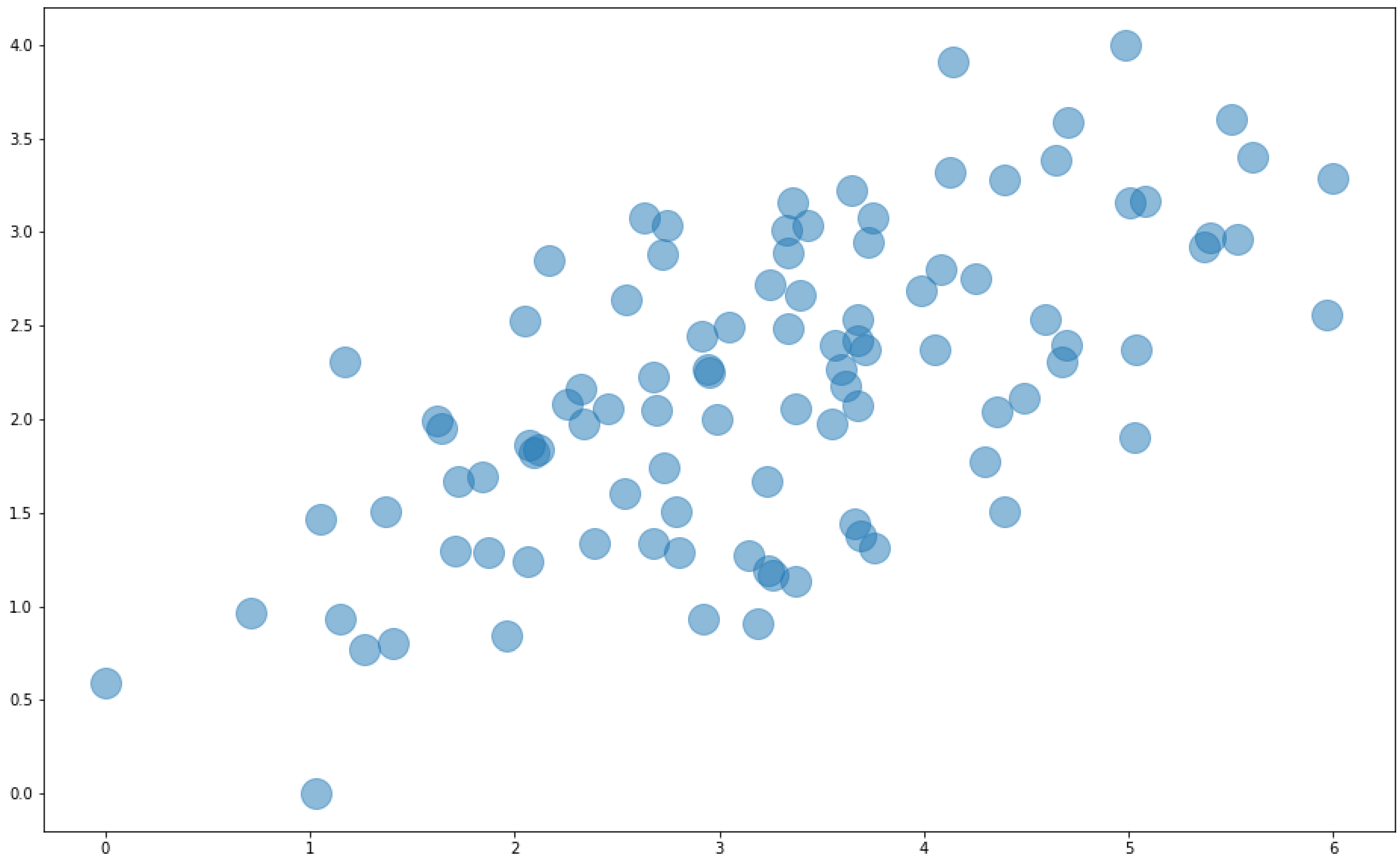
atmosera®

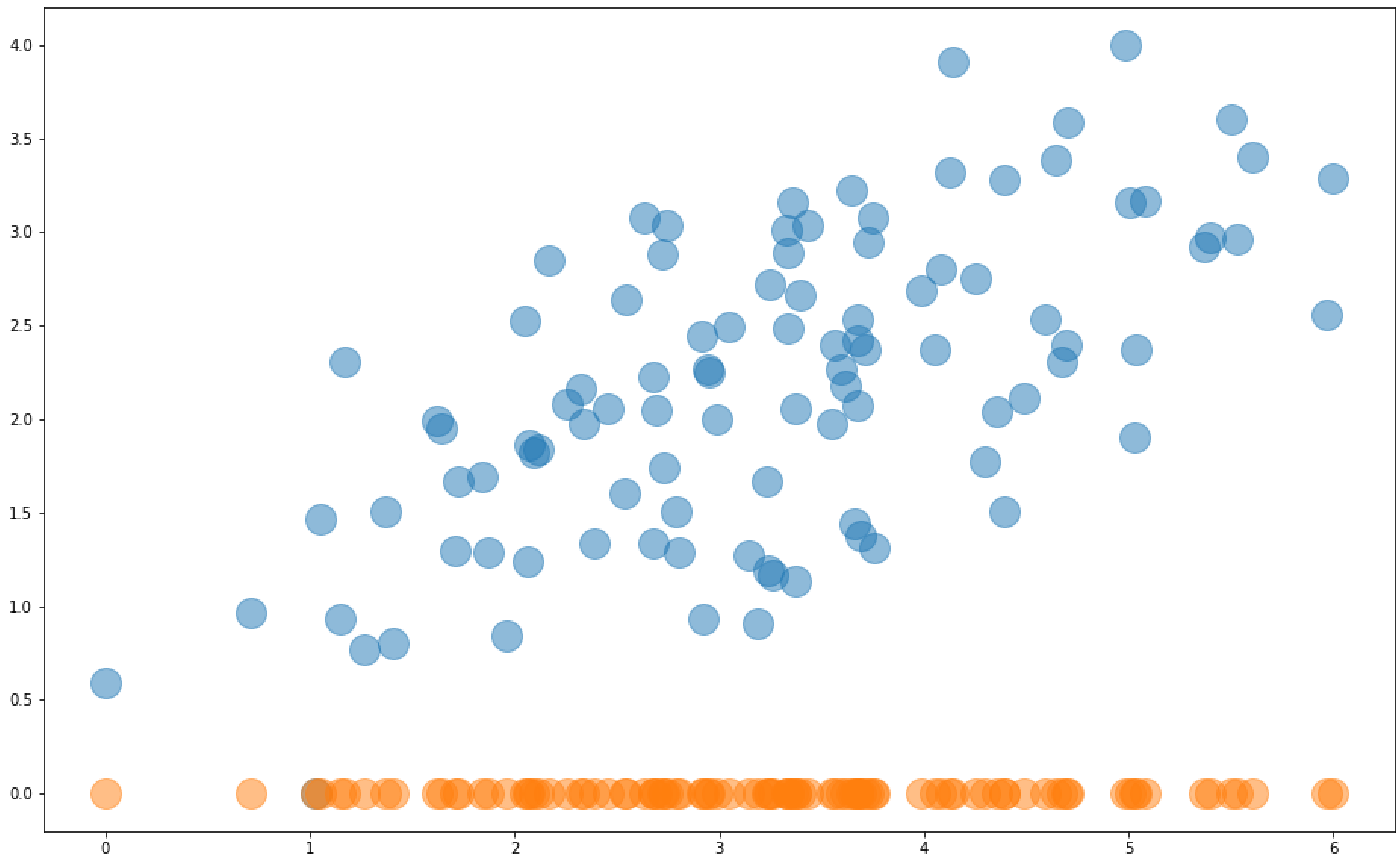
Gold Microsoft Partner Microsoft | Azure Expert MSP

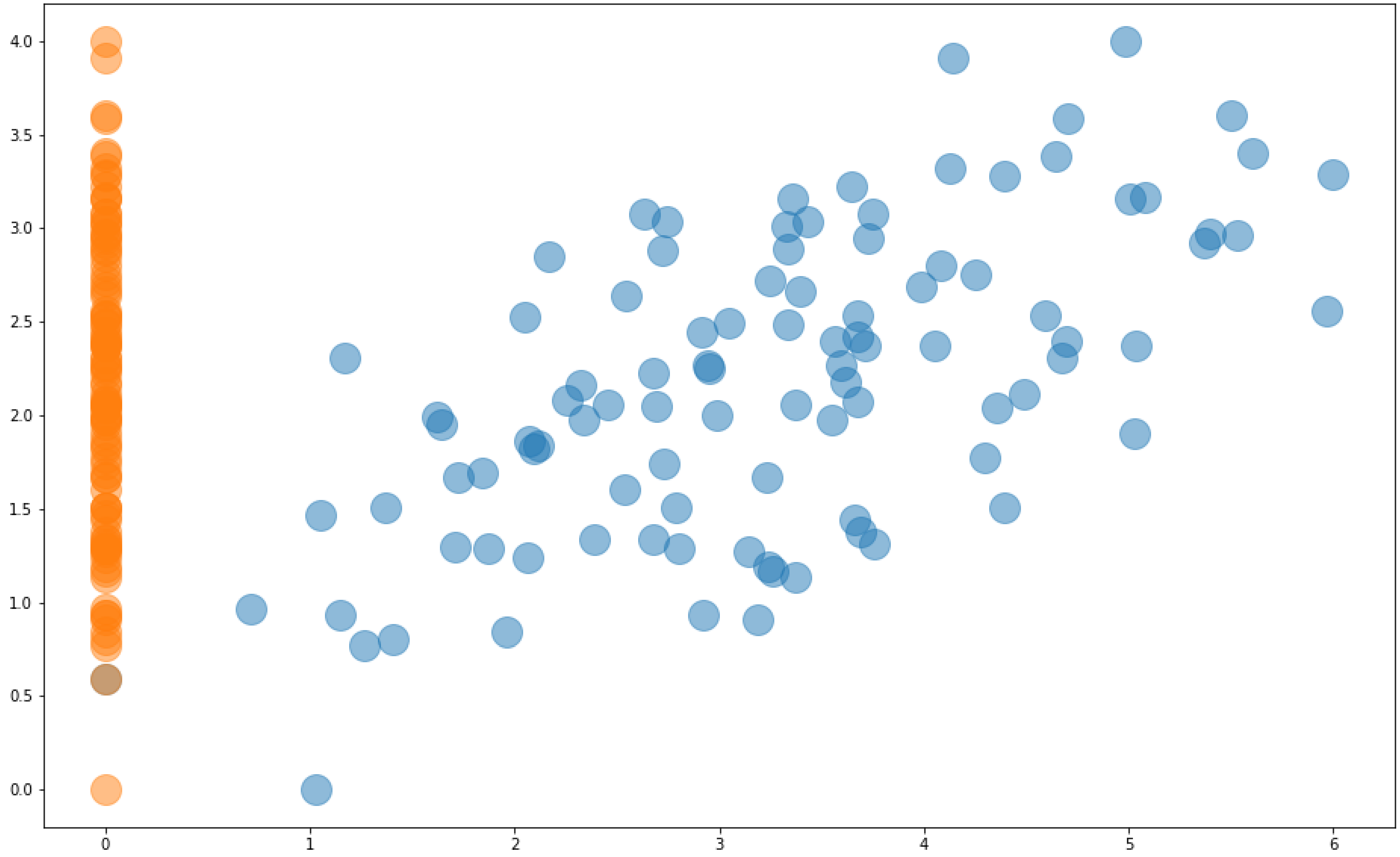
PRINCIPAL COMPONENT ANALYSIS

Jeff Prosise

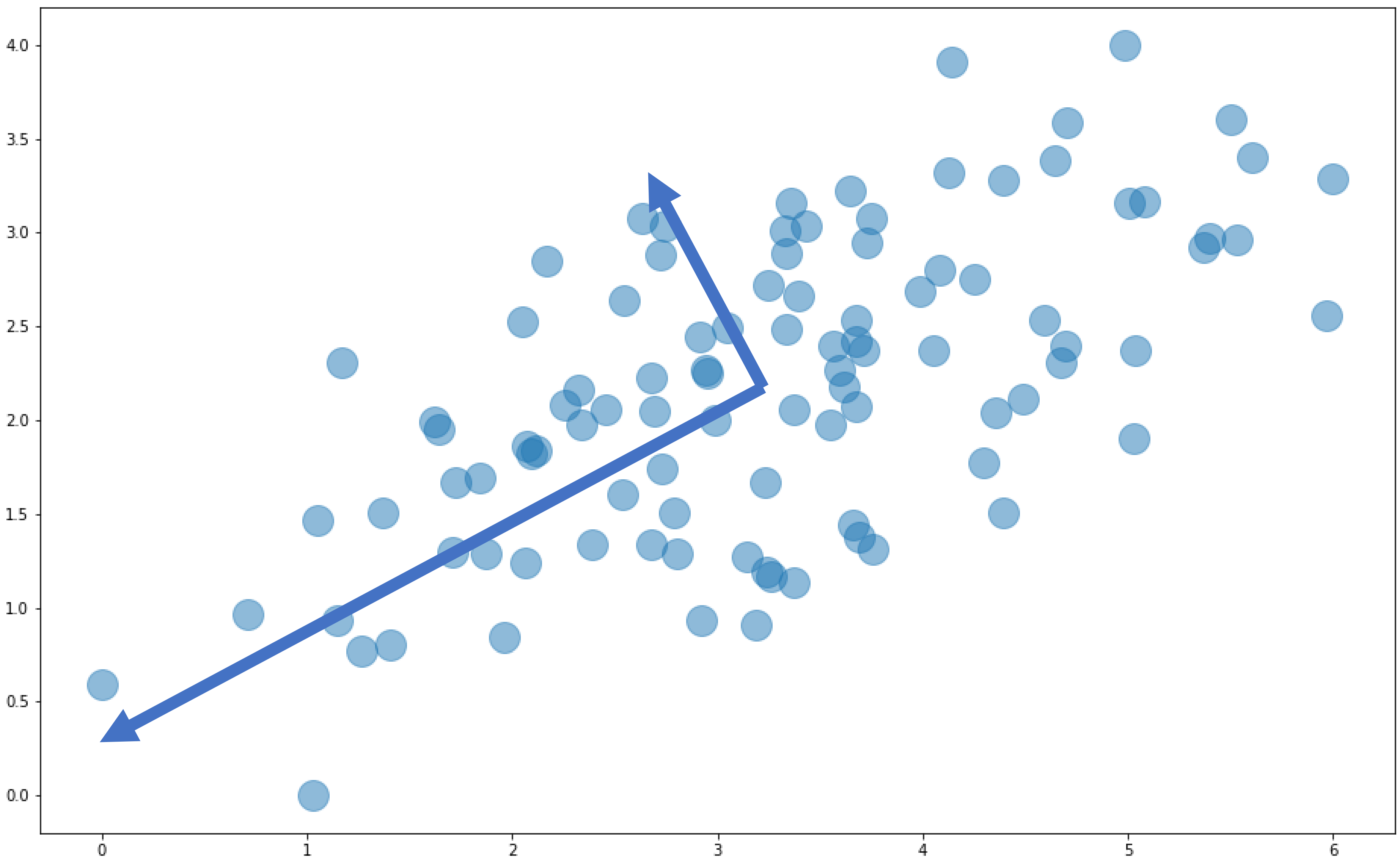


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6

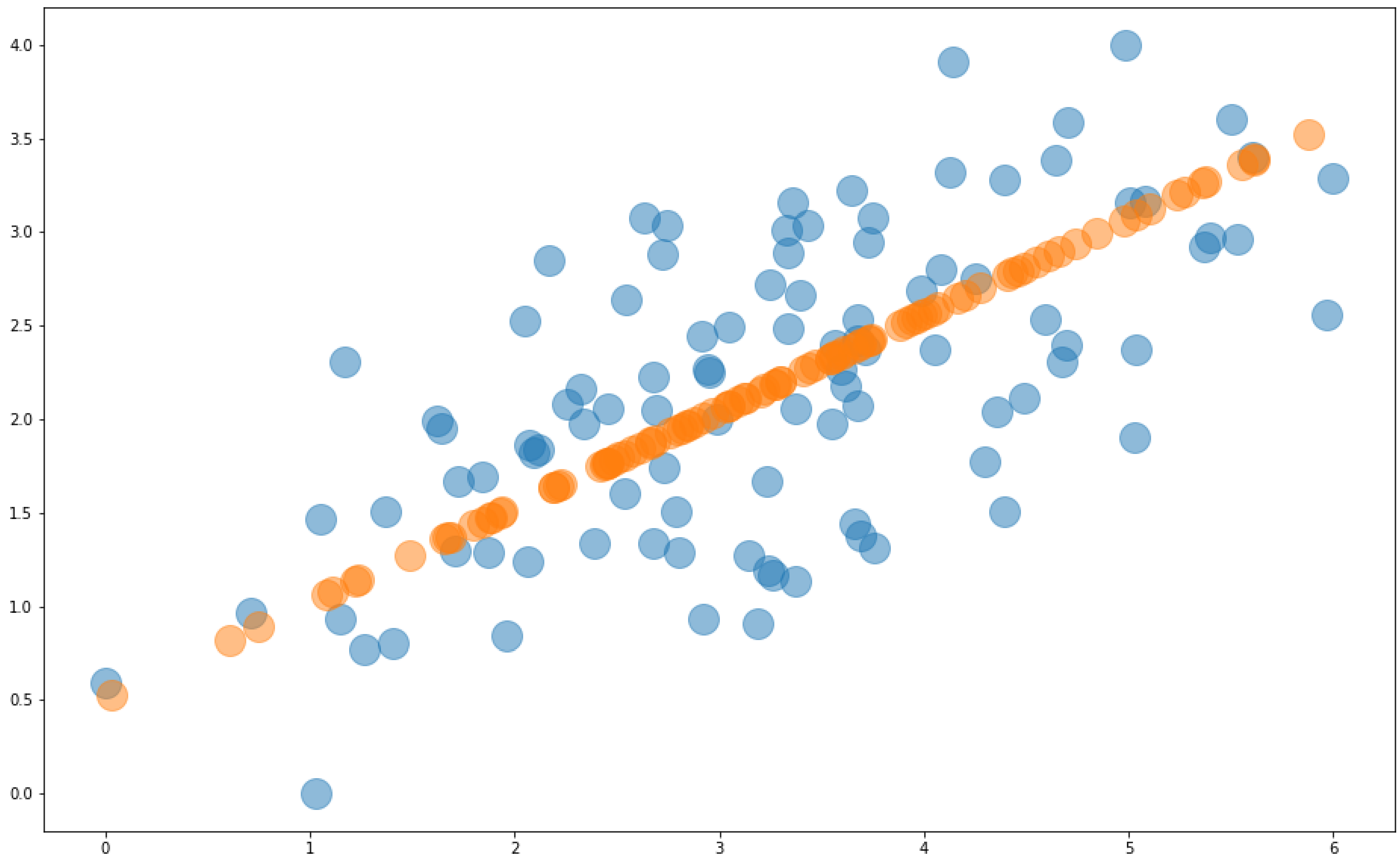

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6



4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6



4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6

Principal Component Analysis (PCA)

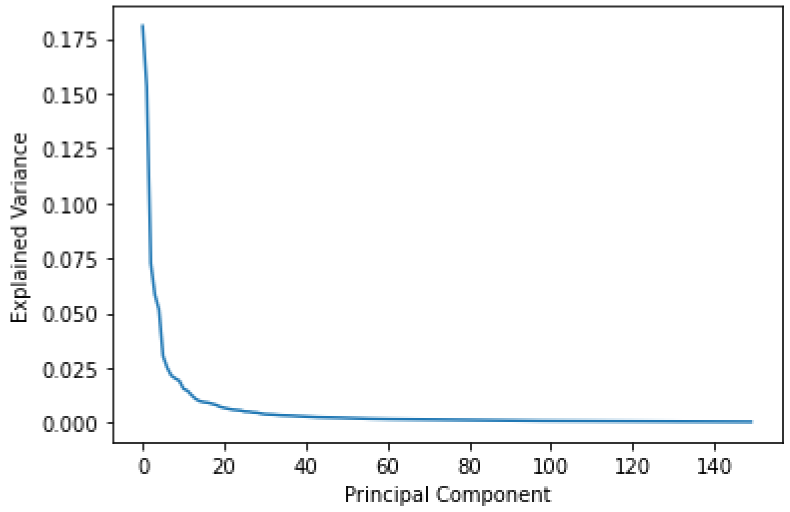
- Commonly used dimensionality-reduction algorithm
  - Reduces number of dimensions without commensurate loss of information
    - Example: Reduce number of dimensions by 90% while retaining 90% of the information
  - Works best with dense data (fewer zeroes); use other algorithms such as Singular Value Decomposition (SVD) for sparse datasets
- Applications include increasing samples/dimensions ratio for small datasets, obfuscating data, filtering noise, eliminating multicollinearity, eliminating irrelevant features, reducing data to 2 or 3 dimensions for plotting and visualization, and anomaly detection
- Scale of all dimensions should be the same before applying PCA

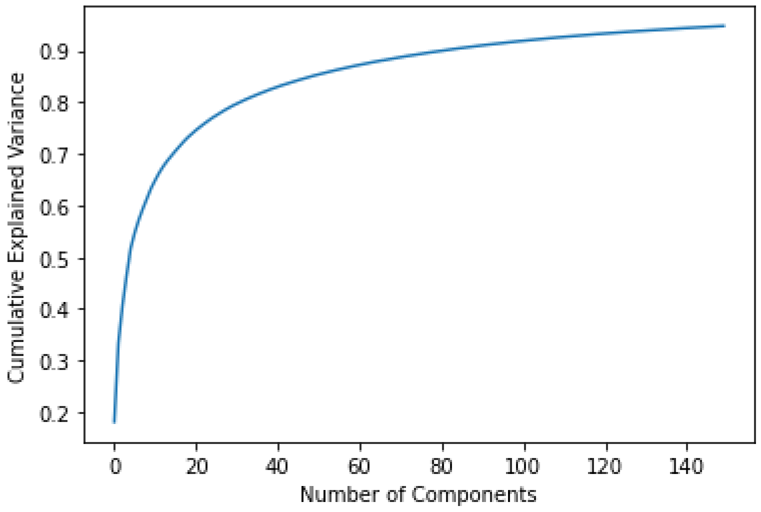
Applying a PCA Transform

```
from sklearn.decomposition import PCA

# Reduce m dimensions to 5
pca = PCA(n_components=5)
pca_data = pca.fit_transform(data)

# Reduce m dimensions to n while retaining 80% of the information
pca = PCA(0.8)
pca_data = pca.fit_transform(data)
n_components = pca.n_components_ # Get the number of components (n)
```

Picking the "Right" Number of Dimensions

- After fitting, **PCA.explained_variance_ratio_** reveals percentage of information contained in each component
- Plot explained variance ratios to find optimum balance between variance and number of components





Demo

Principal Component Analysis

atmosera®

Gold Microsoft Partner Microsoft | Azure Expert MSP

Inverting a PCA Transform

```
# Reduce m dimensions to 5
pca = PCA(n_components=5)
pca_data = pca.fit_transform(data)

# Restore the data to m dimensions
unpca_data = pca.inverse_transform(pca_data)
```

Noise Filtering

- Use PCA to reduce and then restore the number of dimensions
- Least valuable information is discarded, and by definition, noise has little informational value

```
# Reduce m dimensions to 10
pca = PCA(n_components=10)
pca_data = pca.fit_transform(data)

# Restore the data to m dimensions
unpca_data = pca.inverse_transform(pca_data)
```

Anonymizing Data

- Use PCA to "reduce" *m*-dimensional data to *m* dimensions to obfuscate/anonymize the numbers without losing information
- Great for sharing datasets without revealing IP or PII inside them

```
pca = PCA(n_components=30, random_state=0)
pca_data = pca.fit_transform(df)

scaler = StandardScaler()
anon_df = pd.DataFrame(scaler.fit_transform(pca_data))
```

Demo

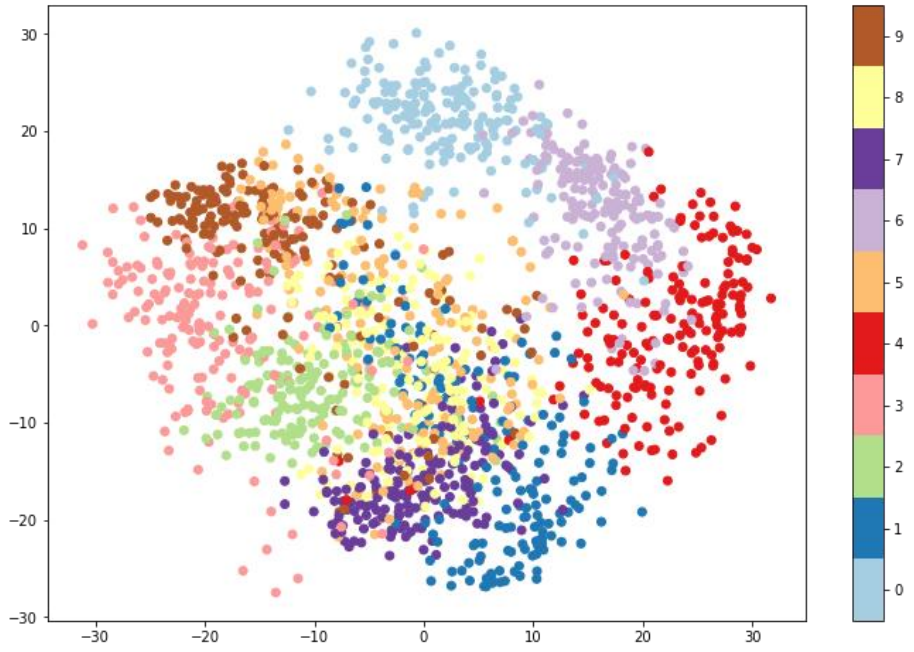
Noise Filtering and Data Anonymization

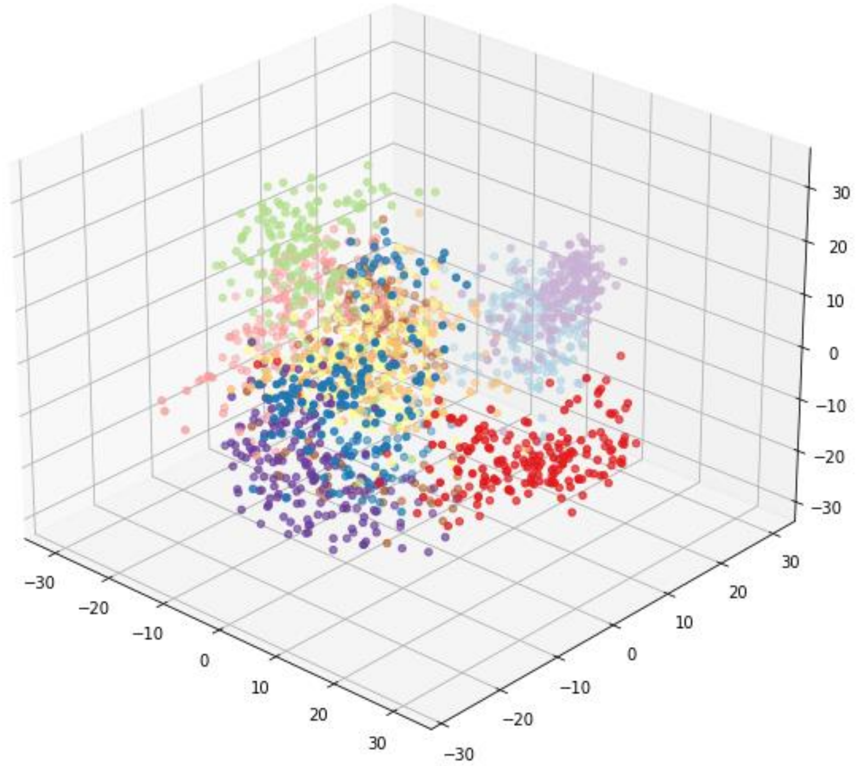
atmosera®

Gold Microsoft Partner Microsoft | Azure Expert MSP

Visualizing High-Dimensional Data

- Humans can't visualize data in more than three dimensions
- Solution: "Squeeze" data down to two or three dimensions and plot it





Plotting in 2D with PCA

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Reduce m dimensions to 2
pca = PCA(n_components=2)
pca_data = pca.fit_transform(data)

# Draw a scatter plot
plt.scatter(pca_data[:, 0], pca_data[:, 1], c=target, cmap='Paired')
```

Color each point based on its target class

t-Distributed Stochastic Neighbor Embedding

- Abbreviated t-SNE and implemented in Scikit's **TSNE** class
- Dimensionality-reduction algorithm that is particularly well suited for 2D and 3D visualizations of high-dimensional data
  - Uses non-linear reduction technique where focus is on keeping similar data points together in low-dimensional space
    - PCA uses linear reduction where focus is on keeping dissimilar points far apart
  - Not impacted by outliers in data (unlike PCA)
- Computationally expensive and impractical to use on large datasets
  - If necessary, use a subset of rows or use PCA to reduce the number of columns before applying t-SNE

Plotting in 2D with t-SNE

```python
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Reduce m dimensions to 2
tsne = TSNE(n_components=2)
tsne_data = tsne.fit_transform(data)

# Draw a scatter plot
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=target, cmap='Paired')
```

Demo

Visualizing High-Dimensional Data

Anomaly Detection

- Anomaly-detection models detect anomalous points in datasets
- PCA-based anomaly detection relies on reconstruction error incurred when a transform is applied and inverted to identify anomalous points
  - Use PCA to reduce a dataset with m dimensions to n dimensions
  - Invert the transform to reconstruct the original dataset minus losses
  - Select a baseline loss from "normal" data and classify points that incur more reconstruction loss as anomalous
  - Assumption: Anomalous points are likely to exhibit more loss
- Learning is unsupervised (labels not required)

Quantifying Loss Due to PCA Transforms

```
loss = np.sum((np.array(df_original) - np.array(df_restored)) ** 2, axis=1)
loss = pd.Series(data=loss, index=df_original.index)
```

atmosera®

Gold Microsoft Partner Microsoft | Azure Expert MSP

Demo

PCA-Based Anomaly Detection

Summary

- PCA reduces dimensions while preserving most of the information
  - Increase ratio of rows to columns (strive for minimum of 5:1)
  - Remove irrelevant features (feature selection)
  - Reduce noise (apply PCA transform and then invert transform)
  - Visualize high-dimensional data in 2D and 3D
  - Obfuscate datasets containing sensitive or proprietary information
  - Detect anomalous data points in datasets and data streams
- Data subjected to PCA should be normalized if scale of values varies
- Scikit's **PCA** class makes Principal Component Analysis easy