN(
 eps = .2,
 metric="euclidean",
 min_samples = 5,
 n_jobs = -1)clusters = outlier_detection.fit_predict(num2)
```

DBSCAN will output an array of -1's and 0's, where -1 indicates an outlier. Below, I visualize outputted outliers in red by plotting two variables.

```
from matplotlib import cm
cmap = cm.get_cmap('Set1')
num.plot.scatter(x='Spend_Score',y='Income', c=clusters,
cmap=cmap,
 colorbar = False)
```




DBSCAN Outliers

**More Information on DBSCAN:**

**Textbook Links**

1. Hands-On Machine Learning with scikit-learn and Scientific Python Toolkits (Released 7/24/2020)
2. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow... (Released 10/2019)
3. Anomaly Detection Principles and Algorithms 2017 Edition

**Web Links**

1. https://mikulskibartosz.name/outlier-detection-with-scikit-learn-d9246b33d352
2. https://blog.dominodatalab.com/topology-and-density-based-clustering/
3. http://mccormickml.com/2016/11/08/dbscan-clustering/
4. https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/
5. https://www.quora.com/How-does-DBSCAN-algorithm-work
6. https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80
7. https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818
8. https://medium.com/@soroush.hashemi76/kmeans-vs-dbscan-d9d5f9dbee8b

## Isolation Forests

For each observation, do the following:

1. Randomly select a feature and randomly select a value for that feature within its range.
2. If the observation's feature value falls above (below) the selected value, then this value becomes the new min (max) of that feature's range.

3. Check if at least one other observation has values in the range of each feature in the dataset, where some ranges were altered via step 2. If no, then the observation is isolated.

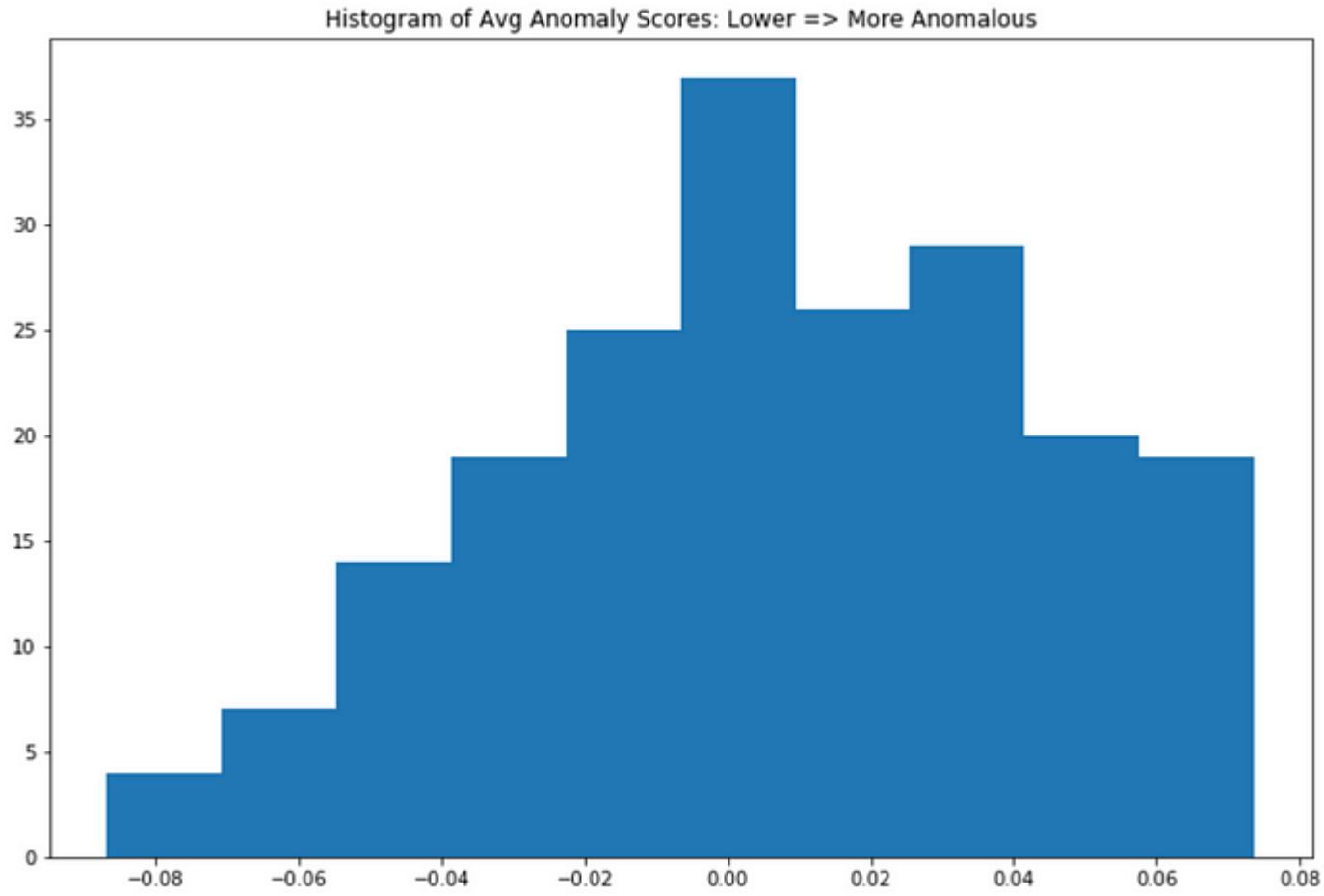
4. Repeat steps 1–3 until the observation is isolated. The number of times you had to go through these steps is the isolation number. The lower the number, the more anomalous the observation is.

## Sklearn Implementation of Isolation Forests:

```
from sklearn.ensemble import IsolationForest
rs=np.random.RandomState(0)
clf = IsolationForest(max_samples=100,random_state=rs,
contamination=.1)
clf.fit(df)
if_scores = clf.decision_function(df)
if_anomalies=clf.predict(df)
if_anomalies=pd.Series(if_anomalies).replace([-1,1],[1,0])
if_anomalies=num[if_anomalies==1];
```

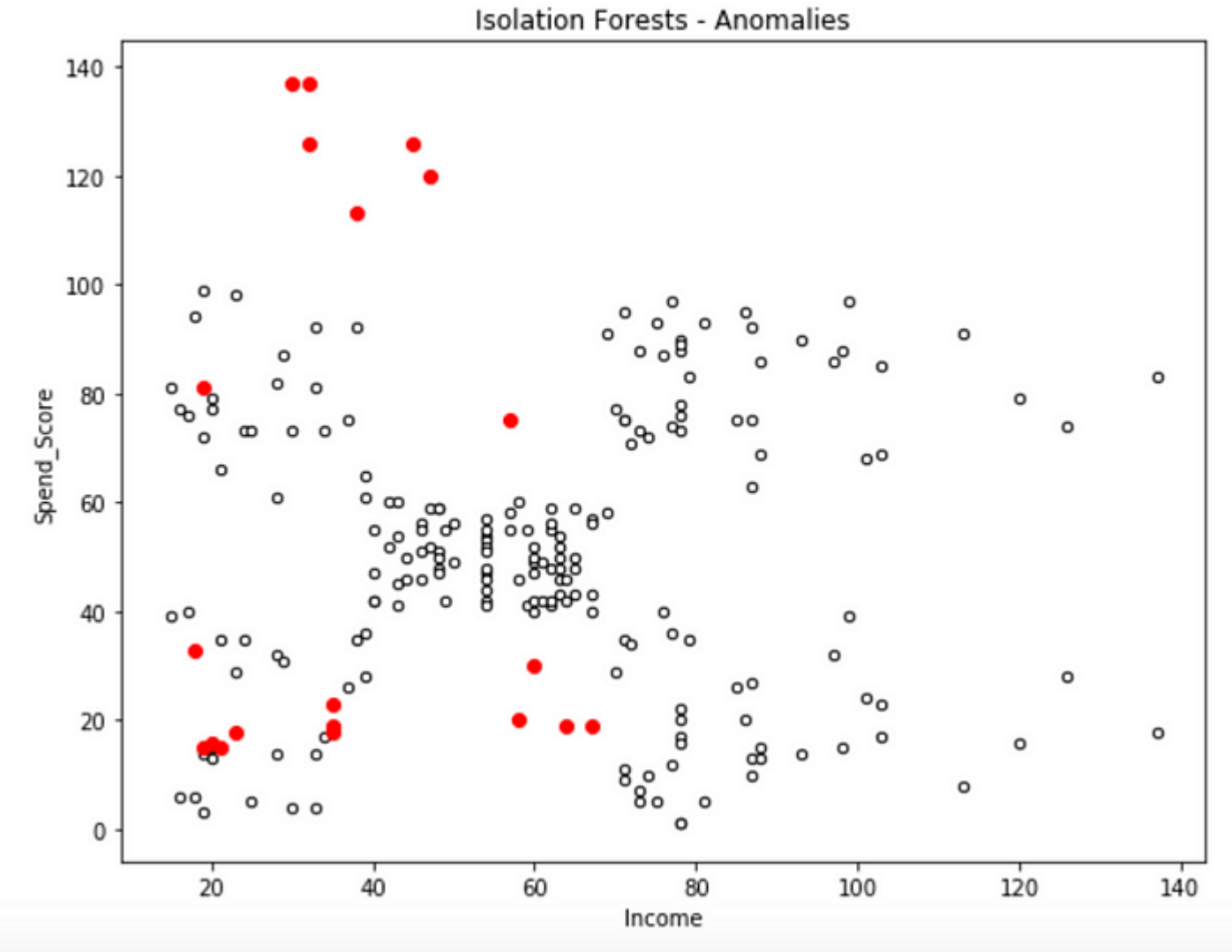
Below, I plot a histogram of *if_scores* values. Lower values indicate observations that are more anomalous.

```
plt.figure(figsize=(12,8))
plt.hist(if_scores);
plt.title('Histogram of Avg Anomaly Scores: Lower => More
Anomalous');
```




Below, I plot observations identified as anomalies. These observations have *if_scores* values below the *clf.threshold_* value.

```
cmap=np.array(['white','red'])
plt.scatter(df.iloc[:,1],df.iloc[:,2],c='white',s=20,edgecolor='
k')
plt.scatter(if_anomalies.iloc[:,0],if_anomalies.iloc[:,1],c='red
')
plt.xlabel('Income')
plt.ylabel('Spend_Score')
plt.title('Isolation Forests - Anomalies')
```



Isolation Forest Outliers

## More Information on Isolation Forests:

### Textbooks

1. Hands-On Machine Learning with scikit-learn and Scientific Python Toolkits (Released 7/24/2020)
2. Beginning Anomaly Detection Using Python-Based Deep Learning: With Keras and PyTorch 1st ed. 2019
3. Outlier Analysis 2nd ed. 2017 Edition

### Web Links

1. https://www.depends-on-the-definition.com/detecting-network-attacks-with-isolation-forests/
2. https://stackoverflow.com/questions/45223921/what-is-the-range-of-scikit-learns-isolationforest-decision-function-scores
3. https://quantdare.com/isolation-forest-algorithm/
4. https://medium.com/@hyunsukim_9320/isolation-forest-step-by-step-341b82923168
5. http://www.ncsa.illinois.edu/Conferences/LSST18/assets/pdfs/hariri_forest.pdf
6. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

## Local Outlier Factor

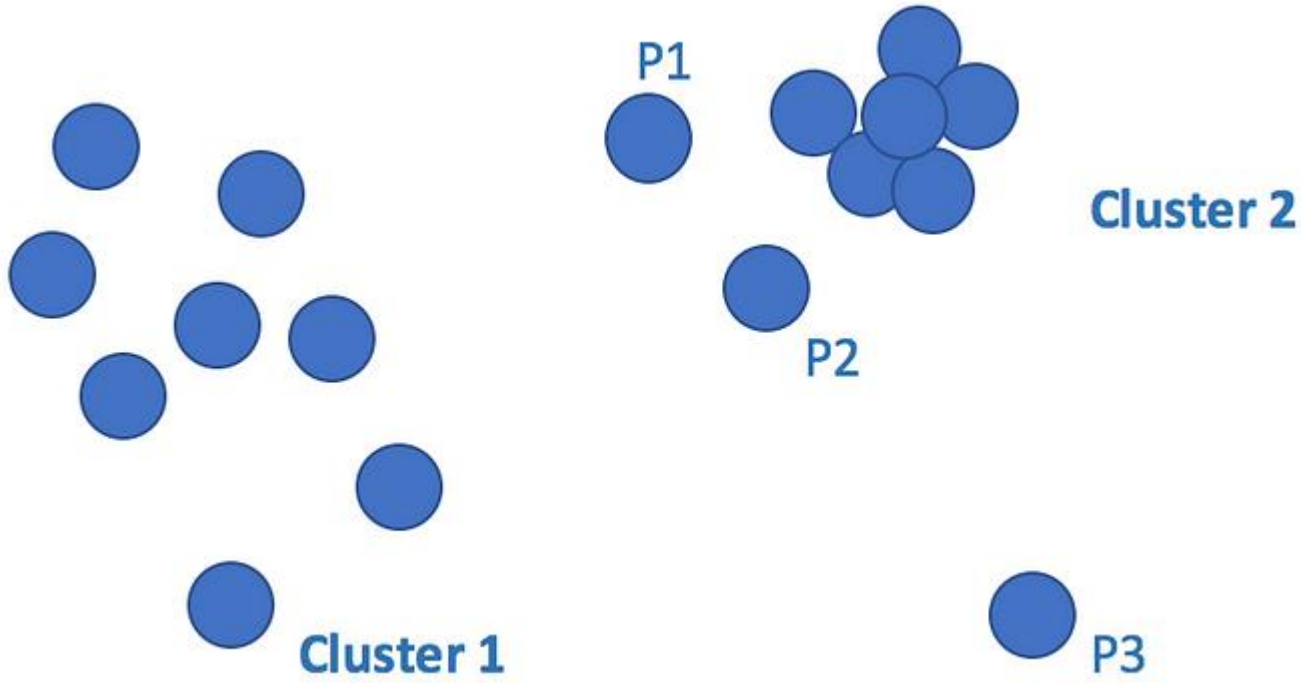
LOF uses density-based outlier detection to identify local outliers, points that are outliers with respect to their local neighborhood, rather than with respect to the global data distribution. The higher the LOF value for an observation, the more anomalous the observation.

This is useful because not all methods will not identify a point that's an outlier relative to a nearby cluster of points (a local outlier) if that whole region is not an outlying region in the global space of data points.

A point is labeled as an outlier if the density around that point is significantly different from the density around its neighbors.

In the below feature space, LOF is able to identify P1 and P2 as outliers, which are local outliers to Cluster 2 (in addition to P3).

**Example of local vs global outliers:**




Local vs Global Outliers

**The method, step-by-step:**

For each point P, do the following:

1. **Calculate distances** between P and every other point (manhattan = |x1-x2| + |y1-y2|) = dist(p1,p2)
2. **Find the Kth closest point** (Kth nearest neighbor's distance=K-Dist(P))

3. **Find the K closest points** (those whose distances are smaller than the Kth point), the K-distance neighborhood of P, Nk(P).
4. **Find its density** (Local Reachability Density= LRDk(p) — a measure of how close its neighbors are to it), basically the inverse of the avg distance between point p and its neighbors. The lower the density, the farther p is from its neighbors.
5. **Find its local outlier factor**, LOFk(p), as sum(reachability distances of neighbors to P) x sum(densities of neighbors). LOFk(P) is basically the sum of the distances between P and its neighboring points, weighted by the sum those points' densities (how far they are from their k neighboring points).

**Additional Details**

** For step 2, If 2 points have the same distance to P, then just select one as the next closest, and the other as the next next closest.

**For step 4, LRD = Local Reachability Density = inverse(avg reachability distance between P and its neighbors) <= 1. The word reachability is used because if a neighbor is closer to P than it's Kth neighbor, then the distance of the Kth neighbor is used instead as a means of smoothing

**For step 4, each reachability distance of a point P's k neighbors is reachdistk(n1<-p) = max(distk(n1), dist(n1,p))

**For step 4, total distances of neighboring points is divided by the number of neighboring points (or ||Nk(P)||), computed using the results of step 3

**Scenarios affecting LOF values:**

Higher LOF values indicate a greater anomaly level and that
LOFk(p) =
sum(reachability distances of its neighbors to P) x sum(neighbor densities)

The LOF for a point P will have a:

1. High value if → P is far from its neighbors and its neighbors have high densities (are close to their neighbors) (LOF = (high distance sum) x (high density sum) = High value)
2. Less high value if -> P is far from its neighbors, but its neighbors have low densities (LOF = (high sum) x (low sum) = middle value)
3. Less high value if -> P is close to its neighbors and its neighbors have low densities (LOF = (low sum) x (low sum) = low value )

**Adjusting K:**

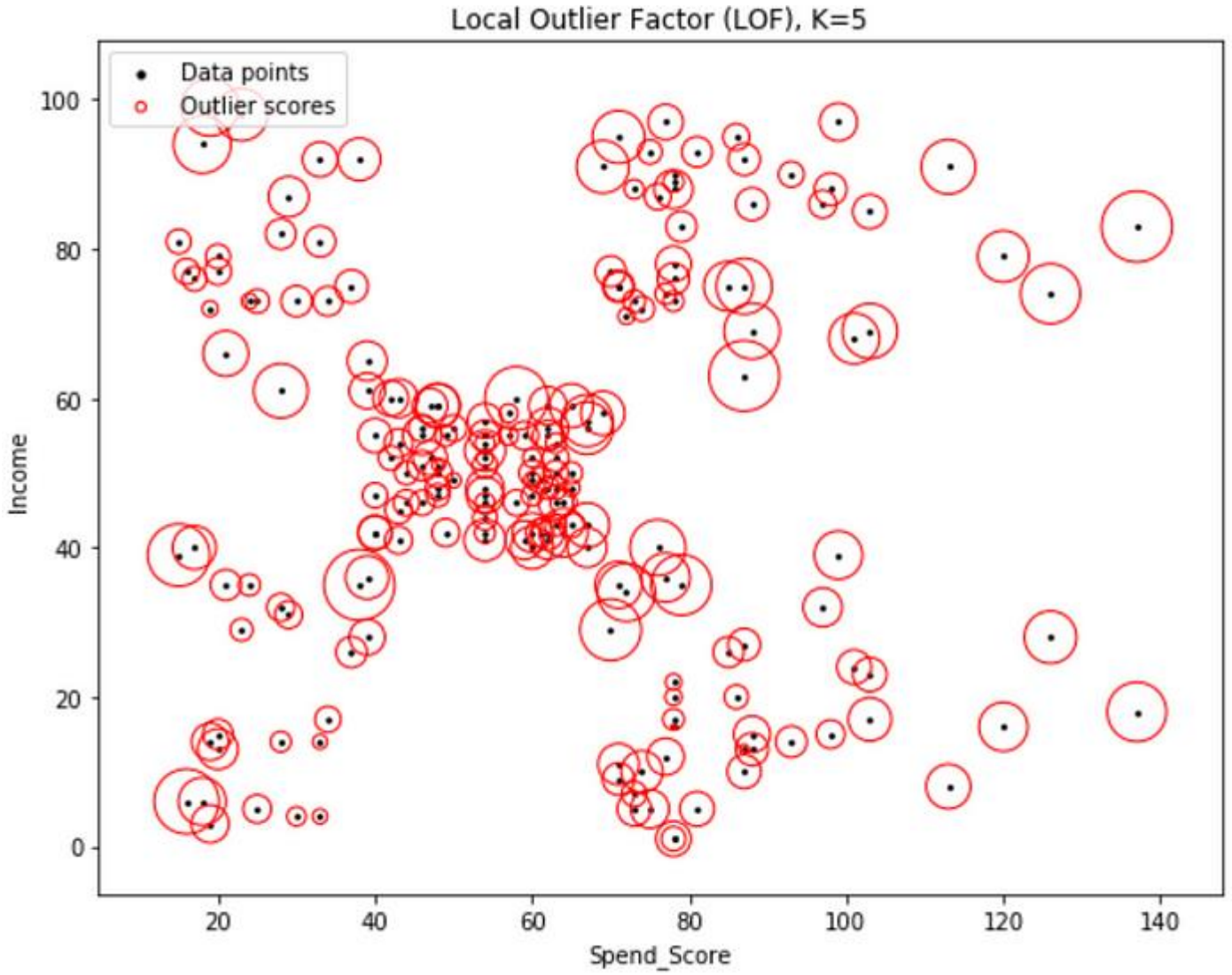
- Increase K too much and you're just looking for outliers with respect to the entire dataset, so points far away from the highest density regions could be misclassified as outliers, even though they themselves reside in a cluster of points.
- Reduce K too much and you're looking for outliers with respect to very small local regions of points. This could also lead to the misclassification as outliers.

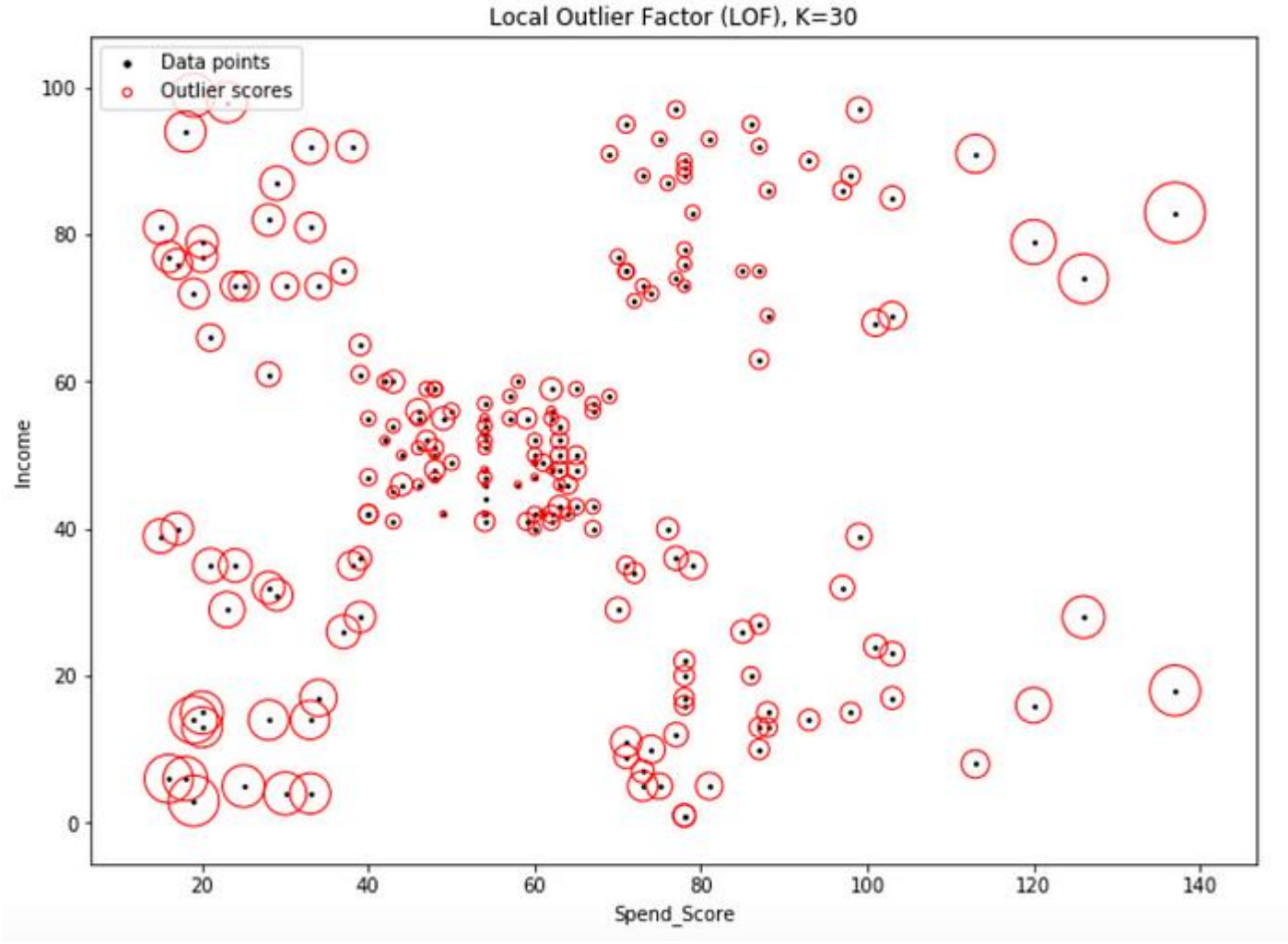
**Observing LOF Results with Varying K:**

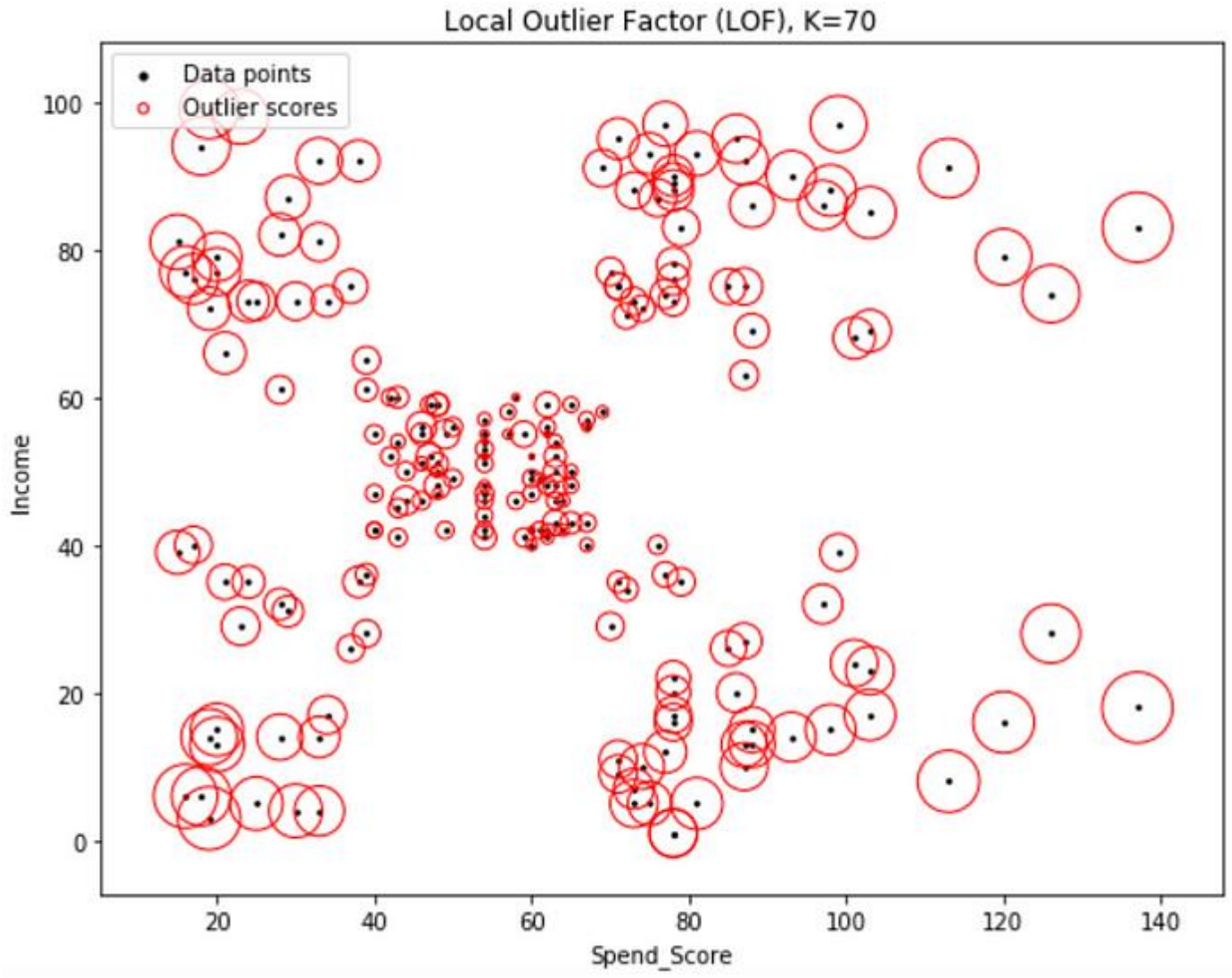
The below code plots LOF scores as red circles around points for each of K=5, 30, and 70. The larger the LOF, the greater the radius of the circle, and the more anomalous the observation.

```
def LOF_plot(k):
 import seaborn as sns
 from sklearn.neighbors import LocalOutlierFactor
 var1,var2=1,2
 clf = LocalOutlierFactor(n_neighbors=k, contamination=.1)
 y_pred = clf.fit_predict(df)
 LOF_Scores = clf.negative_outlier_factor_

 plt.title("Local Outlier Factor (LOF), K={}".format(k))
 plt.scatter(df.iloc[:, var1], df.iloc[:, var2], color='k',
s=3., label='Data points')
 radius = (LOF_Scores.max() — LOF_Scores) / (LOF_Scores.max() —
LOF_Scores.min())
 plt.scatter(df.iloc[:, var1], df.iloc[:, var2], s=1000 *
radius, edgecolors='r',
 facecolors='none', label='Outlier scores') plt.axis('tight')
 plt.ylabel("{}".format(df.columns[var1]))
 plt.xlabel("{}".format(df.columns[var2]))
 legend = plt.legend(loc='upper left')
 legend.legendHandles[0]._sizes = [10]
 legend.legendHandles[1]._sizes = [20]
 plt.show();LOF_plot(5)
LOF_plot(30)
LOF_plot(70)
```


Local Outlier Factor (LOF), K=5
Data points
Outlier scores
Income
Spend_Score
0
20
40
60
80
100
20
40
60
80
100
120
140


Local Outlier Factor (LOF), K=30
Data points
Outlier scores
Income
Spend_Score
0
20
40
60
80
100
20
40
60
80
100
120
140




From the above, observe how too small of a K results in too many points having high LOFs (circle radius does not fluctuate as much as one would think). Too high of a K results in those points in the four outer clusters having high LOFs because of being too far from the main cluster of points.

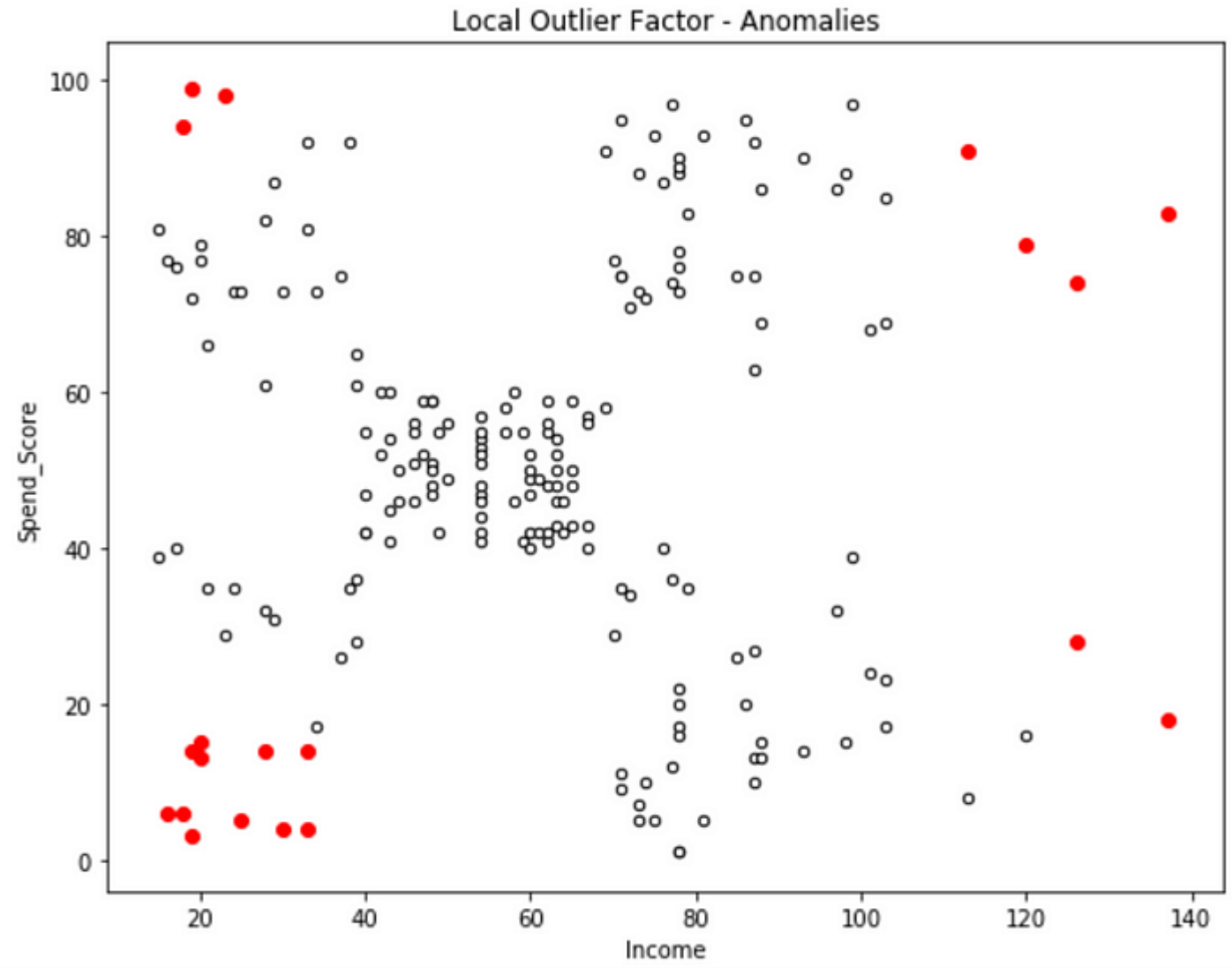
K=30 offers a balance of the two extremes. This value is selected in implementing the method below:

**Sklearn Implementation of Local Outlier Factor:**

```
from sklearn.neighbors import LocalOutlierFactor
clf = LocalOutlierFactor(n_neighbors=30, contamination=.1)
y_pred = clf.fit_predict(df)
LOF_Scores = clf.negative_outlier_factor_
LOF_pred=pd.Series(y_pred).replace([-1,1],[1,0])
LOF_anomalies=df[LOF_pred==1]
```

Observations predicted as anomalies have values of -1 in *clf.fit_predict()*. These observations have LOF scores less than the threshold (*clf.negative_outlier_factor_ < clf.threshold_*). Sklearn outputs negative LOF values.

```
cmap=np.array(['white','red'])
plt.scatter(num.iloc[:,1],num.iloc[:,2],c='white',s=20,edgecolor
='k')
plt.scatter(LOF_anomalies.iloc[:,1],LOF_anomalies.iloc[:,2],c='r
ed')
 #,marker='x',s=100)
plt.title('Local Outlier Factor – Anomalies')
plt.xlabel('Income')
plt.ylabel('Spend_Score')
```




LOF Outliers

**More Information on LOF:**

**Textbooks**

1. Hands-On Machine Learning with scikit-learn and Scientific Python Toolkits (Released 7/24/2020)
2. Outlier Analysis 2nd ed. 2017 Edition
3. Anomaly Detection Principles and Algorithms 2017 Edition
4. Outlier Detection: Techniques and Applications 1st Ed. 2019 Edition

### Web Links

1. http://www.cse.ust.hk/~leichen/courses/comp5331/lectures/LOF_Example.pdf
2. https://towardsdatascience.com/local-outlier-factor-for-anomaly-detection-cc0c770d2ebe
3. https://medium.com/@mtngt/local-outlier-factor-example-by-hand-b57cedb10bd1
4. https://medium.com/@mtngt/local-outlier-factor-simple-python-example-8925dad97fe6
5. https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html#sphx-glr-auto-examples-neighbors-plot-lof-outlier-detection-py

## Elliptic Envelope

The Elliptic Envelope method fits a multivariate gaussian distribution to the dataset. Use the *contamination* hyperparameter to specify the percentage of observations the algorithm will assign as outliers.

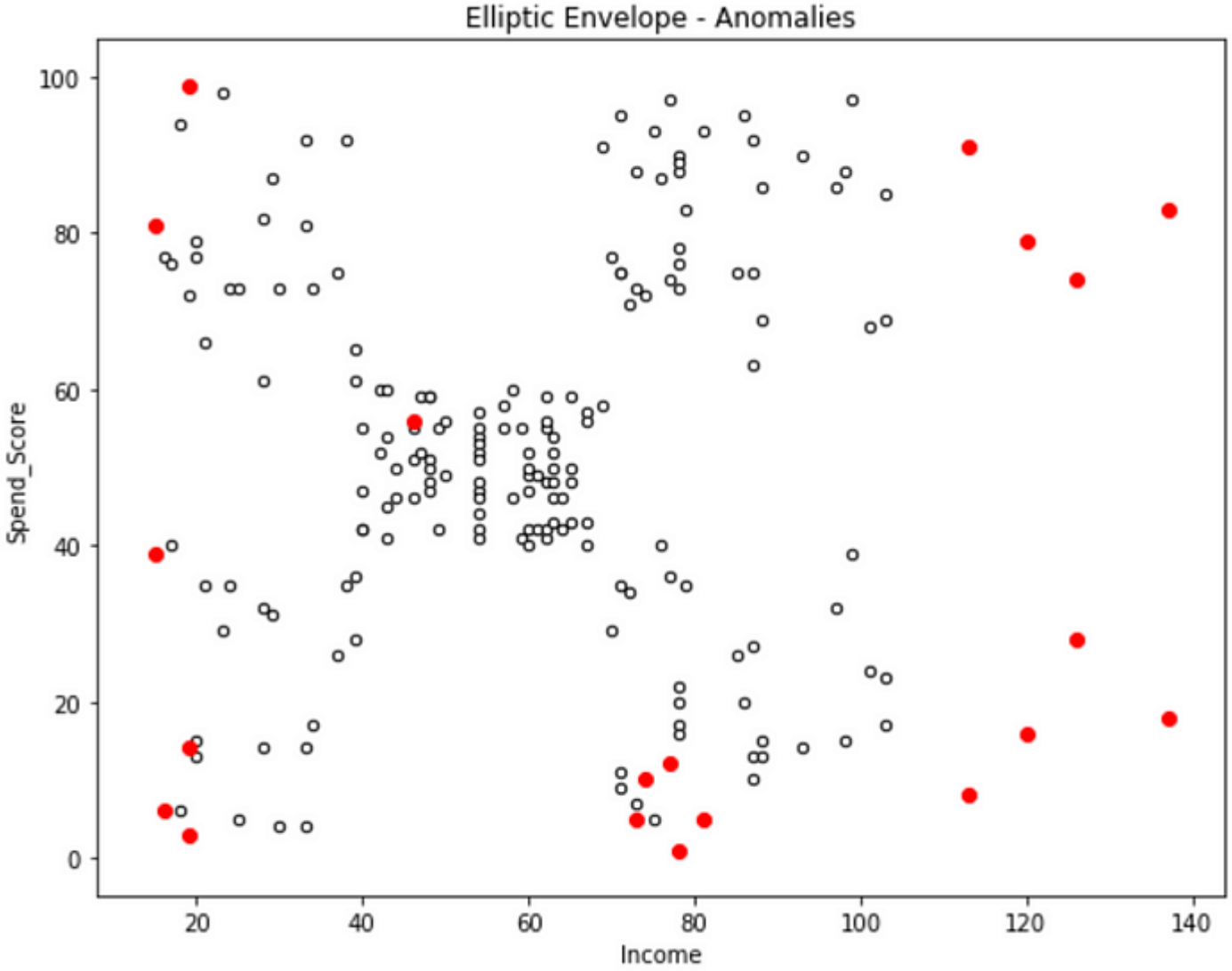
### Sklearn Implementation of Elliptic Envelope:

```
from sklearn.covariance import EllipticEnvelope
clf = EllipticEnvelope(contamination=.1,random_state=0)
clf.fit(df)
ee_scores = pd.Series(clf.decision_function(df))
ee_predict = clf.predict(df)
```

*ee_scores* contains fitted densities.

*ee_predict* contains labels, where -1 indicates an outlier and 1 does not. Labels are calculated based on *clf.threshold_* and *ee_scores*.

```
cmap=np.array(['white','red'])
plt.scatter(num.iloc[:,1],num.iloc[:,2],c='white',s=20,edgecolor
='k')
plt.scatter(ee_anomalies.iloc[:,1],ee_anomalies.iloc[:,2],c='red
')#,marker='x',s=100)
plt.title('Elliptic Envelope - Anomalies')
plt.xlabel('Income')
plt.ylabel('Spend_Score')
```




Elliptic Envelope Outliers

## More information on Elliptic Envelope:

### Text books

1. Hands-On Machine Learning with scikit-learn and Scientific Python Toolkits (Released 7/24/2020)

### Web Links

1. http://sdsawtelle.github.io/blog/output/week9-anomaly-andrew-ng-machine-learning-with-python.htm
2. https://chrisalbon.com/machine_learning/preprocessing_structured_data/detecting_outliers/
3. https://scikit-learn.org/stable/modules/generated/sklearn.covariance.EllipticEnvelope.html

## One-Class Support Vector Machines

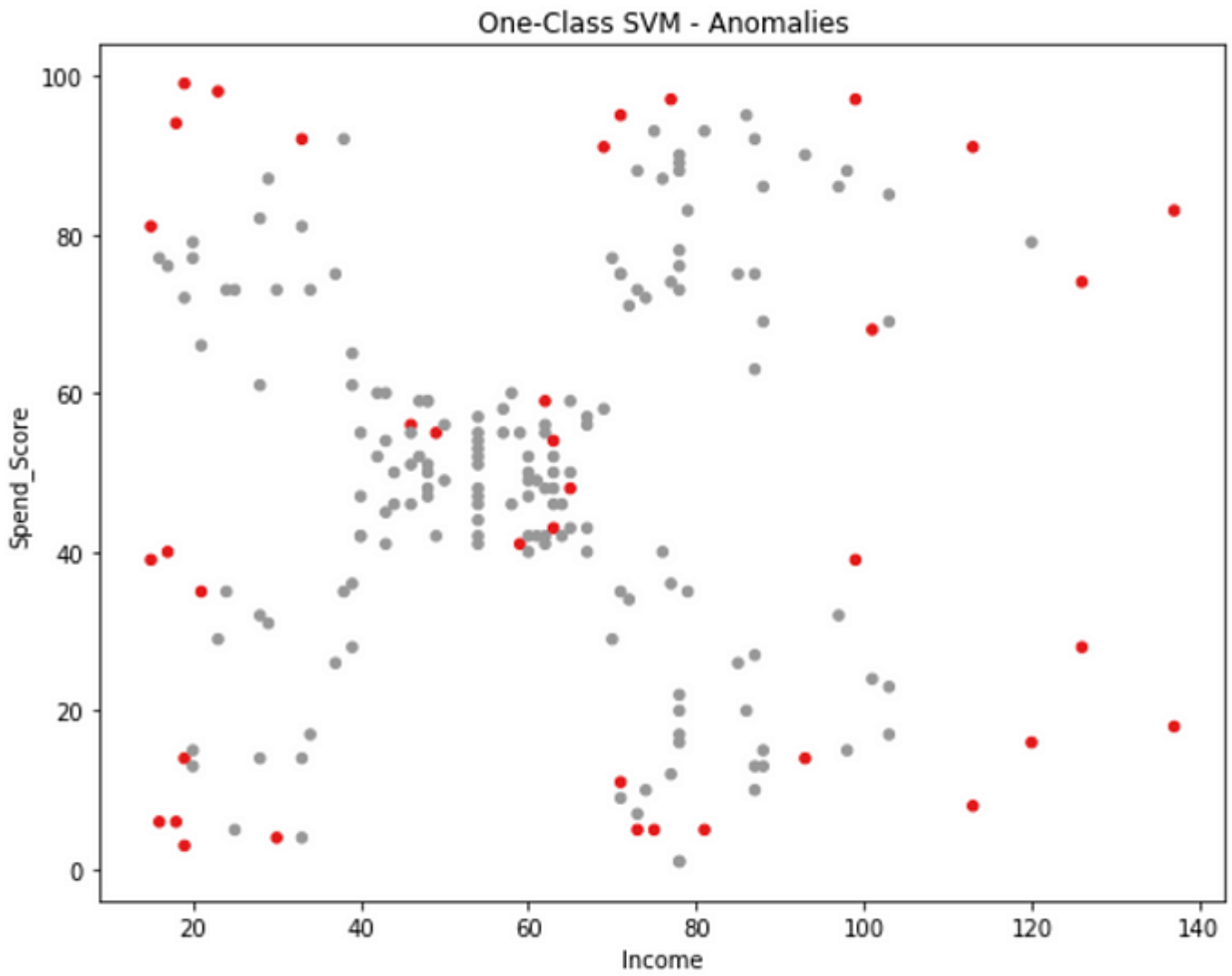
I have not fully looked into this method, but here is a basic implementation. The *nu* hyperparameter seems to be like the contamination hyperparameter in other methods. It sets the % of observations the algorithm will identify as outliers.

### Sklearn Implementation of One-Class SVM:

```
from sklearn import svm
clf=svm.OneClassSVM(nu=.2,kernel='rbf',gamma=.001)
clf.fit(df)
y_pred=clf.predict(df)
```

Below, I plot observations identified as anomalies:

```
num.plot.scatter(x='Income',y='Spend_Score', c=y_pred,
cmap=cmap,
 colorbar = False)
plt.title('One-Class SVM – Anomalies');
```




One-Class SVM Outliers

## More Information on One-Class SVM:

### Textbooks

I. Beginning Anomaly Detection Using Python-Based Deep Learning: With Keras and PyTorch 1st ed. 2019

2. Outlier Analysis 2nd ed. 2017 Edition

### Web Links

1. http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/
2. https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

**Using K-Means Clustering for Anomaly Detection:**

http://amid.fish/anomaly-detection-with-k-means-clustering
https://towardsdatascience.com/time-series-of-price-anomaly-detection-13586cd5ff46