Import Libraries

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import xgboost as xgb
```

Loading the data

In [2]:
```
train = pd.read_csv('train.csv')
```

In [3]:
```
test = pd.read_csv('test.csv')
```

In [4]:
```
features = pd.read_csv('features.csv')
```

In [5]:
```
stores = pd.read_csv('stores.csv')
```

In [6]:
```
train.head()
```

Out[6]:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

In [7]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   Store         421570 non-null  int64  
 1   Dept          421570 non-null  int64  
 2   Date          421570 non-null  object 
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     421570 non-null  bool   
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 13.3+ MB
```

In [8]:
```
train['Date'] = pd.to_datetime(train.Date)
```

In [9]:
```
train['Date'].tail()
```

Out[9]:
```
421565   2012-09-28
421566   2012-10-05
421567   2012-10-12
421568   2012-10-19
421569   2012-10-26
Name: Date, dtype: datetime64[ns]
```

In [10]:
```
test.head()
```

Out[10]:

	Store	Dept	Date	IsHoliday
0	1	1	2012-11-02	False
1	1	1	2012-11-09	False
2	1	1	2012-11-16	False
3	1	1	2012-11-23	True
4	1	1	2012-11-30	False

In [11]:
```
test['Weekly_Sales'] = np.nan
```

In [12]:
```
test.head()
```

Out[12]:

	Store	Dept	Date	IsHoliday	Weekly_Sales
0	1	1	2012-11-02	False	NaN
1	1	1	2012-11-09	False	NaN
2	1	1	2012-11-16	False	NaN
3	1	1	2012-11-23	True	NaN
4	1	1	2012-11-30	False	NaN

In [14]:

```
features.head()
```

Out[14]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

In [15]:

```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         8190 non-null   int64  
 1   Date          8190 non-null   object 
 2   Temperature   8190 non-null   float64
 3   Fuel_Price    8190 non-null   float64
 4   MarkDown1     4032 non-null   float64
 5   MarkDown2     2921 non-null   float64
 6   MarkDown3     3613 non-null   float64
 7   MarkDown4     3464 non-null   float64
 8   MarkDown5     4050 non-null   float64
 9   CPI           7605 non-null   float64
 10  Unemployment  7605 non-null   float64
 11  IsHoliday     8190 non-null   bool   
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 712.0+ KB
```

In [16]:
```
# Assuming 'feature' is your DataFrame
features[['MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5']] = features[['MarkDown1', 'MarkDown2', 'MarkDown3', 'M
```

In [18]:
```
features.CPI.head()
```

Out[18]:
```
0    211.096358
1    211.242170
2    211.289143
3    211.319643
4    211.350143
Name: CPI, dtype: float64
```

In [19]:
```
features['CPI'] = features['CPI'].interpolate()
```

In [20]:
```
features.Unemployment.head()
```

Out[20]:
```
0    8.106
1    8.106
2    8.106
3    8.106
4    8.106
Name: Unemployment, dtype: float64
```

In [21]:
```
features['Unemployment'] = features['Unemployment'].interpolate()
```

In [22]:
```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         8190 non-null   int64  
 1   Date          8190 non-null   object 
 2   Temperature   8190 non-null   float64
 3   Fuel_Price    8190 non-null   float64
 4   MarkDown1     8190 non-null   float64
 5   MarkDown2     8190 non-null   float64
 6   MarkDown3     8190 non-null   float64
 7   MarkDown4     8190 non-null   float64
 8   MarkDown5     8190 non-null   float64
 9   CPI           8190 non-null   float64
 10  Unemployment  8190 non-null   float64
 11  IsHoliday     8190 non-null   bool   
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 712.0+ KB
```

In [23]:
```
features['Date'] = pd.to_datetime(features.Date)
```

In [24]:
```
features.drop('IsHoliday',axis=1,inplace=True)
```

Train set

In [25]:
```
train_set = pd.merge(train,features, on=['Date','Store'],how='inner')
```

In [26]:
```
train_set.head()
```

Out[26]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI
0	1	1	2010-02-05	24924.50	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358
1	1	2	2010-02-05	50605.27	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358
2	1	3	2010-02-05	13740.12	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358
3	1	4	2010-02-05	39954.04	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358
4	1	5	2010-02-05	32229.38	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358

In [27]:

```
# Add the week of the year
train_set['Week_of_Year'] = train_set['Date'].dt.isocalendar().week
```

In [28]:

```
grouped_data= train_set.set_index('Date')
```

In [29]:

```
grouped_data.head()
```

Out[29]:

	Store	Dept	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Ur
Date													
2010-02-05	1	1	24924.50	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	
2010-02-05	1	2	50605.27	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	
2010-02-05	1	3	13740.12	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	
2010-02-05	1	4	39954.04	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	
2010-02-05	1	5	32229.38	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	

Model Training

Random Forest Regression

In [55]:

```
# Import necessary libraries
from sklearn.ensemble import RandomForestRegressor  # For regression tasks
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Assuming your train_set is a DataFrame with features and the target variable
# Replace 'target_column' with the actual name of your continuous target variable

X = grouped_data.drop('Weekly_Sales', axis=1)  # Features (all columns except the target)
y = grouped_data['Weekly_Sales']  # Continuous Target

# Splitting the data into training and validation sets (80% training, 20% validation)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Random Forest model for regression
```

```
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Predict on the validation set
y_pred = model.predict(X_val)

# Evaluate the model using Mean Squared Error
mse = mean_squared_error(y_val, y_pred)
r2_score = r2_score(y_val,y_pred)
mae = mean_absolute_error(y_val,y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R2: {r2_score}')
print(f'MAE: {mae}')
```

```
Mean Squared Error: 17504011.69075895
R2: 0.966705010189062
MAE: 1573.0075135434208
```

XgBoost

In [34]:

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Assuming 'train_set' is your dataset and 'target_column' is your continuous target variable
X = grouped_data.drop('Weekly_Sales', axis=1)  # Features
y = grouped_data['Weekly_Sales']  # Continuous Target variable (e.g., sales values)

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the XGBoost regressor model
xgb = xgb.XGBRegressor(n_estimators=150, learning_rate=0.9, random_state=42)

# Train the model
xgb.fit(X_train, y_train)

# Make predictions on the validation set
y_pred = xgb.predict(X_val)
```

```
# Evaluate the model using Mean Squared Error (or other regression metrics)
mse = mean_squared_error(y_val, y_pred)
mae = mean_absolute_error(y_val, y_pred)
r2 = r2_score(y_val, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R2: {r2}')
```

```
Mean Squared Error: 30834123.71287822
Mean Absolute Error: 3223.7791697690673
R2: 0.9413493401977402
```

Gradient Boosting Regression

In [37]:

```
from sklearn.ensemble import GradientBoostingRegressor

# Initialize Gradient Boosting Regressor model
gbr_model = GradientBoostingRegressor(n_estimators=150, learning_rate=0.1, max_depth=5, random_state=42)

# Train the model
gbr_model.fit(X_train, y_train)

# Predict on the validation set
y_pred_gbr = gbr_model.predict(X_val)

# Evaluate the model
mse_gbr = mean_squared_error(y_val, y_pred_gbr)
r2_gbr = r2_score(y_val, y_pred_gbr)
mae = mean_absolute_error(y_val, y_pred)

print(f"Gradient Boosting - Mean Squared Error: {mse_gbr}")
print(f"Gradient Boosting - R-squared: {r2_gbr}")
print(f'Mean Absolute Error: {mae}')
```

```
Gradient Boosting - Mean Squared Error: 89771913.04434471
Gradient Boosting - R-squared: 0.8292417199596677
Mean Absolute Error: 3223.7791697690673
```

CatBoost Regression

In [39]:
```
from catboost import CatBoostRegressor
# Define the CatBoost Regressor model
catboost_model = CatBoostRegressor(iterations=1000, learning_rate=0.1, depth=6, random_seed=42, verbose=100)

# Train the model
catboost_model.fit(X_train, y_train, eval_set=(X_val, y_val), use_best_model=True, verbose=100)

# Predict on the validation set
y_pred = catboost_model.predict(X_val)

# Evaluate the model performance
mse = mean_squared_error(y_val, y_pred)
r2 = r2_score(y_val, y_pred)
mae = mean_absolute_error(y_val,y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
print(f"MAE: {mae}")
```

```
0:	learn: 22047.0323482	test: 22323.3787665	best: 22323.3787665 (0)	total: 195ms	remaining: 3m 14s
100:	learn: 12195.7417060	test: 12549.4274558	best: 12549.4274558 (100)	total: 3.58s	remaining: 31.9s
200:	learn: 10167.2147604	test: 10507.7174548	best: 10507.7174548 (200)	total: 7.09s	remaining: 28.2s
300:	learn: 9054.6438339	test: 9389.0602462	best: 9389.0602462 (300)	total: 10.6s	remaining: 24.5s
400:	learn: 8426.3834143	test: 8758.7318893	best: 8758.7318893 (400)	total: 14.3s	remaining: 21.3s
500:	learn: 7940.8940850	test: 8278.2113479	best: 8278.2113479 (500)	total: 17.7s	remaining: 17.6s
600:	learn: 7587.2289524	test: 7937.6292392	best: 7937.6292392 (600)	total: 21.2s	remaining: 14.1s
700:	learn: 7297.3198499	test: 7655.4965247	best: 7655.4965247 (700)	total: 24.5s	remaining: 10.5s
800:	learn: 7062.1845232	test: 7428.3580842	best: 7428.3580842 (800)	total: 28.6s	remaining: 7.1s
900:	learn: 6858.4402948	test: 7235.7815428	best: 7235.7815428 (900)	total: 32.1s	remaining: 3.53s
999:	learn: 6688.4540967	test: 7068.3573609	best: 7068.3573609 (999)	total: 35.5s	remaining: 0us

bestTest = 7068.357361
bestIteration = 999

Mean Squared Error: 49961675.876714684
R-squared: 0.9049661575505681
MAE: 4179.714786611311
```

Test

In [57]:
```
test['Date'] = pd.to_datetime(test.Date)
```

In [58]:
```
test_set = pd.merge(test,features, on=['Date','Store'],how='inner')
```

In [59]:
```
test['Weekly_Sales'] = np.nan
```

In [60]:
```
test.head()
```

Out[60]:

	Store	Dept	Date	IsHoliday	Weekly_Sales
0	1	1	2012-11-02	False	NaN
1	1	1	2012-11-09	False	NaN
2	1	1	2012-11-16	False	NaN
3	1	1	2012-11-23	True	NaN
4	1	1	2012-11-30	False	NaN

In [61]:
```
test_set['Week_of_Year'] = train_set['Date'].dt.isocalendar().week
```

In [62]:
```
test_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115064 entries, 0 to 115063
Data columns (total 15 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Store         115064 non-null  int64
 1   Dept          115064 non-null  int64
 2   Date          115064 non-null  datetime64[ns]
 3   IsHoliday     115064 non-null  bool
 4   Weekly_Sales  0 non-null       float64
 5   Temperature   115064 non-null  float64
 6   Fuel_Price    115064 non-null  float64
 7   MarkDown1     115064 non-null  float64
 8   MarkDown2     115064 non-null  float64
 9   MarkDown3     115064 non-null  float64
 10  MarkDown4     115064 non-null  float64
 11  MarkDown5     115064 non-null  float64
 12  CPI           115064 non-null  float64
 13  Unemployment  115064 non-null  float64
 14  Week_of_Year  115064 non-null  UInt32
dtypes: UInt32(1), bool(1), datetime64[ns](1), float64(10), int64(2)
memory usage: 12.1 MB
```

In [63]:
```
test_set.drop('Weekly_Sales',axis=1,inplace=True)
```

In [64]:
```
test_set.set_index('Date', inplace= True)
```

In [65]:
```
test_set.head()
```

Out[65]:

Date	Store	Dept	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
2012-11-02	1	1	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	2	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	3	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	4	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	5	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573

In [66]:

```
test_set['Pred_Sales'] = model.predict(test_set)
```

In [67]:

```
test_set.head()
```

Out[67]:

Date	Store	Dept	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
2012-11-02	1	1	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	2	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	3	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	4	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573
2012-11-02	1	5	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	6.573

In [68]:
```
columns = ['Pred_Sales'] + [col for col in test_set.columns if col != 'Pred_Sales']
test_set = test_set[columns]
```

In [69]:
```
test_set.head()
```

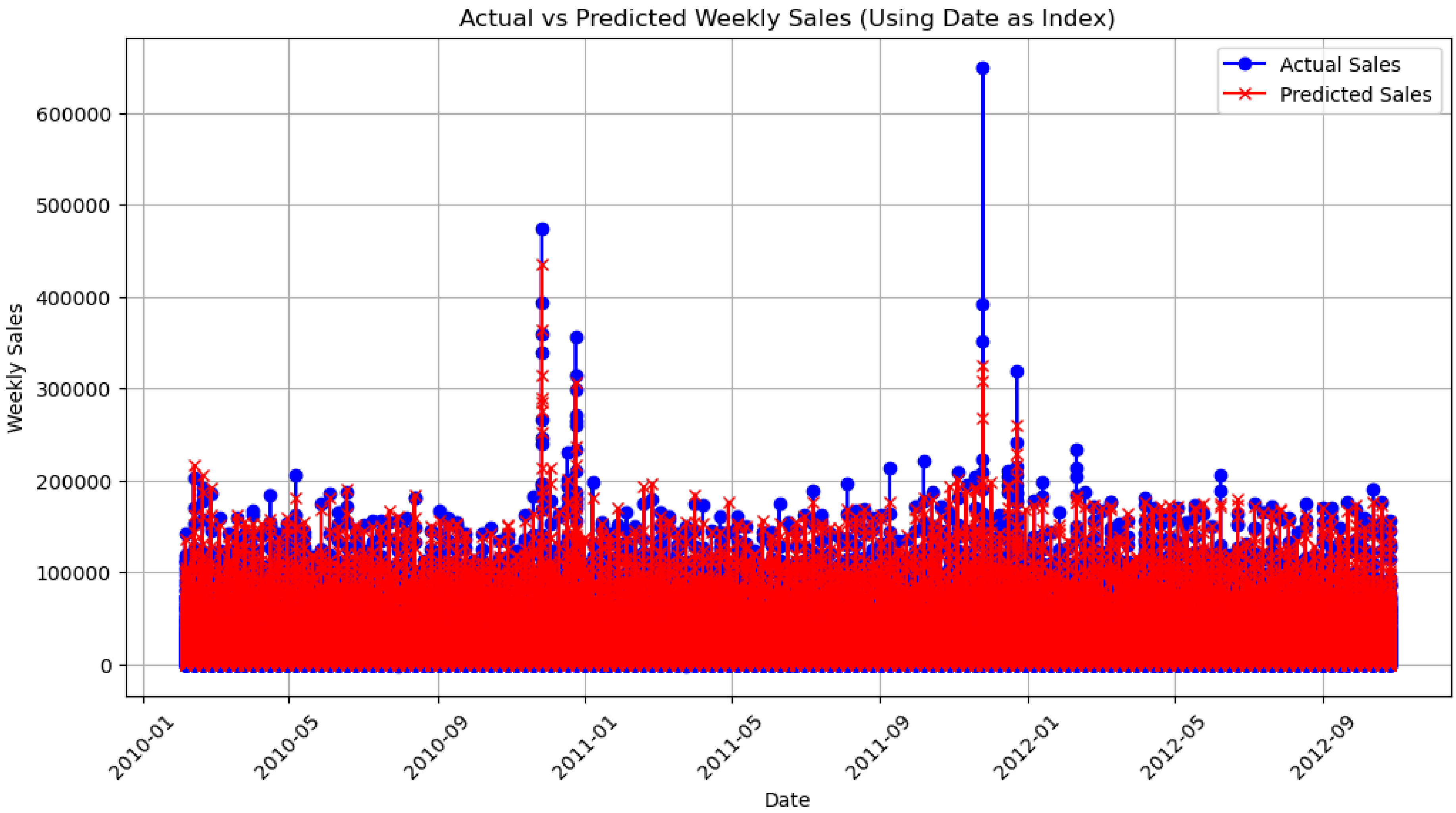
Out[69]:

	Pred_Sales	Store	Dept	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Uner
Date													
2012-11-02	26928.2836	1	1	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	
2012-11-02	50131.9323	1	2	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	
2012-11-02	11056.9892	1	3	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	
2012-11-02	39948.3094	1	4	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	
2012-11-02	23063.3633	1	5	False	55.32	3.386	6766.44	5147.7	50.82	3639.9	2737.42	223.462779	

In [71]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Assuming you already have `y_val` (actual sales) and `y_pred` (predicted sales from your model)
# And that the `Date` column is set as the index.

# Create a DataFrame to hold the actual sales and predicted sales for plotting
result_df = pd.DataFrame({
    'Actual_Sales': y_val,
    'Predicted_Sales': y_pred
}, index=X_val.index)  # Use the index (which is Date)

# Sort the result DataFrame by Date to visualize trends chronologically
result_df = result_df.sort_index()
```

Actual vs Predicted Sales over time

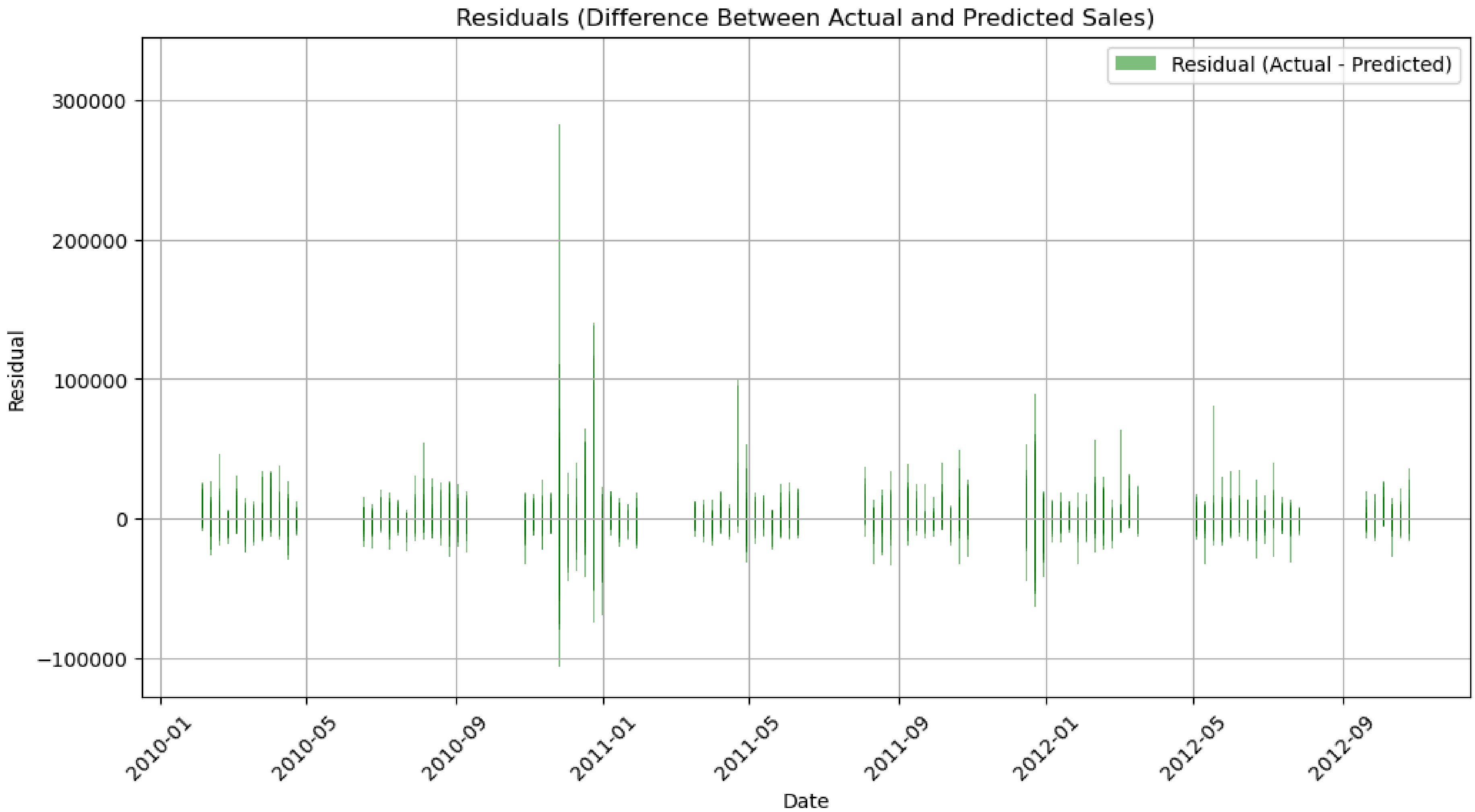
In [72]:
```
# Plot actual vs predicted sales over time
plt.figure(figsize=(12, 6))

# Plot actual sales
plt.plot(result_df.index, result_df['Actual_Sales'], label='Actual Sales', marker='o', color='blue')

# Plot predicted sales
plt.plot(result_df.index, result_df['Predicted_Sales'], label='Predicted Sales', marker='x', color='red')

# Add labels and title
plt.xlabel('Date')
plt.ylabel('Weekly Sales')
plt.title('Actual vs Predicted Weekly Sales (Using Date as Index)')
plt.legend()

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

# Add a grid for easier interpretation
plt.grid(True)

# Show the plot
plt.show()
```



In [73]:

```
# Calculate residuals (Actual - Predicted)
result_df['Residual'] = result_df['Actual_Sales'] - result_df['Predicted_Sales']

# Plot the residuals as bars
plt.figure(figsize=(12, 6))

# Plot residuals as bars
plt.bar(result_df.index, result_df['Residual'], label='Residual (Actual - Predicted)', color='green', alpha=0.5)

# Add labels and title
```

```
plt.xlabel('Date')
plt.ylabel('Residual')
plt.title('Residuals (Difference Between Actual and Predicted Sales)')
plt.legend()

# Rotate the x-axis labels for readability
plt.xticks(rotation=45)

# Show the grid and plot
plt.grid(True)
plt.show()
```

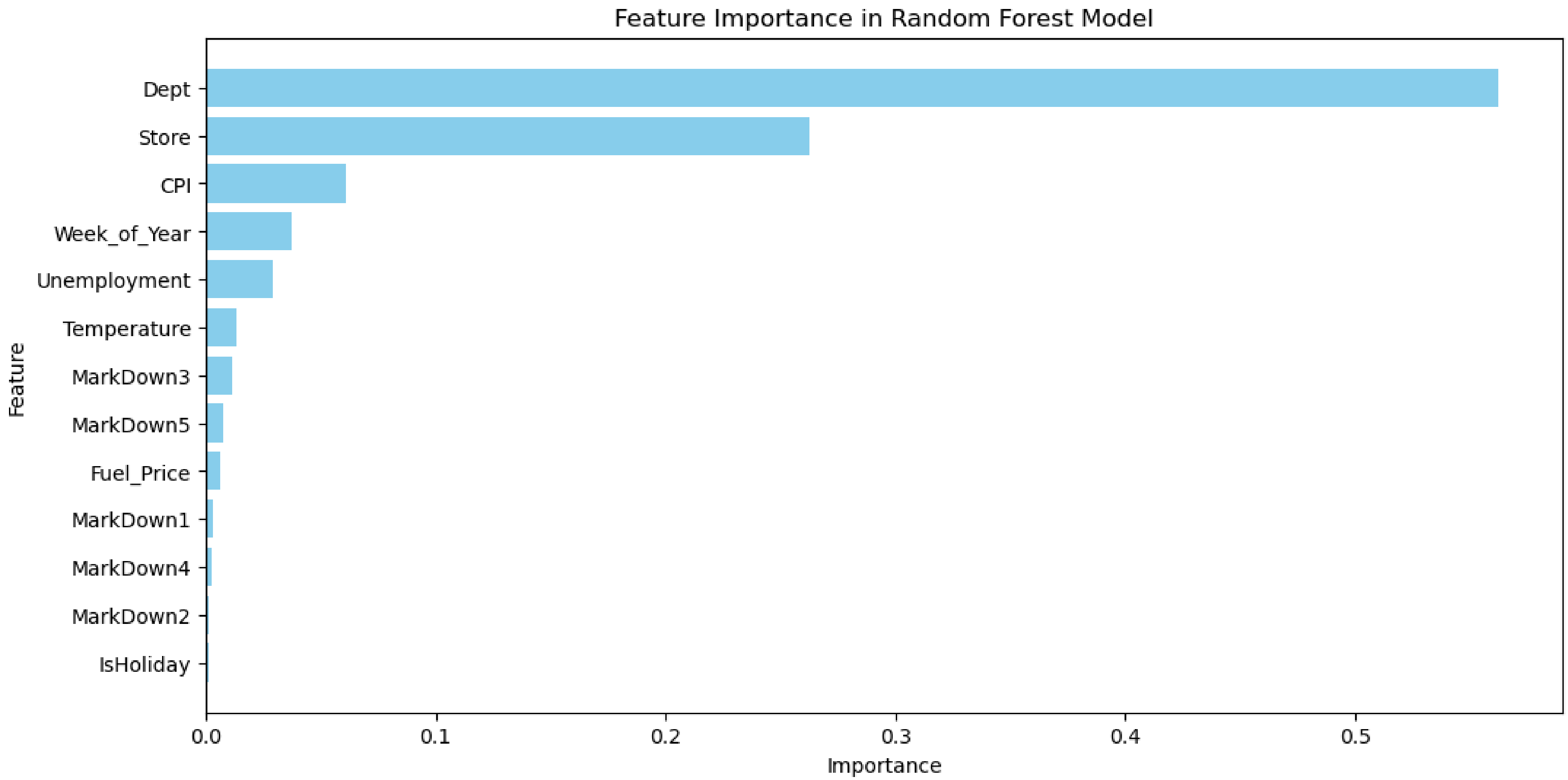


Feature Importance

In [75]:
```
# Get feature importance from the model
importances = model.feature_importances_

# Create a DataFrame for better visualization
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,  # Use the feature names
    'Importance': importances
})

# Sort the DataFrame by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Display the top features
print(feature_importance_df.head(10))
```

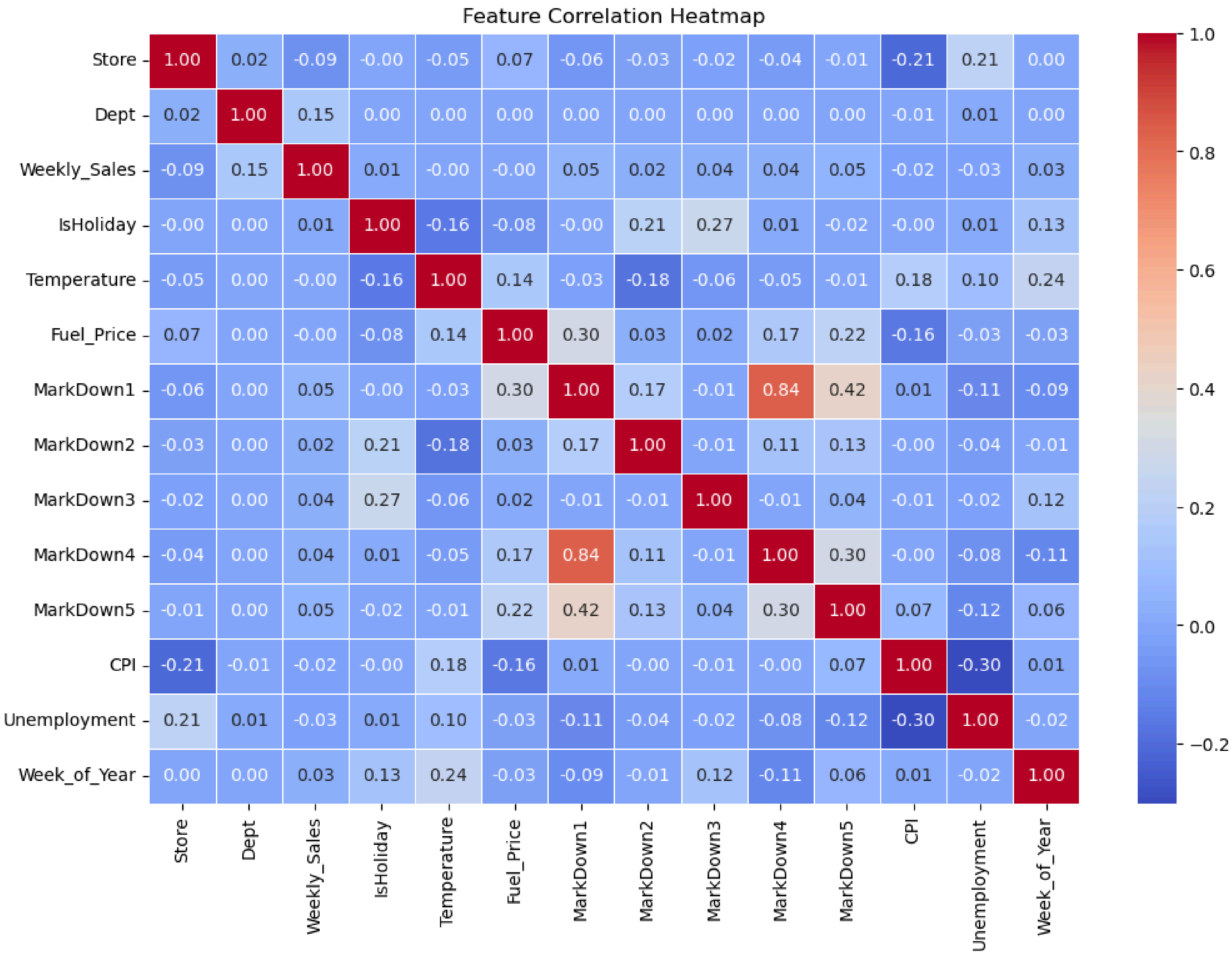
```
        Feature  Importance
1          Dept    0.562048
0         Store    0.262498
10          CPI    0.061139
12  Week_of_Year    0.037682
11  Unemployment    0.029255
3    Temperature    0.013583
7      MarkDown3    0.011101
9      MarkDown5    0.007657
4     Fuel_Price    0.006467
5      MarkDown1    0.003180
```

In [76]:
```
import matplotlib.pyplot as plt

# Plot feature importance
plt.figure(figsize=(12, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance in Random Forest Model')
plt.gca().invert_yaxis()  # To display the highest importance at the top
plt.show()
```



In [78]:
```
correlation_matrix = grouped_data.corr()
```

In [79]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Feature Correlation Heatmap')
plt.show()
```


Feature Correlation Heatmap
Store
Dept
Weekly_Sales
IsHoliday
Temperature
Fuel_Price
MarkDown1
MarkDown2
MarkDown3
MarkDown4
MarkDown5
CPI
Unemployment
Week_of_Year
1.00 0.02 -0.09 -0.00 -0.05 0.07 -0.06 -0.03 -0.02 -0.04 -0.01 -0.21 0.21 0.00
0.02 1.00 0.15 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 -0.01 0.01 0.00
-0.09 0.15 1.00 0.01 -0.00 -0.00 0.05 0.02 0.04 0.04 0.05 -0.02 -0.03 0.03
-0.00 0.00 0.01 1.00 -0.16 -0.08 -0.00 0.21 0.27 0.01 -0.02 -0.00 0.01 0.13
-0.05 0.00 -0.00 -0.16 1.00 0.14 -0.03 -0.18 -0.06 -0.05 -0.01 0.18 0.10 0.24
0.07 0.00 -0.00 -0.08 0.14 1.00 0.30 0.03 0.02 0.17 0.22 -0.16 -0.03 -0.03
-0.06 0.00 0.05 -0.00 -0.03 0.30 1.00 0.17 -0.01 0.84 0.42 0.01 -0.11 -0.09
-0.03 0.00 0.02 0.21 -0.18 0.03 0.17 1.00 -0.01 0.11 0.13 -0.00 -0.04 -0.01
-0.02 0.00 0.04 0.27 -0.06 0.02 -0.01 -0.01 1.00 -0.01 0.04 -0.01 -0.02 0.12
-0.04 0.00 0.04 0.01 -0.05 0.17 0.84 0.11 -0.01 1.00 0.30 -0.00 -0.08 -0.11
-0.01 0.00 0.05 -0.02 -0.01 0.22 0.42 0.13 0.04 0.30 1.00 0.07 -0.12 0.06
-0.21 -0.01 -0.02 -0.00 0.18 -0.16 0.01 -0.00 -0.01 -0.00 0.07 1.00 -0.30 0.01
0.21 0.01 -0.03 0.01 0.10 -0.03 -0.11 -0.04 -0.02 -0.08 -0.12 -0.30 1.00 -0.02
0.00 0.00 0.03 0.13 0.24 -0.03 -0.09 -0.01 0.12 -0.11 0.06 0.01 -0.02 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2