In [1]:
```
# ============================================================================
# CELL 1: IMPORTS AND SETUP
# ============================================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA, FastICA
from sklearn.random_projection import GaussianRandomProjection
from sklearn.metrics import silhouette_score
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from scipy.stats import kurtosis
import time
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
plt.style.use('seaborn-v0_8')

print("="*70)
print("UNSUPERVISED LEARNING AND DIMENSIONALITY REDUCTION")
print("CS7641 Machine Learning - Summer 2025")
print("Max Matkovski")
print("="*70)
```

```
======================================================================
UNSUPERVISED LEARNING AND DIMENSIONALITY REDUCTION
CS7641 Machine Learning - Summer 2025
Max Matkovski
======================================================================
```

In [2]:
```
# ============================================================================
# CELL 2: DATA LOADING AND PREPROCESSING
# ============================================================================
print("\n" + "="*70)
print("DATA LOADING AND PREPROCESSING")
print("="*70)

# Load datasets
cancer_df = pd.read_csv('cancer.csv')
bankrupt_df = pd.read_csv('bankrupt.csv')

print("\n--- Cancer Dataset ---")
print(f"Original shape: {cancer_df.shape}")
print(f"Columns: {list(cancer_df.columns)}")

print("\n--- Bankruptcy Dataset ---")
print(f"Original shape: {bankrupt_df.shape}")
print(f"Number of columns: {len(bankrupt_df.columns)}")
```

```
# Clean and preprocess
cancer_df_clean = cancer_df.dropna()
bankrupt_df_clean = bankrupt_df.dropna()

# Select numeric features only
cancer_X = cancer_df_clean.select_dtypes(include=[np.number])
bankrupt_X = bankrupt_df_clean.select_dtypes(include=[np.number])

# Store labels for later use
cancer_labels = cancer_df_clean['Cancer_Type'].values
bankrupt_labels = bankrupt_df_clean['Bankrupt?'].values

print(f"\nCancer - Samples: {cancer_X.shape[0]}, Features: {cancer_X.shape[1
print(f"Cancer features: {list(cancer_X.columns)}")
print(f"\nBankruptcy - Samples: {bankrupt_X.shape[0]}, Features: {bankrupt_X

# Standardize features
scaler_cancer = StandardScaler()
scaler_bankrupt = StandardScaler()

cancer_X_scaled = scaler_cancer.fit_transform(cancer_X)
bankrupt_X_scaled = scaler_bankrupt.fit_transform(bankrupt_X)

print(f"\n✓ Data preprocessing complete")
print(f"  Cancer: {cancer_X_scaled.shape}")
print(f"  Bankruptcy: {bankrupt_X_scaled.shape}")
```

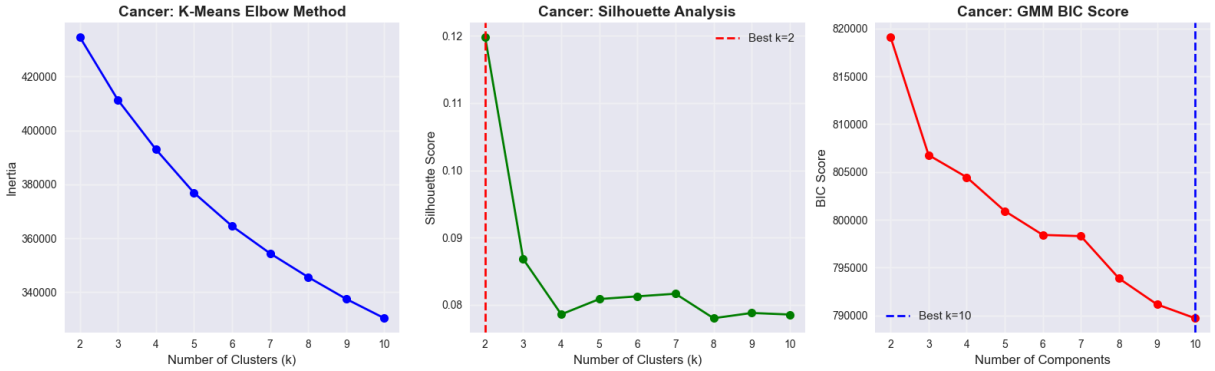
```
======================================================================
DATA LOADING AND PREPROCESSING
======================================================================

--- Cancer Dataset ---
Original shape: (50000, 15)
Columns: ['Patient_ID', 'Age', 'Gender', 'Country_Region', 'Year', 'Genetic_
Risk', 'Air_Pollution', 'Alcohol_Use', 'Smoking', 'Obesity_Level', 'Cancer_T
ype', 'Cancer_Stage', 'Treatment_Cost_USD', 'Survival_Years', 'Target_Severi
ty_Score']

--- Bankruptcy Dataset ---
Original shape: (6819, 96)
Number of columns: 96

Cancer - Samples: 50000, Features: 10
Cancer features: ['Age', 'Year', 'Genetic_Risk', 'Air_Pollution', 'Alcohol_U
se', 'Smoking', 'Obesity_Level', 'Treatment_Cost_USD', 'Survival_Years', 'Ta
rget_Severity_Score']

Bankruptcy - Samples: 6819, Features: 96

✓ Data preprocessing complete
  Cancer: (50000, 10)
  Bankruptcy: (6819, 96)
```

In [3]:
```
# ======================================================================
# CELL 3: STEP 1 - OPTIMAL K SELECTION
# ======================================================================
```

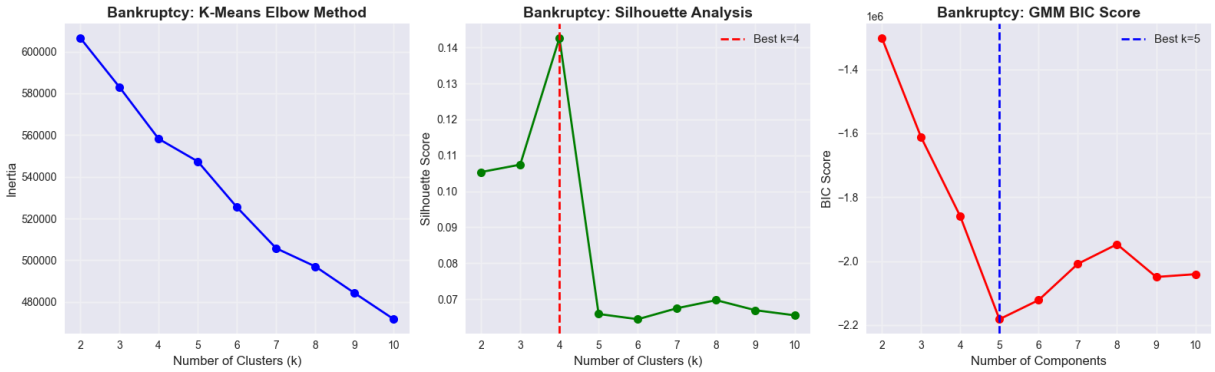
```
print("\n" + "="*70)
print("STEP 1: CLUSTERING ON ORIGINAL DATA")
print("="*70)

def find_optimal_k(X, max_k=10, dataset_name="Dataset"):
    """Find optimal k using elbow method, silhouette score, and BIC"""
    inertias = []
    silhouettes = []
    bics = []
    k_range = range(2, max_k + 1)

    for k in k_range:
        # K-Means
        km = KMeans(n_clusters=k, random_state=42, n_init=10)
        km.fit(X)
        inertias.append(km.inertia_)
        silhouettes.append(silhouette_score(X, km.labels_))

        # GMM
        gm = GaussianMixture(n_components=k, random_state=42, n_init=3)
        gm.fit(X)
        bics.append(gm.bic(X))

    # Create plots
    fig, axes = plt.subplots(1, 3, figsize=(16, 5))

    # Elbow plot
    axes[0].plot(list(k_range), inertias, 'bo-', linewidth=2, markersize=8)
    axes[0].set_xlabel('Number of Clusters (k)', fontsize=12)
    axes[0].set_ylabel('Inertia', fontsize=12)
    axes[0].set_title(f'{dataset_name}: K-Means Elbow Method', fontsize=14,
    axes[0].grid(True, alpha=0.3)

    # Silhouette plot
    axes[1].plot(list(k_range), silhouettes, 'go-', linewidth=2, markersize=
    axes[1].set_xlabel('Number of Clusters (k)', fontsize=12)
    axes[1].set_ylabel('Silhouette Score', fontsize=12)
    axes[1].set_title(f'{dataset_name}: Silhouette Analysis', fontsize=14, f
    best_k_sil = list(k_range)[np.argmax(silhouettes)]
    axes[1].axvline(x=best_k_sil, color='r', linestyle='--', linewidth=2,
                    label=f'Best k={best_k_sil}')
    axes[1].legend(fontsize=11)
    axes[1].grid(True, alpha=0.3)

    # BIC plot
    axes[2].plot(list(k_range), bics, 'ro-', linewidth=2, markersize=8)
    axes[2].set_xlabel('Number of Components', fontsize=12)
    axes[2].set_ylabel('BIC Score', fontsize=12)
    axes[2].set_title(f'{dataset_name}: GMM BIC Score', fontsize=14, fontwei
    best_k_bic = list(k_range)[np.argmin(bics)]
    axes[2].axvline(x=best_k_bic, color='b', linestyle='--', linewidth=2,
                    label=f'Best k={best_k_bic}')
    axes[2].legend(fontsize=11)
    axes[2].grid(True, alpha=0.3)

    plt.tight_layout()
```

```
    plt.savefig(f'{dataset_name.lower()}_optimal_k.png', dpi=150, bbox_inche
    plt.show()

    print(f"\n{dataset_name} Optimal k Analysis:")
    print(f"  Best k by Silhouette: {best_k_sil} (score: {max(silhouettes):.
    print(f"  Best k by BIC: {best_k_bic}")

    return best_k_sil, best_k_bic, silhouettes, bics

# Find optimal k for both datasets
print("\n--- Cancer Dataset: Optimal k Selection ---")
best_k_cancer, _, cancer_sils, cancer_bics = find_optimal_k(cancer_X_scaled,

print("\n--- Bankruptcy Dataset: Optimal k Selection ---")
best_k_bankrupt, _, bankrupt_sils, bankrupt_bics = find_optimal_k(bankrupt_X
```

```
==========================================================================
STEP 1: CLUSTERING ON ORIGINAL DATA
==========================================================================

--- Cancer Dataset: Optimal k Selection ---
```




```
Cancer Optimal k Analysis:
  Best k by Silhouette: 2 (score: 0.1198)
  Best k by BIC: 10

--- Bankruptcy Dataset: Optimal k Selection ---
```




```
Bankruptcy Optimal k Analysis:
  Best k by Silhouette: 4 (score: 0.1427)
  Best k by BIC: 5
```

In [4]:
```
# ============================================================================
# CELL 4: PERFORM CLUSTERING ON ORIGINAL DATA
# ============================================================================
print("\n--- Clustering on Original Data ---")
```

```python
# Use k=3 for consistency
K = 3

def cluster_original(X, k, dataset_name):
    """Perform K-Means and GMM clustering on original data"""
    # K-Means
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km_labels = km.fit_predict(X)
    km_sil = silhouette_score(X, km_labels)

    # GMM
    gm = GaussianMixture(n_components=k, random_state=42, n_init=3)
    gm_labels = gm.fit_predict(X)
    gm_sil = silhouette_score(X, gm_labels)

    print(f"\n{dataset_name} Original Data Clustering (k={k}):")
    print(f"  K-Means Silhouette: {km_sil:.4f}")
    print(f"  K-Means Inertia: {km.inertia_:.2f}")
    print(f"  GMM Silhouette: {gm_sil:.4f}")
    print(f"  GMM BIC: {gm.bic(X):.2f}")

    return km_labels, gm_labels, km_sil, gm_sil

# Cluster both datasets
cancer_km_orig, cancer_gm_orig, cancer_km_sil_orig, cancer_gm_sil_orig = \
    cluster_original(cancer_X_scaled, K, "Cancer")

bankrupt_km_orig, bankrupt_gm_orig, bankrupt_km_sil_orig, bankrupt_gm_sil_or
    cluster_original(bankrupt_X_scaled, K, "Bankruptcy")

# Initialize results storage for comprehensive table
all_results = []

# Add original results
all_results.extend([
    {'Dataset': 'Cancer', 'DR_Method': 'None', 'Clustering': 'K-Means',
     'Silhouette': cancer_km_sil_orig},
    {'Dataset': 'Cancer', 'DR_Method': 'None', 'Clustering': 'GMM',
     'Silhouette': cancer_gm_sil_orig},
    {'Dataset': 'Bankruptcy', 'DR_Method': 'None', 'Clustering': 'K-Means',
     'Silhouette': bankrupt_km_sil_orig},
    {'Dataset': 'Bankruptcy', 'DR_Method': 'None', 'Clustering': 'GMM',
     'Silhouette': bankrupt_gm_sil_orig}
])
```

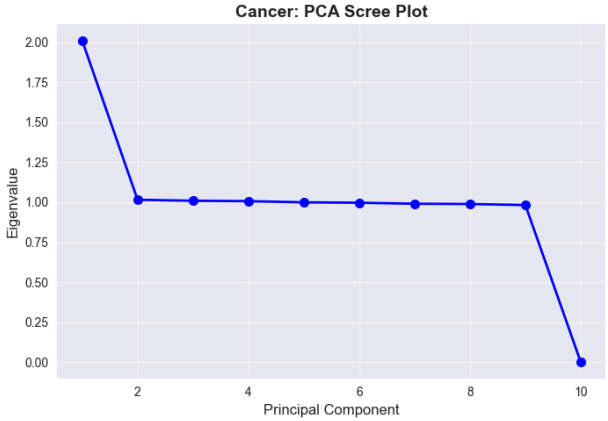
```
--- Clustering on Original Data ---

Cancer Original Data Clustering (k=3):
  K-Means Silhouette: 0.0868
  K-Means Inertia: 411379.10
  GMM Silhouette: 0.0722
  GMM BIC: 806766.82

Bankruptcy Original Data Clustering (k=3):
  K-Means Silhouette: 0.1074
  K-Means Inertia: 583013.28
  GMM Silhouette: 0.0338
  GMM BIC: -1611911.63
```

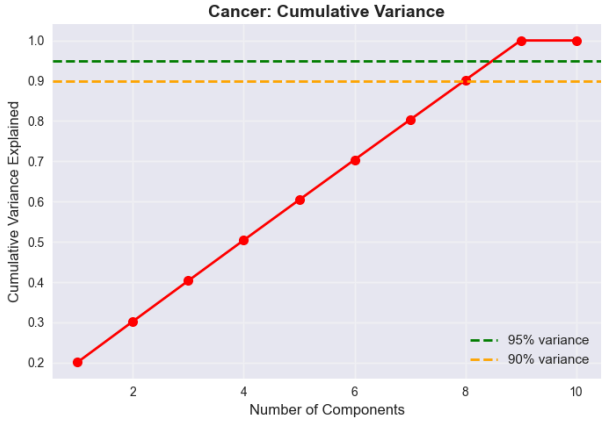
In [5]:

```python
# ============================================================================
# CELL 5: STEP 2a - PCA ANALYSIS WITH SCREE PLOT
# ============================================================================
print("\n" + "="*70)
print("STEP 2: DIMENSIONALITY REDUCTION ANALYSIS")
print("="*70)

def analyze_pca(X, dataset_name):
    """Complete PCA analysis with scree plot"""
    n_comp = min(X.shape[0], X.shape[1])
    pca = PCA(n_components=n_comp, random_state=42)
    pca.fit(X)

    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Scree plot
    eigenvalues = pca.explained_variance_
    n_show = min(20, len(eigenvalues))

    axes[0].plot(range(1, n_show+1), eigenvalues[:n_show], 'bo-', linewidth=
    axes[0].set_xlabel('Principal Component', fontsize=12)
    axes[0].set_ylabel('Eigenvalue', fontsize=12)
    axes[0].set_title(f'{dataset_name}: PCA Scree Plot', fontsize=14, fontwe
    axes[0].grid(True, alpha=0.3)

    # Cumulative variance
    cumvar = np.cumsum(pca.explained_variance_ratio_)
    axes[1].plot(range(1, n_show+1), cumvar[:n_show], 'ro-', linewidth=2, ma
    axes[1].axhline(y=0.95, color='g', linestyle='--', linewidth=2, label='9
    axes[1].axhline(y=0.90, color='orange', linestyle='--', linewidth=2, lab
    axes[1].set_xlabel('Number of Components', fontsize=12)
    axes[1].set_ylabel('Cumulative Variance Explained', fontsize=12)
    axes[1].set_title(f'{dataset_name}: Cumulative Variance', fontsize=14, f
    axes[1].legend(fontsize=11, loc='lower right')
    axes[1].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig(f'{dataset_name.lower()}_pca_scree.png', dpi=150, bbox_inche
    plt.show()

    n_90 = np.argmax(cumvar >= 0.90) + 1
    n_95 = np.argmax(cumvar >= 0.95) + 1
```

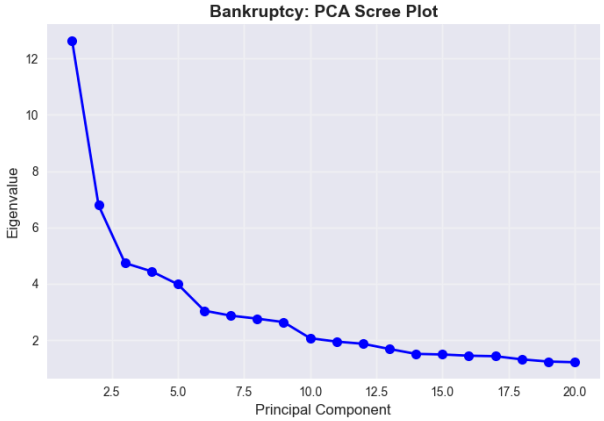
```
    print(f"\n{dataset_name} PCA Analysis:")
    print(f"  Total components: {len(eigenvalues)}")
    print(f"  Components for 90% variance: {n_90}")
    print(f"  Components for 95% variance: {n_95}")
    print(f"  Variance by first 2 PCs: {cumvar[1]*100:.2f}%")
    print(f"  PC1 variance: {pca.explained_variance_ratio_[0]*100:.2f}%")
    print(f"  PC2 variance: {pca.explained_variance_ratio_[1]*100:.2f}%")

    return pca, cumvar

print("\n--- PCA Analysis ---")
pca_cancer_full, cancer_cumvar = analyze_pca(cancer_X_scaled, "Cancer")
pca_bankrupt_full, bankrupt_cumvar = analyze_pca(bankrupt_X_scaled, "Bankrup
```

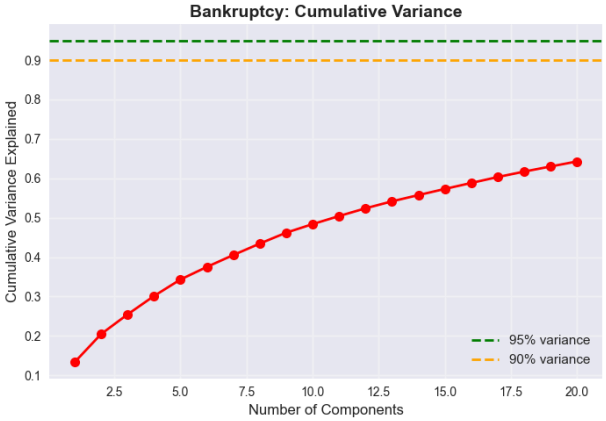
```
======================================================================
STEP 2: DIMENSIONALITY REDUCTION ANALYSIS
======================================================================

--- PCA Analysis ---
```






```
Cancer PCA Analysis:
  Total components: 10
  Components for 90% variance: 8
  Components for 95% variance: 9
  Variance by first 2 PCs: 30.26%
  PC1 variance: 20.11%
  PC2 variance: 10.15%
```





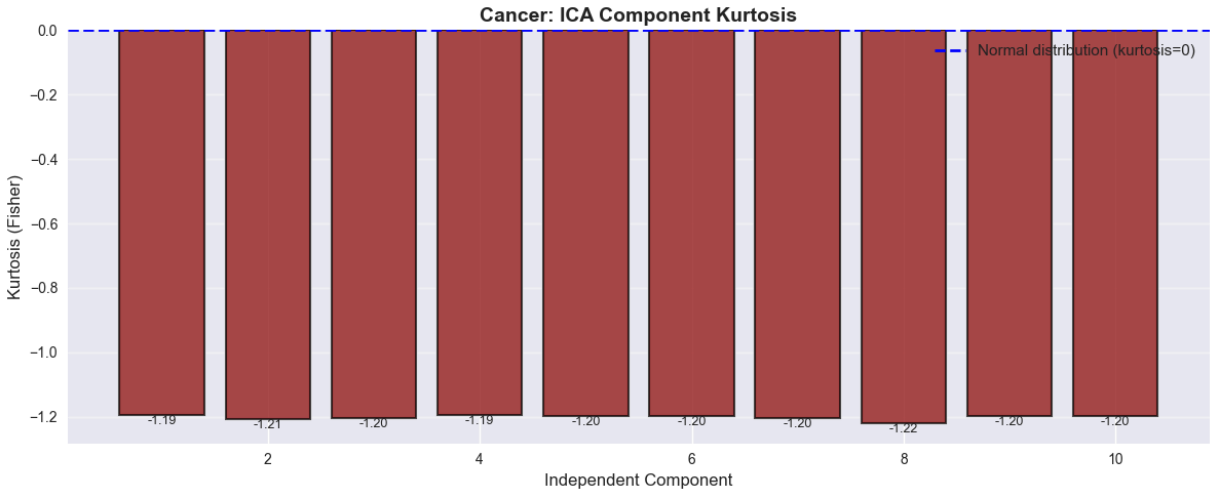
```
Bankruptcy PCA Analysis:
  Total components: 96
  Components for 90% variance: 46
  Components for 95% variance: 54
  Variance by first 2 PCs: 20.45%
  PC1 variance: 13.30%
  PC2 variance: 7.14%
```

In [6]:

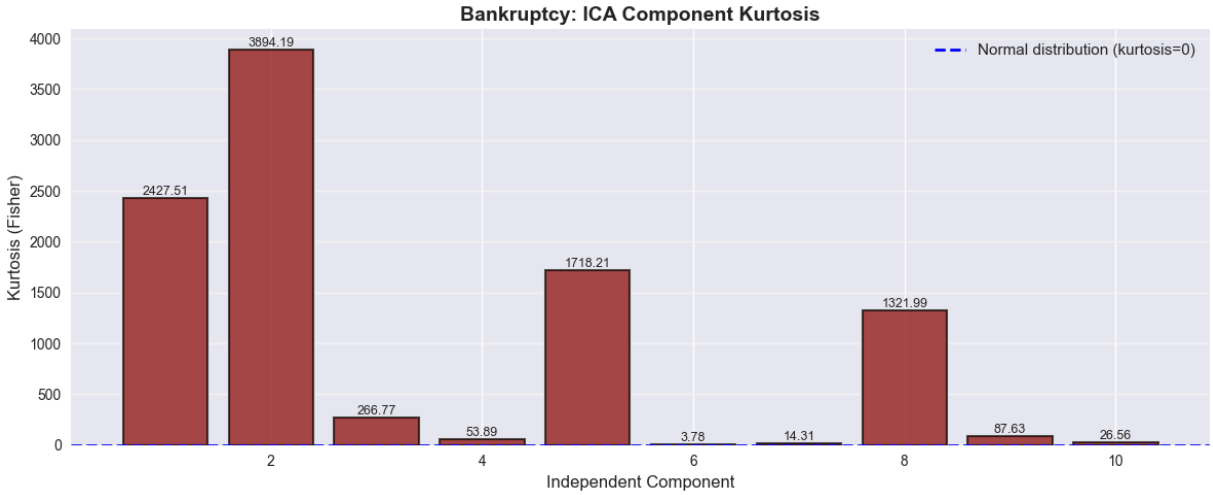
```python
# ============================================================================
# CELL 6: STEP 2b - ICA KURTOSIS ANALYSIS
# ============================================================================
def analyze_ica(X, dataset_name, n_components=10):
    """ICA analysis with kurtosis visualization"""
    n_comp = min(n_components, X.shape[1])

    ica = FastICA(n_components=n_comp, random_state=42, max_iter=1000)
    X_ica = ica.fit_transform(X)

    # Calculate kurtosis
    kurt_vals = [kurtosis(X_ica[:, i], fisher=True) for i in range(n_comp)]

    # Plot
    fig, ax = plt.subplots(figsize=(12, 5))

    colors = ['darkred' if abs(k) > 1 else 'orange' if abs(k) > 0.5 else 'gr
              for k in kurt_vals]

    bars = ax.bar(range(1, n_comp+1), kurt_vals, color=colors, alpha=0.7,
                  edgecolor='black', linewidth=1.5)
    ax.axhline(y=0, color='blue', linestyle='--', linewidth=2,
               label='Normal distribution (kurtosis=0)')
    ax.set_xlabel('Independent Component', fontsize=12)
    ax.set_ylabel('Kurtosis (Fisher)', fontsize=12)
    ax.set_title(f'{dataset_name}: ICA Component Kurtosis', fontsize=14, for
    ax.legend(fontsize=11)
    ax.grid(True, alpha=0.3, axis='y')

    # Add value labels
    for bar, val in zip(bars, kurt_vals):
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{val:.2f}', ha='center', va='bottom' if height > 0 else 't

    plt.tight_layout()
    plt.savefig(f'{dataset_name.lower()}_ica_kurtosis.png', dpi=150, bbox_ir
    plt.show()

    print(f"\n{dataset_name} ICA Kurtosis Analysis:")
    print(f"  Components analyzed: {n_comp}")
    print(f"  Mean |kurtosis|: {np.mean(np.abs(kurt_vals)):.4f}")
    print(f"  Max kurtosis: {np.max(kurt_vals):.4f}")
    print(f"  Components with |kurtosis| > 1: {sum(np.abs(kurt_vals) > 1)}")
    print(f"  Components with |kurtosis| > 0.5: {sum(np.abs(kurt_vals) > 0.5

    return kurt_vals
```

```
print("\n--- ICA Kurtosis Analysis ---")
cancer_kurt = analyze_ica(cancer_X_scaled, "Cancer", n_components=10)
bankrupt_kurt = analyze_ica(bankrupt_X_scaled, "Bankruptcy", n_components=10
```

```
--- ICA Kurtosis Analysis ---
```




```
Cancer ICA Kurtosis Analysis:
  Components analyzed: 10
  Mean |kurtosis|: 1.2012
  Max kurtosis: -1.1930
  Components with |kurtosis| > 1: 10
  Components with |kurtosis| > 0.5: 10
```



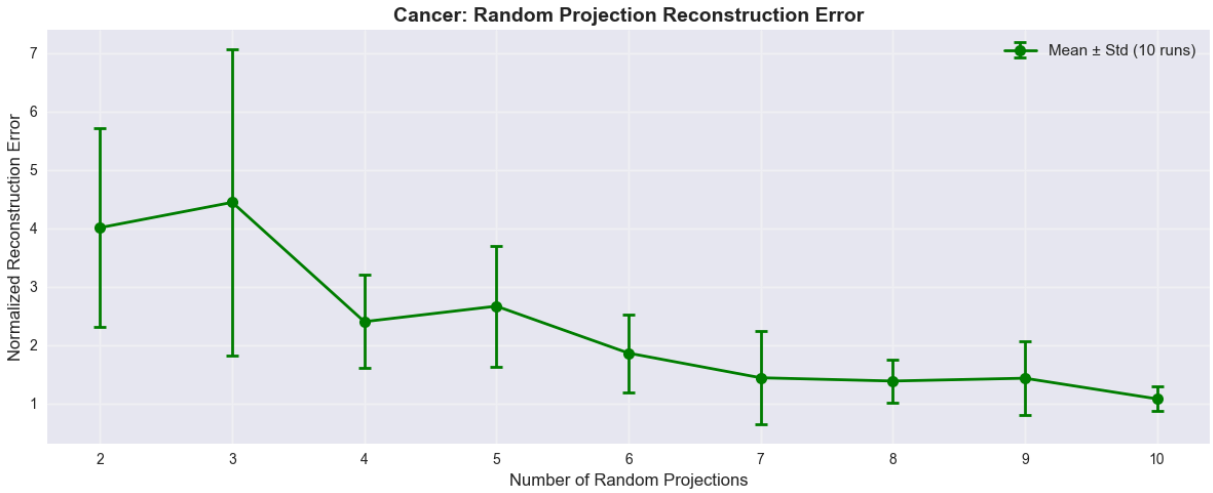

```
Bankruptcy ICA Kurtosis Analysis:
  Components analyzed: 10
  Mean |kurtosis|: 981.4846
  Max kurtosis: 3894.1875
  Components with |kurtosis| > 1: 10
  Components with |kurtosis| > 0.5: 10
```

In [7]:

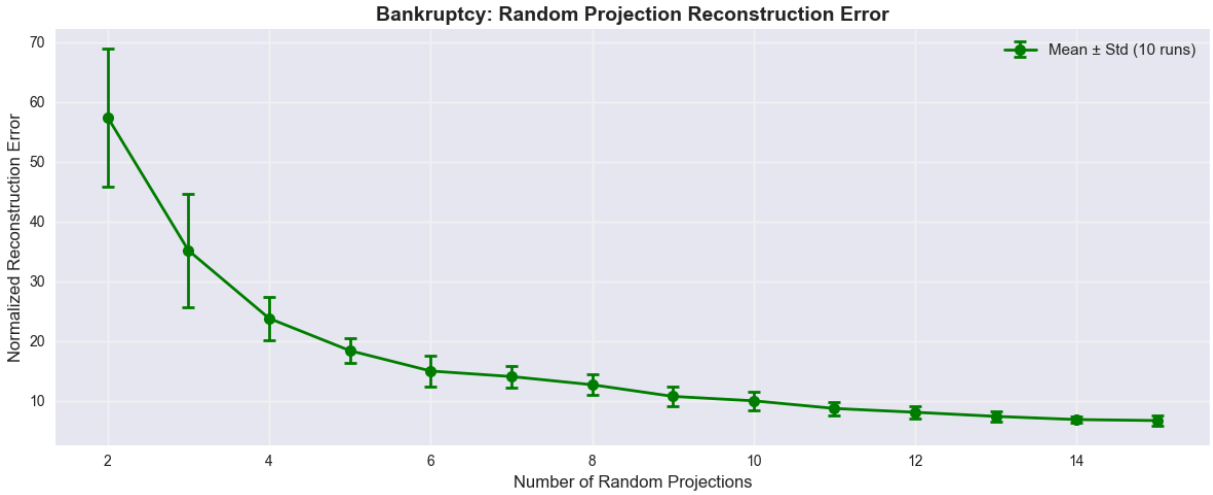
```
# ============================================================================
# CELL 7: STEP 2c - RANDOM PROJECTION RECONSTRUCTION ERROR
# ============================================================================
def analyze_rp(X, dataset_name, n_runs=10):
    """Random projection reconstruction error analysis"""
    max_comp = min(X.shape[1], 15)
    n_comp_range = range(2, max_comp + 1)
```

```
    mean_errors = []
    std_errors = []

    print(f"Analyzing Random Projection for {dataset_name}...")

    for n_comp in n_comp_range:
        errors = []
        for _ in range(n_runs):
            rp = GaussianRandomProjection(n_components=n_comp)
            X_rp = rp.fit_transform(X)
            X_reconstructed = X_rp @ rp.components_
            error = np.mean((X - X_reconstructed) ** 2) / np.mean(X ** 2)
            errors.append(error)

        mean_errors.append(np.mean(errors))
        std_errors.append(np.std(errors))

    # Plot
    fig, ax = plt.subplots(figsize=(12, 5))
    ax.errorbar(list(n_comp_range), mean_errors, yerr=std_errors,
                fmt='go-', capsize=4, capthick=2, linewidth=2, markersize=8,
                label=f'Mean ± Std ({n_runs} runs)', elinewidth=2)
    ax.set_xlabel('Number of Random Projections', fontsize=12)
    ax.set_ylabel('Normalized Reconstruction Error', fontsize=12)
    ax.set_title(f'{dataset_name}: Random Projection Reconstruction Error',
                 fontsize=14, fontweight='bold')
    ax.legend(fontsize=11)
    ax.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig(f'{dataset_name.lower()}_rp_reconstruction.png', dpi=150, bb
    plt.show()

    print(f"\n{dataset_name} Random Projection Analysis:")
    print(f"  Error with 2 components: {mean_errors[0]:.4f} ± {std_errors[0]
    print(f"  Error with 5 components: {mean_errors[3]:.4f} ± {std_errors[3]
    if len(mean_errors) > 8:
        print(f"  Error with 10 components: {mean_errors[8]:.4f} ± {std_erro

    return mean_errors, std_errors

print("\n--- Random Projection Analysis ---")
rp_cancer_err, rp_cancer_std = analyze_rp(cancer_X_scaled, "Cancer", n_runs=
rp_bankrupt_err, rp_bankrupt_std = analyze_rp(bankrupt_X_scaled, "Bankruptcy
```

```
--- Random Projection Analysis ---
Analyzing Random Projection for Cancer...
```




```
Cancer Random Projection Analysis:
  Error with 2 components: 4.0147 ± 1.7039
  Error with 5 components: 2.6694 ± 1.0391
  Error with 10 components: 1.0816 ± 0.2132
Analyzing Random Projection for Bankruptcy...
```




```
Bankruptcy Random Projection Analysis:
  Error with 2 components: 57.3973 ± 11.5976
  Error with 5 components: 18.3481 ± 2.0723
  Error with 10 components: 9.9535 ± 1.5295
```

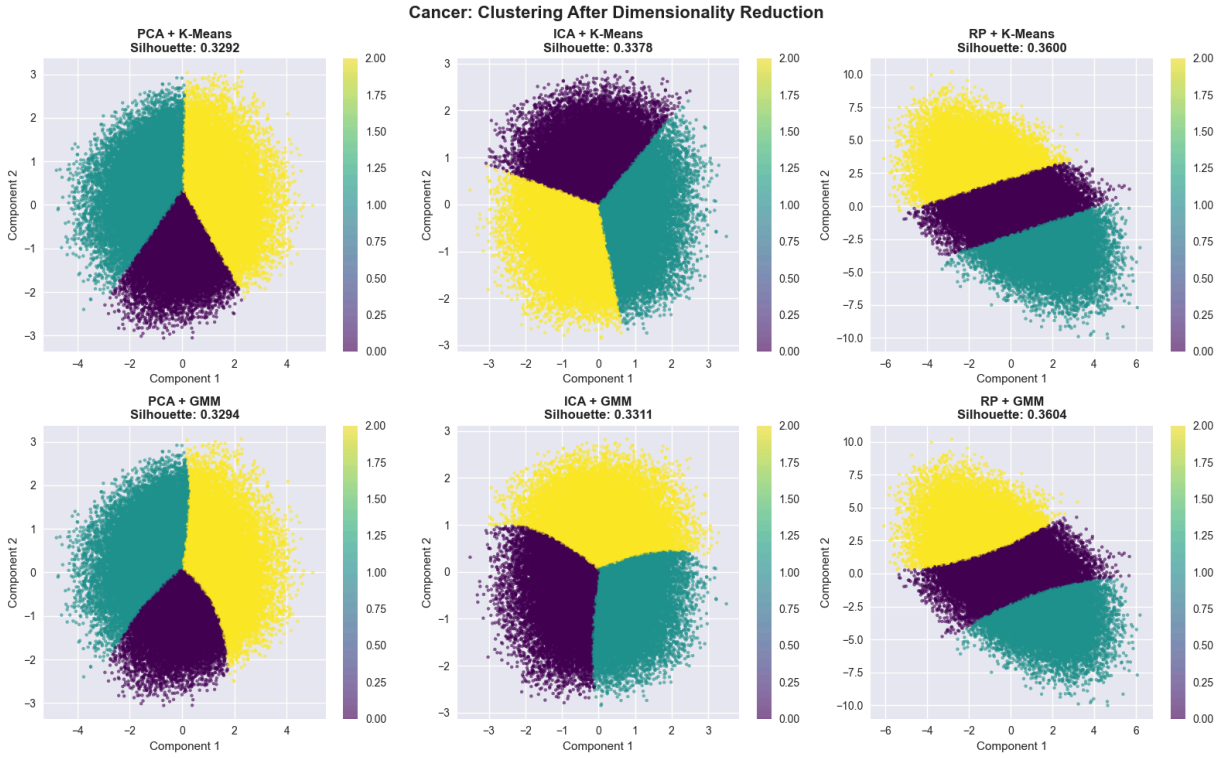
In [8]:

```python
# ============================================================================
# CELL 8: STEP 3 - CLUSTERING AFTER DR (ALL 12 COMBINATIONS)
# ============================================================================
print("\n" + "="*70)
print("STEP 3: CLUSTERING AFTER DIMENSIONALITY REDUCTION")
print("="*70)

def dr_and_cluster(X, k, dataset_name, n_components=2):
    """Apply DR methods then cluster - returns all results and visualizatior

    # Apply DR
    pca = PCA(n_components=n_components, random_state=42)
    X_pca = pca.fit_transform(X)

    ica = FastICA(n_components=n_components, random_state=42, max_iter=1000)
    X_ica = ica.fit_transform(X)
```

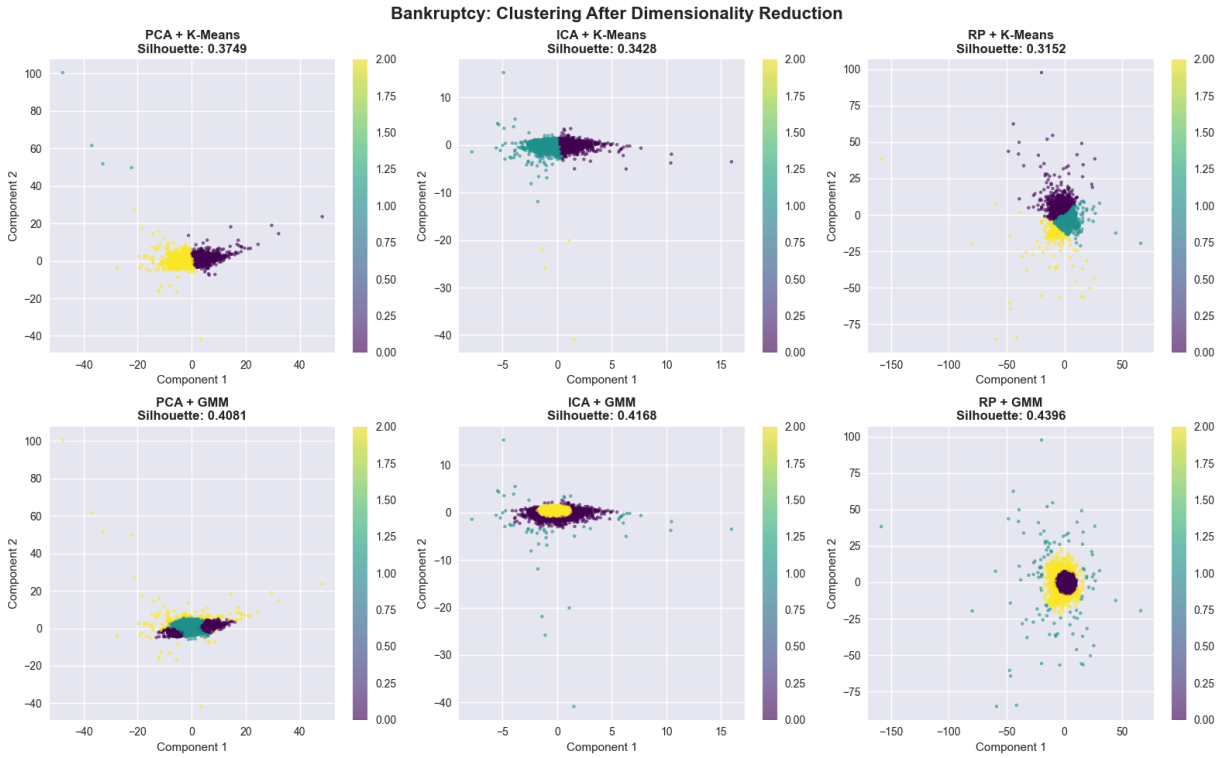
```
    rp = GaussianRandomProjection(n_components=n_components, random_state=42
    X_rp = rp.fit_transform(X)

    dr_data = {'PCA': X_pca, 'ICA': X_ica, 'RP': X_rp}
    results = {}

    print(f"\n{dataset_name} Dataset (k={k}, n_components={n_components}):")
    print("-" * 60)

    for dr_name, X_t in dr_data.items():
        results[dr_name] = {}

        # K-Means
        km = KMeans(n_clusters=k, random_state=42, n_init=10)
        km_labels = km.fit_predict(X_t)
        km_sil = silhouette_score(X_t, km_labels)
        results[dr_name]['kmeans'] = {'labels': km_labels, 'silhouette': km_

        # GMM
        gm = GaussianMixture(n_components=k, random_state=42, n_init=3)
        gm_labels = gm.fit_predict(X_t)
        gm_sil = silhouette_score(X_t, gm_labels)
        results[dr_name]['gmm'] = {'labels': gm_labels, 'silhouette': gm_sil

        print(f"  {dr_name}: K-Means={km_sil:.4f}, GMM={gm_sil:.4f}")

        # Add to global results
        all_results.append({'Dataset': dataset_name, 'DR_Method': dr_name,
                            'Clustering': 'K-Means', 'Silhouette': km_sil})
        all_results.append({'Dataset': dataset_name, 'DR_Method': dr_name,
                            'Clustering': 'GMM', 'Silhouette': gm_sil})

    # Visualization
    fig, axes = plt.subplots(2, 3, figsize=(16, 10))

    for idx, (dr_name, X_t) in enumerate(dr_data.items()):
        # K-Means
        sc = axes[0, idx].scatter(X_t[:, 0], X_t[:, 1],
                                  c=results[dr_name]['kmeans']['labels'],
                                  cmap='viridis', alpha=0.6, s=8)
        axes[0, idx].set_title(f'{dr_name} + K-Means\nSilhouette: {results[d
                               fontsize=12, fontweight='bold')
        axes[0, idx].set_xlabel('Component 1')
        axes[0, idx].set_ylabel('Component 2')
        plt.colorbar(sc, ax=axes[0, idx])

        # GMM
        sc = axes[1, idx].scatter(X_t[:, 0], X_t[:, 1],
                                  c=results[dr_name]['gmm']['labels'],
                                  cmap='viridis', alpha=0.6, s=8)
        axes[1, idx].set_title(f'{dr_name} + GMM\nSilhouette: {results[dr_na
                               fontsize=12, fontweight='bold')
        axes[1, idx].set_xlabel('Component 1')
        axes[1, idx].set_ylabel('Component 2')
        plt.colorbar(sc, ax=axes[1, idx])
```

```
    plt.suptitle(f'{dataset_name}: Clustering After Dimensionality Reductior
                 fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.savefig(f'{dataset_name.lower()}_dr_clustering.png', dpi=150, bbox_i
    plt.show()

    return results, dr_data

# Apply to BOTH datasets
cancer_dr_results, cancer_dr_data = dr_and_cluster(cancer_X_scaled, K, "Canc
bankrupt_dr_results, bankrupt_dr_data = dr_and_cluster(bankrupt_X_scaled, K,
```

```
======================================================================
STEP 3: CLUSTERING AFTER DIMENSIONALITY REDUCTION
======================================================================

Cancer Dataset (k=3, n_components=2):
------------------------------------------------------------
  PCA: K-Means=0.3292, GMM=0.3294
  ICA: K-Means=0.3378, GMM=0.3311
  RP: K-Means=0.3600, GMM=0.3604
```




```
Bankruptcy Dataset (k=3, n_components=2):
------------------------------------------------------------
  PCA: K-Means=0.3749, GMM=0.4081
  ICA: K-Means=0.3428, GMM=0.4168
  RP: K-Means=0.3152, GMM=0.4396
```



In [9]:

```
# ============================================================================
# CELL 9: COMPREHENSIVE RESULTS TABLE
# ============================================================================
print("\n" + "="*70)
print("COMPREHENSIVE RESULTS TABLE")
print("="*70)

results_df = pd.DataFrame(all_results)
print(results_df.to_string(index=False))

# Summary statistics
print("\n--- Summary by Dataset and DR Method ---")
summary = results_df.groupby(['Dataset', 'DR_Method'])['Silhouette'].agg(['m
print(summary)

# Best combinations
print("\n--- Top 5 Combinations by Silhouette Score ---")
print(results_df.nlargest(5, 'Silhouette').to_string(index=False))

# Save results
results_df.to_csv('clustering_results.csv', index=False)
print("\n✓ Results saved to clustering_results.csv")
```

```
======================================================================
COMPREHENSIVE RESULTS TABLE
======================================================================
   Dataset DR_Method Clustering  Silhouette
    Cancer      None    K-Means    0.086816
    Cancer      None        GMM    0.072227
Bankruptcy      None    K-Means    0.107412
Bankruptcy      None        GMM    0.033795
    Cancer       PCA    K-Means    0.329181
    Cancer       PCA        GMM    0.329410
    Cancer       ICA    K-Means    0.337764
    Cancer       ICA        GMM    0.331061
    Cancer        RP    K-Means    0.359981
    Cancer        RP        GMM    0.360354
Bankruptcy       PCA    K-Means    0.374890
Bankruptcy       PCA        GMM    0.408054
Bankruptcy       ICA    K-Means    0.342803
Bankruptcy       ICA        GMM    0.416847
Bankruptcy        RP    K-Means    0.315174
Bankruptcy        RP        GMM    0.439565

--- Summary by Dataset and DR Method ---
                         mean     max
Dataset    DR_Method
Bankruptcy ICA         0.3798  0.4168
           None        0.0706  0.1074
           PCA         0.3915  0.4081
           RP          0.3774  0.4396
Cancer     ICA         0.3344  0.3378
           None        0.0795  0.0868
           PCA         0.3293  0.3294
           RP          0.3602  0.3604

--- Top 5 Combinations by Silhouette Score ---
   Dataset DR_Method Clustering  Silhouette
Bankruptcy        RP        GMM    0.439565
Bankruptcy       ICA        GMM    0.416847
Bankruptcy       PCA        GMM    0.408054
Bankruptcy       PCA    K-Means    0.374890
    Cancer        RP        GMM    0.360354

✓ Results saved to clustering_results.csv
```

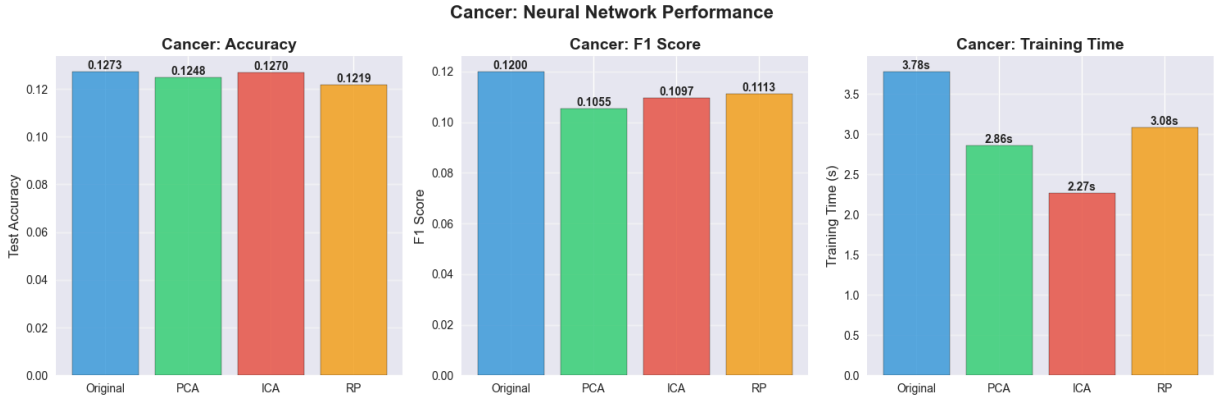
In [10]:

```python
# ======================================================================
# CELL 10: STEP 4 - NEURAL NETWORK WITH DR
# ======================================================================
print("\n" + "="*70)
print("STEP 4: NEURAL NETWORK WITH DIMENSIONALITY REDUCTION")
print("="*70)

def nn_analysis(X, y, dataset_name, n_components=2):
    """NN performance with timing analysis"""

    le = LabelEncoder()
    y_enc = le.fit_transform(y)
```

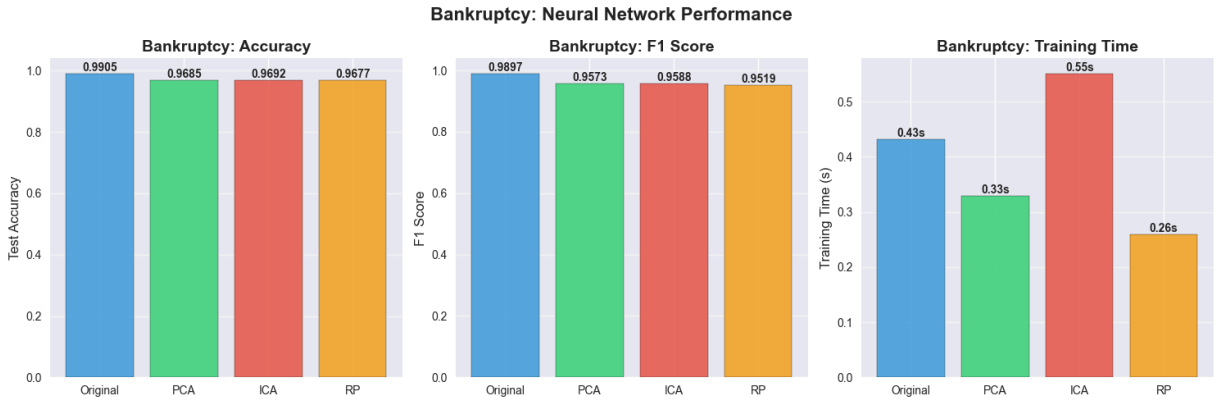
```
    X_train, X_test, y_train, y_test = train_test_split(
        X, y_enc, test_size=0.2, random_state=42, stratify=y_enc
    )

    nn_config = {
        'hidden_layer_sizes': (100, 50),
        'max_iter': 500,
        'random_state': 42,
        'early_stopping': True,
        'validation_fraction': 0.1
    }

    results = {}

    print(f"\n{dataset_name} Neural Network Performance:")
    print("-" * 60)

    # Original
    t0 = time.time()
    mlp = MLPClassifier(**nn_config)
    mlp.fit(X_train, y_train)
    orig_time = time.time() - t0
    orig_acc = mlp.score(X_test, y_test)
    orig_f1 = f1_score(y_test, mlp.predict(X_test), average='weighted')

    results['Original'] = {'accuracy': orig_acc, 'f1': orig_f1, 'time': orig
    print(f"  Original ({X.shape[1]} features):")
    print(f"    Accuracy: {orig_acc:.4f}, F1: {orig_f1:.4f}, Time: {orig_tim

    # DR methods
    for dr_name, dr_model in [
        ('PCA', PCA(n_components=n_components, random_state=42)),
        ('ICA', FastICA(n_components=n_components, random_state=42, max_iter
        ('RP', GaussianRandomProjection(n_components=n_components, random_st
    ]:
        X_train_dr = dr_model.fit_transform(X_train)
        X_test_dr = dr_model.transform(X_test)

        t0 = time.time()
        mlp_dr = MLPClassifier(**nn_config)
        mlp_dr.fit(X_train_dr, y_train)
        dr_time = time.time() - t0
        dr_acc = mlp_dr.score(X_test_dr, y_test)
        dr_f1 = f1_score(y_test, mlp_dr.predict(X_test_dr), average='weighte

        results[dr_name] = {'accuracy': dr_acc, 'f1': dr_f1, 'time': dr_time
        print(f"\n  {dr_name} ({n_components} features):")
        print(f"    Accuracy: {dr_acc:.4f}, F1: {dr_f1:.4f}, Time: {dr_time:
        print(f"    Speedup: {orig_time/dr_time:.2f}x")

    # Visualization
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    methods = list(results.keys())
    accs = [results[m]['accuracy'] for m in methods]
    f1s = [results[m]['f1'] for m in methods]
```

```
    times = [results[m]['time'] for m in methods]
    colors = ['#3498db', '#2ecc71', '#e74c3c', '#f39c12']

    # Accuracy
    bars = axes[0].bar(methods, accs, color=colors, alpha=0.8, edgecolor='bl
    axes[0].set_ylabel('Test Accuracy', fontsize=12)
    axes[0].set_title(f'{dataset_name}: Accuracy', fontsize=14, fontweight='
    axes[0].grid(True, alpha=0.3, axis='y')
    for bar, acc in zip(bars, accs):
        axes[0].text(bar.get_x() + bar.get_width()/2, bar.get_height(),
                     f'{acc:.4f}', ha='center', va='bottom', fontsize=10, for

    # F1 Score
    bars = axes[1].bar(methods, f1s, color=colors, alpha=0.8, edgecolor='bla
    axes[1].set_ylabel('F1 Score', fontsize=12)
    axes[1].set_title(f'{dataset_name}: F1 Score', fontsize=14, fontweight='
    axes[1].grid(True, alpha=0.3, axis='y')
    for bar, f1 in zip(bars, f1s):
        axes[1].text(bar.get_x() + bar.get_width()/2, bar.get_height(),
                     f'{f1:.4f}', ha='center', va='bottom', fontsize=10, font

    # Time
    bars = axes[2].bar(methods, times, color=colors, alpha=0.8, edgecolor='b
    axes[2].set_ylabel('Training Time (s)', fontsize=12)
    axes[2].set_title(f'{dataset_name}: Training Time', fontsize=14, fontwei
    axes[2].grid(True, alpha=0.3, axis='y')
    for bar, t in zip(bars, times):
        axes[2].text(bar.get_x() + bar.get_width()/2, bar.get_height(),
                     f'{t:.2f}s', ha='center', va='bottom', fontsize=10, font

    plt.suptitle(f'{dataset_name}: Neural Network Performance', fontsize=16,
    plt.tight_layout()
    plt.savefig(f'{dataset_name.lower()}_nn_performance.png', dpi=150, bbox_
    plt.show()

    return results

# Run NN analysis
cancer_nn = nn_analysis(cancer_X_scaled, cancer_labels, "Cancer", n_componer
bankrupt_nn = nn_analysis(bankrupt_X_scaled, bankrupt_labels, "Bankruptcy",
```

```
================================================================================
STEP 4: NEURAL NETWORK WITH DIMENSIONALITY REDUCTION
================================================================================

Cancer Neural Network Performance:
------------------------------------------------------------
  Original (10 features):
    Accuracy: 0.1273, F1: 0.1200, Time: 3.78s

  PCA (2 features):
    Accuracy: 0.1248, F1: 0.1055, Time: 2.86s
    Speedup: 1.32x

  ICA (2 features):
    Accuracy: 0.1270, F1: 0.1097, Time: 2.27s
    Speedup: 1.67x

  RP (2 features):
    Accuracy: 0.1219, F1: 0.1113, Time: 3.08s
    Speedup: 1.23x
```




```
Bankruptcy Neural Network Performance:
------------------------------------------------------------
  Original (96 features):
    Accuracy: 0.9905, F1: 0.9897, Time: 0.43s

  PCA (2 features):
    Accuracy: 0.9685, F1: 0.9573, Time: 0.33s
    Speedup: 1.31x

  ICA (2 features):
    Accuracy: 0.9692, F1: 0.9588, Time: 0.55s
    Speedup: 0.78x

  RP (2 features):
    Accuracy: 0.9677, F1: 0.9519, Time: 0.26s
    Speedup: 1.66x
```



In [11]:

```
# ============================================================================
# CELL 11: HYPOTHESIS VALIDATION
# ============================================================================
print("\n" + "="*70)
print("HYPOTHESIS VALIDATION")
print("="*70)

# Hypothesis 1
print("\nHYPOTHESIS 1: DR improves clustering quality")
print("-" * 60)

for ds in ['Cancer', 'Bankruptcy']:
    orig = results_df[(results_df['Dataset'] == ds) & (results_df['DR_Methoc
    after_dr = results_df[(results_df['Dataset'] == ds) & (results_df['DR_Me
    improvement = ((after_dr - orig) / orig) * 100

    print(f"\n{ds}:")
    print(f"  Original: {orig:.4f}")
    print(f"  After DR: {after_dr:.4f}")
    print(f"  Improvement: {improvement:+.1f}%")
    print(f"  Result: {'✓ CONFIRMED' if after_dr > orig else '✗ REJECTED'}")

# Hypothesis 2
print("\n\nHYPOTHESIS 2: RP performs comparably to PCA/ICA")
print("-" * 60)

for ds in ['Cancer', 'Bankruptcy']:
    pca = results_df[(results_df['Dataset'] == ds) & (results_df['DR_Method'
    ica = results_df[(results_df['Dataset'] == ds) & (results_df['DR_Method'
    rp = results_df[(results_df['Dataset'] == ds) & (results_df['DR_Method']

    diff = abs(rp - (pca + ica)/2) / ((pca + ica)/2) * 100

    print(f"\n{ds}:")
    print(f"  PCA: {pca:.4f}, ICA: {ica:.4f}, RP: {rp:.4f}")
    print(f"  RP differs by: {diff:.1f}%")
    print(f"  Result: {'✓ CONFIRMED' if diff < 15 else '~ PARTIAL'}")

# Hypothesis 3
print("\n\nHYPOTHESIS 3: Original space clustering less meaningful")
print("-" * 60)

for ds in ['Cancer', 'Bankruptcy']:
```

```
    orig = results_df[(results_df['Dataset'] == ds) & (results_df['DR_Method
    best = results_df[(results_df['Dataset'] == ds) & (results_df['DR_Method

    print(f"\n{ds}:")
    print(f"  Original: {orig:.4f}")
    print(f"  Best DR: {best:.4f}")
    print(f"  Result: {'✓ CONFIRMED' if orig < 0.2 else '~ PARTIAL'}")

print("\n" + "="*70)
print("ANALYSIS COMPLETE")
print("="*70)
print("\nGenerated files:")
print("  - cancer_optimal_k.png")
print("  - bankruptcy_optimal_k.png")
print("  - cancer_pca_scree.png")
print("  - bankruptcy_pca_scree.png")
print("  - cancer_ica_kurtosis.png")
print("  - bankruptcy_ica_kurtosis.png")
print("  - cancer_rp_reconstruction.png")
print("  - bankruptcy_rp_reconstruction.png")
print("  - cancer_dr_clustering.png")
print("  - bankruptcy_dr_clustering.png")
print("  - cancer_nn_performance.png")
print("  - bankruptcy_nn_performance.png")
print("  - clustering_results.csv")
```

```
================================================================================
HYPOTHESIS VALIDATION
================================================================================

HYPOTHESIS 1: DR improves clustering quality
------------------------------------------------------------

Cancer:
  Original: 0.0795
  After DR: 0.3413
  Improvement: +329.2%
  Result: ✓ CONFIRMED

Bankruptcy:
  Original: 0.0706
  After DR: 0.3829
  Improvement: +442.3%
  Result: ✓ CONFIRMED


HYPOTHESIS 2: RP performs comparably to PCA/ICA
------------------------------------------------------------

Cancer:
  PCA: 0.3293, ICA: 0.3344, RP: 0.3602
  RP differs by: 8.5%
  Result: ✓ CONFIRMED

Bankruptcy:
  PCA: 0.3915, ICA: 0.3798, RP: 0.3774
  RP differs by: 2.1%
  Result: ✓ CONFIRMED


HYPOTHESIS 3: Original space clustering less meaningful
------------------------------------------------------------

Cancer:
  Original: 0.0795
  Best DR: 0.3604
  Result: ✓ CONFIRMED

Bankruptcy:
  Original: 0.0706
  Best DR: 0.4396
  Result: ✓ CONFIRMED

================================================================================
ANALYSIS COMPLETE
================================================================================

Generated files:
  - cancer_optimal_k.png
  - bankruptcy_optimal_k.png
  - cancer_pca_scree.png
  - bankruptcy_pca_scree.png
```

- cancer_ica_kurtosis.png
- bankruptcy_ica_kurtosis.png
- cancer_rp_reconstruction.png
- bankruptcy_rp_reconstruction.png
- cancer_dr_clustering.png
- bankruptcy_dr_clustering.png
- cancer_nn_performance.png
- bankruptcy_nn_performance.png
- clustering_results.csv

In [ ]: