```
#%%
# Importing Required Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn import metrics
#%%
```

```
#Taking the Iris's Dataset
iris_df = sns.load_dataset('iris')
iris_df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
#%%
#Data Presprocessing
#%%
iris_df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
# %%
iris_df.shape
```

```
(150, 5)
```

```
# %%
iris_df.isnull().sum()
```

	0
sepal_length	0
sepal_width	0
petal_length	0
petal_width	0
species	0

dtype: int64

```
#%%

df = iris_df.dropna()  # Drop rows with missing values
 #This won't do anything as there is no null values in the dataset
#%%
```

```
#Selecting the features
X = iris_df.drop(columns=['species'])  # Features
y = iris_df['species']  # Target
print("Training column is: \n ", X.head())
print("Target is: \n", y.head())
```

```
Training column is:
     sepal_length  sepal_width  petal_length  petal_width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
Target is:
 0    setosa
1    setosa
2    setosa
3    setosa
4    setosa
Name: species, dtype: object
```

```
# %%
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
print(X_train.shape)
print(X_test.shape)
```

```
(120, 4)
(30, 4)
```

```
# %%

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print(X_train_scaled)
print(X_test_scaled.shape)
# %%
```

```
 [ 0.40876153  0.75647855  1.16139593  1.31369383]
 [-0.49876152  0.75647855 -1.16139502 -1.31260282]
 [ 0.35451684 -0.58505976  0.15663551  0.15573254]
 [-1.10824606 -1.25582892  0.44316389  0.68967267]
 [-0.01117388  2.09801686 -1.4479234  -1.31260282]
 [-0.01117388 -1.0322392   0.15663551  0.02224751]
 [ 1.57348593 -0.13788033  1.24544335  1.22361279]]
(30, 4)
```

```
# Train classifier on raw data
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train_scaled, y_train)
y_pred = clf.predict(X_test_scaled)
print("Accuracy on Raw Data:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy on Raw Data: 1.0
```

```
# %%

# Apply PCA (2 Components)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

```
# %%

# Train classifier with PCA-transformed data
clf_pca = RandomForestClassifier(n_estimators=100, random_state=42)
clf_pca.fit(X_train_pca, y_train)
y_pred_pca2 = clf_pca.predict(X_test_pca)
print("Accuracy with PCA (2 components):", metrics.accuracy_score(y_test, y_pr
```

```
Accuracy with PCA (2 components): 0.9
```

```
# %%

# Apply PCA (3 Components)
pca = PCA(n_components=3)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

```
# %%
# Train classifier with PCA-transformed data
clf_pca = RandomForestClassifier(n_estimators=100, random_state=42)
clf_pca.fit(X_train_pca, y_train)
y_pred_pca3 = clf_pca.predict(X_test_pca)
print("Accuracy with PCA (3 components):", metrics.accuracy_score(y_test, y_pr
```

```
Accuracy with PCA (3 components): 1.0
```

```
# %%

print (" Accuracy's of all: \n")

print("Accuracy on Raw Data:", metrics.accuracy_score(y_test, y_pred))
print("Accuracy with PCA (2 components):", metrics.accuracy_score(y_test, y_pre

print("Accuracy with PCA (3 components):", metrics.accuracy_score(y_test, y_pre
```

```
 Accuracy's of all:

Accuracy on Raw Data: 1.0
Accuracy with PCA (2 components): 0.9
Accuracy with PCA (3 components): 1.0
```

Taxis Dataset

The Taxis dataset contains information on taxi rides, including fare, distance, pickup and drop-off locations, and payment type. we used it for classification tasks, specifically predicting whether a ride was paid using cash or a card.

Features Used: ● Fare, Distance (Numerical) ● Pickup Borough, Dropoff Borough (Categorical, one-hot encoded) ● Payment Type (Target variable: Cash = 0, Card = 1)

Link: https://www.kaggle.com/datasets/abdmental01/taxis-dataset-yellow-taxi

Taxis Dataset has a mix of numerical & categorical features, but after encoding categorical features (e.g., boroughs), the number of dimensions increases. PCA helps reduce this.

```
#%%
# Importing Required Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn import metrics
#%%
#Taking the Taxi's Dataset
taxis_df = sns.load_dataset('taxis')
taxis_df.head()
```

	pickup	dropoff	passengers	distance	fare	tip	tolls	total	color	payme
0	2019-03-23 20:21:09	2019-03-23 20:27:24	1	1.60	7.0	2.15	0.0	12.95	yellow	cre c
1	2019-03-04 16:11:55	2019-03-04 16:19:00	1	0.79	5.0	0.00	0.0	9.30	yellow	ca

```
#%%
#Data Presprocessing
#%%
taxis_df.describe()
```

	pickup	dropoff	passengers	distance	far
count	6433	6433	6433.000000	6433.000000	6433.00000
mean	2019-03-16 08:31:28.514223616	2019-03-16 08:45:49.491217408	1.539251	3.024617	13.09107
min	2019-02-28 23:29:03	2019-02-28 23:32:35	0.000000	0.000000	1.00000
25%	2019-03-08 15:50:34	2019-03-08 16:12:51	1.000000	0.980000	6.50000
50%	2019-03-15 21:46:58	2019-03-15 22:06:44	1.000000	1.640000	9.50000
75%	2019-03-23 17:41:38	2019-03-23 17:51:56	2.000000	3.210000	15.00000
max	2019-03-31 23:43:45	2019-04-01 00:13:58	6.000000	36.700000	150.00000
std	NaN	NaN	1.203768	3.827867	11.5518

```
# %%
taxis_df.shape
```

```
(6433, 14)
```

```
# %%
taxis_df.isnull().sum()
```

```
                  0
pickup            0
dropoff           0
passengers        0
distance          0
fare              0
tip               0
tolls             0
total             0
color             0
payment          44
pickup_zone      26
dropoff_zone     45
pickup_borough   26
dropoff_borough  45

dtype: int64
```

```
#%%

df = taxis_df.dropna()  # Drop rows with missing values
```

```
#%%
# Converting Target Variable to Numerial
le = LabelEncoder()
taxis_df['payment'] = le.fit_transform(taxis_df['payment'])
```

```
# %%
# Select features for classification
X = taxis_df[['fare', 'distance', 'pickup_borough',
              'dropoff_borough']]
X = pd.get_dummies(X, drop_first=True)  # Encode categorical features
y = taxis_df['payment']  # Target (Cash or Card)
```

```
# %%
# Train-Test Split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2)
```

```
# %%
# Scaling the features
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# %%
# Training a Classifier on Raw Data
clf = RandomForestClassifier(n_estimators=100,
                             random_state=42)
clf.fit(X_train_scaled, y_train)
# %%
```

```
▾     RandomForestClassifier     ⓘ ?

RandomForestClassifier(random_state=42)
```

```
y_pred = clf.predict(X_test_scaled)
print("Accuracy on Raw Data:",
      metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy on Raw Data: 0.6379176379176379
```

```
# %% ##Applying PCA (2 Components)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

```
# %%## Creating ClaSsfier using pca2
clf_pca = RandomForestClassifier(n_estimators=100,
                                 random_state=42)
clf_pca.fit(X_train_pca, y_train)
```

```
▾     RandomForestClassifier     ⓘ ?

RandomForestClassifier(random_state=42)
```

```
# %%
y_pred_pca2 = clf_pca.predict(X_test_pca)
print("Accuracy with PCA 2 components:", metrics.accuracy_score(y_test, y_pred_p
```

```
Accuracy with PCA 2 components: 0.6449106449106449
```

```
# %%
## Applying PCA ( 3 Components)
pca = PCA(n_components=3)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

```
# %%
clf_pca = RandomForestClassifier(n_estimators=100, random_state=42)
clf_pca.fit(X_train_pca, y_train)
```

```
▾     RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# %%
y_pred_pca3 = clf_pca.predict(X_test_pca)
print("Accuracy with PCA 3 components:", metrics.accuracy_score(y_test, y_pred_
```

```
Accuracy with PCA 3 components: 0.6511266511266511
```

```
# %%
print("\nThe Accuracy of all is Respectively:\n")

print("Accuracy on Raw Data:", metrics.accuracy_score(y_test, y_pred))
print("Accuracy with PCA 2 components:", metrics.accuracy_score(y_test, y_pred_
print("Accuracy with PCA 3 components:", metrics.accuracy_score(y_test, y_pred_
```

```

The Accuracy of all is Respectively:

Accuracy on Raw Data: 0.6379176379176379
Accuracy with PCA 2 components: 0.6449106449106449
Accuracy with PCA 3 components: 0.6511266511266511
```