02_feature_analysis

November 30, 2024

[ ]:
```
import warnings
from pathlib import Path

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
from scipy import stats
from sklearn.decomposition import PCA
from sklearn.feature_selection import mutual_info_regression
from sklearn.preprocessing import StandardScaler

matplotlib.use('Agg')  # Use non-interactive backend for PDF export

# Set up the output directory for saving figures
notebook_dir = Path().absolute()
project_root = notebook_dir.parent if notebook_dir.name == 'notebooks' else␣
 ↪notebook_dir
figures_dir = project_root / 'figures'
analysis_dir = figures_dir / 'feature_analysis'
analysis_dir.mkdir(parents=True, exist_ok=True)

# Create directories
(figures_dir / 'exploration').mkdir(parents=True, exist_ok=True)
(figures_dir / 'feature_analysis').mkdir(parents=True, exist_ok=True)

warnings.filterwarnings('ignore')

# Set plotting styles
plt.style.use('bmh')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = [12, 6]
```

[ ]:
```
```

[ ]:
```python
# Load Processed Data from the Pipeline

# Get the current notebook directory and construct the correct path
notebook_dir = Path().absolute()
project_root = notebook_dir.parent if notebook_dir.name == 'notebooks' else␣
↪notebook_dir
processed_data_path = project_root / 'processed_data' / 'final_processed_data.
↪csv'

print(f"Looking for data file at: {processed_data_path}")
df = pd.read_csv(processed_data_path)

# Display basic information about the processed dataset
print("Dataset Overview:")
print("=" * 80)
print(f"\nShape: {df.shape}")
print("\nFeatures:")
for col in df.columns:
    dtype = df[col].dtype
    missing = df[col].isnull().sum()
    print(f"- {col}: {dtype} (Missing: {missing})")
```

[ ]:
```python
# Feature Distribution Analysis
def analyze_feature_distributions():
    """Analyze the distribution of engineered features"""

    # Select numerical columns
    numeric_cols = df.select_dtypes(include=[np.number]).columns

    # Create distribution plots
    for i in range(0, len(numeric_cols), 3):
        cols = numeric_cols[i:i + 3]
        fig, axes = plt.subplots(1, len(cols), figsize=(18, 6))
        if len(cols) == 1:
            axes = [axes]

        for ax, col in zip(axes, cols):
            sns.histplot(data=df, x=col, ax=ax)
            ax.set_title(f'Distribution of {col}')
            ax.tick_params(axis='x', rotation=45)

        plt.tight_layout()
        plt.savefig(analysis_dir / f'distribution_group_{i // 3}.png', dpi=300,␣
↪bbox_inches='tight')
        plt.show()

    # Test for normality
```

```python
    normality_tests = {}
    for col in numeric_cols:
        stat, p_value = stats.normaltest(df[col].dropna())
        normality_tests[col] = {'statistic': stat, 'p_value': p_value}

    return pd.DataFrame(normality_tests).T


# Run distribution analysis
distribution_results = analyze_feature_distributions()
print("\nNormality Test Results:")
display(distribution_results)
```

[ ]:
```python
# Correlation Analysis
def analyze_correlations():
    """Analyze correlations between features"""

    # Filter out non-numerical columns
    numerical_cols = df.select_dtypes(include=[np.number]).columns
    df_numerical = df[numerical_cols]

    # Calculate correlation matrix
    corr_matrix = df_numerical.corr()

    # Plot correlation heatmap
    plt.figure(figsize=(15, 12))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0, fmt='.2f')
    plt.title('Feature Correlation Matrix')
    plt.tight_layout()
    plt.savefig(analysis_dir / 'correlation_matrix.png', dpi=300, 
 ↪bbox_inches='tight')
    plt.show()

    # Identify highly correlated features
    high_corr = np.where(np.abs(corr_matrix) > 0.8)
    high_corr = [(corr_matrix.index[x], corr_matrix.columns[y], corr_matrix.
 ↪iloc[x, y])
                 for x, y in zip(*high_corr) if x != y]

    print("\nHighly Correlated Feature Pairs (|correlation| > 0.8):")
    for feat1, feat2, corr in high_corr:
        print(f"{feat1} - {feat2}: {corr:.3f}")


analyze_correlations()
```

[ ]:
```python
# Feature Importance Analysis
def analyze_feature_importance(target_col='renewable_generation'):  # Changed␣
↪from 'renewable_share'
    """Analyze feature importance using mutual information"""

    # First, verify target column exists
    if target_col not in df.columns:
        print(f"Warning: {target_col} not found. Available columns:")
        print(df.columns)
        return None

    # Prepare data
    X = df.select_dtypes(include=[np.number]).drop(columns=[target_col])
    y = df[target_col]

    # Handle NaN values
    data = pd.concat([X, y], axis=1)
    data = data.dropna()

    X = data.drop(columns=[target_col])
    y = data[target_col]

    # Calculate mutual information scores
    mi_scores = mutual_info_regression(X, y)

    # Create importance DataFrame
    importance_df = pd.DataFrame({
        'feature': X.columns,
        'importance': mi_scores
    }).sort_values('importance', ascending=False)

    # Create output directory if it doesn't exist
    output_dir = Path('figures/feature_analysis')
    output_dir.mkdir(parents=True, exist_ok=True)

    # Plot feature importance
    plt.figure(figsize=(12, 6))
    sns.barplot(data=importance_df, x='importance', y='feature')
    plt.title(f'Feature Importance for {target_col} (Mutual Information)')
    plt.xlabel('Mutual Information Score')
    plt.tight_layout()
    plt.savefig(analysis_dir / 'feature_importance.png', dpi=300,␣
↪bbox_inches='tight')
    plt.show()

    return importance_df
```

```
# Run feature importance analysis
print("Available columns in dataset:")
print(df.columns)
importance_results = analyze_feature_importance('renewable_generation')
print("\nFeature Importance Rankings:")
display(importance_results)
```

[ ]:
```
# Time Series Feature Analysis
def analyze_temporal_features():
    """Analyze temporal features and their relationships"""

    # Plot time series features
    temporal_features = [col for col in df.columns if 'lag' in col or 'rolling'␣
 ↪in col]

    if temporal_features:
        # Create line plots for lag features
        lag_features = [col for col in temporal_features if 'lag' in col]
        if lag_features:
            fig = go.Figure()
            for col in lag_features:
                fig.add_trace(go.Scatter(x=df.index, y=df[col], name=col))
            fig.update_layout(title='Lag Features Over Time')
            fig.write_image(str(analysis_dir / 'lag_features.png'))
            fig.show()

        # Create line plots for rolling features
        rolling_features = [col for col in temporal_features if 'rolling' in␣
 ↪col]
        if rolling_features:
            fig = go.Figure()
            for col in rolling_features:
                fig.add_trace(go.Scatter(x=df.index, y=df[col], name=col))
            fig.update_layout(title='Rolling Features Over Time')
            fig.write_image(str(analysis_dir / 'rolling_features.png'))
            fig.show()

    # Analyze autocorrelation
    if 'renewable_generation' in df.columns:
        plt.figure(figsize=(12, 6))
        pd.plotting.autocorrelation_plot(df['renewable_generation'])
        plt.title('Autocorrelation Plot of Renewable Generation')
        plt.savefig(analysis_dir / 'autocorrelation.png', dpi=300,␣
 ↪bbox_inches='tight')
        plt.show()
```

```
analyze_temporal_features()
```

[ ]:
```
# Geographic Feature Analysis
def analyze_geographic_features():
    """Analyze geographic features and regional patterns"""

    if 'country' in df.columns and 'renewable_share' in df.columns:
        # Calculate regional statistics
        regional_stats = df.groupby('country').agg({
            'renewable_share': ['mean', 'std', 'min', 'max'],
            'total_renewable': ['mean', 'std']
        }).round(3)

        # Plot regional patterns
        fig = px.choropleth(
            df,
            locations='country',
            color='renewable_generation',
            title='Geographic Distribution of Renewable Generation',
            color_continuous_scale='Viridis'
        )
        fig.write_image(str(analysis_dir / 'geographic_distribution.png'))
        fig.show()

        # Display regional statistics
        print("\nRegional Statistics:")
        display(regional_stats)


analyze_geographic_features()
```

[ ]:
```
# Principal Component Analysis
def perform_pca_analysis():
    """Perform PCA on numerical features"""

    # Prepare data
    numeric_cols = df.select_dtypes(include=[np.number]).columns
    X = df[numeric_cols]

    # Handle NaN values
    X = X.dropna(axis=0)

    # Scale the data
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

```python
    # Perform PCA
    pca = PCA()
    X_pca = pca.fit_transform(X_scaled)

    # Calculate explained variance ratio
    explained_variance = pca.explained_variance_ratio_
    cumulative_variance = np.cumsum(explained_variance)

    # Plot explained variance
    plt.figure(figsize=(12, 6))
    plt.plot(range(1, len(explained_variance) + 1), cumulative_variance, 'bo-')
    plt.axhline(y=0.95, color='r', linestyle='--')
    plt.xlabel('Number of Components')
    plt.ylabel('Cumulative Explained Variance Ratio')
    plt.title('PCA Explained Variance')
    plt.savefig(analysis_dir / 'pca_explained_variance.png', dpi=300,␣
 ↪bbox_inches='tight')
    plt.show()

    # Print component loadings
    components_df = pd.DataFrame(
        pca.components_.T,
        columns=[f'PC{i + 1}' for i in range(len(pca.components_))],
        index=numeric_cols
    )

    print("\nPrincipal Component Loadings:")
    display(components_df)

    return pca, components_df


pca_results = perform_pca_analysis()
```

[ ]:
```python
# Feature Interaction Analysis
def analyze_feature_interactions(df: pd.DataFrame):
    """Analyze interactions between important features"""
    # Use actual columns instead of relying on importance results
    feature_cols = [
        'Hydroelectric Power',
        'Solar Energy',
        'Wind Energy',
        'Geothermal Energy',
        'Biomass Energy'
    ]

    # Create scatter matrix
```

```
    fig = px.scatter_matrix(
        df[feature_cols],
        dimensions=feature_cols,
        title='Feature Interactions Matrix'
    )
    fig.write_image(str(analysis_dir / 'feature_interactions.png'))
    fig.show()

    # Calculate interaction terms
    for i in range(len(feature_cols) - 1):
        for j in range(i + 1, len(feature_cols) - 1):
            feat1, feat2 = feature_cols[i], feature_cols[j]
            interaction_name = f'{feat1}_{feat2}_interaction'
            df[interaction_name] = df[feat1] * df[feat2]

    # Create correlation matrix with interactions
    corr_matrix = df.corr()

    # Plot correlation heatmap
    plt.figure(figsize=(12, 10))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
    plt.title('Feature and Interaction Correlations')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig(analysis_dir / 'interaction_correlations.png', dpi=300,␣
↪bbox_inches='tight')
    plt.show()

    return corr_matrix


# Run the analysis
interaction_results = analyze_feature_interactions(df)
```

[ ]:
```
# Summary and Recommendations
def generate_feature_summary():
    """Generate summary of feature analysis and recommendations"""

    summary = """
    Feature Analysis Summary:

    1. Distribution Analysis:
    - Identified non-normal distributions in several features
    - Log transformation recommended for skewed features
    - Some features show clear outliers

    2. Correlation Analysis:
```

```
- Several highly correlated feature pairs identified
- Consider feature selection or dimensionality reduction
- Watch for multicollinearity in modeling

3. Feature Importance:
- Top features identified through mutual information
- Economic indicators show strong predictive power
- Weather features show moderate importance

4. Temporal Features:
- Lag features capture historical patterns
- Rolling features smooth out noise
- Strong autocorrelation present

5. Geographic Analysis:
- Clear regional patterns in renewable adoption
- Significant variation between countries
- Consider regional clustering

6. PCA Analysis:
- First few components explain majority of variance
- Consider dimensionality reduction
- Important feature combinations identified

Recommendations:
1. Feature Selection:
- Remove highly correlated features
- Focus on top important features
- Consider PCA for dimensionality reduction

2. Feature Engineering:
- Create interaction terms for top features
- Log transform skewed features
- Standardize numerical features

3. Modeling Considerations:
- Handle temporal autocorrelation
- Account for geographic patterns
- Consider hierarchical modeling

4. Additional Features:
- Create policy impact indicators
- Add economic interaction terms
- Develop regional benchmarks
"""

from IPython.display import display, HTML
```

```
    display(HTML(f"<pre>{summary}</pre>"))


generate_feature_summary()
```