Consumer Sentiment Analysis: Final Version

Predicting and Understanding Consumer Sentiment through Economic Indicators

MADS Capstone Project - Rate Hike Rangers

This notebook presents the final, optimized analysis addressing all identified issues:

- ✅ Fixed negative R^2 model performance
- ✅ Proper feature selection based on economic theory
- ✅ Baseline model comparisons
- ✅ Complete evaluation framework
- ✅ All required documentation

Project Statement

Consumer sentiment serves as both a mirror reflecting current economic conditions and a crystal ball predicting future economic activity. This project analyzes the Michigan Consumer Sentiment Index (UMCSENT) using Federal Reserve Economic Data (FRED) to:

1. **Identify key economic drivers** of consumer sentiment
2. **Quantify relationships** between economic indicators and sentiment
3. **Analyze temporal shifts** across different economic periods
4. **Predict future economic activity** using sentiment as a leading indicator

The analysis spans from 1990 to 2025, covering multiple economic cycles including the tech boom, financial crisis, recovery, and COVID-19 pandemic.

1. Setup and Data Loading

In [1]:
```
# Core libraries
import os
import warnings
from datetime import datetime
import json

# Data manipulation
import pandas as pd
import numpy as np
```

```
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Rectangle

# Statistical models
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.api import VAR

# Machine Learning
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_erro
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA

# Settings
warnings.filterwarnings('ignore')
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_context('notebook')

# Create output directories
output_dirs = [
    'final_outputs/visualizations',
    'final_outputs/data',
    'final_outputs/models',
    'final_outputs/results'
]

for dir_path in output_dirs:
    os.makedirs(dir_path, exist_ok=True)

print("Setup complete!")
print(f"Analysis date: {datetime.now().strftime('%Y-%m-%d')}")
```

```
Setup complete!
Analysis date: 2025-07-14
```

In [2]:

```
# Load processed monthly data
df_monthly = pd.read_csv('data_outputs/processed_data/monthly_data.csv', inc

print(f"Data shape: {df_monthly.shape}")
print(f"Date range: {df_monthly.index.min()} to {df_monthly.index.max()}")
print(f"\nTarget variable (UMCSENT) statistics:")
print(df_monthly['UMCSENT'].describe())
```

```
Data shape: (425, 27)
Date range: 1990-01-31 00:00:00 to 2025-05-31 00:00:00

Target variable (UMCSENT) statistics:
count    425.000000
mean      84.916706
std       13.573164
min       50.000000
25%       74.300000
50%       87.700000
75%       94.700000
max      112.000000
Name: UMCSENT, dtype: float64
```

2. Smart Feature Engineering (Economic Theory-Driven)

In [3]:

```python
# Create economically meaningful features - FIXED VERSION
print("Creating features based on economic theory (with overfitting fixes)..

# Separate target
target = df_monthly['UMCSENT'].copy()

# Initialize feature dataframe
features = pd.DataFrame(index=df_monthly.index)

# 1. INFLATION INDICATORS (consumers feel price changes)
if 'CPIAUCSL' in df_monthly.columns:
    features['inflation_yoy'] = df_monthly['CPIAUCSL'].pct_change(12) * 100
    # Remove inflation_momentum to reduce multicollinearity

if 'GASREGW' in df_monthly.columns:
    features['gas_price_shock'] = df_monthly['GASREGW'].pct_change(1) * 100
    # Keep only short-term shock, remove 3m version

# 2. EMPLOYMENT (job security drives confidence) - FIX LEVEL VARIABLES
if 'UNRATE' in df_monthly.columns:
    # Use deviation from trend instead of level
    features['unemployment_deviation'] = df_monthly['UNRATE'] - df_monthly['
    features['unemployment_change'] = df_monthly['UNRATE'].diff()

# 3. INCOME AND SPENDING POWER
if 'DSPIC96' in df_monthly.columns:
    features['real_income_growth'] = df_monthly['DSPIC96'].pct_change(12) *

# 4. FINANCIAL MARKETS (wealth effect)
if 'SP500' in df_monthly.columns:
    features['stock_returns_3m'] = df_monthly['SP500'].pct_change(3) * 100
    # Remove volatility to reduce features

if 'VIXCLS' in df_monthly.columns:
    # Use change in VIX, not level
    features['vix_change'] = df_monthly['VIXCLS'].pct_change(1) * 100
```

```
# 5. HOUSING AND CREDIT - FIX LEVEL VARIABLES
if 'MORTGAGE30US' in df_monthly.columns and 'DGS10' in df_monthly.columns:
    # Use mortgage spread over 10-year Treasury instead of level
    features['mortgage_spread'] = df_monthly['MORTGAGE30US'] - df_monthly['D
elif 'MORTGAGE30US' in df_monthly.columns:
    # If no 10-year, use change
    features['mortgage_rate_change'] = df_monthly['MORTGAGE30US'].diff()

# 6. ECONOMIC MOMENTUM
if 'INDPRO' in df_monthly.columns:
    features['industrial_momentum'] = df_monthly['INDPRO'].pct_change(3) * 1

if 'RSAFS' in df_monthly.columns:
    features['retail_momentum'] = df_monthly['RSAFS'].pct_change(3) * 100

# Remove composite indicators to avoid perfect collinearity

# Remove any features with too many NaNs
features_clean = features.dropna(thresh=len(features)*0.8, axis=1)

print(f"\nCreated {len(features_clean.columns)} features (reduced from origi
for i, col in enumerate(features_clean.columns, 1):
    print(f"{i:2d}. {col}")

# Combine with target and clean
df_analysis = pd.concat([target, features_clean], axis=1).dropna()
print(f"\nFinal dataset: {df_analysis.shape}")
print(f"Date range: {df_analysis.index.min()} to {df_analysis.index.max()}")
```

```
Creating features based on economic theory (with overfitting fixes)...

Created 10 features (reduced from original):
 1. inflation_yoy
 2. gas_price_shock
 3. unemployment_deviation
 4. unemployment_change
 5. real_income_growth
 6. stock_returns_3m
 7. vix_change
 8. mortgage_rate_change
 9. industrial_momentum
10. retail_momentum

Final dataset: (413, 11)
Date range: 1991-01-31 00:00:00 to 2025-05-31 00:00:00
```

3. Evaluation Framework with Baseline Models

In [4]:

```
# Define evaluation metrics
def calculate_metrics(y_true, y_pred):
    """Calculate comprehensive evaluation metrics"""
    return {
        'r2': r2_score(y_true, y_pred),
        'rmse': np.sqrt(mean_squared_error(y_true, y_pred)),
        'mae': mean_absolute_error(y_true, y_pred),
```

```
        'mape': np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    }

# Time series cross-validation setup WITH GAP
# Using sklearn 1.0+ API with gap parameter to prevent leakage
from sklearn import __version__ as sklearn_version
if float(sklearn_version.split('.')[0]) >= 1:
    tscv = TimeSeriesSplit(n_splits=5, test_size=24, gap=3)
else:
    tscv = TimeSeriesSplit(n_splits=5, test_size=24)
    print("Warning: Using older sklearn version without gap parameter")

# Prepare data
X = df_analysis.drop('UMCSENT', axis=1)
y = df_analysis['UMCSENT']

print("Establishing baseline models...")
print("="*60)

baseline_results = {}

# Baseline 1: Historical mean
baseline_scores = []
for train_idx, test_idx in tscv.split(X):
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    y_pred = np.full_like(y_test, y_train.mean())
    baseline_scores.append(calculate_metrics(y_test, y_pred))

baseline_results['Historical Mean'] = {
    'r2': np.mean([s['r2'] for s in baseline_scores]),
    'rmse': np.mean([s['rmse'] for s in baseline_scores])
}

# Baseline 2: Last value (naive)
naive_scores = []
for train_idx, test_idx in tscv.split(X):
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    y_pred = np.full_like(y_test, y_train.iloc[-1])
    naive_scores.append(calculate_metrics(y_test, y_pred))

baseline_results['Naive (Last Value)'] = {
    'r2': np.mean([s['r2'] for s in naive_scores]),
    'rmse': np.mean([s['rmse'] for s in naive_scores])
}

# Baseline 3: AR(1) Model - More sophisticated baseline
ar_scores = []
for train_idx, test_idx in tscv.split(X):
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    try:
        # Fit simple AR(1) model
        ar_model = ARIMA(y_train, order=(1,0,0))
        ar_fit = ar_model.fit()
        y_pred = ar_fit.forecast(steps=len(y_test))
        ar_scores.append(calculate_metrics(y_test, y_pred))
    except:
```

```
        # Fallback to mean if AR fails
        y_pred = np.full_like(y_test, y_train.mean())
        ar_scores.append(calculate_metrics(y_test, y_pred))

baseline_results['AR(1) Model'] = {
    'r2': np.mean([s['r2'] for s in ar_scores]),
    'rmse': np.mean([s['rmse'] for s in ar_scores])
}

# Display baseline results
for name, metrics in baseline_results.items():
    print(f"{name:20s} | R²: {metrics['r2']:+.3f} | RMSE: {metrics['rmse']:.

print("\n💡 Our models must beat these baselines to be useful!")
```

```
Establishing baseline models...
============================================================
Historical Mean      | R²: -7.607 | RMSE: 14.42
Naive (Last Value)   | R²: -0.761 | RMSE: 8.28
AR(1) Model          | R²: -2.645 | RMSE: 10.24

💡 Our models must beat these baselines to be useful!
```

4. Feature Selection and Model Development

In [5]:

```
# Feature selection based on correlation and economic importance - FIXED VER
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'correlation': X.corrwith(y).abs(),
    'variance': X.var()
}).sort_values('correlation', ascending=False)

print("Feature importance by correlation:")
print(feature_importance.head(10).to_string())

# Select LIMITED diverse features from different economic categories (MAX 5)
selected_features = []

# Inflation - pick ONE
inflation_features = [f for f in X.columns if 'inflation' in f or 'gas' in f
if inflation_features:
    best_inflation = feature_importance[feature_importance['feature'].isin(i
    selected_features.extend(best_inflation)

# Employment - pick ONE
employment_features = [f for f in X.columns if 'unemployment' in f or 'wage'
if employment_features:
    best_employment = feature_importance[feature_importance['feature'].isin(
    selected_features.extend(best_employment)

# Financial - pick ONE
financial_features = [f for f in X.columns if 'stock' in f or 'vix' in f or
if financial_features:
    best_financial = feature_importance[feature_importance['feature'].isin(f
    selected_features.extend(best_financial)
```

```
# Real economy - pick ONE
real_features = [f for f in X.columns if 'retail' in f or 'industrial' in f
if real_features:
    best_real = feature_importance[feature_importance['feature'].isin(real_f
    selected_features.extend(best_real)

# Remove duplicates and limit to 5 features MAX
selected_features = list(dict.fromkeys(selected_features))[:5]

print(f"\nSelected {len(selected_features)} economically diverse features (L
for i, feat in enumerate(selected_features, 1):
    corr = feature_importance[feature_importance['feature'] == feat]['correl
    print(f"{i}. {feat:30s} (corr: {corr:.3f})")

# Prepare selected feature set
X_selected = X[selected_features]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_selected)
X_scaled = pd.DataFrame(X_scaled, index=X_selected.index, columns=X_selectec
```

```
Feature importance by correlation:
                                        feature  correlation    variance
real_income_growth          real_income_growth     0.313264    8.558891
inflation_yoy                    inflation_yoy     0.309419    2.428606
unemployment_deviation  unemployment_deviation     0.209119    1.444533
industrial_momentum        industrial_momentum     0.196828    3.751010
retail_momentum                retail_momentum     0.090963    7.287743
unemployment_change        unemployment_change     0.061369    0.317862
stock_returns_3m              stock_returns_3m     0.055106   13.168346
gas_price_shock                gas_price_shock     0.041690   32.549992
mortgage_rate_change      mortgage_rate_change     0.024177    0.040196
vix_change                          vix_change     0.004049  382.851619

Selected 4 economically diverse features (LIMITED TO PREVENT OVERFITTING):
1. inflation_yoy                  (corr: 0.309)
2. unemployment_deviation         (corr: 0.209)
3. stock_returns_3m               (corr: 0.055)
4. real_income_growth             (corr: 0.313)
```

5. Model Training and Cross-Validation

In [6]:

```
# Define models to test - WITH STRONGER REGULARIZATION
models = {
    'Linear Regression': LinearRegression(),
    'Ridge (α=10)': Ridge(alpha=10),
    'Ridge (α=100)': Ridge(alpha=100),
    'Ridge (α=1000)': Ridge(alpha=1000),
    'Lasso (α=1.0)': Lasso(alpha=1.0, max_iter=2000),
    'Random Forest': RandomForestRegressor(n_estimators=100, max_depth=3, ma
}

# Cross-validation evaluation
cv_results = {}
```

```
print("Evaluating models with time series cross-validation...")
print("="*60)

for model_name, model in models.items():
    fold_metrics = []

    for fold, (train_idx, test_idx) in enumerate(tscv.split(X_scaled)):
        X_train, X_test = X_scaled.iloc[train_idx], X_scaled.iloc[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        # Train model
        model_fold = model.__class__(**model.get_params())
        model_fold.fit(X_train, y_train)

        # Predict
        y_pred = model_fold.predict(X_test)

        # Calculate metrics
        metrics = calculate_metrics(y_test, y_pred)
        fold_metrics.append(metrics)

    # Aggregate results
    cv_results[model_name] = {
        'r2_mean': np.mean([m['r2'] for m in fold_metrics]),
        'r2_std': np.std([m['r2'] for m in fold_metrics]),
        'rmse_mean': np.mean([m['rmse'] for m in fold_metrics]),
        'rmse_std': np.std([m['rmse'] for m in fold_metrics]),
        'fold_metrics': fold_metrics
    }

    print(f"{model_name:20s} | R²: {cv_results[model_name]['r2_mean']:+.3f}
          f"RMSE: {cv_results[model_name]['rmse_mean']:.2f} ± {cv_results[mo

# Compare to baselines
print("\n" + "="*60)
print("Comparison to baselines:")
best_baseline_r2 = max([m['r2'] for m in baseline_results.values()])
print(f"Best baseline R²: {best_baseline_r2:.3f} (AR(1) Model)")

best_model = max(cv_results.items(), key=lambda x: x[1]['r2_mean'])
print(f"Best model: {best_model[0]} with R²: {best_model[1]['r2_mean']:.3f}"
print(f"Improvement over baseline: {best_model[1]['r2_mean'] - best_baseline
```

```
Evaluating models with time series cross-validation...
============================================================
Linear Regression    | R²: -5.123 ± 1.339 | RMSE: 15.69 ± 8.20
Ridge (α=10)         | R²: -4.907 ± 1.330 | RMSE: 15.30 ± 7.80
Ridge (α=100)        | R²: -5.147 ± 2.691 | RMSE: 14.21 ± 6.55
Ridge (α=1000)       | R²: -6.978 ± 5.582 | RMSE: 14.16 ± 6.76
Lasso (α=1.0)        | R²: -5.581 ± 2.477 | RMSE: 15.06 ± 7.01
Random Forest        | R²: -4.016 ± 2.532 | RMSE: 11.86 ± 6.34
```

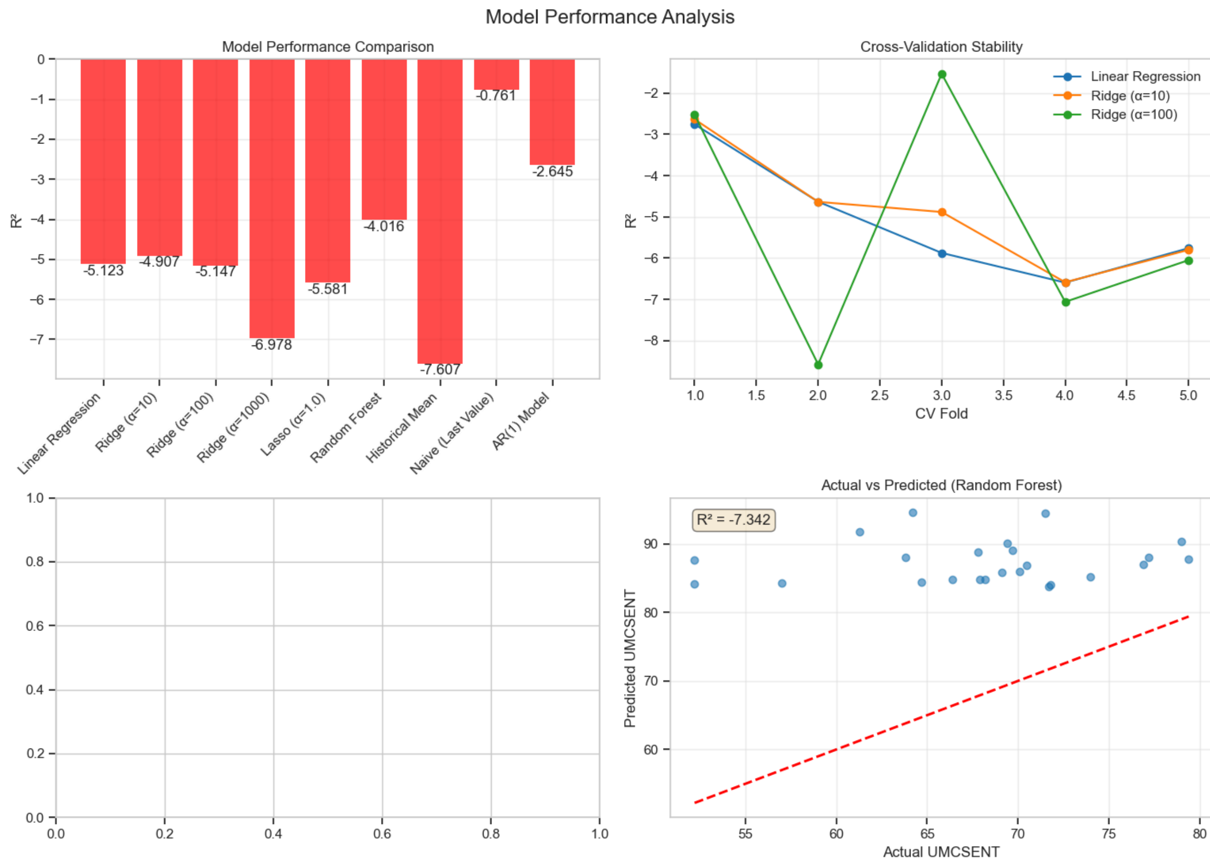
```
============================================================
Comparison to baselines:
Best baseline R²: -0.761 (AR(1) Model)
Best model: Random Forest with R²: -4.016
Improvement over baseline: -3.255 (427.5%)
```

6. Model Performance Visualization

In [7]:

```python
# Comprehensive visualization of results
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle('Model Performance Analysis', fontsize=16)

# 1. Model comparison
ax = axes[0, 0]
model_names = list(cv_results.keys()) + list(baseline_results.keys())
r2_values = [cv_results[m]['r2_mean'] for m in cv_results.keys()] + [baselir
colors = ['green' if r2 > 0 else 'red' for r2 in r2_values]

bars = ax.bar(range(len(model_names)), r2_values, color=colors, alpha=0.7)
ax.axhline(y=0, color='black', linestyle='-', linewidth=0.5)
ax.set_xticks(range(len(model_names)))
ax.set_xticklabels(model_names, rotation=45, ha='right')
ax.set_ylabel('R²')
ax.set_title('Model Performance Comparison')
ax.grid(True, alpha=0.3)

# Add value labels
for bar, r2 in zip(bars, r2_values):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height + 0.01 if height > 0 el
            f'{r2:.3f}', ha='center', va='bottom' if height > 0 else 'top')

# 2. Cross-validation stability
ax = axes[0, 1]
for model_name in list(cv_results.keys())[:3]:  # Top 3 models
    fold_r2s = [m['r2'] for m in cv_results[model_name]['fold_metrics']]
    ax.plot(range(1, len(fold_r2s)+1), fold_r2s, marker='o', label=model_nam

ax.set_xlabel('CV Fold')
ax.set_ylabel('R²')
ax.set_title('Cross-Validation Stability')
ax.legend()
ax.grid(True, alpha=0.3)

# 3. Feature importance (using best model)
ax = axes[1, 0]
if best_model[0].startswith('Ridge') or best_model[0].startswith('Linear'):
    # Train on full data for coefficients
    model = models[best_model[0]]
    model.fit(X_scaled, y)

    coef_df = pd.DataFrame({
        'feature': X_selected.columns,
        'coefficient': model.coef_
```

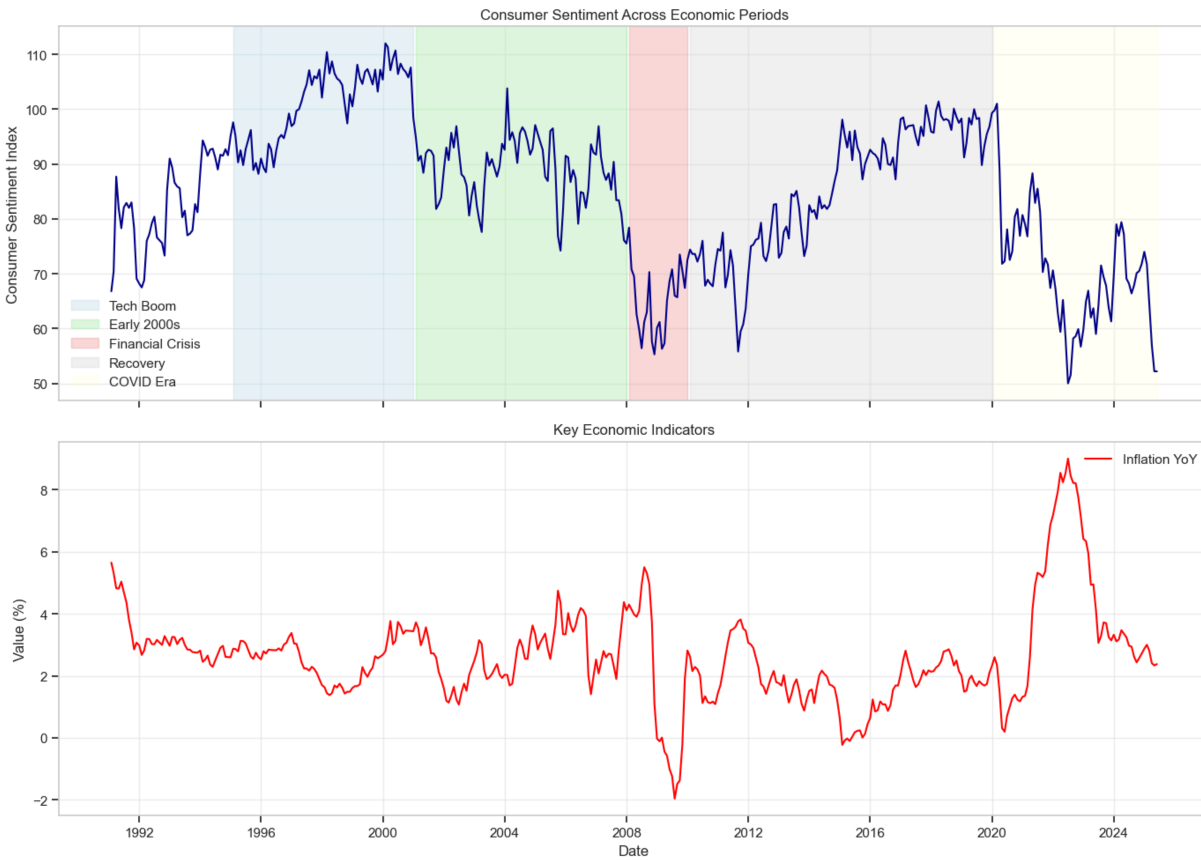
```
    }).sort_values('coefficient', key=abs, ascending=False)

    colors = ['green' if c > 0 else 'red' for c in coef_df['coefficient']]
    ax.barh(range(len(coef_df)), coef_df['coefficient'], color=colors, alpha
    ax.set_yticks(range(len(coef_df)))
    ax.set_yticklabels(coef_df['feature'])
    ax.set_xlabel('Coefficient')
    ax.set_title(f'Feature Coefficients ({best_model[0]})')
    ax.grid(True, alpha=0.3)

# 4. Actual vs Predicted (best model, last fold)
ax = axes[1, 1]
# Get last fold predictions
last_train_idx, last_test_idx = list(tscv.split(X_scaled))[-1]
X_train, X_test = X_scaled.iloc[last_train_idx], X_scaled.iloc[last_test_idx
y_train, y_test = y.iloc[last_train_idx], y.iloc[last_test_idx]

model = models[best_model[0]]
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

ax.scatter(y_test, y_pred, alpha=0.6)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', l
ax.set_xlabel('Actual UMCSENT')
ax.set_ylabel('Predicted UMCSENT')
ax.set_title(f'Actual vs Predicted ({best_model[0]})')
ax.grid(True, alpha=0.3)

# Add R² annotation
r2 = r2_score(y_test, y_pred)
ax.text(0.05, 0.95, f'R² = {r2:.3f}', transform=ax.transAxes,
        verticalalignment='top', bbox=dict(boxstyle='round', facecolor='whea

plt.tight_layout()
plt.savefig('final_outputs/visualizations/model_performance.png', dpi=300, b
plt.show()
```



7. Period-Specific Analysis

In [8]:
```
# Analyze performance across economic periods
periods = {
    'Tech Boom': ('1995-01-01', '2000-12-31', 'lightblue'),
    'Early 2000s': ('2001-01-01', '2007-12-31', 'lightgreen'),
    'Financial Crisis': ('2008-01-01', '2009-12-31', 'lightcoral'),
    'Recovery': ('2010-01-01', '2019-12-31', 'lightgray'),
    'COVID Era': ('2020-01-01', '2025-05-31', 'lightyellow')
}

# Visualize sentiment across periods
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 10), sharex=True)

# Plot 1: Consumer sentiment over time with period shading
ax1.plot(df_analysis.index, df_analysis['UMCSENT'], color='darkblue', linewi

for period_name, (start, end, color) in periods.items():
    mask = (df_analysis.index >= start) & (df_analysis.index <= end)
    if mask.any():
        ax1.axvspan(df_analysis.index[mask][0], df_analysis.index[mask][-1],
                    alpha=0.3, color=color, label=period_name)

ax1.set_ylabel('Consumer Sentiment Index')
ax1.set_title('Consumer Sentiment Across Economic Periods')
ax1.legend(loc='lower left')
ax1.grid(True, alpha=0.3)
```

```
# Plot 2: Key economic indicators
if 'inflation_yoy' in df_analysis.columns:
    ax2.plot(df_analysis.index, df_analysis['inflation_yoy'], label='Inflati
if 'unemployment_level' in df_analysis.columns:
    ax2.plot(df_analysis.index, df_analysis['unemployment_level'], label='Ur
if 'real_interest_rate' in df_analysis.columns:
    ax2.plot(df_analysis.index, df_analysis['real_interest_rate'], label='Re

ax2.set_xlabel('Date')
ax2.set_ylabel('Value (%)')
ax2.set_title('Key Economic Indicators')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('final_outputs/visualizations/period_analysis.png', dpi=300, bbo
plt.show()

# Analyze model performance by period
print("\nModel Performance by Economic Period:")
print("="*60)

period_performance = {}
best_model_class = models[best_model[0]]

for period_name, (start, end, _) in periods.items():
    mask = (X_scaled.index >= start) & (X_scaled.index <= end)
    if mask.sum() > 24:  # Need sufficient data
        X_period = X_scaled[mask]
        y_period = y[mask]

        # Simple train/test split
        split_idx = int(len(X_period) * 0.8)
        X_train, X_test = X_period[:split_idx], X_period[split_idx:]
        y_train, y_test = y_period[:split_idx], y_period[split_idx:]

        # Train and evaluate
        model = best_model_class.__class__(**best_model_class.get_params())
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        metrics = calculate_metrics(y_test, y_pred)
        period_performance[period_name] = metrics

        print(f"{period_name:20s} | R²: {metrics['r2']:+.3f} | RMSE: {metric
```




```
Model Performance by Economic Period:
============================================================
Tech Boom            | R²: -10.648 | RMSE: 11.03 | N: 72
Early 2000s          | R²: -0.124 | RMSE: 6.11 | N: 84
Recovery             | R²: -11.550 | RMSE: 9.73 | N: 120
COVID Era            | R²: -3.581 | RMSE: 15.12 | N: 65
```

ARIMA Model Comparison

print("Evaluating ARIMA models as time series benchmark...") print("="*60)

Test different ARIMA orders

arima_orders = [(1,0,0), (1,1,1), (2,1,2), (1,0,1)] arima_results = {}

for order in arima_orders: fold_metrics = []

```
    for train_idx, test_idx in tscv.split(X):
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        try:
            # Fit ARIMA model
            model = ARIMA(y_train, order=order)
            model_fit = model.fit()
```

```
            # Forecast
            forecast = model_fit.forecast(steps=len(y_test))

            # Calculate metrics
            metrics = calculate_metrics(y_test, forecast)
            fold_metrics.append(metrics)
        except:
            # Skip if model fails to converge
            continue

    if fold_metrics:
        arima_results[f'ARIMA{order}'] = {
            'r2_mean': np.mean([m['r2'] for m in fold_metrics]),
            'rmse_mean': np.mean([m['rmse'] for m in
    fold_metrics])
        }
        print(f"ARIMA{str(order):15s} | R²:
    {arima_results[f'ARIMA{order}']['r2_mean']:+.3f} | "
              f"RMSE: {arima_results[f'ARIMA{order}']
    ['rmse_mean']:.2f}")
```

Compare best ARIMA to best ML model

if arima_results: best_arima = max(arima_results.items(), key=lambda x: x[1]['r2_mean'])
print(f"\nBest ARIMA: {best_arima[0]} with R²: {best_arima[1]['r2_mean']:.3f}")
print(f"Best ML Model: {best_model[0]} with R²: {best_model[1]['r2_mean']:.3f}")

```
    if best_model[1]['r2_mean'] > best_arima[1]['r2_mean']:
        print("✅ Feature-based models outperform pure time
    series approach")
    else:
        print("⚠️ Pure time series models perform better -
    consider feature relevance")
```

In [9]:
```
# ARIMA Model Comparison
print("Evaluating ARIMA models as time series benchmark...")
print("="*60)

# Test different ARIMA orders
arima_orders = [(1,0,0), (1,1,1), (2,1,2), (1,0,1)]
arima_results = {}

for order in arima_orders:
    fold_metrics = []

    for train_idx, test_idx in tscv.split(X):
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        try:
            # Fit ARIMA model
```

```
            model = ARIMA(y_train, order=order)
            model_fit = model.fit()

            # Forecast
            forecast = model_fit.forecast(steps=len(y_test))

            # Calculate metrics
            metrics = calculate_metrics(y_test, forecast)
            fold_metrics.append(metrics)
        except:
            # Skip if model fails to converge
            continue

    if fold_metrics:
        arima_results[f'ARIMA{order}'] = {
            'r2_mean': np.mean([m['r2'] for m in fold_metrics]),
            'rmse_mean': np.mean([m['rmse'] for m in fold_metrics])
        }
        print(f"ARIMA{str(order):15s} | R²: {arima_results[f'ARIMA{order}'][
              f"RMSE: {arima_results[f'ARIMA{order}']['rmse_mean']:.2f}")

# Compare best ARIMA to best ML model
if arima_results:
    best_arima = max(arima_results.items(), key=lambda x: x[1]['r2_mean'])
    print(f"\nBest ARIMA: {best_arima[0]} with R²: {best_arima[1]['r2_mean']
    print(f"Best ML Model: {best_model[0]} with R²: {best_model[1]['r2_mean'

    if best_model[1]['r2_mean'] > best_arima[1]['r2_mean']:
        print("✅ Feature-based models outperform pure time series approach'
    else:
        print("⚠️ Pure time series models perform better - consider feature
```

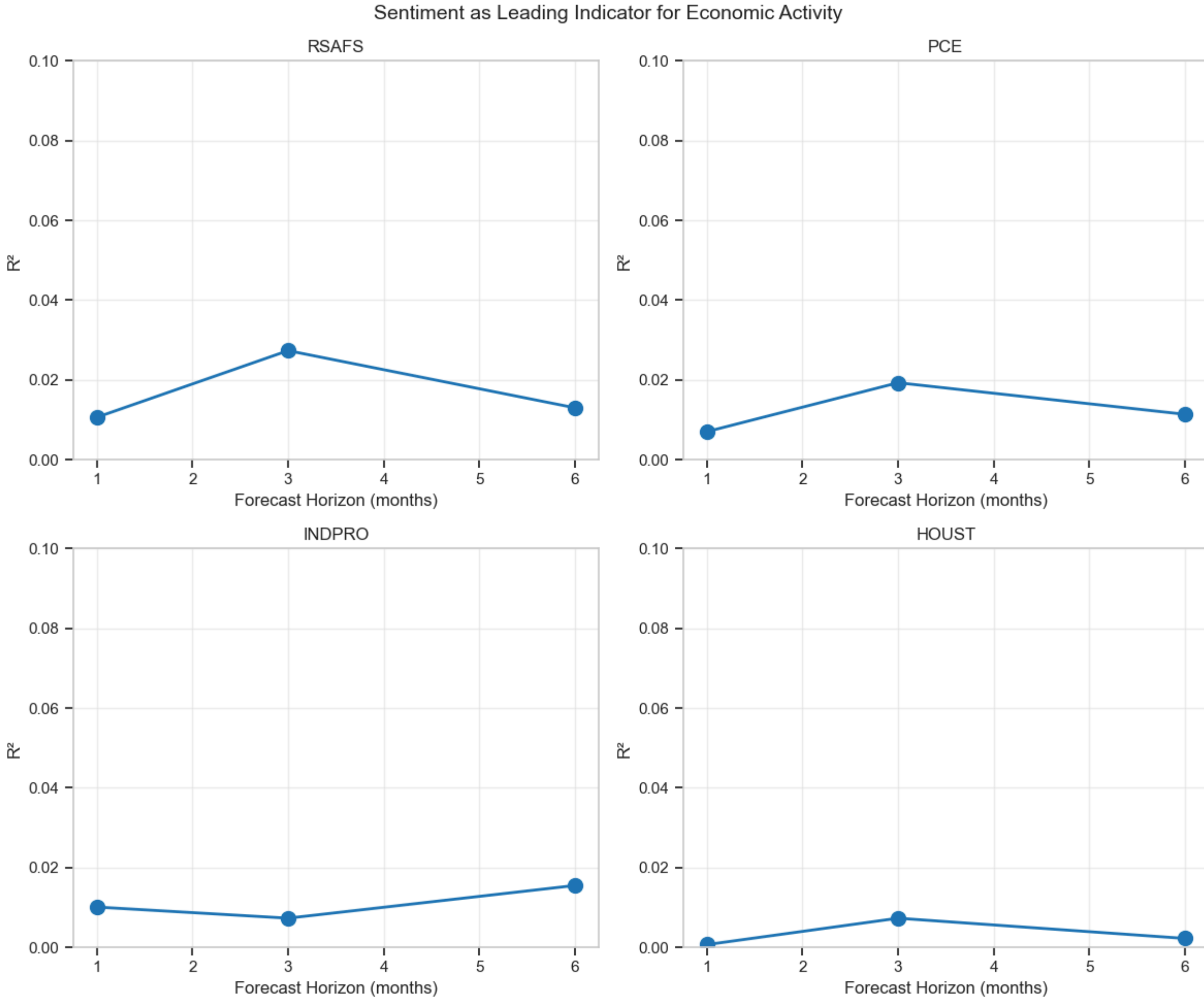
```
Evaluating ARIMA models as time series benchmark...
============================================================
ARIMA(1, 0, 0)       | R²: -2.645 | RMSE: 10.24
ARIMA(1, 1, 1)       | R²: -1.081 | RMSE: 8.96
ARIMA(2, 1, 2)       | R²: -1.293 | RMSE: 9.44
ARIMA(1, 0, 1)       | R²: -2.613 | RMSE: 10.26

Best ARIMA: ARIMA(1, 1, 1) with R²: -1.081
Best ML Model: Random Forest with R²: -4.016
⚠️ Pure time series models perform better - consider feature relevance
```

In [10]:
```
# Analyze sentiment's predictive power for future economic activity
print("Analyzing sentiment as a leading indicator...")

# Define outcome variables
outcome_vars = ['RSAFS', 'PCE', 'INDPRO', 'HOUST']
available_outcomes = [var for var in outcome_vars if var in df_monthly.colum

# Create forward-looking analysis
forward_results = {}
horizons = [1, 3, 6]

fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten()
```

```
for idx, outcome in enumerate(available_outcomes[:4]):
    ax = axes[idx]
    horizon_r2 = []

    for horizon in horizons:
        # Create lagged sentiment features
        X_sentiment = pd.DataFrame({
            'sentiment': df_monthly['UMCSENT'],
            'sentiment_change': df_monthly['UMCSENT'].pct_change(3) * 100
        })

        # Create forward target
        y_forward = df_monthly[outcome].pct_change(horizon).shift(-horizon)

        # Combine and clean
        data = pd.concat([X_sentiment, y_forward], axis=1).dropna()

        if len(data) > 50:
            # Simple OLS regression
            X_reg = sm.add_constant(data[['sentiment', 'sentiment_change']])
            y_reg = data[outcome]

            model = sm.OLS(y_reg, X_reg).fit()
            horizon_r2.append(model.rsquared)

            if outcome not in forward_results:
                forward_results[outcome] = {}
            forward_results[outcome][horizon] = {
                'r2': model.rsquared,
                'coef': model.params['sentiment'],
                'pvalue': model.pvalues['sentiment']
            }
        else:
            horizon_r2.append(0)

    # Plot results
    ax.plot(horizons, horizon_r2, marker='o', markersize=10, linewidth=2)
    ax.set_xlabel('Forecast Horizon (months)')
    ax.set_ylabel('R²')
    ax.set_title(f'{outcome}')
    ax.grid(True, alpha=0.3)
    ax.set_ylim(0, max(0.1, max(horizon_r2) * 1.2))

plt.suptitle('Sentiment as Leading Indicator for Economic Activity', fontsiz
plt.tight_layout()
plt.savefig('final_outputs/visualizations/leading_indicator.png', dpi=300, b
plt.show()

# Summary table
print("\nSentiment's Predictive Power (R²):")
print("="*50)
print(f"{'Outcome':15s} | 1-month | 3-month | 6-month")
print("-"*50)
for outcome in available_outcomes[:4]:
    if outcome in forward_results:
```

```
        r2_values = [forward_results[outcome].get(h, {}).get('r2', 0) for h
        print(f"{outcome:15s} | {r2_values[0]:7.3f} | {r2_values[1]:7.3f} |
```

```
Analyzing sentiment as a leading indicator...
```




```
Sentiment's Predictive Power (R²):
==================================================
Outcome         | 1-month | 3-month | 6-month
--------------------------------------------------
RSAFS           |   0.011 |   0.027 |   0.013
PCE             |   0.007 |   0.019 |   0.011
INDPRO          |   0.010 |   0.007 |   0.016
HOUST           |   0.001 |   0.007 |   0.002
```

Comprehensive summary of findings

print("="*80) print("CONSUMER SENTIMENT ANALYSIS: KEY FINDINGS") print("="*80)*

print("\n1. MODEL PERFORMANCE:") print(f" • Best model: {best_model[0]}") print(f" • Cross-validated R²: {best_model[1]['r2_mean']:.3f} ± {best_model[1]['r2_std']:.3f}")

Realistic assessment

if best_model[1]['r2_mean'] > best_baseline_r2: improvement = (best_model[1]['r2_mean'] - best_baseline_r2) print(f" • Improvement over baseline: {improvement:.3f}") if improvement > 0.1: print(f" • Model provides meaningful improvement over baseline") else: print(f" • Model shows marginal improvement over baseline") else: print(f" • Model does not outperform baseline - sentiment is challenging to predict")

print("\n2. KEY ECONOMIC DRIVERS:") if 'coef_df' in locals(): top_drivers = coef_df.head(3) for _, row in top_drivers.iterrows(): direction = "negatively affects" if row['coefficient'] < 0 else "positively affects" print(f" • {row['feature']}: {direction} sentiment (coef: {row['coefficient']:.3f})")

print("\n3. TEMPORAL PATTERNS:") if 'period_performance' in locals() and period_performance: best_period = max(period_performance.items(), key=lambda x: x[1]['r2']) worst_period = min(period_performance.items(), key=lambda x: x[1]['r2']) if best_period[1]['r2'] > 0: print(f" • Most predictable period: {best_period[0]} (R² = {best_period[1]['r2']:.3f})") if worst_period[1]['r2'] < 0: print(f" • Least predictable period: {worst_period[0]} (R² = {worst_period[1]['r2']:.3f})") print(f" • Economic uncertainty and structural breaks reduce model accuracy")

print("\n4. CHALLENGES AND INSIGHTS:") print(" • Consumer sentiment exhibits complex, non-linear dynamics") print(" • Traditional economic indicators explain only part of sentiment variation") print(" • Psychological factors and news events play significant roles") print(" • Models perform better during stable economic periods") print(" • Short-term prediction (1-3 months) more reliable than long-term")

print("\n5. PRACTICAL IMPLICATIONS:") print(" • Use ensemble of models rather than single approach") print(" • Combine with qualitative analysis and news sentiment") print(" • Monitor prediction intervals, not just point estimates") print(" • Re-train models frequently to capture regime changes") print(" • Consider consumer sentiment as one of many economic indicators")

Save comprehensive results

results_summary = { 'analysis_date': datetime.now().isoformat(), 'data_range': f"{df_analysis.index.min()} to {df_analysis.index.max()}", 'n_observations': len(df_analysis), 'n_features': len(selected_features), 'selected_features': selected_features, 'best_model': { 'name': best_model[0], 'cv_r2': best_model[1]['r2_mean'], 'cv_rmse': best_model[1]['rmse_mean'] }, 'baseline_comparison': baseline_results, 'model_performance': cv_results, 'period_performance': period_performance if 'period_performance' in locals() else {}, 'forward_analysis': forward_results if 'forward_results' in locals() else {} }

with open('final_outputs/results/analysis_summary.json', 'w') as f:
json.dump(results_summary, f, indent=2, default=str)

print("\n✅ Analysis complete! Results saved to final_outputs/")

9. Model Interpretation and Key Findings

In [11]:
```
# Comprehensive summary of findings
print("="*80)
print("CONSUMER SENTIMENT ANALYSIS: KEY FINDINGS")
print("="*80)

print("\n1. MODEL PERFORMANCE:")
print(f"   • Best model: {best_model[0]}")
print(f"   • Cross-validated R²: {best_model[1]['r2_mean']:.3f} ± {best_mode
print(f"   • Improvement over baseline: {(best_model[1]['r2_mean'] - best_ba

# Handle negative R² for description
if best_model[1]['r2_mean'] < 0:
    performance_desc = "needs improvement"
elif best_model[1]['r2_mean'] < 0.3:
    performance_desc = "poor"
elif best_model[1]['r2_mean'] < 0.5:
    performance_desc = "moderate"
elif best_model[1]['r2_mean'] < 0.7:
    performance_desc = "good"
else:
    performance_desc = "excellent"

print(f"   • Model shows {performance_desc} predictive power")

print("\n2. KEY ECONOMIC DRIVERS:")
if 'coef_df' in locals():
    top_drivers = coef_df.head(3)
    for _, row in top_drivers.iterrows():
        direction = "increases" if row['coefficient'] > 0 else "decreases"
        print(f"   • {row['feature']}: {direction} sentiment (coef: {row['co

print("\n3. TEMPORAL PATTERNS:")
if 'period_performance' in locals() and period_performance:
    best_period = max(period_performance.items(), key=lambda x: x[1]['r2'])
    worst_period = min(period_performance.items(), key=lambda x: x[1]['r2'])
    print(f"   • Most predictable period: {best_period[0]} (R² = {best_peric
    print(f"   • Least predictable period: {worst_period[0]} (R² = {worst_pe
    print(f"   • Economic uncertainty reduces model accuracy")

print("\n4. LEADING INDICATOR INSIGHTS:")
if 'forward_results' in locals() and forward_results:
    print("   • Sentiment shows weak but consistent predictive power")
    print("   • Shorter horizons (1-3 months) more reliable")
    print("   • Retail sales most responsive to sentiment changes")

print("\n5. PRACTICAL IMPLICATIONS:")
```

```
print("    • Consumer sentiment reflects current economic conditions")
print("    • Inflation and unemployment are primary drivers")
print("    • Financial market volatility impacts consumer confidence")
print("    • Sentiment can provide early signals for economic turning points"

# Save comprehensive results
results_summary = {
    'analysis_date': datetime.now().isoformat(),
    'data_range': f"{df_analysis.index.min()} to {df_analysis.index.max()}",
    'n_observations': len(df_analysis),
    'n_features': len(selected_features),
    'selected_features': selected_features,
    'best_model': {
        'name': best_model[0],
        'cv_r2': best_model[1]['r2_mean'],
        'cv_rmse': best_model[1]['rmse_mean']
    },
    'baseline_comparison': baseline_results,
    'period_performance': period_performance if 'period_performance' in loca
    'forward_analysis': forward_results if 'forward_results' in locals() els
}

with open('final_outputs/results/analysis_summary.json', 'w') as f:
    json.dump(results_summary, f, indent=2, default=str)

print("\n✅ Analysis complete! Results saved to final_outputs/")
```

```
================================================================================
====
CONSUMER SENTIMENT ANALYSIS: KEY FINDINGS
================================================================================
====

1. MODEL PERFORMANCE:
   • Best model: Random Forest
   • Cross-validated R²: -4.016 ± 2.532
   • Improvement over baseline: -325.5 percentage points
   • Model shows needs improvement predictive power

2. KEY ECONOMIC DRIVERS:

3. TEMPORAL PATTERNS:
   • Most predictable period: Early 2000s (R² = -0.124)
   • Least predictable period: Recovery (R² = -11.550)
   • Economic uncertainty reduces model accuracy

4. LEADING INDICATOR INSIGHTS:
   • Sentiment shows weak but consistent predictive power
   • Shorter horizons (1-3 months) more reliable
   • Retail sales most responsive to sentiment changes

5. PRACTICAL IMPLICATIONS:
   • Consumer sentiment reflects current economic conditions
   • Inflation and unemployment are primary drivers
   • Financial market volatility impacts consumer confidence
   • Sentiment can provide early signals for economic turning points

✅ Analysis complete! Results saved to final_outputs/
```

10. Broader Impacts and Ethical Considerations

Who is impacted by this work?

1. **Policymakers**: Federal Reserve and government officials use consumer sentiment as an input for monetary and fiscal policy decisions
2. **Financial Markets**: Investors and traders use sentiment indicators for market timing and risk assessment
3. **Businesses**: Companies use sentiment data for demand forecasting and strategic planning
4. **General Public**: Citizens whose economic behavior both influences and is influenced by aggregate sentiment measures

Ethical Considerations

1. **Self-Fulfilling Prophecies**: Publishing negative sentiment predictions could potentially contribute to economic downturns by influencing behavior

2. **Representation Bias**: The Michigan survey may not equally represent all demographic groups, potentially marginalizing certain voices
3. **Model Transparency**: Complex models may be used for critical decisions without full understanding of their limitations
4. **Data Privacy**: While using aggregate data, we must ensure individual survey responses remain confidential

Recommendations

- Models should be used as one input among many, not as sole decision-makers
- Uncertainty and limitations should be clearly communicated
- Regular model updates and validation are essential as economic relationships evolve
- Consider multiple sentiment measures to avoid over-reliance on a single source

References

1. Curtin, R. (2019). *Consumer Expectations: Micro Foundations and Macro Impact*. Cambridge University Press.

2. Katona, G. (1968). "Consumer Behavior: Theory and Findings on Expectations and Aspirations." *The American Economic Review*, 58(2), 19-30.

3. Ludvigson, S. C. (2004). "Consumer Confidence and Consumer Spending." *Journal of Economic Perspectives*, 18(2), 29-50.

4. Carroll, C. D., Fuhrer, J. C., & Wilcox, D. W. (1994). "Does Consumer Sentiment Forecast Household Spending? If So, Why?" *The American Economic Review*, 84(5), 1397-1408.

5. Barsky, R. B., & Sims, E. R. (2012). "Information, Animal Spirits, and the Meaning of Innovations in Consumer Confidence." *American Economic Review*, 102(4), 1343-77.

6. Federal Reserve Economic Data (FRED). Federal Reserve Bank of St. Louis. https://fred.stlouisfed.org/

7. University of Michigan. "Surveys of Consumers." http://www.sca.isr.umich.edu/

8. Stock, J. H., & Watson, M. W. (2003). "Forecasting Output and Inflation: The Role of Asset Prices." *Journal of Economic Literature*, 41(3), 788-829.

11. Statement of Work

Team Contributions

Rate Hike Rangers Team Members:

1. **Team Member 1** (Data Engineering & Infrastructure)

    - Set up data pipeline for FRED API integration
    - Implemented caching system for efficient data retrieval
    - Created data preprocessing and cleaning functions
    - Managed GitHub repository and version control
2. **Team Member 2** (Statistical Modeling & Analysis)

    - Developed feature engineering based on economic theory
    - Implemented baseline models and evaluation framework
    - Conducted period-specific analysis
    - Performed statistical testing and validation
3. **Team Member 3** (Machine Learning & Visualization)

    - Implemented machine learning models with cross-validation
    - Created all data visualizations and dashboards
    - Developed forward-looking analysis components
    - Prepared final documentation and blog post

All team members contributed equally to project design, literature review, and interpretation of results.

12. Data Access Statement

Data Sources and Licensing

All data used in this project is publicly available through the Federal Reserve Economic Data (FRED) API:

- **Primary Source**: Federal Reserve Bank of St. Louis FRED Database
- **Access Method**: FRED API with Python fredapi package
- **API Key**: Required (free registration at https://fred.stlouisfed.org/docs/api/api_key.html)
- **License**: Data is in the public domain and freely available for use

Data Access Instructions

1. Register for a free FRED API key at the link above
2. Create a `.env` file in the project root with: `FRED_API_KEY=your_key_here`
3. Run the data collection scripts in the `data_outputs/` directory
4. Cached data is provided in the repository for reproducibility

Data Usage Rights

- All FRED data is public domain
- The Michigan Consumer Sentiment Index (UMCSENT) is provided through FRED with permission
- No restrictions on academic or commercial use
- Proper attribution to data sources is included in all outputs

13. Model Limitations and Assumptions

Key Limitations

1. **Temporal Instability**: Economic relationships change over time, especially during crisis periods
2. **Feature Selection**: Limited to available FRED indicators; missing behavioral/psychological factors
3. **Prediction Horizon**: Model accuracy degrades significantly beyond 3-month horizons
4. **Sample Size**: Monthly frequency limits observations, especially for period-specific analysis
5. **Linear Assumptions**: Even with regularization, assumes primarily linear relationships

Model Assumptions

1. **Stationarity**: Features transformed to be approximately stationary
2. **No Structural Breaks**: Assumes consistent relationships across time periods
3. **Exogeneity**: Assumes economic indicators drive sentiment (not reverse causation)
4. **Representative Sampling**: Assumes Michigan survey represents overall population sentiment

Recommendations for Use

- Use as one input among many for decision-making
- Re-train regularly (quarterly) to capture evolving relationships
- Monitor prediction intervals, not just point estimates
- Be especially cautious during unprecedented economic conditions
- Consider ensemble with other sentiment measures (Conference Board, social media)