In [83]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

columns = ["id", "diagnosis"]
x_columns = []
for i in range(1, 31):
    columns.append("feature_" + str(i))
    x_columns.append("feature_" + str(i))

df = pd.read_csv("data/wdbc.data", names=columns)
```

Data preprocessing

Apply MinMaxScaler to the features to minimize the effect of features with large values

In [84]:
```python
features = df[x_columns]
features = MinMaxScaler().fit_transform(features)

target = df["diagnosis"]
target_name = df["diagnosis"].copy()
target = target.map({"M": 1, "B": 0})
```

K-Means implementation

In [85]:
```python
def k_means(data, k_num, max_iters=100):
    cent = data[np.random.choice(data.shape[0], k_num, replace=False)]

    label = np.array([])

    for _ in range(max_iters):
        distances = np.sqrt(((data - cent[:, np.newaxis]) ** 2).sum(axis=2))
        label = np.argmin(distances, axis=0)

        new_centroids = np.array([data[label == j].mean(axis=0) for j in range(k_num)])

        if np.allclose(cent, new_centroids):
            break

        cent = new_centroids

    return cent, label
```

In [86]:
```python
def calculate_accuracy(l, t):
    acc = np.mean(l == t)
    if acc < 0.5:
        acc = 1 - acc

    return acc
```

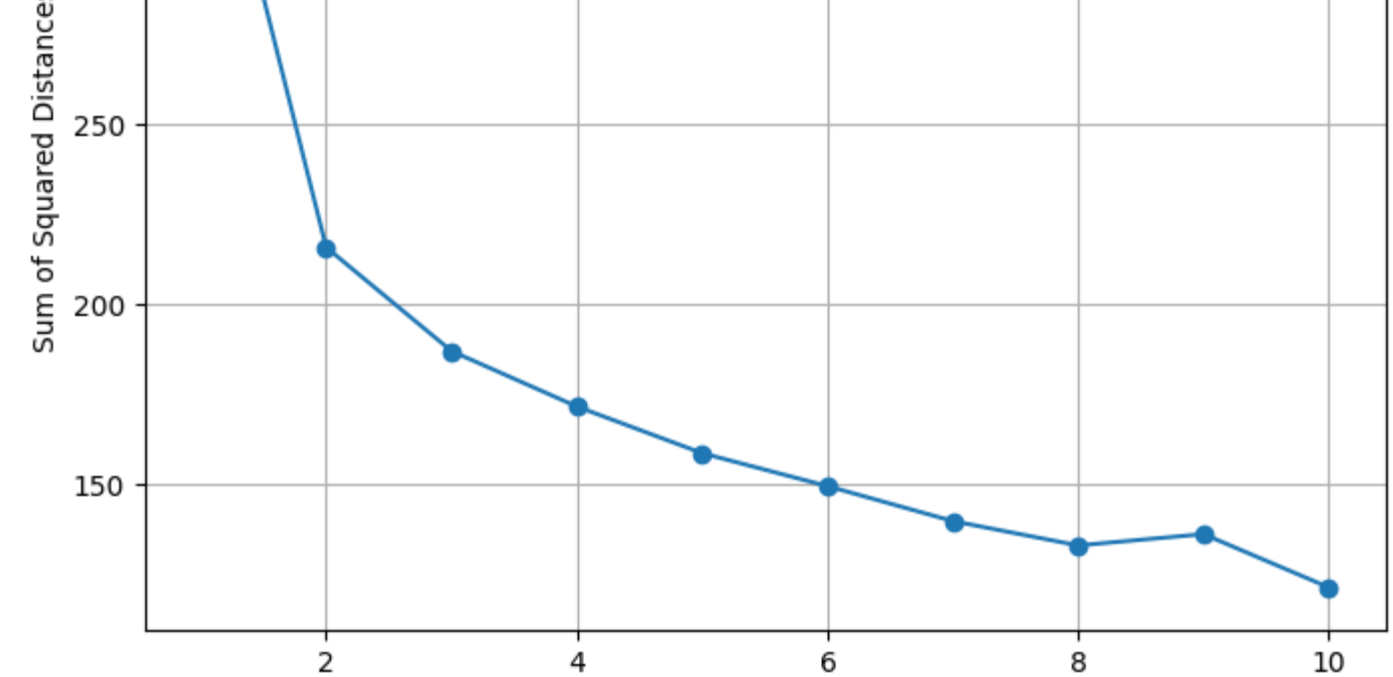
Elbow method

In [87]:
```python
def elbow_method(data, max_k=10, title="Elbow"):
    sum_squared_err = []

    for current_k in range(1, max_k + 1):
        centroids, labels = k_means(data, k_num=current_k)
        sum_squared_err.append(np.sum((data - centroids[labels]) ** 2))

    plt.figure(figsize=(8, 6))
    plt.plot(range(1, max_k + 1), sum_squared_err, marker='o')
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Sum of Squared Distances')
    plt.title(title)
    plt.grid()
    plt.show()
```

Trying different k-means values and visualizing the results

In [98]:
```python
k_nums = [2, 3, 4, 5, 6, 7, 8, 9, 10]

plots_per_row = 3
num_rows = int(np.ceil(len(k_nums) / plots_per_row))
fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(15, 5 * num_rows))

for i, k in enumerate(k_nums):
    centroids, labels = k_means(features, k_num=k)
    row = i // plots_per_row
    col = i % plots_per_row

    if num_rows == 1:
        ax = axes[col]
    else:
        ax = axes[row, col]

    ax.scatter(features[:, 0], features[:, 1], c=labels, cmap='viridis')
    ax.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, alpha=0.5)
    ax.set_title(f'K = {k}')

# Hide empty subplots if there are any
for i in range(len(k_nums), num_rows * plots_per_row):
    row = i // plots_per_row
    col = i % plots_per_row

    if num_rows == 1:
        ax = axes[col]
    else:
        ax = axes[row, col]

    ax.axis('off')

plt.tight_layout()
plt.show()
```

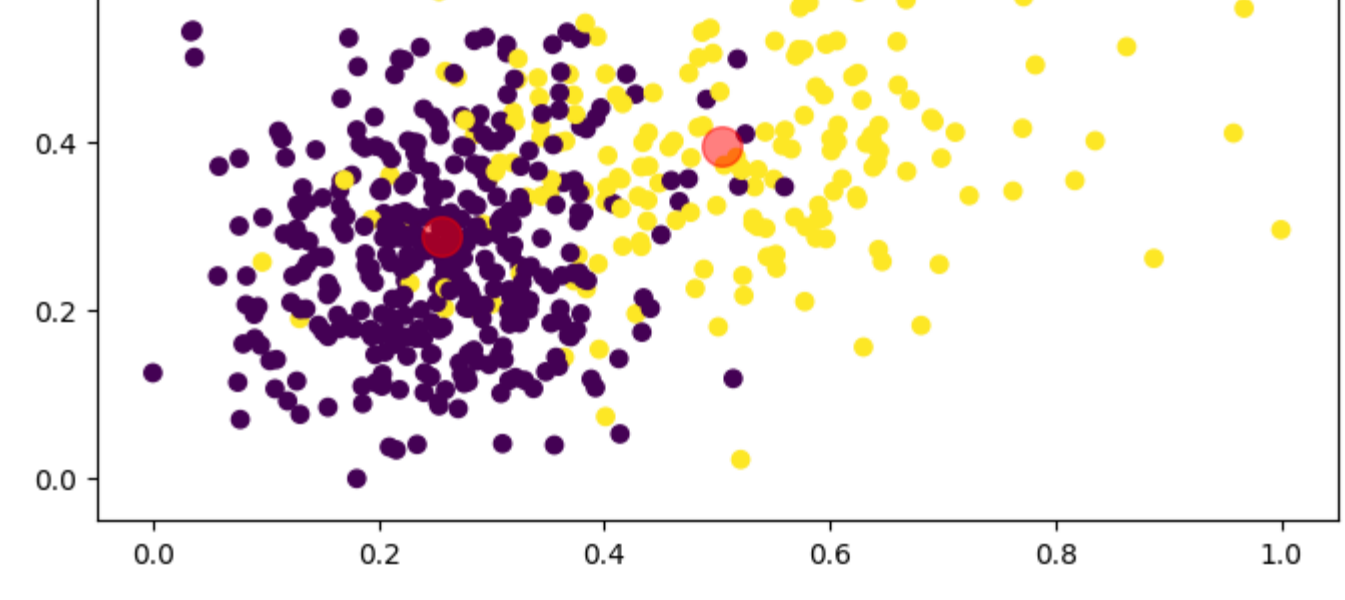
In [88]:
```python
elbow_method(features, title="Elbow Exp 1 before PCA")
```



Observations

We can see that the elbow is at k=2, which is the number of classes in the dataset.

In [89]:
```python
k = 2 # Best number of clusters from elbow method

centroids, labels = k_means(features, k_num=k)

# plt
plt.figure(figsize=(8, 6))
plt.scatter(features[:, 0], features[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, alpha=0.5)
plt.show()

# Calculate accuracy
accuracy = calculate_accuracy(labels, target)

print("Accuracy: " + str(round(accuracy * 100, 2)) + "%")

sse_exp_1 = np.sum((features - centroids[labels]) ** 2)
```

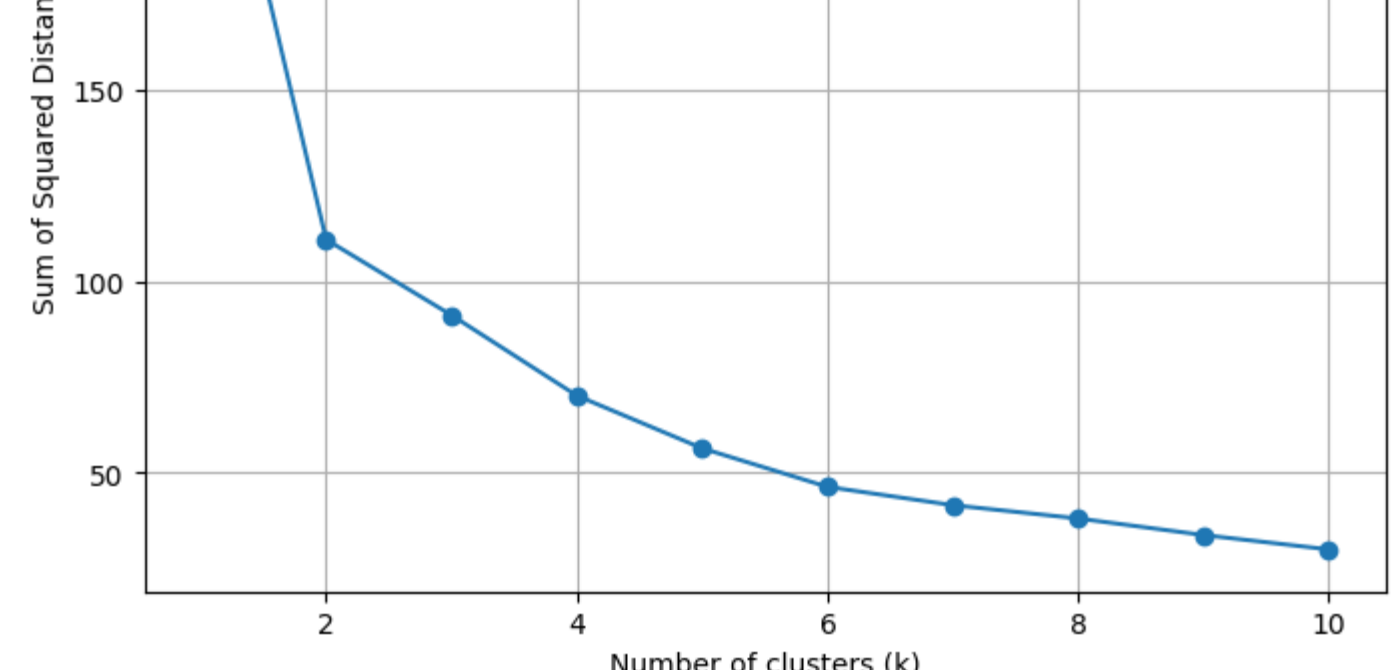


```
Accuracy: 92.79%
```

PCA implementation

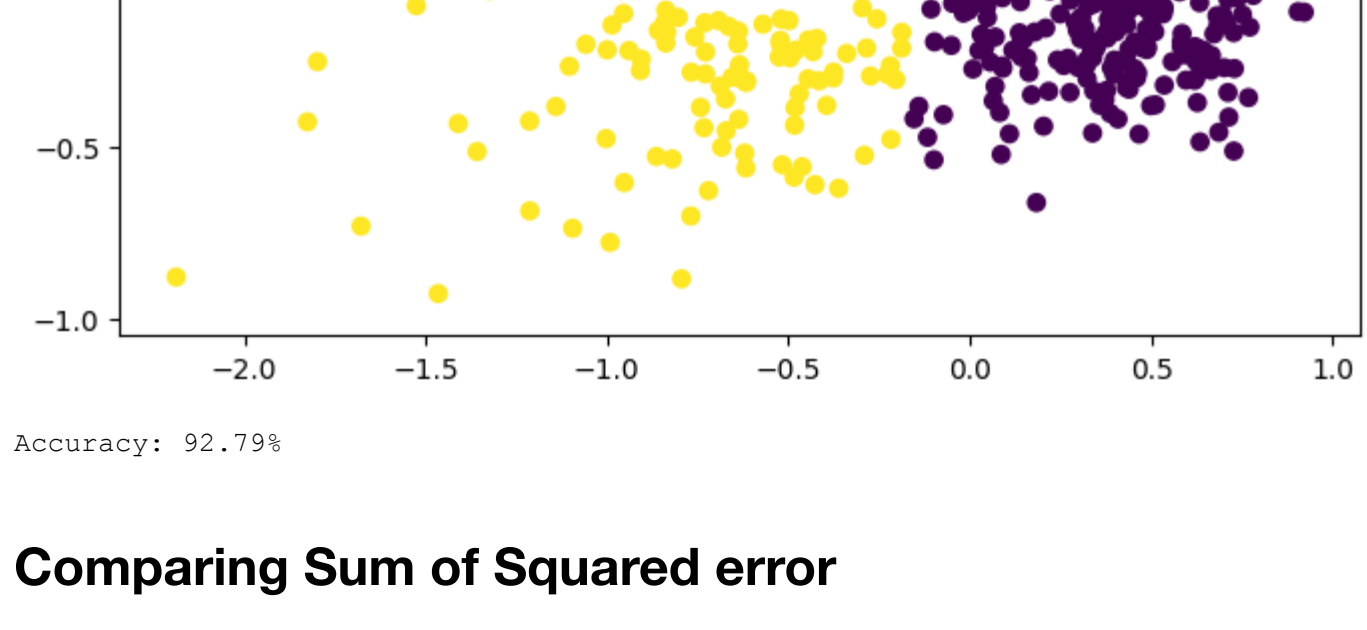
In [90]:
```python
def pca(data, num_components):
    # Step 1: Standardize the data
    data_mean = np.mean(data, axis=0)
    standardized_data = data - data_mean

    # Step 2: Compute the covariance matrix
    covariance_matrix = np.cov(standardized_data, rowvar=False)

    # Step 3: Calculate eigenvalues and eigenvectors
    eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

    # Step 4: Select top k eigenvectors
    sorted_indices = np.argsort(eigenvalues)[::-1]
    top_k_indices = sorted_indices[:num_components]
    principal_components = eigenvectors[:, top_k_indices]

    # Step 5: Project the data onto the selected principal components
    transformed_data = np.dot(standardized_data, principal_components)

    return transformed_data
```

In [91]:
```python
t_data = pca(features, num_components=2)

elbow_method(t_data, title="Elbow Ex 2 with PCA")
```



In [92]:
```python
k = 2 # Best number of clusters from elbow method
centroids, labels = k_means(t_data, k)
```

In [93]:
```python
plt.figure(figsize=(8, 6))
plt.scatter(t_data[:, 0], t_data[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, alpha=0.5)
plt.show()

sse_exp_2 = np.sum((t_data - centroids[labels]) ** 2)

accuracy = calculate_accuracy(labels, target)
print("Accuracy: " + str(round(accuracy * 100, 2)) + "%")
```



```
Accuracy: 92.79%
```

Comparing Sum of Squared error

Expectations

Experiment 1 (Without PCA):

- SSE reflects distances in the original feature space without dimensionality reduction.
- Higher SSE may indicate more scattered or dispersed clusters due to the higher dimensionality.

Experiment 2 (With PCA):

- Lower SSE compared to Experiment 1 due to dimensionality reduction.
- PCA attempts to retain the most relevant information in fewer dimensions.
- Lower SSE suggests that clustering in a reduced-dimensional space captures the inherent structure better.

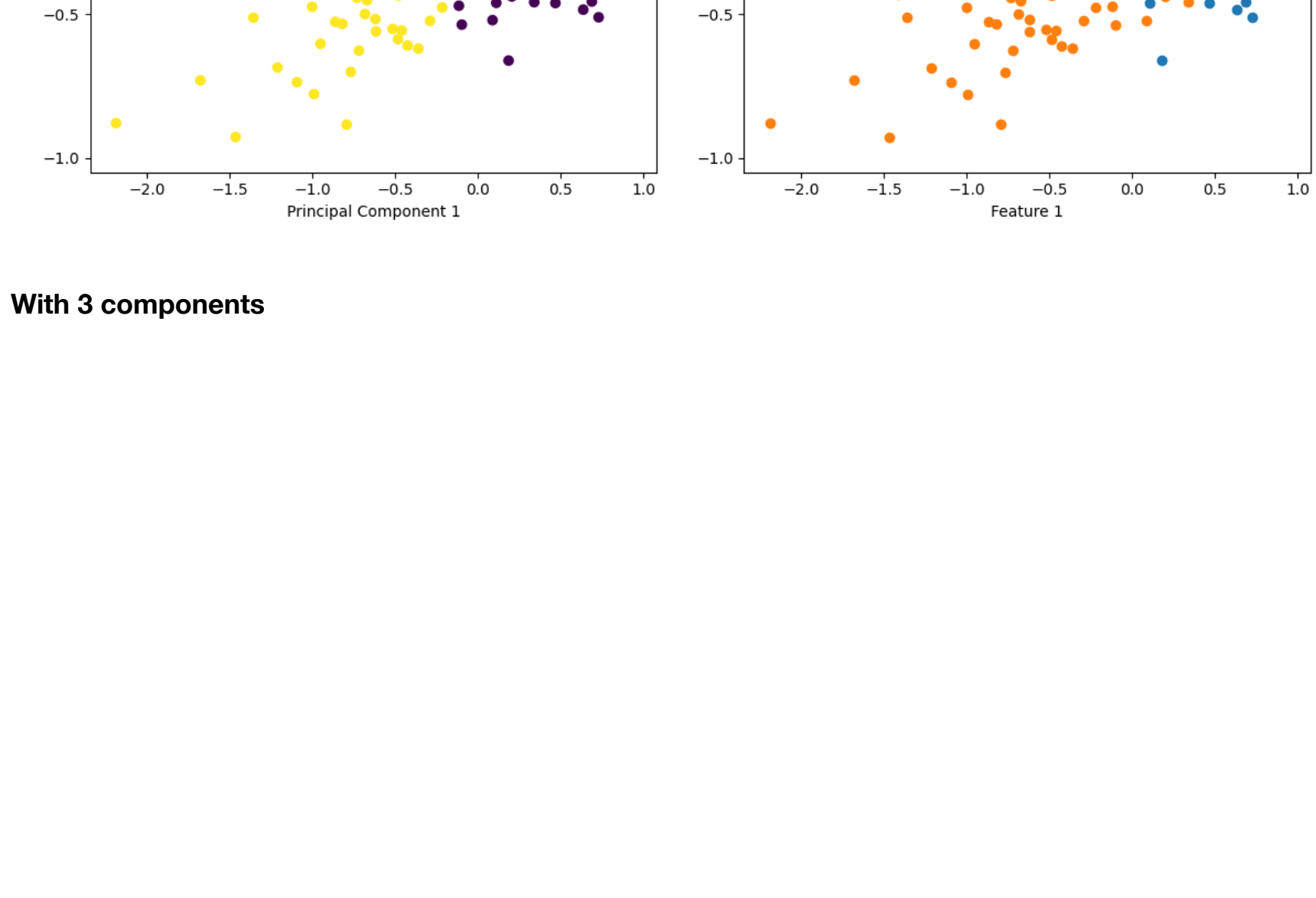
In [94]:
```python
print(f"SSE Experiment 1 (Without PCA): {round(sse_exp_1, 2)}")
print(f"SSE Experiment 2 (With PCA): {round(sse_exp_2, 2)}")
```

```
SSE Experiment 1 (Without PCA): 215.84
SSE Experiment 2 (With PCA): 110.97
```

Trying different principale components and visualizing the results

With 2 components

In [95]:
```python
n_components = 2

_, axes = plt.subplots(1, 2, figsize=(12, 8))

t_data = pca(features, n_components)
centroids, labels_pca = k_means(t_data, k_num=2)
ax = axes[0]

ax.scatter(t_data[:, 0], t_data[:, 1], c=labels_pca)
ax.set_title(f'PCA with {n_components} Components')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')

# Plot original labels
ax_original = axes[1]

map_dict = {0: 'B', 1: 'M'}
for i in [0, 1]:
    pca_points = t_data[np.where(target == i)]
    ax_original.scatter(pca_points[:, 0], pca_points[:, 1], label=map_dict[i])

ax_original.set_title('Original Labels')
ax_original.set_xlabel('Feature 1')
ax_original.set_ylabel('Feature 2')

plt.tight_layout()
plt.show()
```



With 3 components

In [961]:
```
from mpl_toolkits.mplot3d import Axes3D
n_components = 3

fig = plt.figure(figsize=(14, 6))

# Subplot for PCA with 3 components
ax_pca = fig.add_subplot(121, projection='3d')
x_pca = pca_features[:, :n_components]
y_labels_pca = k_means(k_data, k_num=2)
ax_pca.scatter(k_data[:, 0], k_data[:, 1], k_data[:, 2], c=labels_pca)
ax_pca.set_title('PCA with {} Components'.format(n_components))
ax_pca.set_xlabel('Principal Component 1')
ax_pca.set_ylabel('Principal Component 2')
ax_pca.set_zlabel('Principal Component 3')

# Subplot for Original Labels
ax_original = fig.add_subplot(122, projection='3d')

map_dict = {0: 'No', 1: 'Yes'}
for i in [0, 1]:
    pca_points = k_data[np.where(target == i)]
    ax_original.scatter(pca_points[:, 0], pca_points[:, 1], pca_points[:, 2], label=map_dict[i])

ax_original.set_title('Original Labels')
ax_original.set_xlabel('Feature 1')
ax_original.set_ylabel('Feature 2')
ax_original.set_zlabel('Feature 3')

plt.tight_layout()
plt.show()
```