NOOBS Project Report **MULTIVARIATE STATISTICS**

21BDA61 Zeenat Zahoor

21BDA43 M Hisham T

21BDA17 Alen Roy

PCA ... Only About Dimensionality Reduction?

Why we chose this project, what was the proposed scope?

Mr Srinavas sir suggested this topic to us. At first it seemed difficult as the only thing that we knew about PCA was reduction of data. We had no compulsion from his side to choose this particular topic but one day he suggested a video too to get the idea for what we should do. After watching that video it was really clear to us what exactly we wanted to do. In fact, by watching that video we came across so many things that we didn't know before. We were really very happy because when we planned what we should do, it made good sense and we knew it was going to be stepwise procedure and we will overall do it with ease.

So basically when we watched the YouTube video that sir send us, it amazed us! Before that no one among in our group members had idea what possible things we can do with PCA other than Dimensionality Reduction. PCA is an unsupervised Machine Learning Technique.We studied PCA in this semester for 2 subjects but the main focus of those courses was only the dimensionality reduction. I feel like if we didn't do this project we would be not able to discover all this beauty of PCA and we won't be able to convey it to others too. So the main focus of our project

is so explain everyone that PCA is not all about Dimensionality Reduction but much more! In total we took 4 applications of PCA i.e:

1. Dimensionality Reduction: Dimensionality-reduction is a technique that reduces the number of dimensions in a dataset without giving up a lot of information.
2. Obfuscating Data: Obfuscating Data in literal sense is hiding the data. The confidential data that some organizations don't want to share can be hidden using PCA. These organizations can be banks, defense, etc.
3. Noise Reduction: Noise is the extra information in the data which makes it hard to extract inferences from the data. PCA helps to reduce that.
4. Anomaly Detection: Anomaly detection is the identification of rare events, items, or observations which are suspicious because they differ significantly from standard behaviors or patterns. Anomalies in data are also called standard deviations, outliers, noise, novelties, and exceptions.

For each application, we took different data sets. The objective of our project is to explain each application of PCA in a most simple possible way. I hope that whatever we are trying to convey will reach to everyone and everyone will be as amazed as we were at first when we came to know.

How did we obtain or create the data?

We didn’t create the data, we used the already available Data Sets from different online sources like Kaggle and Sk Learn. Because we needed 4 Data sets for 4 different applications we had a little difficulty to find the Data sets.

The 4 Datasets information:

1. Dimensionality Reduction: For this we, took a Data set called Labeled Faces in the Wild. This data set contains labeled pictures of famous people. Loading an image data is difficult because the files are heavy. This was a big file, so we loaded images of people for which there were at least 100 faces per person. we found 5 famous people data like that. Then we took the first 24 images. Each image was 62x47 pixels which equals to 2914 columns which is approximately 3000-Dimensional data.

2. Obfuscating Data: We took a Data set from Kaggle named Credit Card Fraud detection. It included Transactions made by European cardholders using credit cards. There were 31 columns and 284808 rows in data set. In 31 columns only 3 were labeled i.e The "Time" column reveals the number of seconds elapsed since the first transaction. "Class" tells us whether the transaction is legitimate (0) or fraudulent (1), and the "Amount" column shows the amount of the transaction. The remaining columns "V1" through "V28" were generated using PCA from information that isn't revealed to us. It most likely includes information about what was purchased, where it was purchased, and who purchased it. The latter might include information such as how long the member has been a cardholder, their credit score, their spending habits, their age, and their annual income.

3. Noise Reduction: We took the Scanned Handwritten Digits Data set. The data was already normalized to make all pixel values in the range (0, 1). Again here we had to load the data so we took the first 100 images out of the data set.

4. Anomaly Detection: We took a Data set named New York Taxi with two columns Time and Values. It was clearer to go with two columns only to see the behavior of Moving Averages and Anomalies.

Which methodology, approach or technique did we use, and why?

Application vise methodology:

1. Dimensionality Reduction: For this application we took a facial image data set and first we performed PCA to reduce 3000-Dimensional data to 150-Dimensional Data. Then we UnPCA it or we can say we performed PCA inverse transform to calculate how much total information was lost.
2. Obfuscating Data: We did the Fraud detection because unfortunately, we could not make predictions with this model because we didn't know the meaning of the numbers in the "V1" through "V28" columns, and we could not generate them because we didn't have the PCA transform applied to the original dataset. Nor do we know what the original dataset looked like. However, we have proved the principle that given the right features, we can build a PCA-based model that is reasonably accurate at detecting credit-card fraud.
3. Noise Reduction: After loading the data we split our data into 80:20 ratio i.e 80% data for training and 20% for testing. First we added the noise to the Data and then we reduced the noise using PCA and Kernal PCA (an extension of PCA) and we saw the difference between two. Which will be seen later in this report.

4. Anomaly Detection: After importing the NY Taxi data we first Saw the behavior of the moving Averages and the we took the value vs time plot and saw the behavior of the anomaly. The inferences will be explained later using the graph screenshots to this document.

What went right, what didn't go right?

The fact that we wanted to keep our project simple is what went right because we took data sets but we tried to keep it more and more simple so that each one watching our presentation will get clear idea what's happening in the project with all the datasets.

The thing that didn't go right was that we couldn't add the exploratory data analysis because we didn't have enough time. We just focused on the thing that we need to explain the above mentioned 4 aplications.

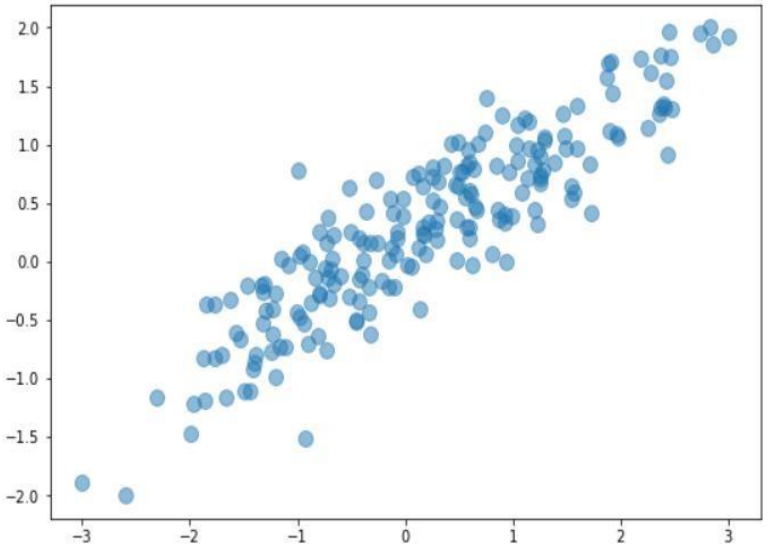
Evidence of activity: show sample code, sample dashboard, sample output

1. Dimensionality Reduction:

To understand what PCA is and how it works, begin by using it to reduce a 2-dimensional dataset to one dimension. The first step is to generate a dataset consisting of *x* and *y* co-ordinate pairs that roughly form a line. Then use PCA to find two principal components in the dataset. After that, plot the two principal components. The primary component is the axis that contains most of the information. The secondary component is the axis that contains the remaining information. Because the data points roughly form a line, the primary component lies along that line.

In [1]:
```
import numpy as np
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
%matplotlib inline

x, y = make_regression(n_samples=200, n_features=1, noise=50, random_state=0)
x = np.interp(x, (x.min(), x.max()), (-3, 3))
y = np.interp(y, (y.min(), y.max()), (-2, 2))
xy = np.column_stack((x, y))

plt.figure(figsize=(9, 6))
plt.scatter(x, y, s=100, alpha=0.5)
```

Out[1]: <matplotlib.collections.PathCollection at 0x27f44001af0>

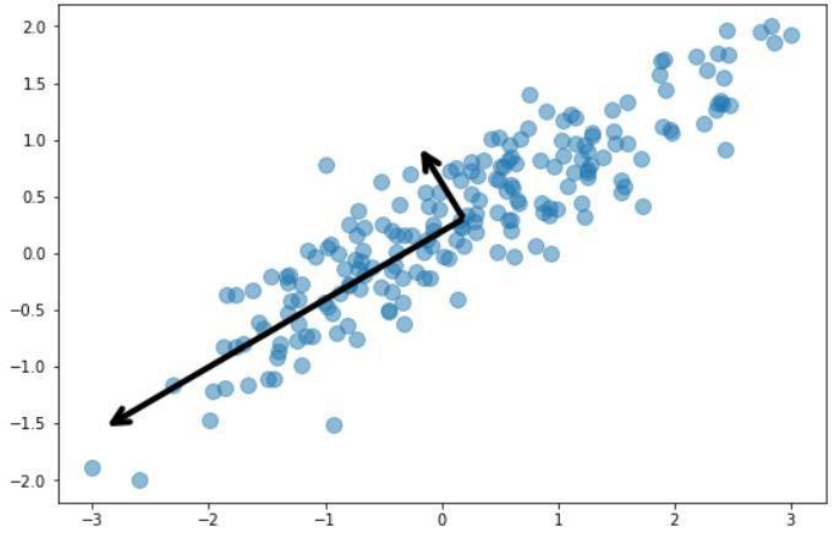


Use the PCA class's explained_variance_ratio_ attribute to quantify the amount of information contained in each component.

In [2]:
```
#Use PCA to find 2 Principal Components
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(xy)
```

Out[2]:
```
PCA(n_components=2)
```

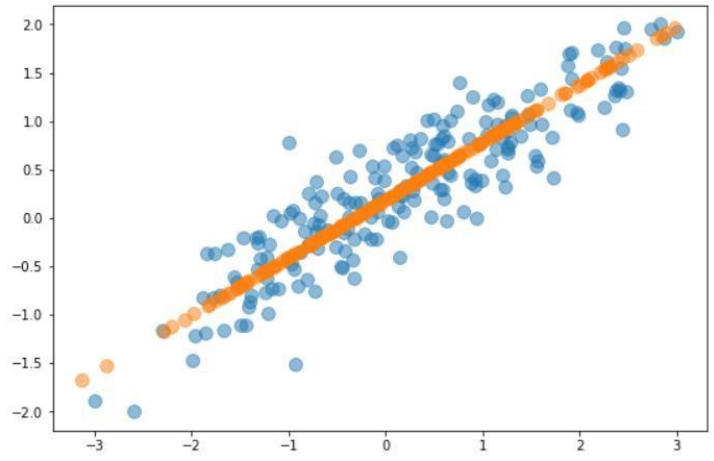
In [3]:
```
#Plot the PCs
def draw_vector(v0, v1):
    ax = plt.gca()
    props = dict(arrowstyle='->', linewidth=4, mutation_scale=25)
    ax.annotate('', v1, v0, arrowprops=props)

plt.figure(figsize=(9, 6))
plt.scatter(x, y, s=100, alpha=0.5)

for len, vector in zip(pca.explained_variance_, pca.components_):
    draw_vector(pca.mean_, pca.mean_ + (vector * 2.5 * np.sqrt(len)))
```



More than 95% of the information in the dataset is contained in one axis (the primary component). Use PCA to reduce the dataset to one principal component.

Now project all the data points onto the principal component axis by inverting the PCA transform, restoring the transformed data to two dimensions with information from just the first dimension. The original points are shown in blue, while the projected points are shown in orange. We can see that the orange points retain most of the information contained in the blue points, even though the size of the dataset has been cut in half (from two dimensions to one). That's the gist of PCA: reducing the number of dimensions without incurring a commensurate loss of information.

```
In [4]: pca.explained_variance_ratio_

Out[4]: array([0.95860561, 0.04139439])

In [5]: pca = PCA(n_components=1)
        pca_data = pca.fit_transform(xy)

In [6]: unpca_data = pca.inverse_transform(pca_data)

        plt.figure(figsize=(9, 6))
        plt.scatter(x, y, s=100, alpha=0.5)
        plt.scatter(unpca_data[:, 0], unpca_data[:, 1], s=100, alpha=0.5)

Out[6]: <matplotlib.collections.PathCollection at 0x27f46547370>
```



Taking the Image dataset

DIMENSIONALITY REDUCTION with facial images

In [7]:
```
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_lfw_people

faces = fetch_lfw_people(min_faces_per_person=100)
print(faces.target_names)
print(faces.images.shape)
```

```
['Colin Powell' 'Donald Rumsfeld' 'George W Bush' 'Gerhard Schroeder'
 'Tony Blair']
(1140, 62, 47)
```

Plot the first 24 images in the dataset along with their labels.

Plot the first 24 images in the dataset along with their labels.

In [8]:
```
#Plotting the first 24 images in the dataset along with their labels.
fig, ax = plt.subplots(3, 8, figsize=(18, 10))

for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='gist_gray')
    axi.set(xticks=[], yticks=[], xlabel=faces.target_names[faces.target[i]])
```

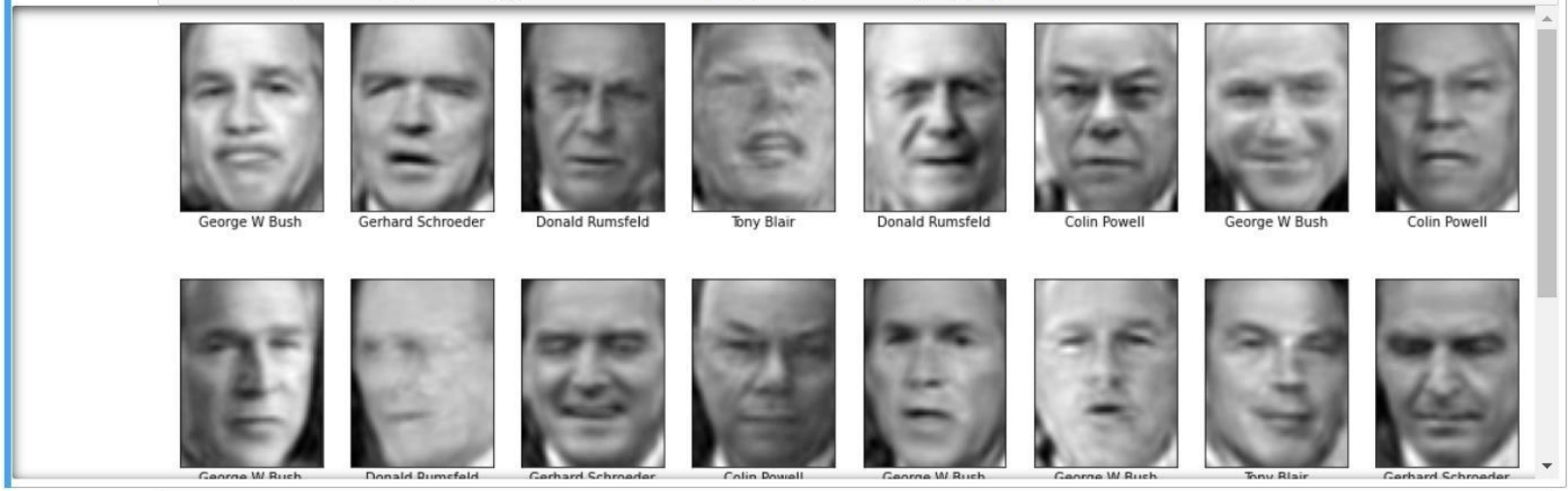
Reducing 2,914 dimensions (62 x 47) to 150 using PCA and invert the transform to restore the facial images.

Now reduce 2,914 dimensions (62 x 47) to 150 using PCA and invert the transform to restore the facial images.

In [9]:
```
#Now reduce 2,914 dimensions (62 x 47) to 150 using PCA
pca = PCA(n_components=150, random_state=0)
pca_faces = pca.fit_transform(faces.data)
```

In [10]:
```
#Inverting the transform to restore the facial images.
unpca_faces = pca.inverse_transform(pca_faces).reshape(1140, 62, 47)
```

In [11]:
```
fig, ax = plt.subplots(3, 8, figsize=(18, 10))

for i, axi in enumerate(ax.flat):
    axi.imshow(unpca_faces[i], cmap='gist_gray')
    axi.set(xticks=[], yticks=[], xlabel=faces.target_names[faces.target[i]])
```



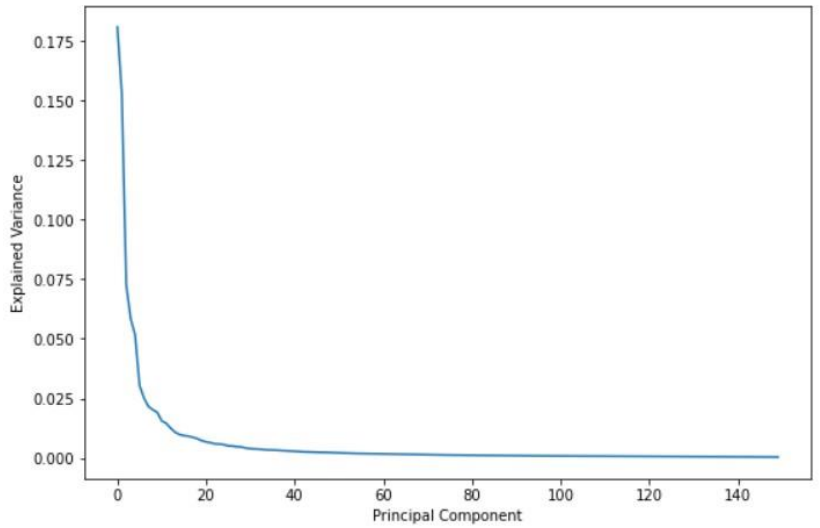
In [12]:
```
np.sum(pca.explained_variance_ratio_)
```

Out[12]:
```
0.9480244
```

Compare the before and after images. Does it look as if 95% of the information was discarded? In fact, you can compute how much of the original information was retained by summing the explained variances. In this case, it's almost 95%.

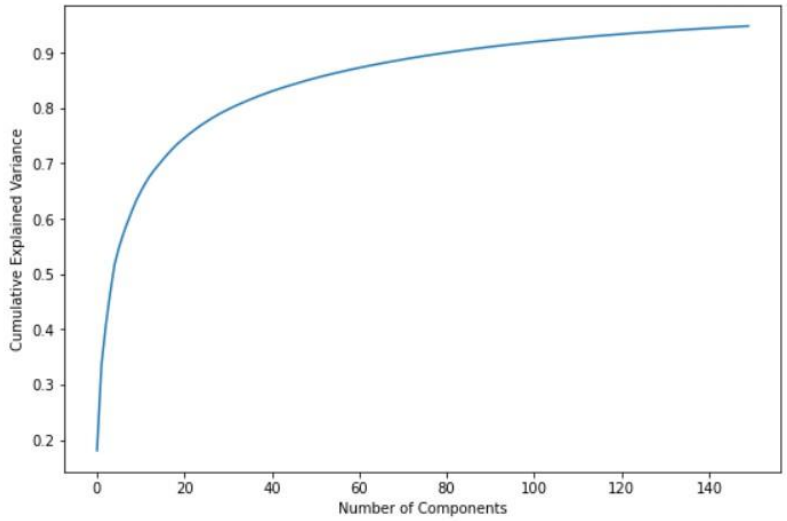
```
In [13]: #"Right" number of components?
         plt.figure(figsize=(9, 6))
         plt.plot(pca.explained_variance_ratio_)
         plt.xlabel('Principal Component')
         plt.ylabel('Explained Variance')
```

```
Out[13]: Text(0, 0.5, 'Explained Variance')
```



A logical question to ask is what is the "right" number of components? In other words, what number of components strikes the best balance between reducing the number of dimensions in the dataset and retaining most of the information? One way to answer that question is a scree plot, which plots the proportion of explained variance for each dimension. Here is a scree plot for the PCA transform used on the facial images.

```
In [14]: plt.figure(figsize=(9, 6))
         plt.plot(np.cumsum(pca.explained_variance_ratio_))
         plt.xlabel('Number of Components')
         plt.ylabel('Cumulative Explained Variance')

Out[14]: Text(0, 0.5, 'Cumulative Explained Variance')
```



Another way to look at it is to plot the sum of the variances as a function of component count.

2. Obfuscating Data:

The "Time" column reveals the number of seconds elapsed since the first transaction. "Class" tells us whether the transaction is legitimate (0) or fraudulent (1), and the "Amount" column shows the amount of the transaction. The remaining columns "V1" through "V28" were generated using PCA from information that isn't revealed to us.

Obfuscating Data using PCA

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

In [2]:
```
df = pd.read_csv("C:/Users/DELL/Downloads/creditcard.csv")
df.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

Checking whether any of those rows having missing values:

```
In [4]: #No of rows representing 0 and 1
        df['Class'].value_counts()

Out[4]: 0    284315
        1       492
        Name: Class, dtype: int64

In [5]: # Separate the samples by class
        legit = df[df['Class'] == 0]
        fraud = df[df['Class'] == 1]

        # Drop the "Time" and "Class" columns
        legit = legit.drop(['Time', 'Class'], axis=1)
        fraud = fraud.drop(['Time', 'Class'], axis=1)

In [6]: from sklearn.decomposition import PCA

        pca = PCA(n_components=26, random_state=0)
        legit_pca = pd.DataFrame(pca.fit_transform(legit), index=legit.index)
        fraud_pca = pd.DataFrame(pca.transform(fraud), index=fraud.index)

In [7]: legit_restored = pd.DataFrame(pca.inverse_transform(legit_pca), index=legit_pca.index)
        fraud_restored = pd.DataFrame(pca.inverse_transform(fraud_pca), index=fraud_pca.index)

In [8]: #Function for measuring loss due to PCA.
        def get_anomaly_scores(df_original, df_restored):
            loss = np.sum((np.array(df_original) - np.array(df_restored)) ** 2, axis=1)
            loss = pd.Series(data=loss, index=df_original.index)
            return loss

In [9]: legit_scores = get_anomaly_scores(legit, legit_restored)
        fraud_scores = get_anomaly_scores(fraud, fraud_restored)
```

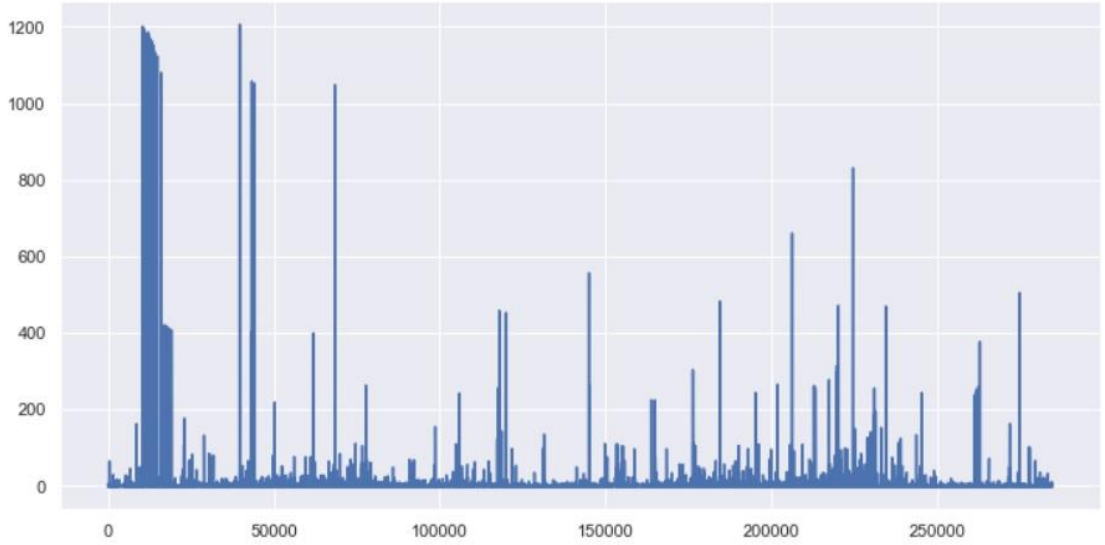
The dataset contains 284808 rows, and none are missing values. Now check the balance: the number of rows representing legitimate transactions (Class=0) vs. the number of rows representing fraudulent transactions (Class=1).

The dataset is highly imbalanced, which isn't surprising given that legitimate credit-card transactions are far more common than fraudulent transactions. A machine-learning model trained on this data will be more accurate at identifying legitimate transactions than fraudulent transactions.

Define a function for measuring loss due to PCA.

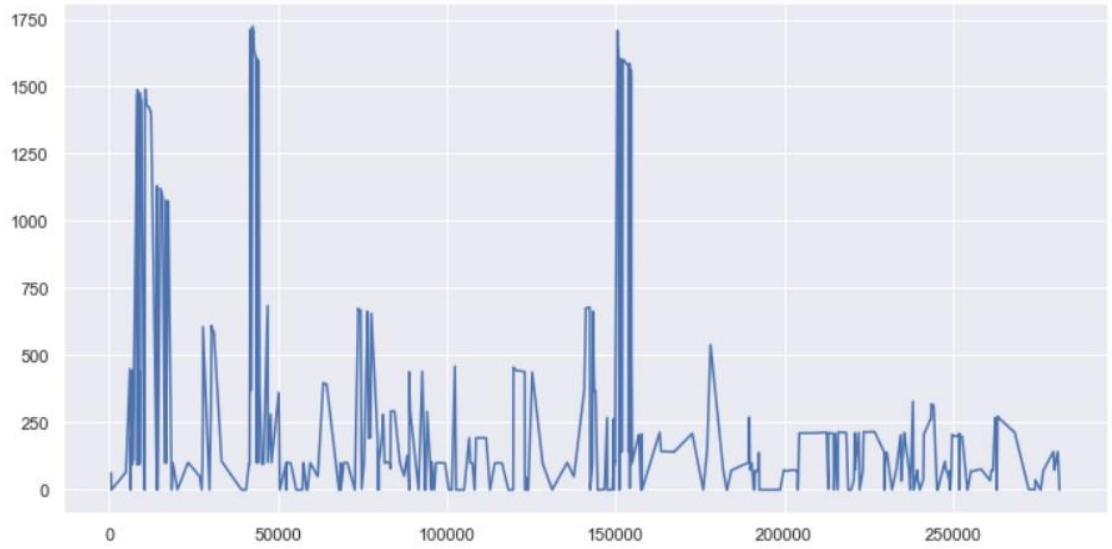
In [10]:
```
#Plot the loss for the legitimate transactions.
legit_scores.plot(figsize = (12, 6))
```

Out[10]: <AxesSubplot:>



In [11]:
```
#Plot the loss for the fraudulent transactions.
fraud_scores.plot(figsize = (12, 6))
```

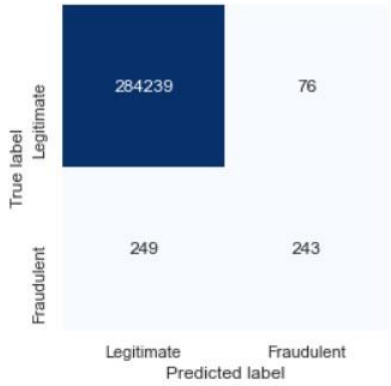
Out[11]: <AxesSubplot:>



0 50000 100000 150000 200000 250000

In [12]:
```
#Confusion matrix
threshold = 200

true_neg = legit_scores[legit_scores < threshold].count()
false_pos = legit_scores[legit_scores >= threshold].count()
true_pos = fraud_scores[fraud_scores >= threshold].count()
false_neg = fraud_scores[fraud_scores < threshold].count()

labels = ['Legitimate', 'Fraudulent']
mat = [[true_neg, false_pos], [false_neg, true_pos]]

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False, cmap='Blues',
            xticklabels=labels, yticklabels=labels)

plt.xlabel('Predicted label')
plt.ylabel('True label')
```

Out[12]:
```
Text(89.133125, 0.5, 'True label')
```



Out of 284808 legitimate transactions, the model correctly classified 284,239 of them as legitimate while misclassifying 76 of them as fraudulent. This means that legitimate transactions are classified correctly more than 99.97% of the time. Meanwhile, the model caught about half of the fraudulent transactions.

3. Noise Reduction: We will use both a PCA and a kernel-based PCA to solve this problem.

Noise Reduction

In [1]:
```
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

X, y = fetch_openml(data_id=41082, as_frame=False, return_X_y=True)
X = MinMaxScaler().fit_transform(X)
```

In [2]:
```
#TRaining and testing data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=0, train_size=1_000, test_size=100
)

rng = np.random.RandomState(0)
noise = rng.normal(scale=0.25, size=X_test.shape)
X_test_noisy = X_test + noise

noise = rng.normal(scale=0.25, size=X_train.shape)
X_train_noisy = X_train + noise
```

In [3]:
```
import matplotlib.pyplot as plt


def plot_digits(X, title):
    """Small helper function to plot 100 digits."""
    fig, axs = plt.subplots(nrows=10, ncols=10, figsize=(8, 8))
    for img, ax in zip(X, axs.ravel()):
        ax.imshow(img.reshape((16, 16)), cmap="Greys")
        ax.axis("off")
    fig.suptitle(title, fontsize=24)
```

We will create a helper function to qualitatively assess the image reconstruction by plotting the test images.

In [4]:
```
#Difference between noise free and noisy image
plot_digits(X_test, "Uncorrupted test images")
plot_digits(
    X_test_noisy, f"Noisy test images\nMSE: {np.mean((X_test - X_test_noisy) ** 2):.2f}"
)
```

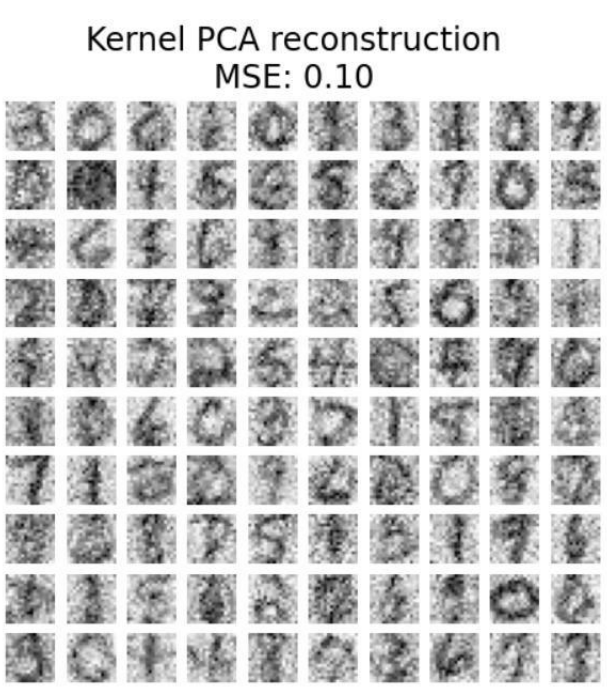
Let’s first have a look to see the difference between noise-free and noisy images.

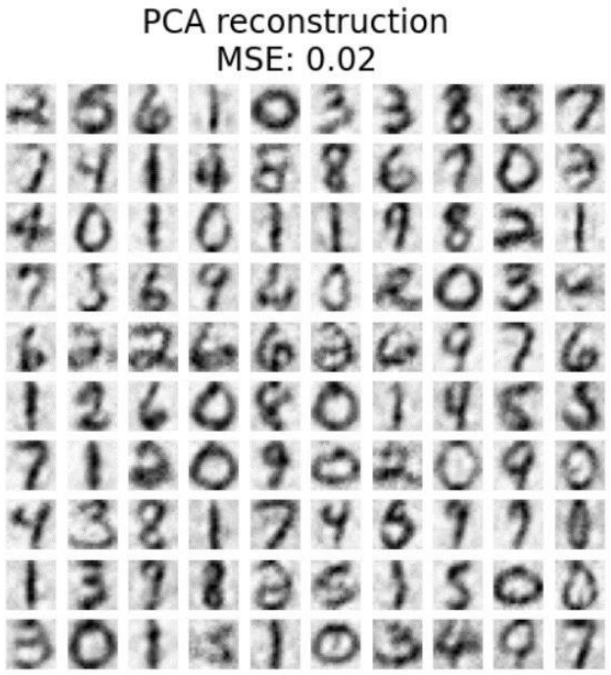
Now, we can transform and reconstruct the noisy test set.

In [5]:
```python
#Using PCA
from sklearn.decomposition import PCA, KernelPCA

pca = PCA(n_components=32)
kernel_pca = KernelPCA(
    n_components=400, kernel="rbf", gamma=1e-3, fit_inverse_transform=True, alpha=5e-3
)

pca.fit(X_train_noisy)
_ = kernel_pca.fit(X_train_noisy)
```

In [6]:
```python
#transform and reconstruct the noisy
X_reconstructed_kernel_pca = kernel_pca.inverse_transform(
    kernel_pca.transform(X_test_noisy)
)
X_reconstructed_pca = pca.inverse_transform(pca.transform(X_test_noisy))
```

In [7]:
```
plot_digits(X_test, "Uncorrupted test images")
plot_digits(
    X_reconstructed_pca,
    f"PCA reconstruction\nMSE: {np.mean((X_test - X_reconstructed_pca) ** 2):.2f}",
)
plot_digits(
    X_reconstructed_kernel_pca,
    "Kernel PCA reconstruction\n"
    f"MSE: {np.mean((X_test - X_reconstructed_kernel_pca) ** 2):.2f}",
)
```


Kernel PCA reconstruction
MSE: 0.10



PCA has a lower MSE than kernel PCA. However, the qualitative analysis might not favor PCA instead of kernel PCA. We observe that kernel PCA is able to remove background noise and provide a smoother image.

4. Anomaly Detection:

Anomaly Detection

In [2]:
```
import pandas as pd
data = pd.read_csv("C:/Users/DELL/Downloads/nyc_taxi.csv")
data['timestamp'] = pd.to_datetime(data['timestamp'])
data.head()
```

Out[2]:

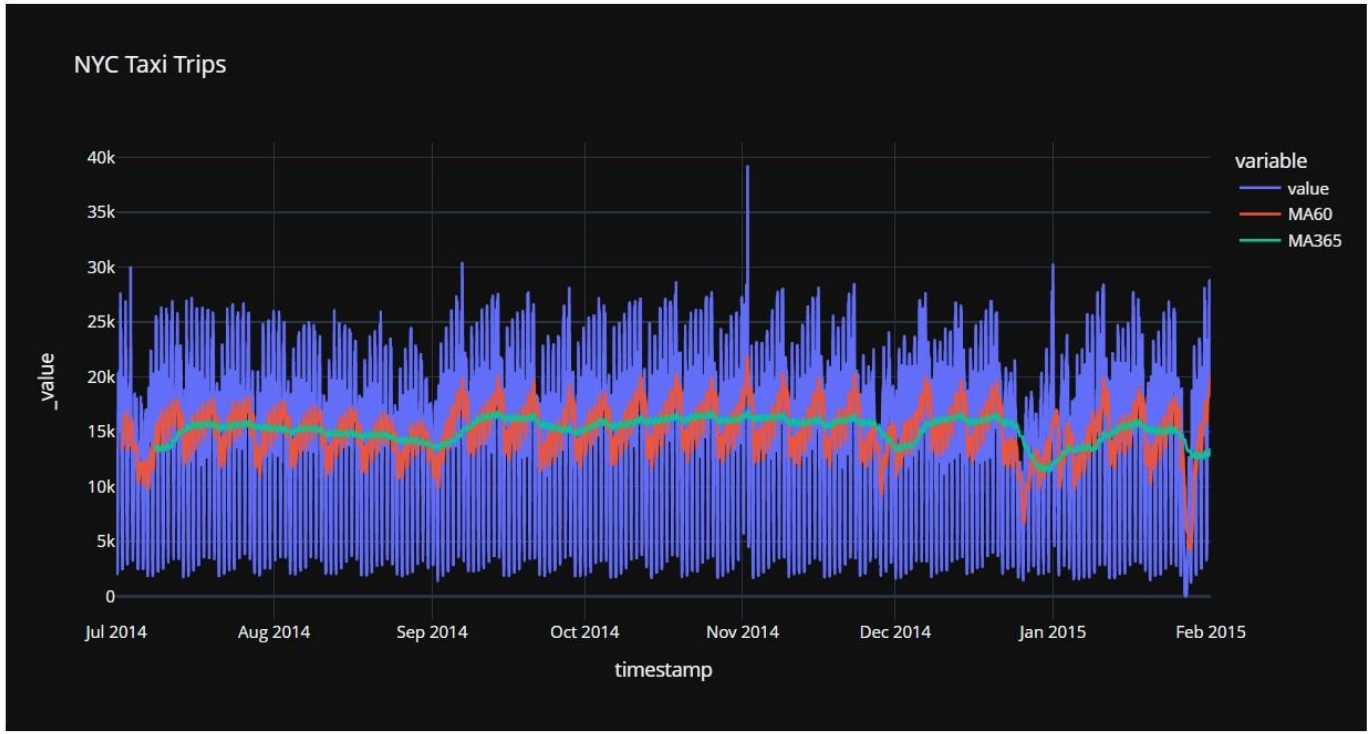
	timestamp	value
0	2014-07-01 00:00:00	10844
1	2014-07-01 00:30:00	8127
2	2014-07-01 01:00:00	6210
3	2014-07-01 01:30:00	4656
4	2014-07-01 02:00:00	3820

In [3]:
```
# create moving-averages
data['MA60'] = data['value'].rolling(60).mean()
data['MA365'] = data['value'].rolling(365).mean()
data.tail()
```

Out[3]:

	timestamp	value	MA60	MA365
10315	2015-01-31 21:30:00	24670	19638.816667	13308.419178
10316	2015-01-31 22:00:00	25721	19807.800000	13361.350685
10317	2015-01-31 22:30:00	27309	20017.633333	13415.520548
10318	2015-01-31 23:00:00	26591	20168.000000	13460.742466
10319	2015-01-31 23:30:00	26288	20255.916667	13501.473973

In [4]:
```
# plot
import plotly.express as px
fig = px.line(data, x="timestamp", y=['value', 'MA60', 'MA365'], title='NYC Taxi Trips', template = 'plotly_dark')
fig.show()
```



In [5]:
```
# drop moving-average columns
data.drop(['MA60', 'MA365'], axis=1, inplace=True)
data.head()
```

Out[5]:

	timestamp	value
0	2014-07-01 00:00:00	10844
1	2014-07-01 00:30:00	8127
2	2014-07-01 01:00:00	6210
3	2014-07-01 01:30:00	4656
4	2014-07-01 02:00:00	3820

In [6]:
```
# set timestamp to index
data.set_index('timestamp', drop=True, inplace=True)
data.head()
```

Out[6]:

	value
timestamp	
2014-07-01 00:00:00	10844
2014-07-01 00:30:00	8127
2014-07-01 01:00:00	6210
2014-07-01 01:30:00	4656
2014-07-01 02:00:00	3820

In [7]:
```
# resample timeseries to hourly
data = data.resample('H').sum()
data.head()
```

Out[7]:

timestamp	value
2014-07-01 00:00:00	18971
2014-07-01 01:00:00	10866
2014-07-01 02:00:00	6693
2014-07-01 03:00:00	4433
2014-07-01 04:00:00	4379

In [8]:
```
# creature features from date
data['day'] = [i.day for i in data.index]
data['day_name'] = [i.day_name() for i in data.index]
data['day_of_year'] = [i.dayofyear for i in data.index]
data['week_of_year'] = [i.weekofyear for i in data.index]
data['hour'] = [i.hour for i in data.index]
data['is_weekday'] = [i.isoweekday() for i in data.index]
data.head()
```

Out[8]:

timestamp	value	day	day_name	day_of_year	week_of_year	hour	is_weekday
2014-07-01 00:00:00	18971	1	Tuesday	182	27	0	2
2014-07-01 01:00:00	10866	1	Tuesday	182	27	1	2
2014-07-01 02:00:00	6693	1	Tuesday	182	27	2	2
2014-07-01 03:00:00	4433	1	Tuesday	182	27	3	2
2014-07-01 04:00:00	4379	1	Tuesday	182	27	4	2

In [9]:
```
# init setup
from pycaret.anomaly import *
s = setup(data, session_id = 42,
          ordinal_features = {'day_name' : ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
       'Friday','Sunday','Saturday',]},
          numeric_features=['is_weekday'])
```

0	session_id	42
1	Original Data	(5160, 7)
2	Missing Values	False
3	Numeric Features	6
4	Categorical Features	1
5	Ordinal Features	True
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(5160, 7)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	anomaly-default-name

In [5]:
```
# drop moving-average columns
data.drop(['MA60', 'MA365'], axis=1, inplace=True)
data.head()
```

Out[5]:

	timestamp	value
0	2014-07-01 00:00:00	10844
1	2014-07-01 00:30:00	8127
2	2014-07-01 01:00:00	6210
3	2014-07-01 01:30:00	4656
4	2014-07-01 02:00:00	3820

In [6]:
```
# set timestamp to index
data.set_index('timestamp', drop=True, inplace=True)
data.head()
```

Out[6]:

	value
timestamp	
2014-07-01 00:00:00	10844
2014-07-01 00:30:00	8127
2014-07-01 01:00:00	6210
2014-07-01 01:30:00	4656
2014-07-01 02:00:00	3820

In [11]:
```
# train model
pca = create_model('pca')
pca_results = assign_model(pca)
pca_results.head()
```

Out[11]:

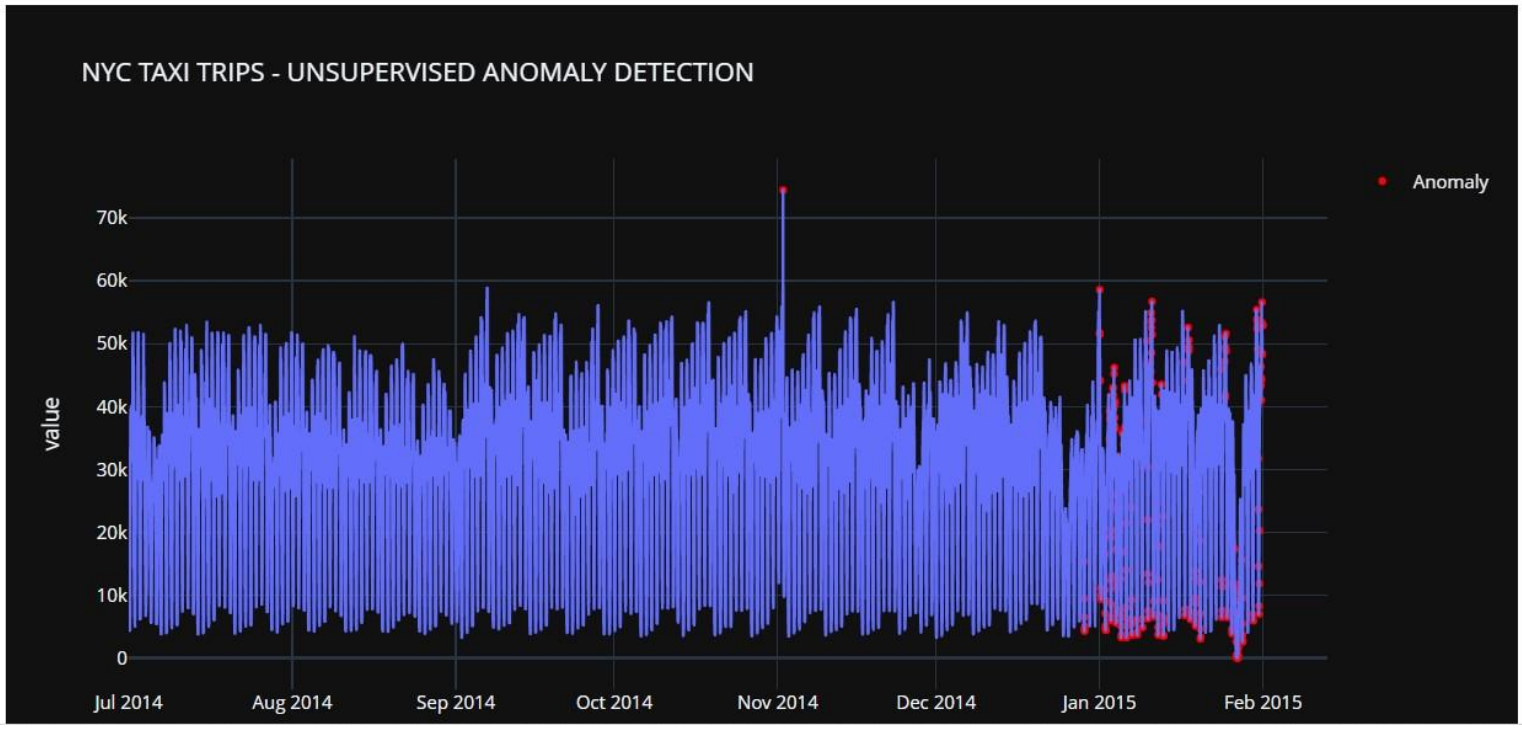
timestamp	value	day	day_name	day_of_year	week_of_year	hour	is_weekday	Anomaly	Anomaly_Score
2014-07-01 00:00:00	18971	1	Tuesday	182	27	0	2	0	1014.236530
2014-07-01 01:00:00	10866	1	Tuesday	182	27	1	2	0	1062.836288
2014-07-01 02:00:00	6693	1	Tuesday	182	27	2	2	0	1091.848640
2014-07-01 03:00:00	4433	1	Tuesday	182	27	3	2	0	1104.146923
2014-07-01 04:00:00	4379	1	Tuesday	182	27	4	2	0	1089.556476

In [12]:
```
# check anomalies
pca_results[pca_results['Anomaly'] == 1].head()
```

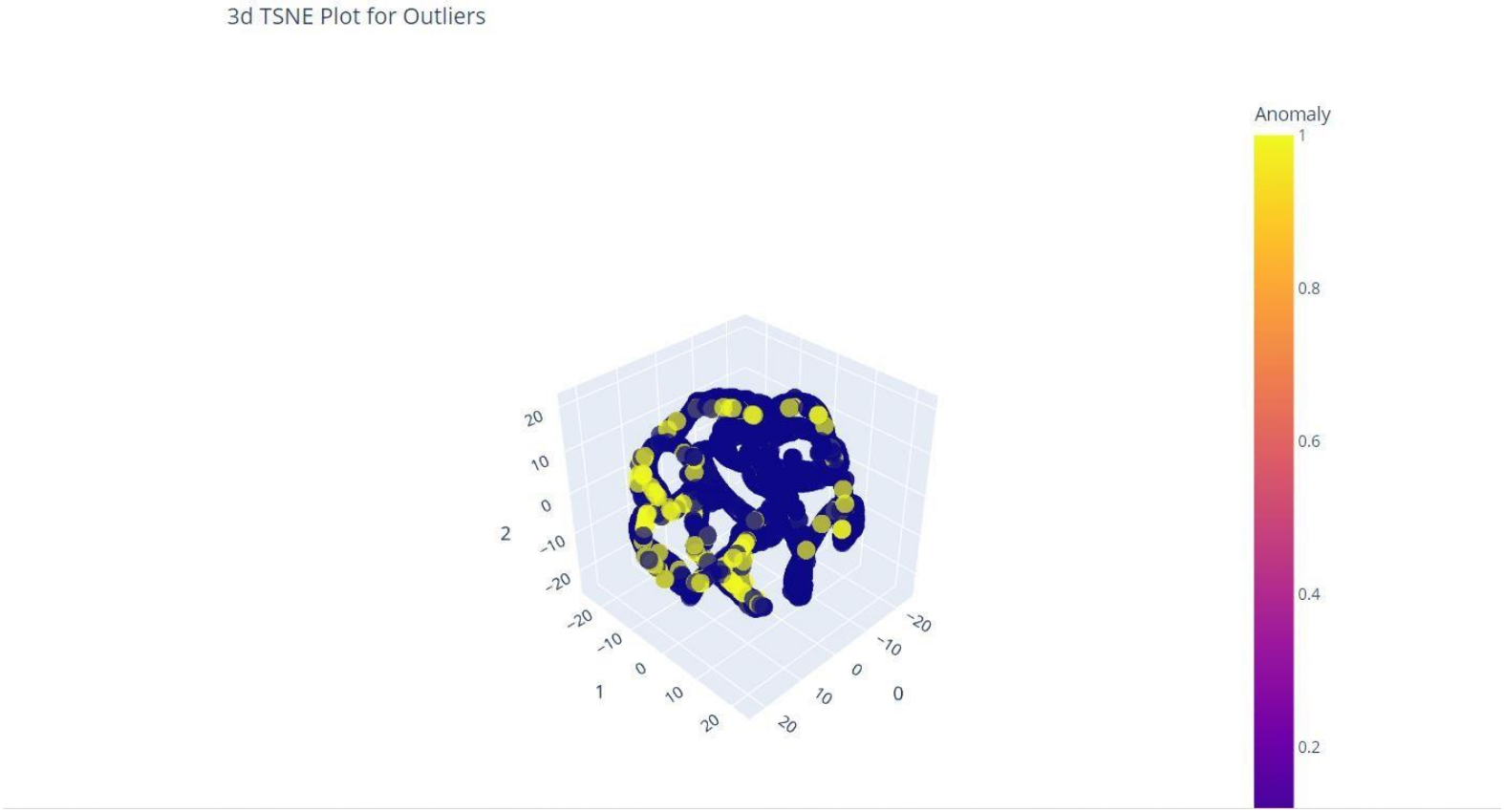
Out[12]:

timestamp	value	day	day_name	day_of_year	week_of_year	hour	is_weekday	Anomaly	Anomaly_Score
2014-11-02 01:00:00	74409	2	Sunday	306	44	1	7	1	1436.756752
2014-12-29 00:00:00	15314	29	Monday	363	1	0	1	1	1325.651108
2014-12-29 01:00:00	9440	29	Monday	363	1	1	1	1	1354.394925
2014-12-29 02:00:00	6503	29	Monday	363	1	2	1	1	1366.901125
2014-12-29 03:00:00	4589	29	Monday	363	1	3	1	1	1373.112759

In [13]:
```
import plotly.graph_objects as go
# plot value on y-axis and date on x-axis
fig = px.line(pca_results, x=pca_results.index, y="value", title='NYC TAXI TRIPS - UNSUPERVISED ANOMALY DETECTION', template = 'p
# create list of outlier_dates
outlier_dates = pca_results[pca_results['Anomaly'] == 1].index
# obtain y value of anomalies to plot
y_values = [pca_results.loc[i]['value'] for i in outlier_dates]
fig.add_trace(go.Scatter(x=outlier_dates, y=y_values, mode = 'markers',
                name = 'Anomaly',
                marker=dict(color='red',size=5)))
fig.show()
```



In [14]:
```
plot_model(pca)
```



uMAP Plot

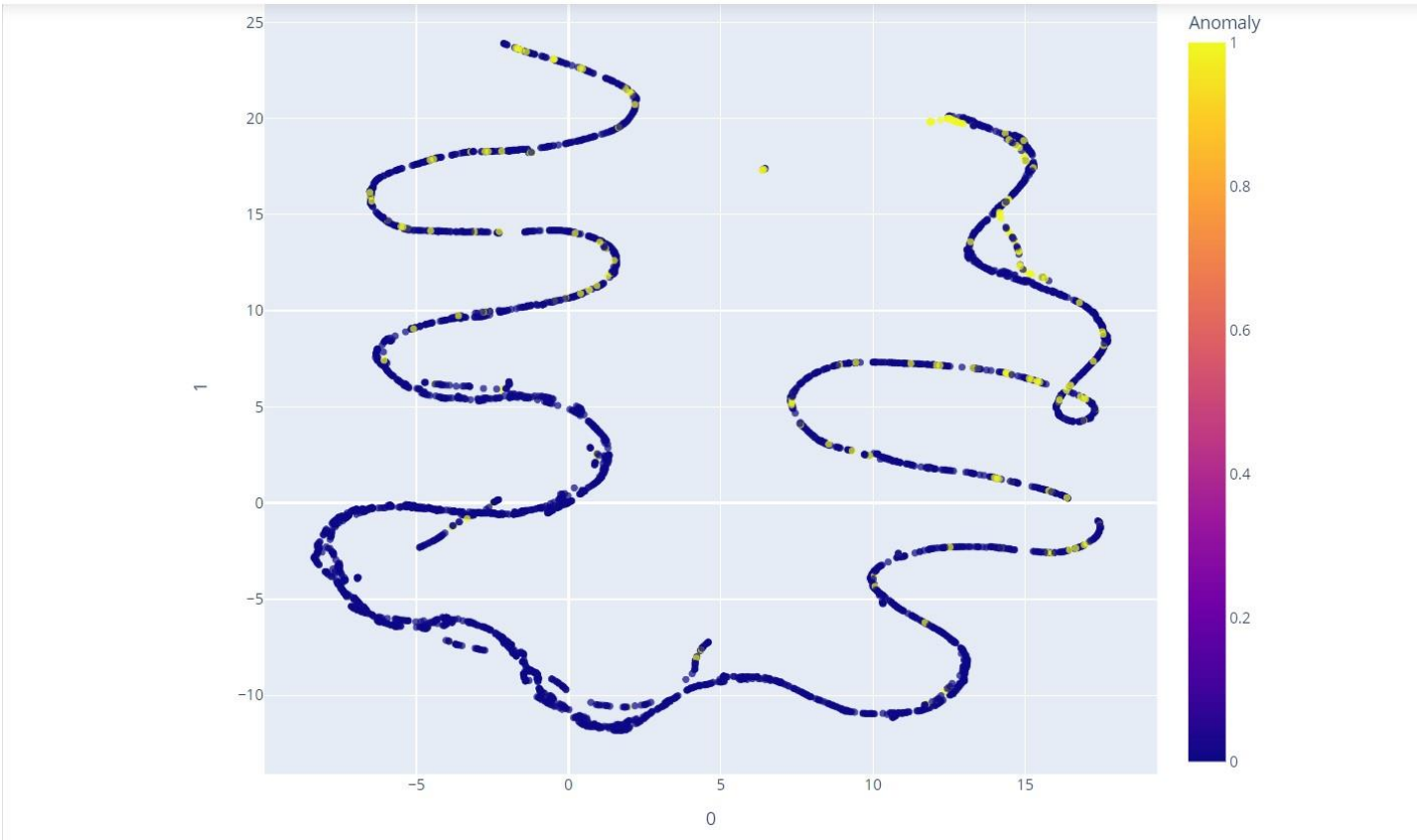
A uMAP plot is a graph displaying the "Uniform Manifold Approximation and Projection",

which visually shows how separable the classes under consideration are with respect to the selected group of features.

It is a 2D plot and represents each class as a cluster of points in a unique color.

You want to have these clusters as far away from each other as possible for the best classification. You can visually inspect UMAP plots for your selected sensors and feature groups. We recommend choosing sensors and feature groups that result in maximally separated clusters.

In [14]:
```
plot_model(pca, plot = 'umap')
```



Describe the “Eureka!”, and the “Oh no!”

“Eureka!” moment for us was the moment when we figured out what we exactly need to do after watching that YouTube video, it definitely showed us the right path.

Well, “Oh No!” definitely for everyone is the moment when they get error in their codes, for us we were not able to resolve one error in anomaly detection file for a long time, even after installing so many packages. But finally we did it.

What did we learn?

When we say we learned something that is obviously something “new”. But in our case it’s a “lot of new things”. Well one thing that we learned about PCA is that, it’s vast. Don’t keep it till dimensionality reduction.

Other thing we learned is that we have so many Supervised and Unsupervised techniques in ML and for every technique we know the functionality but if we change our perspective maybe we can discover new things about them.

What more needs to be done in the future?

The things that we did are just the first step. As we mentioned earlier we made it simple to make everyone understand it. There is lot more one can do like:

1. Take more complex datasets.
2. Perform different modeling after applying PCA.