In [7]:
```python
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.linear_model import LogisticRegression
from ipca_classes_update import IPCA_v1
import warnings
from xgboost import XGBClassifier
warnings.filterwarnings('ignore')

def prepare_earnings_surprise_data(filepath):
    """
    Load and prepare data for earnings surprise prediction using IPCA
    """
    # Load data
    data = pd.read_csv(filepath)
    print(f"Loaded data: {data.shape[0]} observations, {data.shape[1]} features")

    # Create date column and sort
    data['date'] = pd.to_datetime(data[['year', 'month']].assign(day=1))
    data = data.sort_values(['date', 'permno']).reset_index(drop=True)

    # Create earnings surprise binary target
    # 1 if actual EPS > median estimate, 0 otherwise
    data['earnings_surprise'] = np.where(
        (data['eps_actual'].notna()) & (data['eps_medest'].notna()),
        (data['eps_actual'] > data['eps_medest']).astype(int),
        np.nan
    )

    print(f"Earnings surprise distribution:")
    print(data['earnings_surprise'].value_counts(dropna=False))

    # Define characteristic variables (excluding forward-looking and identifiers)
    exclude_vars = [
        # Target/forward-looking variables
        'ret_eom', 'stock_exret', 'earnings_surprise',
        'eps_medest', 'eps_meanest', 'eps_stdevest', 'eps_actual',

        # Identifiers and date variables
        'permno', 'CUSIP', 'stock_ticker', 'comp_name',
        'year', 'month', 'date', 'SHRCD', 'EXCHCD',

        # Market-wide variables
        'RF', 'size_port'
    ]

    # Get characteristic variables
    char_vars = [col for col in data.columns if col not in exclude_vars]
    print(f"Using {len(char_vars)} characteristic variables for IPCA")

    return data, char_vars

def run_ipca_earnings_prediction(data, char_vars, K=6, min_obs_per_date=50,
                                oos_start_year=2010, oos_window=60):
    """
    Run IPCA analysis for earnings surprise prediction
    """
    # Filter data with valid earnings surprise and sufficient characteristics
    valid_data = data.dropna(subset=['earnings_surprise'] + char_vars[:20])  # Require at least 20 non-missing chars
    print(f"Data after filtering: {valid_data.shape[0]} observations")

    # Filter dates with sufficient cross-section
    date_counts = valid_data.groupby('date').size()
    valid_dates = date_counts[date_counts >= min_obs_per_date].index
    valid_data = valid_data[valid_data['date'].isin(valid_dates)]
    print(f"Using {len(valid_dates)} dates with sufficient cross-section")

    # Create multi-index dataset for IPCA
    ipca_data = valid_data.set_index(['date', 'permno'])[['earnings_surprise'] + char_vars]

    # Handle missing values by forward-filling within each stock
    ipca_data = ipca_data.groupby(level=1).fillna(method='ffill')

    # Rank transform characteristics to [-0.5, 0.5] by date
    char_data = ipca_data[char_vars].copy()
    for date in char_data.index.get_level_values(0).unique():
        date_mask = char_data.index.get_level_values(0) == date
        for var in char_vars:
            if char_data.loc[date_mask, var].notna().sum() > 10:  # Sufficient non-missing
                ranks = char_data.loc[date_mask, var].rank(method='dense') - 1
                max_rank = ranks.max()
                if max_rank > 0:
                    char_data.loc[date_mask, var] = (ranks / max_rank) - 0.5
                else:
                    char_data.loc[date_mask, var] = 0

    # Combine target with transformed characteristics
    ipca_input = pd.concat([ipca_data[['earnings_surprise']], char_data], axis=1)
    ipca_input = ipca_input.dropna()

    print(f"Final IPCA dataset: {ipca_input.shape[0]} observations")
```

```python
    # Initialize IPCA
    ipca = IPCA_v1(ipca_input, return_column='earnings_surprise', add_constant=True)

    # Split data for OOS analysis
    oos_dates = ipca_input.index.get_level_values(0) >= pd.to_datetime(f'{oos_start_year}-01-01')

    if oos_dates.sum() > 0:
        print("Running out-of-sample IPCA estimation...")
        # Out-of-sample estimation
        results = ipca.fit(
            K=K,
            OOS=True,
            OOS_window='recursive',
            OOS_window_specs=oos_window,
            R_fit=True,
            dispIters=True,
            dispItersInt=50
        )
    else:
        print("Running in-sample IPCA estimation...")
        # In-sample estimation
        results = ipca.fit(K=K, R_fit=True, dispIters=True)

    return results, ipca, ipca_input

def select_optimal_k(data, char_vars, k_range=range(4, 16),
                     min_obs_per_date=100, oos_start_year=2010, oos_window=60):
    """
    Select optimal number of factors K using cross-validation
    """
    print("\n" + "="*50)
    print("SELECTING OPTIMAL NUMBER OF FACTORS (K)")
    print("="*50)

    k_results = {}

    for k in k_range:
        print(f"\nTesting K = {k}...")
        try:
            # Run IPCA with current K
            results, ipca, ipca_input_k = run_ipca_earnings_prediction(
                data, char_vars, K=k, min_obs_per_date=min_obs_per_date,
                oos_start_year=oos_start_year, oos_window=oos_window
            )

            # Quick evaluation
            prediction_results = evaluate_ipca_factors_for_prediction(
                results, ipca_input_k, ipca, K=k, verbose=False
            )

            if prediction_results and 'metrics' in prediction_results:
                auc = prediction_results['metrics']['auc']
                r2_managed = results['xfits']['R2_Total']
                r2_returns = results['rfits']['R2_Total']

                k_results[k] = {
                    'auc': auc,
                    'r2_managed': r2_managed,
                    'r2_returns': r2_returns,
                    'score': auc  # Primary metric for selection
                }

                print(f"  AUC: {auc:.3f}, R²(managed): {r2_managed:.3f}, R²(returns): {r2_returns:.3f}")
            else:
                k_results[k] = {'auc': 0, 'r2_managed': 0, 'r2_returns': 0, 'score': 0}
                print(f"  Failed to evaluate K={k}")

        except Exception as e:
            print(f"  Error with K={k}: {str(e)}")
            k_results[k] = {'auc': 0, 'r2_managed': 0, 'r2_returns': 0, 'score': 0}

    # Find optimal K
    if k_results:
        optimal_k = max(k_results.keys(), key=lambda x: k_results[x]['score'])

        print(f"\n" + "="*30)
        print("K SELECTION RESULTS:")
        print("="*30)
        for k in sorted(k_results.keys()):
            metrics = k_results[k]
            marker = " ← OPTIMAL" if k == optimal_k else ""
            print(f"K={k:2d}: AUC={metrics['auc']:.3f}, R²(mgd)={metrics['r2_managed']:.3f}, R²(ret)={metrics['r2_returns']:.3f}{marker}"

        print(f"\nSelected optimal K = {optimal_k}")
        return optimal_k, k_results
    else:
        print("No valid results found, defaulting to K=6")
        return 6, {}

def evaluate_ipca_factors_for_prediction(results, ipca_input, ipca, K=6, verbose=True):
    """
    Use IPCA factors to predict earnings surprise
    """
    if verbose:
```

```
        print("\n" + "="*50)
        print("EVALUATING IPCA FACTORS FOR EARNINGS PREDICTION")
        print("="*50)

    # Extract factor loadings and create factor scores
    gamma = results['Gamma']
    if isinstance(gamma.index, pd.MultiIndex):
        # OOS case - use latest Gamma
        latest_date = gamma.index.get_level_values(0).max()
        gamma_final = gamma.loc[latest_date]
    else:
        # In-sample case
        gamma_final = gamma

    print(f"Factor loadings shape: {gamma_final.shape}")
    if verbose:
        print(f"Top characteristics for each factor:")
        for i in range(min(K, gamma_final.shape[1])):
            factor_name = gamma_final.columns[i]
            top_chars = gamma_final.iloc[:, i].abs().nlargest(5)
            print(f"\nFactor {factor_name}:")
            for char, loading in top_chars.items():
                print(f"  {char}: {loading:.3f}")

    # Create factor scores for prediction
    char_vars = gamma_final.index.tolist()
    if 'Constant' in char_vars:
        char_vars.remove('Constant')

    # Calculate factor scores
    factor_scores_list = []

    for date in ipca_input.index.get_level_values(0).unique():
        date_data = ipca_input.loc[date]
        if date_data.shape[0] > 10:  # Sufficient observations
            # Get characteristics matrix
            X = date_data[char_vars].values
            # Add constant
            X_with_const = np.column_stack([X, np.ones(X.shape[0])])

            # Calculate factor scores
            factor_scores = X_with_const @ gamma_final.values

            # Create DataFrame
            factor_df = pd.DataFrame(
                factor_scores,
                index=date_data.index,
                columns=gamma_final.columns
            )
            factor_df['date'] = date
            factor_df['earnings_surprise'] = date_data['earnings_surprise'].values
            factor_scores_list.append(factor_df)

    if len(factor_scores_list) == 0:
        if verbose:
            print("No valid data for factor score calculation")
        return None

    # Combine all factor scores
    all_factor_scores = pd.concat(factor_scores_list, ignore_index=True)
    all_factor_scores = all_factor_scores.dropna()

    if verbose:
        print(f"\nFactor scores dataset: {all_factor_scores.shape[0]} observations")

    # Split into train/test
    train_data = all_factor_scores[all_factor_scores['date'] < pd.to_datetime('2015-01-01')]
    test_data = all_factor_scores[all_factor_scores['date'] >= pd.to_datetime('2015-01-01')]

    if len(train_data) == 0 or len(test_data) == 0:
        if verbose:
            print("Insufficient data for train/test split")
        return all_factor_scores

    if verbose:
        print(f"Train set: {len(train_data)} observations")
        print(f"Test set: {len(test_data)} observations")

    # Prepare features (exclude date and target)
    factor_cols = gamma_final.columns.tolist()
    X_train = train_data[factor_cols]
    y_train = train_data['earnings_surprise']
    X_test = test_data[factor_cols]
    y_test = test_data['earnings_surprise']

    # Train logistic regression
# With this:
    xgb = XGBClassifier(random_state=42, eval_metric='logloss')
    xgb.fit(X_train, y_train)
    y_pred_proba = xgb.predict_proba(X_test)[:, 1]
    y_pred = xgb.predict(X_test)
    # Evaluate
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
```

```python
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred_proba)

    if verbose:
        print(f"\nPREDICTION RESULTS:")
        print(f"Accuracy: {accuracy:.3f}")
        print(f"Precision: {precision:.3f}")
        print(f"Recall: {recall:.3f}")
        print(f"F1-Score: {f1:.3f}")
        print(f"AUC-ROC: {auc:.3f}")

        # Feature importance
        print(f"\nFACTOR IMPORTANCE (Logistic Regression Coefficients):")
        for factor, coef in zip(factor_cols, lr.coef_[0]):
            print(f"{factor}: {coef:.3f}")

    return {
        'factor_scores': all_factor_scores,
        'gamma': gamma_final,
        'model': lr,
        'metrics': {
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1,
            'auc': auc
        }
    }

def main():
    """
    Main execution function
    """
    print("IPCA EARNINGS SURPRISE PREDICTION")
    print("="*50)

    # Load and prepare data
    filepath = '/teamspace/studios/this_studio/goup_project_sample_v3.csv'  # Replace with your actual file path
    data, char_vars = prepare_earnings_surprise_data(filepath)

    # Select optimal K
    optimal_k, k_results = select_optimal_k(
        data, char_vars,
        k_range=range(4, 16),  # Test K from 4 to 15
        min_obs_per_date=100,
        oos_start_year=2012,
        oos_window=60
    )

    # Run IPCA analysis with optimal K
    print(f"\nRunning final analysis with K = {optimal_k}...")
    results, ipca, ipca_input = run_ipca_earnings_prediction(
        data, char_vars,
        K=optimal_k,
        min_obs_per_date=100,
        oos_start_year=2012,
        oos_window=60
    )

    # Print IPCA results
    print(f"\nFINAL IPCA ESTIMATION RESULTS (K={optimal_k}):")
    print(f"Managed Portfolio R²: {results['xfits']['R2_Total']:.3f}")
    print(f"Returns R²: {results['rfits']['R2_Total']:.3f}")

    # Detailed evaluation with optimal K
    prediction_results = evaluate_ipca_factors_for_prediction(
        results, ipca_input, ipca, K=optimal_k, verbose=True
    )

    if prediction_results:
        # Save results
        prediction_results['factor_scores'].to_csv('ipca_factor_scores.csv', index=False)
        results['Gamma'].to_csv('ipca_factor_loadings.csv')

        # Save K selection results
        k_results_df = pd.DataFrame(k_results).T
        k_results_df.to_csv('k_selection_results.csv')

        print(f"\nResults saved:")
        print(f"- Factor scores: ipca_factor_scores.csv")
        print(f"- Factor loadings: ipca_factor_loadings.csv")
        print(f"- K selection results: k_selection_results.csv")
        print(f"- Optimal K used: {optimal_k}")

if __name__ == '__main__':
    main()
```

```
IPCA EARNINGS SURPRISE PREDICTION
==================================================
Loaded data: 911537 observations, 165 features
Earnings surprise distribution:
earnings_surprise
1.0    310507
NaN    307811
0.0    293219
Name: count, dtype: int64
Using 148 characteristic variables for IPCA

==================================================
SELECTING OPTIMAL NUMBER OF FACTORS (K)
==================================================

Testing K = 4...
Data after filtering: 403294 observations
Using 295 dates with sufficient cross-section
Final IPCA dataset: 127199 observations
Running out-of-sample IPCA estimation...
iters 50: tol = 0.10351827181307294
iters 100: tol = 0.01997162832782351
iters 150: tol = 0.0004490765502374039
iters 50: tol = 0.0009626866873225781
iters 50: tol = 0.0002038675146982527
iters 50: tol = 0.0002737770873626477
iters 50: tol = 0.00016373562731397673
iters 50: tol = 0.0008937911715094504
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[7], line 367
    364         print(f"- Optimal K used: {optimal_k}")
    366 if __name__ == '__main__':
--> 367     main()

Cell In[7], line 323, in main()
    320 data, char_vars = prepare_earnings_surprise_data(filepath)
    322 # Select optimal K
--> 323 optimal_k, k_results = select_optimal_k(
    324     data, char_vars,
    325     k_range=range(4, 16),  # Test K from 4 to 15
    326     min_obs_per_date=100,
    327     oos_start_year=2012,
    328     oos_window=60
    329 )
    331 # Run IPCA analysis with optimal K
    332 print(f"\nRunning final analysis with K = {optimal_k}...")

Cell In[7], line 133, in select_optimal_k(data, char_vars, k_range, min_obs_per_date, oos_start_year, oos_window)
    130 print(f"\nTesting K = {k}...")
    131 try:
    132     # Run IPCA with current K
--> 133     results, ipca, ipca_input_k = run_ipca_earnings_prediction(
    134         data, char_vars, K=k, min_obs_per_date=min_obs_per_date,
    135         oos_start_year=oos_start_year, oos_window=oos_window
    136     )
    138     # Quick evaluation
    139     prediction_results = evaluate_ipca_factors_for_prediction(
    140         results, ipca_input_k, ipca, K=k, verbose=False
    141     )

Cell In[7], line 102, in run_ipca_earnings_prediction(data, char_vars, K, min_obs_per_date, oos_start_year, oos_window)
    100     print("Running out-of-sample IPCA estimation...")
    101     # Out-of-sample estimation
--> 102     results = ipca.fit(
    103         K=K,
    104         OOS=True,
    105         OOS_window='recursive',
    106         OOS_window_specs=oos_window,
    107         R_fit=True,
    108         dispIters=True,
    109         dispItersInt=50
    110     )
    111 else:
    112     print("Running in-sample IPCA estimation...")

File ~/ipca_classes_update.py:749, in IPCA_v1.fit(self, K, OOS, gFac, normalization_choice, normalization_choice_specs, OOS_window, OOS_wi
ndow_specs, factor_mean, R_fit, Beta_fit, dispIters, minTol, maxIters, F_names, G_names, R2_bench, dispItersInt)
    746 iters += 1
    747 # for first t, Gamma0 will be from _svd_initial outside loop; for subsequent t, will be that last
    748 #  Gamma0 obtained in the previous t's iterative "while" stmt
--> 749 Gamma1, Factor1 = self._linear_als_estimation(
    750     Gamma0=Gamma0.copy(),
    751     gFac=gFac,  # make _gFac for ndarray-based
    752     K=K,
    753     M=M,
    754     KM=KM,
    755     normalization_choice=normalization_choice,
    756     normalization_choice_specs=normalization_choice_specs,
    757     Dates=datest,
    758 )
    760 tolGam = np.max(np.abs(Gamma1 - Gamma0))
    761 tolFac = np.max(np.abs(Factor1 - Factor0))

File ~/ipca_classes_update.py:998, in IPCA_v1._linear_als_estimation(self, Gamma0, K, M, KM, normalization_choice, normalization_choice_sp
ecs, gFac, Dates)
    995 for t in Dates:
    996     numer += np.kron(self.X[t].values, Factor[:, ct]) * self.Nts[t]
    997     denom += (
--> 998         np.kron(self.W.loc[t].values, np.outer(Factor[:, ct], Factor[:, ct]))
    999         * self.Nts[t]
   1000     )
   1001     ct += 1
   1002 # # ndarray-based
   1003 # ct=0
   1004 # for t in Dates:
   1005 #     numer += np.kron(self._X[:, t], Factor[:, ct]) * self.Nts[t]
   1006 #     denom += np.kron(self._W[:, :, t], np.outer(Factor[:, ct], Factor[:, ct])) * self.Nts[t]
   1007 #     ct+=1

File /home/zeus/miniconda3/envs/cloudspace/lib/python3.10/site-packages/numpy/lib/shape_base.py:1173, in kron(a, b)
   1171 b_arr = expand_dims(b_arr, axis=tuple(range(0, nd*2, 2)))
   1172 # In case of `mat`, convert result to `array`
-> 1173 result = _nx.multiply(a_arr, b_arr, subok=(not is_any_mat))
   1175 # Reshape back
   1176 result = result.reshape(_nx.multiply(as_, bs))

KeyboardInterrupt: 
```

In [1]:
```
import pandas as pd
import numpy as np
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from ipca_classes_update import IPCA_v1
import statsmodels.api as sm
from datetime import datetime
from dateutil.relativedelta import relativedelta
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

def prepare_earnings_surprise_data(filepath):
    """
    Load and prepare data for earnings surprise prediction using IPCA
    """
    # Load data
    data = pd.read_csv(filepath)
    print(f"Loaded data: {data.shape[0]} observations, {data.shape[1]} features")

    # Create date column and sort
    data['date'] = pd.to_datetime(data[['year', 'month']].assign(day=1))
    data = data.sort_values(['date', 'permno']).reset_index(drop=True)

    # Create earnings surprise binary target
    # 1 if actual EPS > median estimate, 0 otherwise
    data['earnings_surprise'] = np.where(
        (data['eps_actual'].notna()) & (data['eps_medest'].notna()),
        (data['eps_actual'] > data['eps_medest']).astype(int),
        np.nan
    )

    print(f"Earnings surprise distribution:")
    print(data['earnings_surprise'].value_counts(dropna=False))

    # Define characteristic variables (excluding forward-looking and identifiers)
    exclude_vars = [
        # Target/forward-looking variables
        'ret_eom', 'stock_exret', 'earnings_surprise',
        'eps_medest', 'eps_meanest', 'eps_stdevest', 'eps_actual',

        # Identifiers and date variables
        'permno', 'CUSIP', 'stock_ticker', 'comp_name',
        'year', 'month', 'date', 'SHRCD', 'EXCHCD',

        # Market-wide variables
        'RF', 'size_port'
    ]

    # Get characteristic variables
    char_vars = [col for col in data.columns if col not in exclude_vars]
    print(f"Using {len(char_vars)} characteristic variables for IPCA")

    return data, char_vars

def run_ipca_earnings_prediction(data, char_vars, K=6, min_obs_per_date=50,
                                oos_start_year=2010, oos_window=60):
    """
    Run IPCA analysis for earnings surprise prediction
    """
    # Filter data with valid earnings surprise and sufficient characteristics
    valid_data = data.dropna(subset=['earnings_surprise'] + char_vars[:20])  # Require at least 20 non-missing chars
    print(f"Data after filtering: {valid_data.shape[0]} observations")

    # Filter dates with sufficient cross-section
    date_counts = valid_data.groupby('date').size()
    valid_dates = date_counts[date_counts >= min_obs_per_date].index
    valid_data = valid_data[valid_data['date'].isin(valid_dates)]
    print(f"Using {len(valid_dates)} dates with sufficient cross-section")

    # Create multi-index dataset for IPCA
    ipca_data = valid_data.set_index(['date', 'permno'])[['earnings_surprise'] + char_vars]

    # Handle missing values by forward-filling within each stock
    ipca_data = ipca_data.groupby(level=1).fillna(method='ffill')

    # Rank transform characteristics to [-0.5, 0.5] by date
    char_data = ipca_data[char_vars].copy()
    for date in char_data.index.get_level_values(0).unique():
        date_mask = char_data.index.get_level_values(0) == date
        for var in char_vars:
            if char_data.loc[date_mask, var].notna().sum() > 10:  # Sufficient non-missing
                ranks = char_data.loc[date_mask, var].rank(method='dense') - 1
                max_rank = ranks.max()
                if max_rank > 0:
                    char_data.loc[date_mask, var] = (ranks / max_rank) - 0.5
                else:
                    char_data.loc[date_mask, var] = 0

    # Combine target with transformed characteristics
    ipca_input = pd.concat([ipca_data[['earnings_surprise']], char_data], axis=1)
    ipca_input = ipca_input.dropna()

    print(f"Final IPCA dataset: {ipca_input.shape[0]} observations")
```

```python
    # Initialize IPCA
    ipca = IPCA_v1(ipca_input, return_column='earnings_surprise', add_constant=True)

    # Split data for OOS analysis
    oos_dates = ipca_input.index.get_level_values(0) >= pd.to_datetime(f'{oos_start_year}-01-01')

    if oos_dates.sum() > 0:
        print("Running out-of-sample IPCA estimation...")
        # Out-of-sample estimation
        results = ipca.fit(
            K=K,
            OOS=True,
            OOS_window='recursive',
            OOS_window_specs=oos_window,
            R_fit=True,
            dispIters=True,
            dispItersInt=50
        )
    else:
        print("Running in-sample IPCA estimation...")
        # In-sample estimation
        results = ipca.fit(K=K, R_fit=True, dispIters=True)

    return results, ipca, ipca_input

def select_optimal_k(data, char_vars, k_range=range(4, 16),
                    min_obs_per_date=100, oos_start_year=2010, oos_window=60):
    """
    Select optimal number of factors K using cross-validation
    """
    print("\n" + "="*50)
    print("SELECTING OPTIMAL NUMBER OF FACTORS (K)")
    print("="*50)

    k_results = {}

    for k in k_range:
        print(f"\nTesting K = {k}...")
        try:
            # Run IPCA with current K
            results, ipca, ipca_input_k = run_ipca_earnings_prediction(
                data, char_vars, K=k, min_obs_per_date=min_obs_per_date,
                oos_start_year=oos_start_year, oos_window=oos_window
            )

            # Quick evaluation
            prediction_results = evaluate_ipca_factors_for_prediction(
                results, ipca_input_k, ipca, K=k, verbose=False
            )

            if prediction_results and 'metrics' in prediction_results:
                auc = prediction_results['metrics']['auc']
                r2_managed = results['xfits']['R2_Total']
                r2_returns = results['rfits']['R2_Total']

                k_results[k] = {
                    'auc': auc,
                    'r2_managed': r2_managed,
                    'r2_returns': r2_returns,
                    'score': auc  # Primary metric for selection
                }

                print(f"  AUC: {auc:.3f}, R²(managed): {r2_managed:.3f}, R²(returns): {r2_returns:.3f}")
            else:
                k_results[k] = {'auc': 0, 'r2_managed': 0, 'r2_returns': 0, 'score': 0}
                print(f"  Failed to evaluate K={k}")

        except Exception as e:
            print(f"  Error with K={k}: {str(e)}")
            k_results[k] = {'auc': 0, 'r2_managed': 0, 'r2_returns': 0, 'score': 0}

    # Find optimal K
    if k_results:
        optimal_k = max(k_results.keys(), key=lambda x: k_results[x]['score'])

        print(f"\n" + "="*30)
        print("K SELECTION RESULTS:")
        print("="*30)
        for k in sorted(k_results.keys()):
            metrics = k_results[k]
            marker = " ← OPTIMAL" if k == optimal_k else ""
            print(f"K={k:2d}: AUC={metrics['auc']:.3f}, R²(mgd)={metrics['r2_managed']:.3f}, R²(ret)={metrics['r2_returns']:.3f}{marker}"

        print(f"\nSelected optimal K = {optimal_k}")
        return optimal_k, k_results
    else:
        print("No valid results found, defaulting to K=6")
        return 6, {}

def evaluate_ipca_factors_for_prediction(results, ipca_input, ipca, K=6, verbose=True):
    """
    Use IPCA factors to predict earnings surprise
    """
```

```
    if verbose:
        print("\n" + "="*50)
        print("EVALUATING IPCA FACTORS FOR EARNINGS PREDICTION")
        print("="*50)

    # Extract factor loadings and create factor scores
    gamma = results['Gamma']
    if isinstance(gamma.index, pd.MultiIndex):
        # OOS case - use latest Gamma
        latest_date = gamma.index.get_level_values(0).max()
        gamma_final = gamma.loc[latest_date]
    else:
        # In-sample case
        gamma_final = gamma

    print(f"Factor loadings shape: {gamma_final.shape}")
    if verbose:
        print(f"Top characteristics for each factor:")
        for i in range(min(K, gamma_final.shape[1])):
            factor_name = gamma_final.columns[i]
            top_chars = gamma_final.iloc[:, i].abs().nlargest(5)
            print(f"\nFactor {factor_name}:")
            for char, loading in top_chars.items():
                print(f"  {char}: {loading:.3f}")

    # Create factor scores for prediction
    char_vars = gamma_final.index.tolist()
    if 'Constant' in char_vars:
        char_vars.remove('Constant')

    # Calculate factor scores
    factor_scores_list = []

    for date in ipca_input.index.get_level_values(0).unique():
        date_data = ipca_input.loc[date]
        if date_data.shape[0] > 10:  # Sufficient observations
            # Get characteristics matrix
            X = date_data[char_vars].values
            # Add constant
            X_with_const = np.column_stack([X, np.ones(X.shape[0])])

            # Calculate factor scores
            factor_scores = X_with_const @ gamma_final.values

            # Create DataFrame
            factor_df = pd.DataFrame(
                factor_scores,
                index=date_data.index,
                columns=gamma_final.columns
            )
            factor_df['date'] = date
            factor_df['year'] = date.year
            factor_df['month'] = date.month
            factor_df['permno'] = date_data.index  # Add permno
            factor_df['earnings_surprise'] = date_data['earnings_surprise'].values
            factor_scores_list.append(factor_df)

    if len(factor_scores_list) == 0:
        if verbose:
            print("No valid data for factor score calculation")
        return None

    # Combine all factor scores
    all_factor_scores = pd.concat(factor_scores_list, ignore_index=True)
    all_factor_scores = all_factor_scores.dropna()

    if verbose:
        print(f"\nFactor scores dataset: {all_factor_scores.shape[0]} observations")

    # Split into train/test
    train_data = all_factor_scores[all_factor_scores['date'] < pd.to_datetime('2015-01-01')]
    test_data = all_factor_scores[all_factor_scores['date'] >= pd.to_datetime('2015-01-01')]

    if len(train_data) == 0 or len(test_data) == 0:
        if verbose:
            print("Insufficient data for train/test split")
        return all_factor_scores

    if verbose:
        print(f"Train set: {len(train_data)} observations")
        print(f"Test set: {len(test_data)} observations")

    # Prepare features (exclude date and target)
    factor_cols = gamma_final.columns.tolist()
    X_train = train_data[factor_cols]
    y_train = train_data['earnings_surprise']
    X_test = test_data[factor_cols]
    y_test = test_data['earnings_surprise']

    # Train XGBoost classifier
    xgb = XGBClassifier(random_state=42, eval_metric='logloss')
    xgb.fit(X_train, y_train)

    # Predictions
```

```python
    y_pred_proba = xgb.predict_proba(X_test)[:, 1]
    y_pred = xgb.predict(X_test)

    # Evaluate
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred_proba)

    if verbose:
        print(f"\nPREDICTION RESULTS:")
        print(f"Accuracy: {accuracy:.3f}")
        print(f"Precision: {precision:.3f}")
        print(f"Recall: {recall:.3f}")
        print(f"F1-Score: {f1:.3f}")
        print(f"AUC-ROC: {auc:.3f}")

        # Feature importance
        print(f"\nFACTOR IMPORTANCE (XGBoost):")
        for factor, importance in zip(factor_cols, xgb.feature_importances_):
            print(f"{factor}: {importance:.3f}")

    return {
        'factor_scores': all_factor_scores,
        'gamma': gamma_final,
        'model': xgb,
        'metrics': {
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1,
            'auc': auc
        }
    }

def backtest_ipca_strategy(data, prediction_results, start_year=2015, end_year=2023,
                          top_n_stocks=50, hold_period=1):
    """
    Backtest IPCA-based earnings surprise strategy
    """
    print("\n" + "="*50)
    print("BACKTESTING IPCA EARNINGS SURPRISE STRATEGY")
    print("="*50)

    # Get factor scores with predictions
    factor_scores = prediction_results['factor_scores'].copy()
    model = prediction_results['model']
    gamma = prediction_results['gamma']

    # Merge with original data to get returns
    factor_scores['date'] = pd.to_datetime(factor_scores[['year', 'month']].assign(day=1))
    data_with_dates = data.copy()
    data_with_dates['date'] = pd.to_datetime(data_with_dates[['year', 'month']].assign(day=1))

    # Add stock returns to factor scores
    backtest_data = factor_scores.merge(
        data_with_dates[['permno', 'date', 'stock_exret', 'RF']],
        on=['permno', 'date'],
        how='left'
    )

    # Generate predictions for all data
    factor_cols = gamma.columns.tolist()
    X_all = backtest_data[factor_cols].fillna(0)
    backtest_data['pred_proba'] = model.predict_proba(X_all)[:, 1]
    backtest_data['pred_binary'] = model.predict(X_all)

    # Create trading periods
    trade_periods = []
    current_date = datetime(start_year, 1, 1)
    end_date = datetime(end_year, 12, 31)

    while current_date <= end_date:
        trade_periods.append((current_date.year, current_date.month))
        current_date += relativedelta(months=1)

    # Portfolio construction and backtesting
    portfolio_returns = []
    portfolio_compositions = []
    benchmark_returns = []

    for i, (year, month) in enumerate(trade_periods[:-hold_period]):
        # Selection data (current month)
        selection_data = backtest_data[
            (backtest_data['year'] == year) &
            (backtest_data['month'] == month)
        ].copy()

        if selection_data.empty:
            portfolio_returns.append(0.0)
            benchmark_returns.append(0.0)
            continue
```

```python
        # Select top stocks based on earnings surprise probability
        top_stocks = selection_data.nlargest(top_n_stocks, 'pred_proba')

        if len(top_stocks) == 0:
            portfolio_returns.append(0.0)
            benchmark_returns.append(0.0)
            continue

        # Equal weight portfolio
        weights = np.ones(len(top_stocks)) / len(top_stocks)

        # Get returns for holding period (next month)
        hold_year, hold_month = trade_periods[i + hold_period]
        return_data = backtest_data[
            (backtest_data['year'] == hold_year) &
            (backtest_data['month'] == hold_month) &
            (backtest_data['permno'].isin(top_stocks['permno']))
        ]

        # Calculate portfolio return
        if len(return_data) > 0:
            # Match weights to available returns
            portfolio_return = 0.0
            total_weight = 0.0

            for j, (idx, stock) in enumerate(top_stocks.iterrows()):
                stock_return_data = return_data[return_data['permno'] == stock['permno']]
                if len(stock_return_data) > 0:
                    stock_return = stock_return_data['stock_exret'].iloc[0]
                    if pd.notna(stock_return):
                        portfolio_return += weights[j] * stock_return
                        total_weight += weights[j]

            # Normalize if some stocks missing
            if total_weight > 0:
                portfolio_return = portfolio_return / total_weight

            # Add risk-free rate
            rf_rate = return_data['RF'].iloc[0] if len(return_data) > 0 and pd.notna(return_data['RF'].iloc[0]) else 0.0
            portfolio_returns.append(portfolio_return + rf_rate)

        else:
            portfolio_returns.append(0.0)

        # Benchmark return (risk-free rate for now - can be replaced with market return)
        benchmark_returns.append(rf_rate if 'rf_rate' in locals() else 0.0)

        # Store composition
        portfolio_compositions.append({
            'year': year,
            'month': month,
            'num_stocks': len(top_stocks),
            'avg_pred_proba': top_stocks['pred_proba'].mean(),
            'portfolio_return': portfolio_returns[-1]
        })

    # Create results DataFrame
    results_df = pd.DataFrame(portfolio_compositions)
    results_df['date'] = pd.to_datetime(results_df[['year', 'month']].assign(day=1))

    return {
        'portfolio_returns': portfolio_returns,
        'benchmark_returns': benchmark_returns,
        'trade_periods': trade_periods[:-hold_period],
        'portfolio_compositions': results_df
    }

def calculate_performance_metrics(portfolio_returns, benchmark_returns, risk_free_rates=None):
    """
    Calculate comprehensive performance metrics
    """
    portfolio_returns = pd.Series(portfolio_returns)
    benchmark_returns = pd.Series(benchmark_returns)

    if risk_free_rates is None:
        risk_free_rates = pd.Series([0.0] * len(portfolio_returns))
    else:
        risk_free_rates = pd.Series(risk_free_rates)

    # Create DataFrame and remove NaN values
    df = pd.DataFrame({
        'portfolio': portfolio_returns,
        'benchmark': benchmark_returns,
        'rf': risk_free_rates
    }).dropna()

    if len(df) < 2:
        return {k: np.nan for k in ["Annual Return (%)", "Annualized Volatility (%)",
                                    "Annualized Sharpe Ratio", "Max Drawdown (%)",
                                    "Max Monthly Loss (%)", "Beta vs Benchmark"]}

    # Annualized return
    ann_ret = (1 + df['portfolio']).prod()**(12 / len(df)) - 1
```

```python
    # Annualized volatility
    ann_vol = df['portfolio'].std() * np.sqrt(12)

    # Risk-free rate
    ann_rf = (1 + df['rf'].mean())**12 - 1

    # Sharpe ratio
    sharpe = (ann_ret - ann_rf) / ann_vol if ann_vol > 1e-9 else np.nan

    # Alpha and Beta
    excess_portfolio = df['portfolio'] - df['rf']
    excess_benchmark = df['benchmark'] - df['rf']

    alpha_monthly, beta = np.nan, np.nan
    if len(excess_portfolio) >= 2 and excess_benchmark.std() > 1e-9:
        try:
            model = sm.OLS(excess_portfolio, sm.add_constant(excess_benchmark)).fit()
            alpha_monthly = model.params[0]
            beta = model.params[1]
        except:
            alpha_monthly = excess_portfolio.mean() - excess_benchmark.mean()
            beta = np.nan

    ann_alpha = alpha_monthly * 12 if pd.notna(alpha_monthly) else np.nan

    # Drawdown
    cumulative_returns = (1 + df['portfolio']).cumprod()
    peak = cumulative_returns.expanding().max()
    drawdown = (cumulative_returns - peak) / peak
    max_drawdown = drawdown.min()

    # Max monthly loss
    max_loss = df['portfolio'].min()

    # Information Ratio
    active_return = df['portfolio'] - df['benchmark']
    tracking_error = active_return.std() * np.sqrt(12)
    mean_active_return = active_return.mean() * 12
    info_ratio = mean_active_return / tracking_error if tracking_error > 1e-9 else np.nan

    return {
        "Annual Return (%)": ann_ret * 100,
        "Annualized Volatility (%)": ann_vol * 100,
        "Annualized Sharpe Ratio": sharpe,
        "Annualized Alpha vs Benchmark (%)": ann_alpha * 100,
        "Beta vs Benchmark": beta,
        "Max Drawdown (%)": max_drawdown * 100,
        "Max Monthly Loss (%)": max_loss * 100,
        "Annualized Information Ratio vs Benchmark": info_ratio,
        "Annualized Tracking Error vs Benchmark (%)": tracking_error * 100
    }

def create_performance_summary(backtest_results, strategy_name="IPCA_Earnings_Strategy"):
    """
    Create performance summary table
    """
    portfolio_returns = backtest_results['portfolio_returns']
    benchmark_returns = backtest_results['benchmark_returns']

    # Calculate metrics for both strategy and benchmark
    strategy_metrics = calculate_performance_metrics(portfolio_returns, benchmark_returns)
    benchmark_metrics = calculate_performance_metrics(benchmark_returns, benchmark_returns)

    # Create summary DataFrame
    metrics_df = pd.DataFrame({
        'Benchmark': benchmark_metrics,
        strategy_name: strategy_metrics
    })

    # Format numbers
    for col in metrics_df.columns:
        for idx in metrics_df.index:
            val = metrics_df.loc[idx, col]
            if pd.isna(val):
                metrics_df.loc[idx, col] = 'nan'
            elif 'Ratio' in idx:
                metrics_df.loc[idx, col] = f"{val:.3f}"
            elif 'Beta' in idx:
                metrics_df.loc[idx, col] = f"{val:.3f}"
            else:
                metrics_df.loc[idx, col] = f"{val:.2f}"

    return metrics_df

def plot_cumulative_returns(backtest_results, strategy_name="IPCA_Earnings_Strategy"):
    """
    Plot cumulative returns
    """
    portfolio_returns = pd.Series(backtest_results['portfolio_returns'])
    benchmark_returns = pd.Series(backtest_results['benchmark_returns'])
    trade_periods = backtest_results['trade_periods']

    # Create date index
    dates = [datetime(year, month, 1) for year, month in trade_periods]
```

```python
    # Calculate cumulative returns
    cum_portfolio = (1 + portfolio_returns).cumprod() * 100
    cum_benchmark = (1 + benchmark_returns).cumprod() * 100

    # Create DataFrame for plotting
    cum_returns_df = pd.DataFrame({
        strategy_name: cum_portfolio.values,
        'Benchmark': cum_benchmark.values
    }, index=dates)

    # Plot
    plt.figure(figsize=(12, 8))
    cum_returns_df.plot(kind='line')
    plt.title('Cumulative Returns Comparison')
    plt.xlabel('Date')
    plt.ylabel('Cumulative Return Index (Base = 100)')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    return cum_returns_df
    """
    Main execution function
    """
    print("IPCA EARNINGS SURPRISE PREDICTION")
    print("="*50)

    # Load and prepare data
    filepath = 'your_dataset.csv'  # Replace with your actual file path
    data, char_vars = prepare_earnings_surprise_data(filepath)

    # Select optimal K
    optimal_k, k_results = select_optimal_k(
        data, char_vars,
        k_range=range(4, 16),  # Test K from 4 to 15
        min_obs_per_date=100,
        oos_start_year=2012,
        oos_window=60
    )

    # Run IPCA analysis with optimal K
    print(f"\nRunning final analysis with K = {optimal_k}...")
    results, ipca, ipca_input = run_ipca_earnings_prediction(
        data, char_vars,
        K=optimal_k,
        min_obs_per_date=100,
        oos_start_year=2012,
        oos_window=60
    )

    # Print IPCA results
    print(f"\nFINAL IPCA ESTIMATION RESULTS (K={optimal_k}):")
    print(f"Managed Portfolio R²: {results['xfits']['R2_Total']:.3f}")
    print(f"Returns R²: {results['rfits']['R2_Total']:.3f}")

    # Detailed evaluation with optimal K
    prediction_results = evaluate_ipca_factors_for_prediction(
        results, ipca_input, ipca, K=optimal_k, verbose=True
    )

    if prediction_results:
        # Save results
        prediction_results['factor_scores'].to_csv('ipca_factor_scores.csv', index=False)
        results['Gamma'].to_csv('ipca_factor_loadings.csv')

        # Save K selection results
        k_results_df = pd.DataFrame(k_results).T
        k_results_df.to_csv('k_selection_results.csv')

        print(f"\nResults saved:")
        print(f"- Factor scores: ipca_factor_scores.csv")
        print(f"- Factor loadings: ipca_factor_loadings.csv")
        print(f"- K selection results: k_selection_results.csv")
        print(f"- Optimal K used: {optimal_k}")

#if __name__ == '__main__':
#    main()
```

In [2]:

```python
# 1. Load and prepare data
filepath = 'goup_project_sample_v3.csv'  # Replace with your actual file path
data, char_vars = prepare_earnings_surprise_data(filepath)

# 2. Run IPCA with fixed K=8 (skip optimization for now)
results, ipca, ipca_input = run_ipca_earnings_prediction(
    data, char_vars, K=8, min_obs_per_date=100,
    oos_start_year=2012, oos_window=60
)

# 3. Get predictions
prediction_results = evaluate_ipca_factors_for_prediction(
    results, ipca_input, ipca, K=8, verbose=True
```

```
)

# 4. Run backtest
if prediction_results:
    backtest_results = backtest_ipca_strategy(
        data, prediction_results, start_year=2015, end_year=2023,
        top_n_stocks=50, hold_period=1
    )

    # 5. Show performance
    performance_summary = create_performance_summary(
        backtest_results, strategy_name="IPCA_Earnings_Strategy"
    )
    print("\nPERFORMANCE METRICS:")
    print(performance_summary.T)
```

```
Loaded data: 911537 observations, 165 features
Earnings surprise distribution:
earnings_surprise
1.0    310507
NaN    307811
0.0    293219
Name: count, dtype: int64
Using 148 characteristic variables for IPCA
Data after filtering: 403294 observations
Using 295 dates with sufficient cross-section
Final IPCA dataset: 127199 observations
Running out-of-sample IPCA estimation...
iters 50: tol = 0.08065946189690842
iters 100: tol = 0.06626441854350565
iters 150: tol = 0.004592182742092721
iters 200: tol = 0.0005191526454617623
iters 50: tol = 0.037003190546523346
iters 100: tol = 0.030077222830379502
iters 150: tol = 0.015600795422230584
iters 200: tol = 0.016244122876550054
iters 250: tol = 0.016137740145162005
iters 300: tol = 0.011160074237905215
iters 350: tol = 0.007115623367877633
iters 400: tol = 0.004611988097601952
iters 450: tol = 0.003059289988542324
iters 500: tol = 0.0020819830235951353
iters 550: tol = 0.0014440186226952756
iters 600: tol = 0.001029008829104533
iters 650: tol = 0.0007670173692280319
iters 700: tol = 0.0005663844727101519
iters 750: tol = 0.0004151763691390975
iters 800: tol = 0.0003026370348466756
iters 850: tol = 0.00021967690743196489
iters 900: tol = 0.00015896089452877016
iters 950: tol = 0.00011476199270471499
iters 50: tol = 0.010788499320705247
iters 100: tol = 0.002053338757406986
iters 150: tol = 0.0004946754598947889
iters 200: tol = 0.00013229472261788722
iters 50: tol = 0.008821327712032367
iters 100: tol = 0.0005646622437873727
iters 50: tol = 0.007681430903717978
iters 100: tol = 0.0015209131295217393
iters 150: tol = 0.00046493879707365515
iters 200: tol = 0.00015777971737046537
iters 50: tol = 0.02465396186111357
iters 100: tol = 0.012981116662177161
iters 150: tol = 0.019292334273246503
iters 200: tol = 0.024572449164001475
iters 250: tol = 0.045244216014279415
iters 300: tol = 0.009483665201616809
iters 350: tol = 0.007888457070984511
iters 400: tol = 0.014157602532393398
iters 450: tol = 0.014794475998481382
iters 500: tol = 0.0313235008391603
iters 550: tol = 0.00967627246806857
iters 600: tol = 0.001663214982941641
iters 650: tol = 0.000247564897058572
iters 50: tol = 0.008742818270767394
iters 100: tol = 0.004452360871662009
iters 150: tol = 0.002217697089484716
iters 200: tol = 0.0010380277106506464
iters 250: tol = 0.0004756290832303445
iters 300: tol = 0.00021629936449246712
iters 50: tol = 0.014042135596038063
iters 100: tol = 0.012475424127992962
iters 150: tol = 0.007555617161090433
iters 200: tol = 0.0032861207825556904
iters 250: tol = 0.0017858587102645984
iters 300: tol = 0.0009330636149529337
iters 350: tol = 0.0004150336388135667
iters 400: tol = 0.00017609292202580562
iters 50: tol = 0.012226493134589678
iters 100: tol = 0.008045477862600348
iters 150: tol = 0.004540119520415775
iters 200: tol = 0.002244650876745524
iters 250: tol = 0.0010442840346265303
iters 300: tol = 0.00047445458575423594
iters 350: tol = 0.00021356800832883494
iters 50: tol = 0.02081236818490506
iters 100: tol = 0.005978064715516096
iters 150: tol = 0.0018056850604175612
iters 200: tol = 0.0005633388021286656
iters 250: tol = 0.0001779231054139796
iters 50: tol = 0.008043289553272409
iters 100: tol = 0.007828462609907896
iters 150: tol = 0.011055874574236402
iters 200: tol = 0.019801997230406654
iters 250: tol = 0.0251307521754196
iters 300: tol = 0.036398942341965446
iters 350: tol = 0.01665463185328675
iters 400: tol = 0.01345505511076217
iters 450: tol = 0.0030621818139406898
iters 500: tol = 0.0015276900279674877
```

```
iters 550: tol = 0.0006613797381429531
iters 600: tol = 0.0002918579267078836
iters 650: tol = 0.0001313576040827824
iters 50: tol = 0.005391860527068859
iters 100: tol = 0.001392567371205411
iters 150: tol = 0.0002992795379963553
2005-12-01 00:00:00 is done and took 184 iterations and 117.95 seconds
iters 50: tol = 0.0025603109704039895
iters 50: tol = 0.00043102668997385685
iters 50: tol = 0.00021392363412992
iters 50: tol = 0.00030586135624399713
iters 50: tol = 0.004326902194147664
iters 100: tol = 0.0001753651580020943
iters 50: tol = 0.0011572411639441949
iters 50: tol = 0.0008765052597831113
iters 50: tol = 0.0009453161674954913
iters 100: tol = 0.00011773223382149922
iters 50: tol = 0.0011861553194061347
iters 100: tol = 0.00010241034738123211
iters 50: tol = 0.0004062479112002526
iters 50: tol = 0.00036771458717488326
iters 50: tol = 0.0002418886583741564
2006-12-01 00:00:00 is done and took 62 iterations and 97.92 seconds
iters 50: tol = 0.0011236851669783193
iters 50: tol = 0.0472939969569012
iters 100: tol = 0.08608960962903517
iters 150: tol = 0.013560803482594877
iters 200: tol = 0.0020189420776883515
iters 250: tol = 0.0001569284039418528
iters 50: tol = 0.0011491798789286037
iters 50: tol = 0.04350271888230471
iters 100: tol = 6.6450627228972765
iters 150: tol = 0.01109841740338302
iters 200: tol = 0.0005776222934330555
iters 50: tol = 0.00027956842995946474
iters 50: tol = 0.00039355123595741226
iters 50: tol = 0.00034340303266622296
iters 50: tol = 0.0011774705069946823
iters 100: tol = 0.00011452305489378922
iters 50: tol = 0.0003927300610615525
iters 50: tol = 0.00026908390134461335
iters 50: tol = 0.00014645080108177666
iters 50: tol = 0.000835818937491295
2007-12-01 00:00:00 is done and took 86 iterations and 134.00 seconds
iters 50: tol = 0.0014205900847987785
iters 100: tol = 0.00017125232290493564
iters 50: tol = 0.0016995184156075593
iters 100: tol = 0.0004865279144332657
iters 150: tol = 0.00015311835640585691
iters 50: tol = 0.0020723270541112004
iters 100: tol = 0.00030831473768289097
iters 50: tol = 0.0011755318708370766
iters 100: tol = 0.0002505990760547361
iters 50: tol = 0.0012796946043172053
iters 100: tol = 0.00032929909233914145
iters 50: tol = 0.031767382666843424
iters 100: tol = 0.008057774253608874
iters 150: tol = 0.001732238429586308
iters 200: tol = 0.00035828569741003236
iters 50: tol = 0.011654571660645985
iters 100: tol = 0.00721451118422245
iters 150: tol = 0.007628318641984211
iters 200: tol = 0.014932114028965882
iters 250: tol = 0.520459727156283
iters 300: tol = 0.009437288493640228
iters 350: tol = 0.002706031846161694
iters 400: tol = 0.0006565667398116259
iters 450: tol = 0.00016526816681439183
iters 50: tol = 0.003523861552909846
iters 100: tol = 0.0007353283802163102
iters 150: tol = 0.0001751604667493134
iters 50: tol = 0.0026788378151290093
iters 100: tol = 0.0006406824199495231
iters 150: tol = 0.00016680637061150527
iters 50: tol = 0.0007603433311719754
iters 100: tol = 0.00022085712656688683
iters 50: tol = 0.00152121879635847
iters 100: tol = 0.0003365472847600137
iters 150: tol = 0.00021876488670491412
iters 200: tol = 0.00013847622333851284
iters 50: tol = 0.003647159977978487
iters 100: tol = 0.0014104538102525654
iters 150: tol = 0.0007996127067449454
iters 200: tol = 0.0004617569647039077
iters 250: tol = 0.00026133087021631973
iters 300: tol = 0.00014566210816122083
2008-12-01 00:00:00 is done and took 332 iterations and 428.98 seconds
iters 50: tol = 0.0035564163525838577
iters 100: tol = 0.0012627921615729898
iters 150: tol = 0.00044579733731087146
iters 200: tol = 0.00017227174358613873
iters 50: tol = 0.0003472747783790364
iters 50: tol = 0.00042640301730068053
iters 100: tol = 0.000248289935256496
```

```
iters 150: tol = 0.0001424188078753552
iters 50: tol = 0.005329805388767372
iters 100: tol = 0.003996703767551291
iters 150: tol = 0.0026451622718364487
iters 200: tol = 0.0015211112062306423
iters 250: tol = 0.000872391982986942
iters 300: tol = 0.0004643654041083245
iters 350: tol = 0.0002336621270953021
iters 400: tol = 0.00011437023150989711
iters 50: tol = 0.008627357608397723
iters 100: tol = 0.004070821461870988
iters 150: tol = 0.002046466562225513
iters 200: tol = 0.0011421643706880369
iters 250: tol = 0.0006806033802451705
iters 300: tol = 0.0004129565170855609
iters 350: tol = 0.00025353211835021927
iters 400: tol = 0.00015682185882831545
iters 50: tol = 0.015751090705429106
iters 100: tol = 0.00576210628001661
iters 150: tol = 0.001023280471978305
iters 200: tol = 0.00016511913479755336
iters 50: tol = 0.002539130340517892
iters 100: tol = 0.0003484546155231305
iters 150: tol = 0.00011885317432969167
iters 50: tol = 0.16007247632166433
iters 100: tol = 0.011130771168362363
iters 150: tol = 0.004814366841401574
iters 200: tol = 0.0018697697678185174
iters 250: tol = 0.0006596838998498555
iters 300: tol = 0.0002243376620869736
iters 50: tol = 0.014771301361441225
iters 100: tol = 0.0008098536394756106
iters 150: tol = 0.00018332487273220455
iters 50: tol = 0.0025724137145215487
iters 100: tol = 0.0007001726712460021
iters 150: tol = 0.00018628418130384183
iters 50: tol = 0.005094464423163747
iters 100: tol = 0.0025752047114669663
iters 150: tol = 0.0014129697563207422
iters 200: tol = 0.0008218725014282058
iters 250: tol = 0.0004953220767181937
iters 300: tol = 0.00030496978018961646
iters 350: tol = 0.000190230197314116
iters 400: tol = 0.00011961604805124995
iters 50: tol = 0.0023833916003226374
iters 100: tol = 0.0012442297090929921
iters 150: tol = 0.0008835406299055881
iters 200: tol = 0.0006157719012429563
iters 250: tol = 0.0004321958080975574
iters 300: tol = 0.0003057749049897862
iters 350: tol = 0.00021762057961094428
iters 400: tol = 0.0001555299017654277
iters 450: tol = 0.00011149539583548557
2009-12-01 00:00:00 is done and took 467 iterations and 366.65 seconds
iters 50: tol = 0.0027512639867048883
iters 100: tol = 0.0013176963893680727
iters 150: tol = 0.0005801599830603243
iters 200: tol = 0.00024697529108780314
iters 250: tol = 0.00010365107117527028
iters 50: tol = 0.0009113030293024238
iters 100: tol = 0.00032829547212359644
iters 150: tol = 0.00011753905293804268
iters 50: tol = 0.0006089800022701652
iters 100: tol = 0.00021737714298675215
iters 50: tol = 0.012156149879989975
iters 100: tol = 0.006897108680361308
iters 150: tol = 0.003645516469479526
iters 200: tol = 0.0021787208604655967
iters 250: tol = 0.001393463516934923
iters 300: tol = 0.0009297833513699227
iters 350: tol = 0.0006426617062604834
iters 400: tol = 0.0004552036076479471
iters 450: tol = 0.00032722694036246835
iters 500: tol = 0.00023803416947798528
iters 550: tol = 0.0001749435606192007
iters 600: tol = 0.00012927653246952442
iters 50: tol = 0.0032082388563527964
iters 100: tol = 0.001975634122764802
iters 150: tol = 0.0013263310080575685
iters 200: tol = 0.0009567201345452303
iters 250: tol = 0.0008095303350267868
iters 300: tol = 0.0006600361817696576
iters 350: tol = 0.0005144827695452991
iters 400: tol = 0.0003844222132469155
iters 450: tol = 0.0002773923659941946
iters 500: tol = 0.00019490678048494914
iters 550: tol = 0.00013433157292649933
iters 50: tol = 0.05496632203768326
iters 100: tol = 0.003309668365617302
iters 150: tol = 0.0006559089254184469
iters 200: tol = 0.0001668765872616662
iters 50: tol = 0.001132593171463192
iters 100: tol = 0.0001010667065679538
iters 50: tol = 0.007768526875881435
```

```
iters 100: tol = 0.003308190050544002
iters 150: tol = 0.001767046650153703
iters 200: tol = 0.0011471534591339372
iters 250: tol = 0.0008188321044504399
iters 300: tol = 0.0006159306227393191
iters 350: tol = 0.0004801161544393667
iters 400: tol = 0.0003837118762370223
iters 450: tol = 0.0003122112331015933
iters 500: tol = 0.00025738758576154125
iters 550: tol = 0.0002142687618383876
iters 600: tol = 0.00017968630142152758
iters 650: tol = 0.00015152832136879253
iters 700: tol = 0.00012833147218062856
iters 750: tol = 0.00010904695145608906
iters 50: tol = 0.026125273031309626
iters 100: tol = 0.0015352052315886766
iters 150: tol = 0.0002801356406697958
iters 50: tol = 0.0024509559229475286
iters 100: tol = 0.0012896501439334518
iters 150: tol = 0.0009431488602604787
iters 200: tol = 0.0006871199947664497
iters 250: tol = 0.00046933870378029763
iters 300: tol = 0.00030845156467007584
iters 350: tol = 0.0001977872095918487
iters 400: tol = 0.0001248537950023776
iters 50: tol = 0.002603875701885361
iters 100: tol = 0.0012046053862494466
iters 150: tol = 0.0006547814515844783
iters 200: tol = 0.0003624241821903948
iters 250: tol = 0.00019784953116622553
iters 300: tol = 0.0001075281914205764
iters 50: tol = 0.0011462374758580349
iters 100: tol = 0.0006362830811129511
iters 150: tol = 0.00037672738777722037
iters 200: tol = 0.0002290827314081617
iters 250: tol = 0.00014066908116784627
2010-12-01 00:00:00 is done and took 286 iterations and 197.69 seconds
iters 50: tol = 0.001088536657601058
iters 100: tol = 0.0007582481488867254
iters 150: tol = 0.0005708186484909006
iters 200: tol = 0.0004387321327637972
iters 250: tol = 0.00034084390090410865
iters 300: tol = 0.0002667786095704727
iters 350: tol = 0.00020997846163262057
iters 400: tol = 0.00016598162418701712
iters 450: tol = 0.0001316399723185202
iters 500: tol = 0.0001046746569395296
iters 50: tol = 0.0021788817488767043
iters 100: tol = 0.0016630578110866656
iters 150: tol = 0.0009900239400274513
iters 200: tol = 0.0005659295783176921
iters 250: tol = 0.0003210542335065625
iters 300: tol = 0.00018133142712076222
iters 350: tol = 0.00010182486737050911
iters 50: tol = 0.0001802657934935059
iters 50: tol = 0.0023613395644357174
iters 100: tol = 0.0009260806334311471
iters 150: tol = 0.00042861166645975085
iters 200: tol = 0.00021052856266737252
iters 250: tol = 0.00010552465007772349
iters 50: tol = 0.018280348580321015
iters 100: tol = 0.007485493081895056
iters 150: tol = 0.004897247113649916
iters 200: tol = 0.004057682286329994
iters 250: tol = 0.004046610744604149
iters 300: tol = 0.0046034453745502635
iters 350: tol = 0.0056537895242455005
iters 400: tol = 0.007037764979917149
iters 450: tol = 0.008078802688395115
iters 500: tol = 0.008019019894372481
iters 550: tol = 0.006489452012528041
iters 600: tol = 0.005034610590141098
iters 650: tol = 0.005321884908737418
iters 700: tol = 0.0066926214552150975
iters 750: tol = 0.013456812103794569
iters 800: tol = 0.05061846695381256
iters 850: tol = 0.025542287459104696
iters 900: tol = 0.005177704056390509
iters 950: tol = 0.0015302554485492337
iters 1000: tol = 0.0005667276007690347
iters 1050: tol = 0.00021148838407913928
iters 50: tol = 0.0017174019333520696
iters 100: tol = 0.0002609935494476412
iters 50: tol = 0.0012111846366229528
iters 100: tol = 0.00026652528663762
iters 50: tol = 0.003055544200731175
iters 100: tol = 0.000782312290361159
iters 150: tol = 0.00022415213073995188
iters 50: tol = 0.0021439591605791897
iters 100: tol = 5.766650562948763
iters 150: tol = 0.00012363080076860378
iters 50: tol = 0.0031938729708730906
iters 100: tol = 0.0005432090658384348
iters 50: tol = 0.0022320340763926083
```

```
iters 100: tol = 9.50191471391637e-05
iters 50: tol = 0.00011711960950211431
2011-12-01 00:00:00 is done and took 53 iterations and 88.29 seconds
iters 50: tol = 0.000364314489019725
iters 50: tol = 0.0006902183866510647
iters 50: tol = 0.0007806199453250784
iters 100: tol = 9.769882389165652e-05
iters 50: tol = 0.0011409028944363575
iters 100: tol = 0.00010655223668765146
iters 50: tol = 0.0003021785063632709
iters 50: tol = 0.00030251295978905857
iters 50: tol = 0.0007794294636666588
iters 100: tol = 0.0001442395167027266
iters 50: tol = 0.0034777875860947383
iters 100: tol = 0.001192233374763485
iters 150: tol = 0.0005065714266742072
iters 200: tol = 0.00022644282113470915
iters 250: tol = 0.0001021968725481992
iters 50: tol = 0.0022815574759592616
iters 100: tol = 0.001207690061122424
iters 150: tol = 0.0006322276861963072
iters 200: tol = 0.00033933180907635974
iters 250: tol = 0.0001789973055281724
iters 50: tol = 0.0015739393803911872
iters 100: tol = 0.0003866676918866663
iters 150: tol = 0.0003101789349639894
iters 200: tol = 0.00018680300807599748
iters 250: tol = 0.00010765235197024436
iters 50: tol = 0.008492976663015628
iters 100: tol = 0.01758964205775082
iters 150: tol = 0.011503298061352668
iters 200: tol = 0.004415456706400311
iters 250: tol = 0.0017519073990447387
iters 300: tol = 0.0007442654033992802
iters 350: tol = 0.00032824779424356354
iters 400: tol = 0.0001473842914773149
iters 50: tol = 0.012355604101175026
iters 100: tol = 0.007051615457826621
iters 150: tol = 0.0028029737437870184
iters 200: tol = 0.001108700812392005
iters 250: tol = 0.00044958932717720224
iters 300: tol = 0.0001848617923193574
2012-12-01 00:00:00 is done and took 335 iterations and 247.38 seconds
iters 50: tol = 0.011843310430691811
iters 100: tol = 0.0046536060737027984
iters 150: tol = 0.001950687217435798
iters 200: tol = 0.0009180463924111371
iters 250: tol = 0.00046356221716215007
iters 300: tol = 0.00024331824207921016
iters 350: tol = 0.0001304278478893306
iters 50: tol = 0.017475306239433053
iters 100: tol = 0.017258442698964677
iters 150: tol = 0.005854032321684299
iters 200: tol = 0.001761800430151117
iters 250: tol = 0.0007034783592798632
iters 300: tol = 0.0003645583864333446
iters 350: tol = 0.00017494849685373293
iters 50: tol = 0.007345366053783164
iters 100: tol = 0.003851870361081644
iters 150: tol = 0.0018092964198908844
iters 200: tol = 0.0008190003893817979
iters 250: tol = 0.0003676067021600171
iters 300: tol = 0.00016469936173911515
iters 50: tol = 0.006801486405745183
iters 100: tol = 0.002623579356317851
iters 150: tol = 0.001202950227093158
iters 200: tol = 0.0005280954641545543
iters 250: tol = 0.0002290012733995317
iters 300: tol = 9.89769928112505e-05
iters 50: tol = 0.0828592679878397
iters 100: tol = 0.005098501373788533
iters 150: tol = 0.002623857222726844
iters 200: tol = 0.0012038649432726256
iters 250: tol = 0.0005130743350422184
iters 300: tol = 0.00021181106933365612
iters 50: tol = 0.003988834959512633
iters 100: tol = 0.0017447274959509484
iters 150: tol = 0.0006022111740693892
iters 200: tol = 0.00019174971313073996
iters 50: tol = 0.0082344664590418
iters 100: tol = 0.0018173708324161764
iters 150: tol = 0.00042724863582788153
iters 200: tol = 0.00011313984727978621
iters 50: tol = 0.003808788558909426
iters 100: tol = 0.004647013338053174
iters 150: tol = 0.01929366931841732
iters 200: tol = 0.043512868923712984
iters 250: tol = 0.0045491752393916896
iters 300: tol = 0.0002984428972205855
iters 50: tol = 0.008793423694566621
iters 100: tol = 0.00029107766302460814
iters 50: tol = 0.0009592618164171451
iters 50: tol = 0.0003821821265097558
iters 50: tol = 0.0008027831181999234
```

```
2013-12-01 00:00:00 is done and took 86 iterations and 200.49 seconds
iters 50: tol = 0.001239345637938316
iters 100: tol = 0.0001555725301749522
iters 50: tol = 0.00280250332671525
iters 100: tol = 0.00046353104410333523
iters 50: tol = 0.00044842626329044677
iters 50: tol = 0.001222288721412168
iters 100: tol = 0.00014540810343377508
iters 50: tol = 0.0005586156872599002
iters 50: tol = 0.0007083450112890244
iters 100: tol = 0.00012449700800831742
iters 50: tol = 0.005923954886400906
iters 100: tol = 0.006989767424114168
iters 150: tol = 0.0077641297496439665
iters 200: tol = 0.002500321429574376
iters 250: tol = 0.0007524156032588936
iters 300: tol = 0.0002711665194797819
iters 50: tol = 0.0017022444213892207
iters 100: tol = 0.0002902995972855482
iters 50: tol = 0.004703638999333959
iters 100: tol = 0.0008950741838383847
iters 150: tol = 0.00013203284328422438
iters 50: tol = 0.0005166083093045559
iters 50: tol = 0.0006343365382137645
iters 50: tol = 0.004533910501173133
iters 100: tol = 0.0006170149539842518
2014-12-01 00:00:00 is done and took 143 iterations and 115.97 seconds
iters 50: tol = 0.005760977217817098
iters 100: tol = 0.0008002337437253981
iters 50: tol = 0.0028412429767890046
iters 100: tol = 0.0002497131054268742
iters 50: tol = 0.020428389323409443
iters 100: tol = 0.0012076236516166405
iters 150: tol = 0.00010931417633308627
iters 50: tol = 0.00770779062519622
iters 100: tol = 0.00026817972332970896
iters 50: tol = 0.0033951966489638163
iters 100: tol = 0.0002387636033669427
iters 50: tol = 0.0013013345162262713
iters 100: tol = 0.00012459119756980108
iters 50: tol = 0.006648702494983505
iters 100: tol = 0.0017859989125743603
iters 150: tol = 0.0011278570740245186
iters 200: tol = 0.0007135472715531499
iters 250: tol = 0.00045186696819676797
iters 300: tol = 0.0002862618015160301
iters 350: tol = 0.00018139545204620688
iters 400: tol = 0.00011497060379284108
iters 50: tol = 0.008941642114154824
iters 100: tol = 0.003034990033568885
iters 150: tol = 0.0016777529004948222
iters 200: tol = 0.0022091279088068405
iters 250: tol = 0.003830393908173635
iters 300: tol = 0.004411233520596247
iters 350: tol = 0.0022842124932158647
iters 400: tol = 0.0007628762188582883
iters 450: tol = 0.000254784480674064
iters 50: tol = 0.0010556776169230453
iters 100: tol = 0.00023032567756986477
iters 50: tol = 0.0009915792424278758
iters 100: tol = 0.00014052628344768392
iters 50: tol = 0.000899653794708577
iters 100: tol = 0.0001285801836277134
iters 50: tol = 0.002454452976513505
iters 100: tol = 0.00045210104422738207
iters 150: tol = 9.983713265304672e-05
2015-12-01 00:00:00 is done and took 150 iterations and 133.60 seconds
iters 50: tol = 0.00157214229969882
iters 100: tol = 0.0006327632645497516
iters 150: tol = 0.0002906378893765549
iters 200: tol = 0.0001340155902136430 7
iters 50: tol = 0.007130314196635235
iters 100: tol = 0.002762661343162387
iters 150: tol = 0.0012235932909132607
iters 200: tol = 0.0006228665993295301
iters 250: tol = 0.0003025197705908145
iters 300: tol = 0.00014375404907740474
iters 50: tol = 0.0009646197204643547
iters 100: tol = 0.0006112063480219337
iters 150: tol = 0.00031746262173328044
iters 200: tol = 0.00015351148574629936
iters 50: tol = 0.005286709430038894
iters 100: tol = 0.0024213154038772444
iters 150: tol = 0.001407366826746248
iters 200: tol = 0.0008917574284224017
iters 250: tol = 0.0006495776196867586
iters 300: tol = 0.0006208708517396744
iters 350: tol = 0.0007511641272830438
iters 400: tol = 0.0011392194780250886
iters 450: tol = 0.0020429619037252156
iters 500: tol = 0.003953372987981485
iters 550: tol = 0.006169939914870762
iters 600: tol = 0.004419987728323083
iters 650: tol = 0.0063330351218476855
```

```
iters 700: tol = 0.011327376203480344
iters 750: tol = 0.009099573851782616
iters 800: tol = 0.013863401536633796
iters 850: tol = 0.005831004179173047
iters 900: tol = 0.008758340693076017
iters 950: tol = 0.03436913399165912
iters 1000: tol = 0.025867682880682474
iters 1050: tol = 0.00267858472774668
iters 1100: tol = 0.00020218583887512964
iters 50: tol = 0.0054450716011767986
iters 100: tol = 0.0015075523692409387
iters 150: tol = 0.0004405204677554453
iters 200: tol = 0.00013078917218101171
iters 50: tol = 0.004899291059884181
iters 100: tol = 0.0020372793957341706
iters 150: tol = 0.0009072361065518564
iters 200: tol = 0.0004131105433571314
iters 250: tol = 0.0001899929589650462
iters 50: tol = 0.00478595688213912
iters 100: tol = 0.0016533946937968835
iters 150: tol = 0.0005820971086044624
iters 200: tol = 0.00020031288789412738
iters 50: tol = 0.007363132410416917
iters 100: tol = 0.001890934594811322
iters 150: tol = 0.000581752332664609
iters 200: tol = 0.0002538795824408302
iters 250: tol = 0.00012115970913395557
iters 50: tol = 0.0022980794263450233
iters 100: tol = 0.0010343513875842314
iters 150: tol = 0.0008300219872079406
iters 200: tol = 0.0006517610974199339
iters 250: tol = 0.0005682993598437047
iters 300: tol = 0.0006750523917701257
iters 350: tol = 0.0008313963263252844
iters 400: tol = 0.0010494376822135498
iters 450: tol = 0.0013531704285795199
iters 500: tol = 0.0017915798553187479
iters 550: tol = 0.0024400604117931213
iters 600: tol = 0.003393960384537431
iters 650: tol = 0.004667855493382889
iters 700: tol = 0.0064490355288353285
iters 750: tol = 0.0073128761282835325
iters 800: tol = 0.006219450422608408
iters 850: tol = 0.004588162105875648
iters 900: tol = 0.003331412182742133
iters 950: tol = 0.0024033801428912094
iters 1000: tol = 0.0017747959986376127
iters 1050: tol = 0.0013513968057060621
iters 1100: tol = 0.0010602339430370589
iters 1150: tol = 0.0008544793932164785
iters 1200: tol = 0.0007051074751256436
iters 1250: tol = 0.0005939823810204858
iters 1300: tol = 0.0005095221638795056
iters 1350: tol = 0.00044413892472866534
iters 1400: tol = 0.0003927330264105189
iters 1450: tol = 0.000351794784055115
iters 1500: tol = 0.00031885598892891776
iters 1550: tol = 0.00029214721754328155
iters 1600: tol = 0.0002717368996692781
iters 1650: tol = 0.0002556180132729513
iters 1700: tol = 0.00024283322072463087
iters 1750: tol = 0.00023290969299097353
iters 1800: tol = 0.0002255153697455492
iters 1850: tol = 0.00022217856338568875
iters 1900: tol = 0.00022472554017871627
iters 1950: tol = 0.00023060792628554205
iters 2000: tol = 0.00024341926849325013
iters 2050: tol = 0.00026549954536791986
iters 2100: tol = 0.00032341215864374606
iters 2150: tol = 0.00040602916246876286
iters 2200: tol = 0.0005289177697967162
iters 2250: tol = 0.000721694817088607
iters 2300: tol = 0.00104407563764998
iters 2350: tol = 0.0016207350836073692
iters 2400: tol = 0.002676114929334561
iters 2450: tol = 0.004129180599258764
iters 2500: tol = 0.005512305659902461
iters 2550: tol = 0.005145595572243811
iters 2600: tol = 0.003043662632419092
iters 2650: tol = 0.0019559319166775613
iters 2700: tol = 0.00132465415518912
iters 2750: tol = 0.0009210237789615272
iters 2800: tol = 0.0006343626007661585
iters 2850: tol = 0.0004341298428375828
iters 2900: tol = 0.0002957378032629743
iters 2950: tol = 0.00020081580110675734
iters 3000: tol = 0.0001360596111332213
iters 50: tol = 0.007777526293120007
iters 100: tol = 0.0029662608397644785
iters 150: tol = 0.0011421186411171291
iters 200: tol = 0.0004360574042631127
iters 250: tol = 0.00016746660587274587
iters 50: tol = 0.001502173177856836
iters 100: tol = 0.00033642301866487756
```

```
iters 150: tol = 0.00014638339961725588
iters 50: tol = 0.0012893907076660938
iters 100: tol = 0.00026103089309037486
iters 150: tol = 0.00011907562894863943
2016-12-01 00:00:00 is done and took 162 iterations and 166.82 seconds
iters 50: tol = 0.0016879056719663055
iters 100: tol = 0.00042744182883749926
iters 150: tol = 0.000148069500182757
iters 50: tol = 0.003593220305317363
iters 100: tol = 0.0021938626304155717
iters 150: tol = 0.0018701195771290702
iters 200: tol = 0.0018800102306940625
iters 250: tol = 0.0020968277764769477
iters 300: tol = 0.0024252831324027696
iters 350: tol = 0.0025265660818580005
iters 400: tol = 0.002306966234150387
iters 450: tol = 0.0018308649973824975
iters 500: tol = 0.0015584471338092598
iters 550: tol = 0.001586115057642598
iters 600: tol = 0.0012439872315590378
iters 650: tol = 0.0008031249396102114
iters 700: tol = 0.00046571305752662884
iters 750: tol = 0.00025602877823691084
iters 800: tol = 0.00013725959014390665
iters 50: tol = 0.00484743743720184
iters 100: tol = 0.004323477821914445
iters 150: tol = 0.0026223076217525287
iters 200: tol = 0.0015250525567830908
iters 250: tol = 0.0009194128225233356
iters 300: tol = 0.0005779578495304083
iters 350: tol = 0.00037580487848948296
iters 400: tol = 0.00025048152320145123
iters 450: tol = 0.00016987682843977447
iters 500: tol = 0.00011659683499826157
iters 50: tol = 0.0021642684816865487
iters 100: tol = 0.0007114117588888913
iters 150: tol = 0.000318680681285044
iters 200: tol = 0.00014791066223085458
iters 50: tol = 0.0018177129353931232
iters 100: tol = 0.0012283265364137763
iters 150: tol = 0.0007975002038160678
iters 200: tol = 0.000501859518169005
iters 250: tol = 0.0003101088058309921
iters 300: tol = 0.00018974720705258047
iters 350: tol = 0.00011550895685120965
iters 50: tol = 0.0018098997649805826
iters 100: tol = 0.0005986186179306463
iters 150: tol = 0.00033610786214403887
iters 200: tol = 0.00020663793160292931
iters 250: tol = 0.00012408889626826236
iters 50: tol = 0.003954877325055117
iters 100: tol = 0.003244923845368841
iters 150: tol = 0.0023992619832624573
iters 200: tol = 0.0016274780578352036
iters 250: tol = 0.0010698536582911672
iters 300: tol = 0.0006917450328092142
iters 350: tol = 0.0004452831343118649
iters 400: tol = 0.000287279056430334
iters 450: tol = 0.00018501342140864185
iters 500: tol = 0.00011895450002696872
iters 50: tol = 0.002948682052085816
iters 100: tol = 0.0010479840355147596
iters 150: tol = 0.0005197078833495405
iters 200: tol = 0.0002613500879132258
iters 250: tol = 0.00013091922367602926
iters 50: tol = 0.004192679265345944
iters 100: tol = 0.002368565883108742
iters 150: tol = 0.0013433848103269752
iters 200: tol = 0.0007460385340710418
iters 250: tol = 0.0004119168601295631
iters 300: tol = 0.00022723791483392208
iters 350: tol = 0.0001253948675068728
iters 50: tol = 0.0038321066771117296
iters 100: tol = 0.0015623246809713387
iters 150: tol = 0.0006018888090268404
iters 200: tol = 0.00022858063874098278
iters 50: tol = 0.0015534118760924809
iters 100: tol = 0.0005801768656714112
iters 150: tol = 0.00020449812755515828
iters 50: tol = 0.0011368981995533156
iters 100: tol = 0.00035349105111026624
iters 150: tol = 0.00013281159986133773
2017-12-01 00:00:00 is done and took 165 iterations and 143.34 seconds
iters 50: tol = 0.0031273868728260226
iters 100: tol = 0.0009212059334419986
iters 150: tol = 0.00027288000011051194
iters 50: tol = 0.004487713796094028
iters 100: tol = 0.0017478101253030898
iters 150: tol = 0.0007081621626992551
iters 200: tol = 0.00029625179855871653
iters 250: tol = 0.00013085601338724828
iters 50: tol = 0.0012533294755860958
iters 100: tol = 0.00029767381733236675
iters 50: tol = 0.0017808025446500175
```

```
iters 100: tol = 0.000558376829142039
iters 150: tol = 0.0002310378778112021
iters 200: tol = 0.00010550570662393177
iters 50: tol = 0.0023361701560761228
iters 100: tol = 0.0012405732965595773
iters 150: tol = 0.0008425710219855942
iters 200: tol = 0.0006179199560543136
iters 250: tol = 0.0004755213823660065
iters 300: tol = 0.0003811064176843515
iters 350: tol = 0.00031199067606968445
iters 400: tol = 0.00025949093800703427
iters 450: tol = 0.00021845810799470122
iters 500: tol = 0.00018565847086210452
iters 550: tol = 0.0001589656840000897
iters 600: tol = 0.00013692673575504966
iters 650: tol = 0.00011851557143960356
iters 700: tol = 0.00010298675705555649
iters 50: tol = 0.006545684180799205
iters 100: tol = 0.003587182968343777
iters 150: tol = 0.001206098093634922
iters 200: tol = 0.00034578231220194766
iters 50: tol = 0.0020490129108252098
iters 100: tol = 0.0004169811254471911
iters 50: tol = 0.0007692509666488156
iters 100: tol = 0.00011965560437499079
iters 50: tol = 0.0017944654613086808
iters 100: tol = 0.0008266265923955185
iters 150: tol = 0.0003840050120513805
iters 200: tol = 0.0001820995266608172
iters 50: tol = 0.0030565144074017336
iters 100: tol = 0.0020831822708877734
iters 150: tol = 0.001406089411833955
iters 200: tol = 0.0009113987529948409
iters 250: tol = 0.0005741688661000666
iters 300: tol = 0.0003542701265775161
iters 350: tol = 0.00021548870756915584
iters 400: tol = 0.0001298635226361089
iters 50: tol = 0.0019786593092242233
iters 100: tol = 0.0005357867671740291
iters 150: tol = 0.00014200258204022376
iters 50: tol = 0.0002595445872354274
2018-12-01 00:00:00 is done and took 77 iterations and 69.54 seconds
iters 50: tol = 0.0014074070825212948
iters 100: tol = 0.0006315743419743614
iters 150: tol = 0.00030634239920990236
iters 200: tol = 0.00014862814492466736
iters 50: tol = 0.0030323383072245746
iters 100: tol = 0.0010575696332999485
iters 150: tol = 0.00035398735672903525
iters 200: tol = 0.00011511848383355394
iters 50: tol = 0.0007707068569242082
iters 100: tol = 0.00028522117378093625
iters 150: tol = 0.00010254078572624614
iters 50: tol = 0.00245911694457307
iters 100: tol = 0.0006511165200833238
iters 150: tol = 0.00020805970621510378
iters 50: tol = 0.0014002732822381292
iters 100: tol = 0.000565233509173868
iters 150: tol = 0.0002288115077898334
iters 50: tol = 0.0012415999100144903
iters 100: tol = 0.00062722676744012
iters 150: tol = 0.00028302717395956023
iters 200: tol = 0.00012678602361532176
iters 50: tol = 0.0005471760673958492
iters 100: tol = 0.00023783096220025834
iters 150: tol = 0.00010048752699143293
iters 50: tol = 0.0003439777118227072
iters 50: tol = 0.00037960165431893955
iters 100: tol = 0.0001093192209213889
iters 50: tol = 0.0017718092739404945
iters 100: tol = 0.0008627458309654404
iters 150: tol = 0.000473698959042727
iters 200: tol = 0.00026839417845795494
iters 250: tol = 0.00015387657635540508
iters 50: tol = 0.0010264979008598601
iters 100: tol = 0.0002533132047067843
iters 150: tol = 0.00019928260794325765
iters 200: tol = 0.00017976589331685378
iters 250: tol = 0.0001461388122725709
iters 300: tol = 0.00011506670536992267
iters 50: tol = 0.0015801216254102551
iters 100: tol = 0.0011262224053573446
iters 150: tol = 0.0008115379043529131
iters 200: tol = 0.0005937842701072282
iters 250: tol = 0.0004394430762700019
iters 300: tol = 0.00032866961890992696
iters 350: tol = 0.0002482020409962371
iters 400: tol = 0.00018901085825873132
iters 450: tol = 0.0001449443256150429
iters 500: tol = 0.00011178511451434558
2019-12-01 00:00:00 is done and took 522 iterations and 478.12 seconds
iters 50: tol = 0.0022737626705129355
iters 100: tol = 0.0013303960092401201
iters 150: tol = 0.0009638783194904355
```

```
iters 200: tol = 0.0007963925725593263
iters 250: tol = 0.0007099457808763185
iters 300: tol = 0.0006620076343860151
iters 350: tol = 0.0006412115731461304
iters 400: tol = 0.0006398093220189305
iters 450: tol = 0.0006491528035672267
iters 500: tol = 0.0006802800275146903
iters 550: tol = 0.0007232558170267083
iters 600: tol = 0.0007738481481268239
iters 650: tol = 0.0008293780888896651
iters 700: tol = 0.0008853741958342598
iters 750: tol = 0.0009409769643918542
iters 800: tol = 0.0009920137389741457
iters 850: tol = 0.0010179237055270851
iters 900: tol = 0.0010087464138447966
iters 950: tol = 0.000959699636047584
iters 1000: tol = 0.0008739184600731753
iters 1050: tol = 0.0007619948922046316
iters 1100: tol = 0.0006383121790799473
iters 1150: tol = 0.0005164364904542418
iters 1200: tol = 0.00040601941460827184
iters 1250: tol = 0.0003120328544835016
iters 1300: tol = 0.00023563989036362587
iters 1350: tol = 0.00017561605993426932
iters 1400: tol = 0.00012960496367603325
iters 50: tol = 0.010147643469394008
iters 100: tol = 0.003933833081928753
iters 150: tol = 0.00130448504736332
iters 200: tol = 0.0005011813531183434
iters 250: tol = 0.0002256299668189632
iters 300: tol = 0.00012305710099047573
iters 50: tol = 0.006925899273331426
iters 100: tol = 0.0037313271047099833
iters 150: tol = 0.0016057749718332293
iters 200: tol = 0.0006099622722766929
iters 250: tol = 0.00022511167173205893
iters 50: tol = 0.001617037966119561
iters 100: tol = 0.000339603807169242
iters 150: tol = 0.0001279996359558888
iters 50: tol = 0.003998772732001665
iters 100: tol = 0.001394514709452288
iters 150: tol = 0.0007733433943510892
iters 200: tol = 0.0004182435348596414
iters 250: tol = 0.0002134836081777447
iters 300: tol = 0.00010568199842508896
iters 50: tol = 0.002261931494849545
iters 100: tol = 0.0009637483884985809
iters 150: tol = 0.00040752408826082165
iters 200: tol = 0.00018761695207687723
iters 50: tol = 0.002455858865535898
iters 100: tol = 0.0009008308477989058
iters 150: tol = 0.0004804649591915733
iters 200: tol = 0.00027454691455147673
iters 250: tol = 0.00015556789483639477
iters 50: tol = 0.0012059432090860689
iters 100: tol = 0.00036998045064629004
iters 150: tol = 0.00015902380077159606
iters 50: tol = 0.001140401061786811
iters 100: tol = 0.00041084762507992423
iters 150: tol = 0.00016291873956664205
iters 50: tol = 0.0010248504162064243
iters 100: tol = 0.0002870989621437947
iters 50: tol = 0.0011243006572714265
iters 100: tol = 0.00027836347108811665
iters 50: tol = 0.0007638261655565359
iters 100: tol = 0.00023099119753213371
2020-12-01 00:00:00 is done and took 137 iterations and 128.09 seconds
iters 50: tol = 0.0013389411983387722
iters 100: tol = 0.00047729431358023433
iters 150: tol = 0.00014870124148536057
iters 50: tol = 0.0004978362543457404
iters 100: tol = 0.0001861006677326138
iters 50: tol = 0.0007041324059750276
iters 50: tol = 0.0023386544557391087
iters 100: tol = 0.0009722992483989223
iters 150: tol = 0.0003993189672883979
iters 200: tol = 0.0001630093273539135
iters 50: tol = 0.0005735262985167022
iters 100: tol = 0.0001938925725479823
iters 50: tol = 0.00013540887516755307
iters 50: tol = 0.00045803182216253013
iters 100: tol = 0.00013325637653482936
iters 50: tol = 0.012764112290726659
iters 100: tol = 0.002911979961158162
iters 150: tol = 0.0005643315065773646
iters 200: tol = 0.00012486829662572418
iters 50: tol = 0.0030954543936417
iters 100: tol = 0.000741106452175358
iters 150: tol = 0.00017435375458750568
iters 50: tol = 0.016237643172082872
iters 100: tol = 0.004257110877238984
iters 150: tol = 0.001072175846017008
iters 200: tol = 0.00031967093425195464
iters 250: tol = 0.00010104669335118076
```

```
iters 50: tol = 0.005126933896540398
iters 100: tol = 0.001308907235160639
iters 150: tol = 0.00033346386864924193
iters 50: tol = 0.00035429677296677786
2021-12-01 00:00:00 is done and took 96 iterations and 152.96 seconds
iters 50: tol = 0.0016854290894368074
iters 100: tol = 0.0008406150962223524
iters 150: tol = 0.00042518327738960693
iters 200: tol = 0.00021347522479642222
iters 250: tol = 0.00010664938828797155
iters 50: tol = 0.0007839216955551342
iters 100: tol = 0.0005406888485888217
iters 150: tol = 0.0003827382951258062б
iters 200: tol = 0.0002657774351264574б
iters 250: tol = 0.0001837951273752081
iters 300: tol = 0.0001272584723190917
iters 50: tol = 0.025014948708546758
iters 100: tol = 0.011499896427077605
iters 150: tol = 0.0025671023530025594
iters 200: tol = 0.0010256975257880718
iters 250: tol = 0.0007338471423570248
iters 300: tol = 0.0007866437357979539
iters 350: tol = 0.0010142062916276284
iters 400: tol = 0.0013356036647191871
iters 450: tol = 0.0018623221764229037
iters 500: tol = 0.0028127182833952435
iters 550: tol = 0.004660042856767133
iters 600: tol = 0.010107235172833295
iters 650: tol = 0.02133063284206166
iters 700: tol = 0.027587061455178974
iters 750: tol = 0.00992556151304691
iters 800: tol = 0.004059197192671338
iters 850: tol = 0.0012969737497772194
iters 900: tol = 0.00041323481310673316
iters 950: tol = 0.00014369899272381748
iters 50: tol = 0.001648546911940052
iters 100: tol = 0.0009402706647851922
iters 150: tol = 0.0006422835727365045
iters 200: tol = 0.00042314951594023265
iters 250: tol = 0.0002805838683388323
iters 300: tol = 0.00018840368080957903
iters 350: tol = 0.0001277819689398385
iters 50: tol = 0.0012003322187969512
iters 100: tol = 0.00011705934211425628
iters 50: tol = 0.0011077969975534785
iters 100: tol = 0.0010546027150410975
iters 150: tol = 0.0009678307072986669
iters 200: tol = 0.0008436182262437919
iters 250: tol = 0.0007100557507370153
iters 300: tol = 0.0005833918514294156
iters 350: tol = 0.000471089982049977
iters 400: tol = 0.0003757269921959294
iters 450: tol = 0.00029704370531077884
iters 500: tol = 0.00023337425689609614
iters 550: tol = 0.00018253574773263725
iters 600: tol = 0.000142315108725255
iters 650: tol = 0.00011069956357673272
iters 50: tol = 0.004532614492227416
iters 100: tol = 0.0052986895190133
iters 150: tol = 0.004026492424667016
iters 200: tol = 0.002899029427156141
iters 250: tol = 0.0020836857092384475
iters 300: tol = 0.0014517519460529593
iters 350: tol = 0.0009929604393147384
iters 400: tol = 0.000673748170473637
iters 450: tol = 0.00045563896692454864
iters 500: tol = 0.0003077409409245613
iters 550: tol = 0.0002077599755552495
iters 600: tol = 0.00014024843465462733
iters 50: tol = 0.004596403565226503
iters 100: tol = 0.002738858424595092
iters 150: tol = 0.0014386466850327961
iters 200: tol = 0.0006515213393518682
iters 250: tol = 0.00027704636615866196
iters 300: tol = 0.00011486757360101851
iters 50: tol = 0.0032346455284154585
iters 100: tol = 0.0011538365893806746
iters 150: tol = 0.00038397314414001515
iters 200: tol = 0.00012167260413181724
iters 50: tol = 0.004348575341053484
iters 100: tol = 0.000678850244871132
iters 150: tol = 0.00014175467039878598
iters 50: tol = 0.0020490654047482515
iters 100: tol = 0.00045404293937939544
iters 150: tol = 0.0001684511773119013
iters 50: tol = 0.0010295609870650813
iters 100: tol = 0.00026214107332361847
2022-12-01 00:00:00 is done and took 135 iterations and 133.12 seconds
iters 50: tol = 0.0008923686662739017
iters 100: tol = 0.00016800973002439878
iters 50: tol = 0.0009625340143055161
iters 100: tol = 0.00013454409291899228
iters 50: tol = 0.001457661379783426
iters 100: tol = 0.00014272490581102026
```

```
iters 50: tol = 0.0013317685176140737
iters 100: tol = 0.00021644112962881934
iters 50: tol = 0.008776730874115755
iters 100: tol = 0.0027140145684114558
iters 150: tol = 0.0011789566092793718
iters 200: tol = 0.000519586622555579
iters 250: tol = 0.00024085153657371627
iters 300: tol = 0.0001121873480163238
iters 50: tol = 0.0022784129747372983
iters 100: tol = 0.001114327637506407
iters 150: tol = 0.0007431512754474956
iters 200: tol = 0.0005753326514064128
iters 250: tol = 0.00047254918384781464
iters 300: tol = 0.0004100696428556705
iters 350: tol = 0.0003740677400324022
iters 400: tol = 0.00035737941249477934
iters 450: tol = 0.000356884234221283
iters 500: tol = 0.0003723753756637582
iters 550: tol = 0.00040841790249473986
iters 600: tol = 0.000476675776301394
iters 650: tol = 0.0005884801683224972
iters 700: tol = 0.000776253643739816
iters 750: tol = 0.0011109572231094988
iters 800: tol = 0.001765516556543667
iters 850: tol = 0.003223859123724415
iters 900: tol = 0.007150268463884543
iters 950: tol = 0.03626737285557685
iters 1000: tol = 0.0052351213828 21228
iters 1050: tol = 0.005643464473513715
iters 1100: tol = 0.0031077176279209473
iters 1150: tol = 0.0014312624420993458
iters 1200: tol = 0.0006679764864423454
iters 1250: tol = 0.0003225831109989752
iters 1300: tol = 0.00016023788725577637
iters 50: tol = 0.00300375185953472
iters 100: tol = 0.0011246984768439328
iters 150: tol = 0.00044304681927342937
iters 200: tol = 0.00018274014081753887
iters 50: tol = 0.0023751042542300427
iters 100: tol = 0.000463470633691343
iters 50: tol = 0.001341592011578907
iters 100: tol = 0.00045557763262149553
iters 150: tol = 0.00014677669622031875
iters 50: tol = 0.0017852688244330839
iters 100: tol = 0.0005595541290243089
iters 150: tol = 0.00018383079922801304
iters 50: tol = 0.0018532952180413398
iters 100: tol = 0.0004698181496081699
iters 150: tol = 0.00015452750417610517
iters 50: tol = 0.0019279103189401752
iters 100: tol = 0.00044518187448928936
iters 150: tol = 0.00012556012805067795
2023-12-01 00:00:00 is done and took 160 iterations and 183.53 seconds
iters 50: tol = 0.0017256528753282555
iters 100: tol = 0.000636691727140315
iters 150: tol = 0.0002553103452029859
iters 200: tol = 0.00010904322546079204
iters 50: tol = 0.0020940892600460614
iters 100: tol = 0.0002778469776729686
iters 50: tol = 0.0016273641576503017
iters 100: tol = 0.0017363799420081483
iters 150: tol = 0.002013402359125449
iters 200: tol = 0.0028481643272205703
iters 250: tol = 0.004031416299784318
iters 300: tol = 0.0069177183273746445
iters 350: tol = 0.012420372348139294
iters 400: tol = 0.015941696420054385
iters 450: tol = 0.009133293714503748
iters 500: tol = 0.0029387870811419248
iters 550: tol = 0.001895823854918044
iters 600: tol = 0.0008808520075562765
iters 650: tol = 0.0003701990463713667
iters 700: tol = 0.00015253355907085542
iters 50: tol = 0.0003705934107262887
iters 50: tol = 0.0017053173040619818
iters 100: tol = 0.0005067050616569535
iters 150: tol = 0.00018004164561060895
iters 50: tol = 0.007810794243863661
iters 100: tol = 0.002826532778881452
iters 150: tol = 0.0010918178313155114
iters 200: tol = 0.00044737846595377384
iters 250: tol = 0.0001911272305885614
iters 50: tol = 0.007969657105043326
iters 100: tol = 0.0019069659162163077
iters 150: tol = 0.0011567630043436417
iters 200: tol = 0.000757882107196628
iters 250: tol = 0.0004850855884462879
iters 300: tol = 0.0003065068156097306
iters 350: tol = 0.00019313467596265843
iters 400: tol = 0.00012159042972598177

==================================================
EVALUATING IPCA FACTORS FOR EARNINGS PREDICTION
==================================================
```

```
Factor loadings shape: (149, 8)
Top characteristics for each factor:

Factor 0:
  Constant: 0.384
  op_at: 0.380
  op_atl1: 0.338
  gp_atl1: 0.329
  gp_at: 0.266

Factor 1:
  dolvol_126d: 0.371
  at_turnover: 0.319
  ami_126d: 0.257
  sale_me: 0.230
  eqnpo_12m: 0.219

Factor 2:
  cop_at: 0.303
  qmj_safety: 0.293
  qmj_prof: 0.270
  cop_atl1: 0.265
  gp_atl1: 0.240

Factor 3:
  turnover_126d: 0.499
  zero_trades_126d: 0.312
  Constant: 0.282
  market_equity: 0.258
  gp_at: 0.180

Factor 4:
  ami_126d: 0.353
  turnover_126d: 0.242
  gp_atl1: 0.234
  at_gr1: 0.207
  zero_trades_126d: 0.203

Factor 5:
  mispricing_mgmt: 0.266
  qmj: 0.244
  eqnpo_me: 0.217
  noa_at: 0.216
  debt_me: 0.213

Factor 6:
  Constant: 0.355
  qmj: 0.255
  op_atl1: 0.224
  o_score: 0.221
  turnover_126d: 0.200

Factor 7:
  mispricing_perf: 0.234
  Constant: 0.221
  inv_gr1: 0.208
  nncoa_gr1a: 0.206
  cop_at: 0.201

Factor scores dataset: 127199 observations
Train set: 79917 observations
Test set: 47282 observations

PREDICTION RESULTS:
Accuracy: 0.576
Precision: 0.653
Recall: 0.591
F1-Score: 0.620
AUC-ROC: 0.600

FACTOR IMPORTANCE (XGBoost):
0: 0.286
1: 0.102
2: 0.112
3: 0.106
4: 0.099
5: 0.096
6: 0.099
7: 0.100

==================================================
BACKTESTING IPCA EARNINGS SURPRISE STRATEGY
==================================================

PERFORMANCE METRICS:
                       Annual Return (%) Annualized Volatility (%)  \
Benchmark                           1.32                      0.45
IPCA_Earnings_Strategy             15.54                     28.98

                       Annualized Sharpe Ratio  \
Benchmark                                2.912
IPCA_Earnings_Strategy                   0.536

                       Annualized Alpha vs Benchmark (%) Beta vs Benchmark  \
```

```
Benchmark                                         0.00               nan
IPCA_Earnings_Strategy                           17.04               nan

                      Max Drawdown (%) Max Monthly Loss (%)  \
Benchmark                         0.00                 0.00
IPCA_Earnings_Strategy          -33.50               -21.35

                      Annualized Information Ratio vs Benchmark  \
Benchmark                                                   nan
IPCA_Earnings_Strategy                                    0.587

                      Annualized Tracking Error vs Benchmark (%)
Benchmark                                                   0.00
IPCA_Earnings_Strategy                                     29.02
```