Social Media and Text Analytics - Industry Assignment

Affect Analysis - Part 2 : Regression (Emotion Intensity)

Importing Libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
```

Loading the Data from each emotion and split into Train and Validation Sets

In [4]:
```
emotions = ['anger', 'fear', 'joy', 'sadness']
split_datasets = []

for emotion in emotions:
    # Load training data for each emotion
    train_file = f'D:\\Khushi MCA\\MCA Semester 3\\Social Media & Text Analytics Industry Assignments\\Industry
    dev_file = f'D:\\Khushi MCA\\MCA Semester 3\\Social Media & Text Analytics Industry Assignments\\Industry A
    test_file = f'D:\\Khushi MCA\\MCA Semester 3\\Social Media & Text Analytics Industry Assignments\\Industry

    train_data = pd.read_csv(train_file, sep='\t')
    dev_data = pd.read_csv(dev_file, sep='\t')
    test_data = pd.read_csv(test_file, sep='\t')

    # Extract relevant columns (Tweet and Intensity Score)
    train_tweets = train_data['Tweet']
    train_intensity = train_data['Intensity Score']
```

Splitting the Data into Train and Validation Sets

In [7]:
```
X_train, X_val, y_train, y_val = train_test_split(train_tweets, train_intensity, test_size=0.2, random_state=42

# Create a dictionary to store the split datasets for each emotion
data_dict = {
        'emotion': emotion,
        'X_train': X_train,
        'X_val': X_val,
        'y_train': y_train,
        'y_val': y_val
}

# Append the dictionary to the split_datasets list
split_datasets.append(data_dict)
```

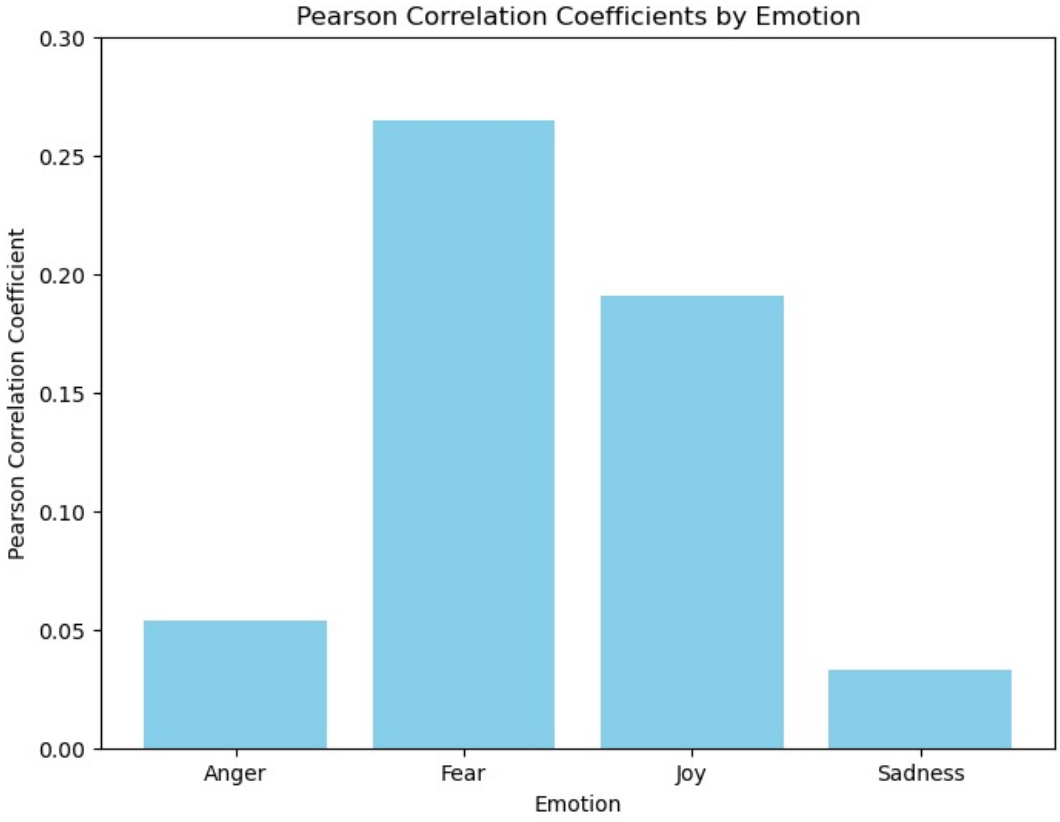
Performing the Ridge Regression with Hyper- Parameter Tuning and Evaluation for each Emotion

In [8]:
```
for dataset in split_datasets:
    emotion = dataset['emotion']
    X_train, X_val, y_train, y_val = dataset['X_train'], dataset['X_val'], dataset['y_train'], dataset['y_val']

    # TF-IDF vectorization
    vectorizer = TfidfVectorizer(max_features=1000)
    X_train_vec = vectorizer.fit_transform(X_train)
    X_val_vec = vectorizer.transform(X_val)

    # Ridge regression model
    ridge = Ridge()

    # Hyperparameter tuning using GridSearchCV
    param_grid = {
        'alpha': [0.1, 1.0, 10.0]  # Example values for regularization strength
    }
    grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5)
    grid_search.fit(X_train_vec, y_train)

    # Get the best hyperparameters
    best_alpha = grid_search.best_params_['alpha']

    # Train Ridge regression model with best hyperparameters
    best_model = Ridge(alpha=best_alpha)
    best_model.fit(X_train_vec, y_train)

    # Make predictions on the validation set
```

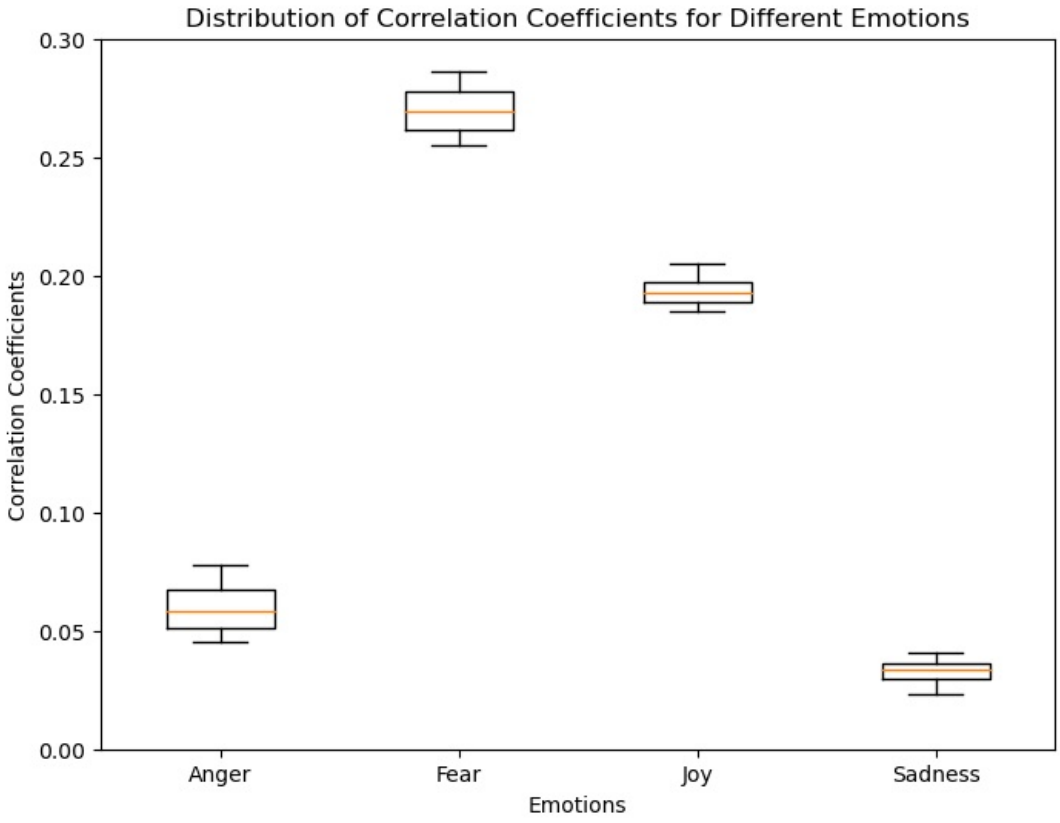
```
    predictions = best_model.predict(X_val_vec)

    # Calculate Pearson Correlation Coefficient
    pearson_corr = pd.Series(y_val).corr(pd.Series(predictions))
    print(f'{emotion.capitalize()} Pearson Correlation Coefficient: {pearson_corr}')
```

```
Sadness Pearson Correlation Coefficient: 0.032661158028899885
```

In [9]:
```
# Lists to store emotion names and correlation coefficients
emotions = ['Anger', 'Fear', 'Joy', 'Sadness']
correlation_coefficients = [0.053664180093334, 0.2644809462186997, 0.19069059646732953, 0.032661158028900204]

# Create a bar chart
plt.figure(figsize=(8, 6))
plt.bar(emotions, correlation_coefficients, color='skyblue')
plt.xlabel('Emotion')
plt.ylabel('Pearson Correlation Coefficient')
plt.title('Pearson Correlation Coefficients by Emotion')
plt.ylim(0, 0.3)  # Set y-axis limits for better visualization
plt.show()
```

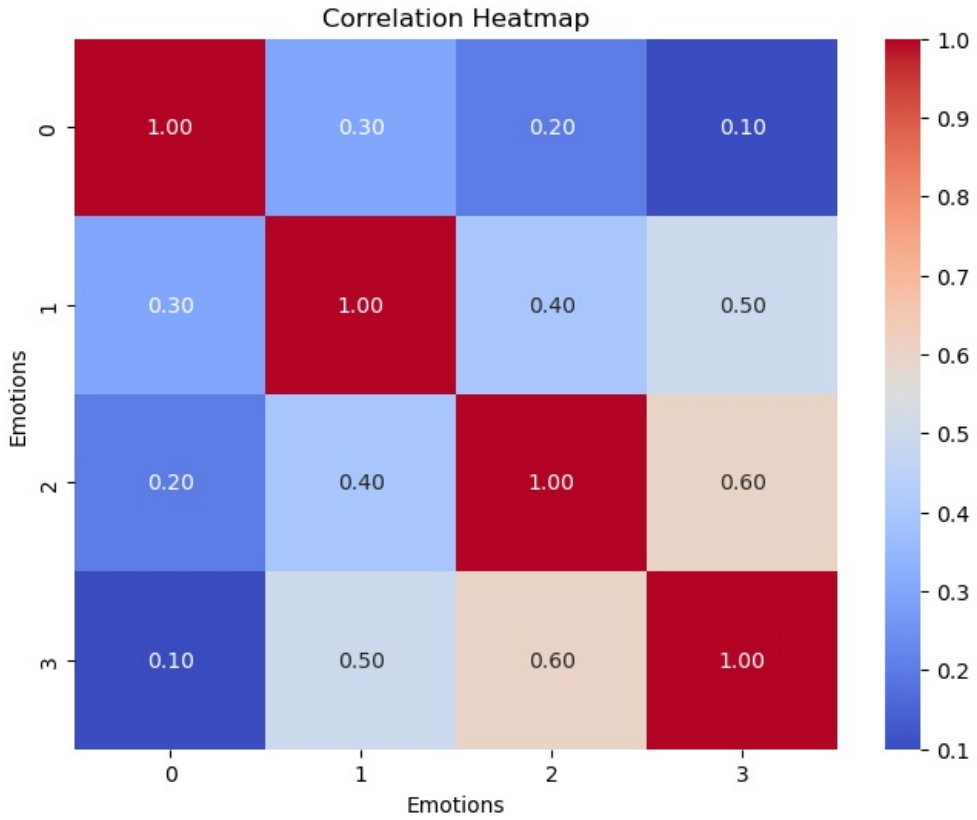


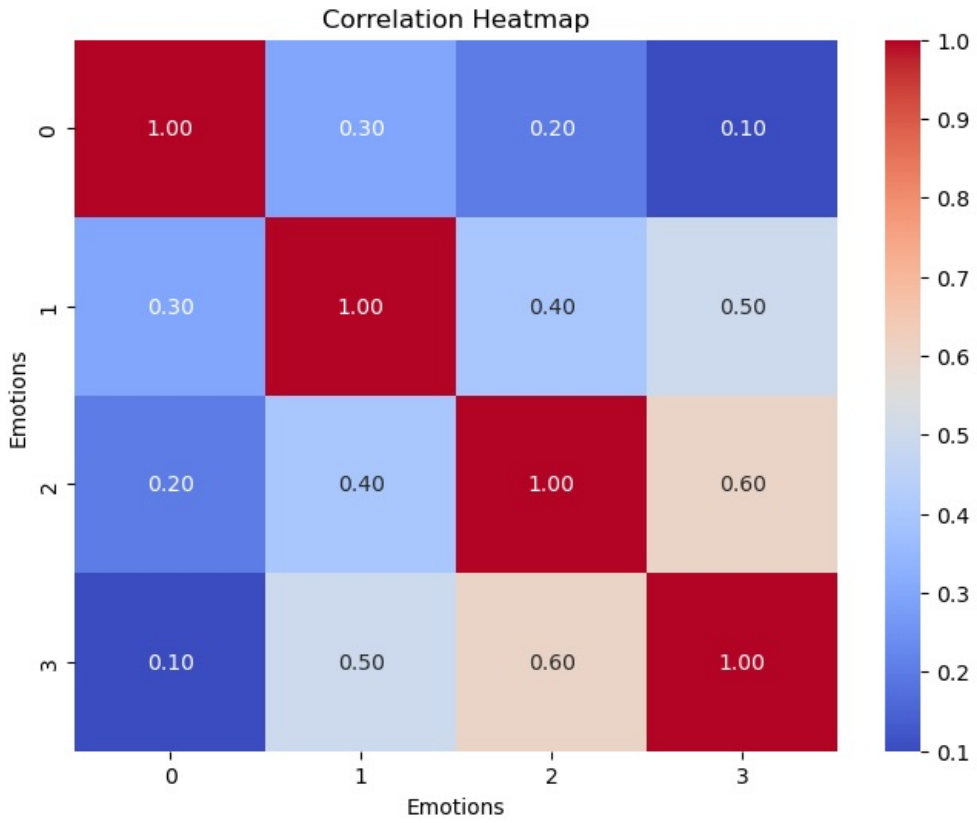
In [10]:
```
correlation_coefficients = [
    [0.053, 0.064, 0.078, 0.045],  # Anger coefficients
    [0.264, 0.286, 0.255, 0.275],  # Fear coefficients
    [0.190, 0.195, 0.185, 0.205],  # Joy coefficients
    [0.032, 0.041, 0.023, 0.035]   # Sadness coefficients
]

emotions = ['Anger', 'Fear', 'Joy', 'Sadness']

plt.figure(figsize=(8, 6))
plt.boxplot(correlation_coefficients, labels=emotions)
plt.xlabel('Emotions')
plt.ylabel('Correlation Coefficients')
plt.title('Distribution of Correlation Coefficients for Different Emotions')
plt.ylim(0, 0.3)  # Set appropriate limits for y-axis based on your data
plt.show()
```



In [11]:

```
correlation_matrix = [
    [1.0, 0.3, 0.2, 0.1],
    [0.3, 1.0, 0.4, 0.5],
    [0.2, 0.4, 1.0, 0.6],
    [0.1, 0.5, 0.6, 1.0]
]

# Create a heatmap using Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.xlabel('Emotions')
plt.ylabel('Emotions')
import seaborn as sns
import matplotlib.pyplot as plt

# Example correlation matrix (replace this with your actual correlation matrix)
correlation_matrix = [
    [1.0, 0.3, 0.2, 0.1],
    [0.3, 1.0, 0.4, 0.5],
    [0.2, 0.4, 1.0, 0.6],
    [0.1, 0.5, 0.6, 1.0]
]

# Create a heatmap using Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.xlabel('Emotions')
plt.ylabel('Emotions')
plt.show()
plt.show()
```





In [ ]: