In [88]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from xgboost import XGBRegressor
import matplotlib.pyplot as plt
```

In [89]:

```
# Load train and test data
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

In [90]:

```
# 1. Remove columns with zero variance
train_df = train_df.loc[:, train_df.apply(pd.Series.nunique) != 1]
test_df = test_df.loc[:, test_df.apply(pd.Series.nunique) != 1]
```

In [91]:

```
# 2. Check for null and unique values for test and train sets
print('Train set:')
print('Null values: ', train_df.isnull().sum().sum())
print('Unique values: ', train_df.nunique())
print('\nTest set:')
print('Null values: ', test_df.isnull().sum().sum())
print('Unique values: ', test_df.nunique())
```

```
Train set:
Null values:  0
Unique values:  ID      4209
y       2545
X0        47
X1        27
X2        44
        ...
X380       2
X382       2
X383       2
X384       2
X385       2
Length: 366, dtype: int64

Test set:
Null values:  0
Unique values:  ID      4209
X0        49
X1        27
X2        45
X3         7
        ...
X380       2
X382       2
X383       2
X384       2
X385       2
Length: 372, dtype: int64
```

In [92]:

```
train_df_2 = train_df
```

In [93]:

```
# Ensuring train and test datasets have the same columns
train_df, test_df = train_df.align(test_df, join='inner', axis=1)
```

In [94]:

```
# 3. Apply Label Encoder
for c in train_df.columns:
    if train_df[c].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(train_df[c].values) + list(test_df[c].values))
        train_df[c] = lbl.transform(list(train_df[c].values))
        test_df[c] = lbl.transform(list(test_df[c].values))
```

In [95]:

```
# 4. Perform dimensionality reduction
n_comp = 12 # Number of components you want to reduce to
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(train_df)
pca2_results_test = pca.transform(test_df)
```

In [96]:

```
# 5. Predict test_df values using XGBoost
y_train = train_df_2['y'].values
xgb = XGBRegressor(n_estimators=500, learning_rate=0.05, gamma=0, subsample=0.75, colsample_bytree=1, max_depth
xgb.fit(pca2_results_train, y_train)
```

Out[96]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None, colsample_bytree=
1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=0, gpu_id=None,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=7, max_leaves=None,
```

In [97]:

```
prediction = xgb.predict(pca2_results_test)
print('Predictions: ', prediction)
```

```
Predictions:  [ 82.230934 101.91376   90.335495 ...  97.99921  111.22856   94.04689 ]
```

In [98]:

```
# Assuming y_test are the actual values for the test set
y_test = train_df_2['y'].values

plt.figure(figsize=(12, 6))

# Scatter plot for actual values
plt.scatter(range(len(y_test)), y_test, color='blue', label='Actual values')

# Scatter plot for predicted values
plt.scatter(range(len(prediction)), prediction, color='red', label='Predicted values')

plt.xlabel('Index')
plt.ylabel('Values')
plt.title('Actual vs Predicted values')
plt.legend(loc='upper right')
plt.show()
```