SEAS6414_HW6_Doran

February 22, 2024

Question 1: Financial Sentiment Analysis

Data Preparation: Load and convert the text file into a DataFrame with columns: 'Text' and 'Sentiment':

[1]:
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

# Initialize lists to hold the text and sentiment values
texts = []
sentiments = []

# Open the file and parse lines according to the structure "This is a sentence .
↪@sentiment"
with open('Q1_FinancialDataset_Sentences_AllAgree.txt', 'r') as file:
    for line in file:
        parts = line.strip().split('.@')
        if len(parts) == 2:  # Ensure line has both parts
            texts.append(parts[0])
            sentiments.append(parts[1])

# Create the DataFrame
df = pd.DataFrame({
    'Text': texts,
    'Sentiment': sentiments
})

df.head()
```

```
/tmp/ipykernel_280008/651493850.py:1: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better
interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466
```

```
import pandas as pd
```

[1]:

```
                                                Text Sentiment
0  According to Gran , the company has no plans t…   neutral
1  For the last quarter of 2010 , Componenta 's n…  positive
2  In the third quarter of 2010 , net sales incre…  positive
3  Operating profit rose to EUR 13.1 mn from EUR …  positive
4  Operating profit totalled EUR 21.1 mn , up fro…  positive
```

Exploratory Data Analysis (EDA): Perform EDA and plot bar charts for the frequency of the top 20 words in each sentiment category.

[2]:

```
df.describe()
```

[2]:

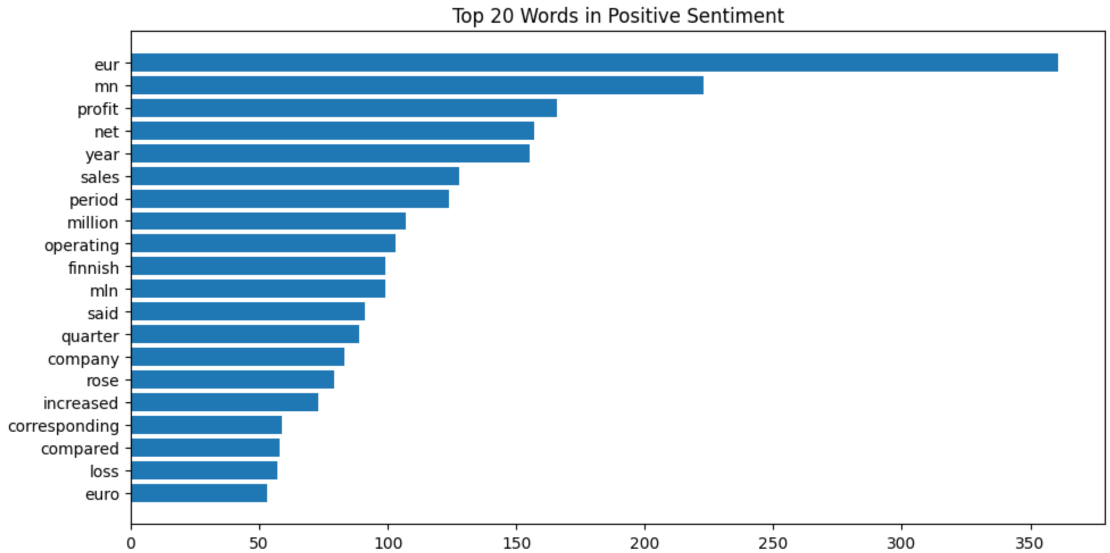
```
                                                     Text Sentiment
count                                                2208      2208
unique                                               2203         3
top     The report profiles 614 companies including ma…   neutral
freq                                                    2      1358
```

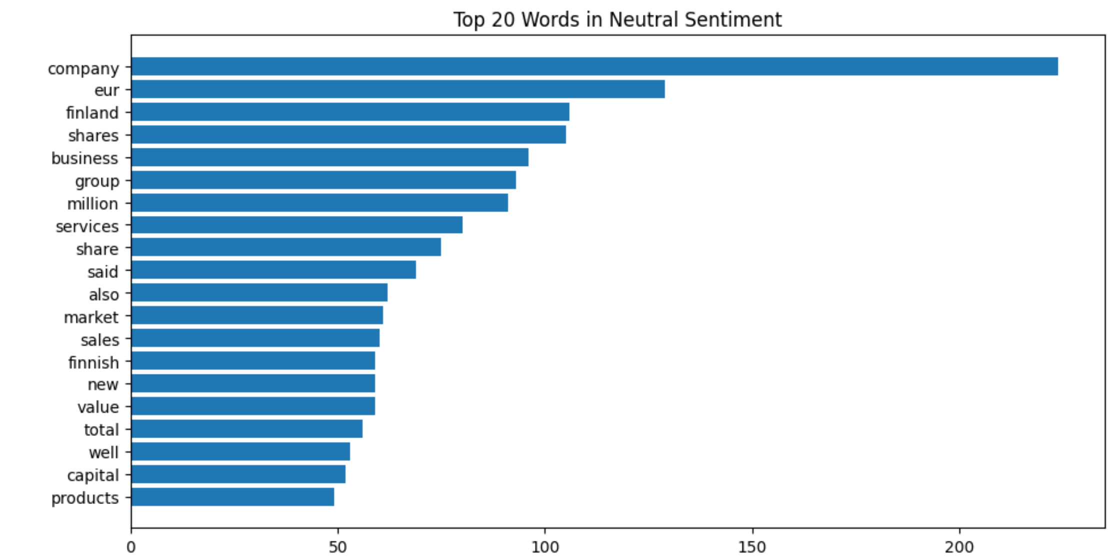
[3]:

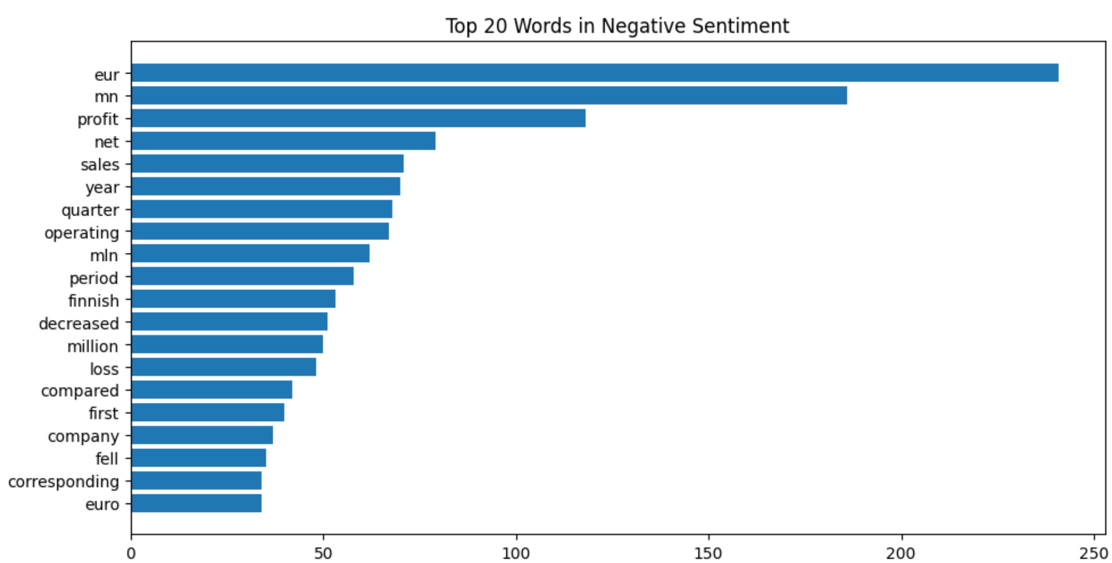
```
import pandas as pd
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from collections import Counter
import re
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Preprocess and tokenize
def preprocess(text):
    tokens = re.findall(r'\b\w+\b', text.lower())  # Tokenize and convert to␣
↪lowercase
    return [word for word in tokens if word not in stop_words and not word.
↪isdigit()]

df['Tokens'] = df['Text'].apply(preprocess)

# Count word frequencies by sentiment
frequencies = df.groupby('Sentiment')['Tokens'].sum().apply(lambda x:␣
↪Counter(x))

# Plot the results
fig, axes = plt.subplots(3, 1, figsize=(10, 15))
sentiments = ['positive', 'neutral', 'negative']

for i, sentiment in enumerate(sentiments):
    top_words = frequencies[sentiment].most_common(20)
```

```
    words, counts = zip(*reversed(top_words))  # Reverse the order for␣
↪descending plot
    axes[i].barh(words, counts)
    axes[i].set_title(f'Top 20 Words in {sentiment.capitalize()} Sentiment')

plt.tight_layout()
plt.show()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/danrdoran/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

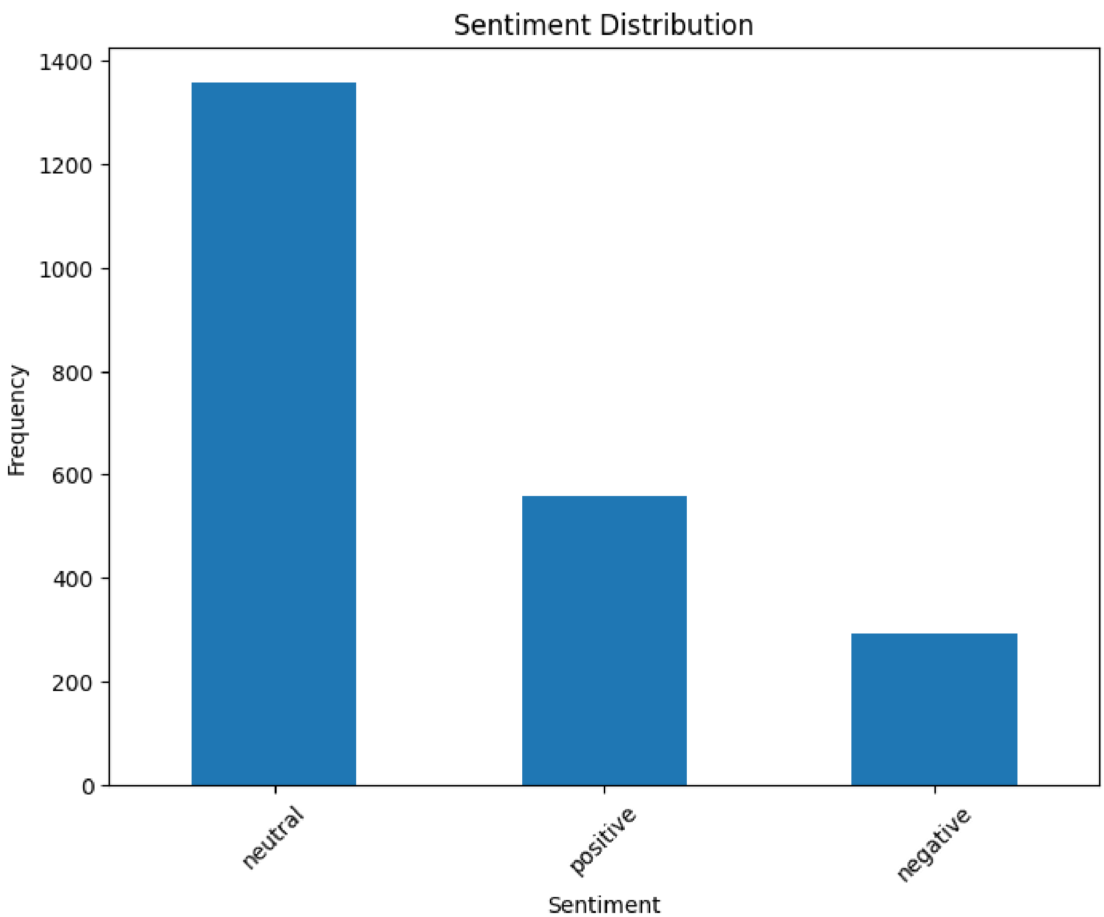

Top 20 Words in Positive Sentiment
eur
mn
profit
net
year
sales
period
million
operating
finnish
mln
said
quarter
company
rose
increased
corresponding
compared
loss
euro
0
50
100
150
200
250
300
350


Top 20 Words in Neutral Sentiment
company
eur
finland
shares
business
group
million
services
share
said
also
market
sales
finnish
new
value
total
well
capital
products
0
50
100
150
200


Top 20 Words in Negative Sentiment
eur
mn
profit
net
sales
year
quarter
operating
mln
period
finnish
decreased
million
loss
compared
first
company
fell
corresponding
euro
0
50
100
150
200
250

Class Imbalance Analysis: Compute and visualize the frequency of each sentiment label with a bar graph. Discuss class imbalance.

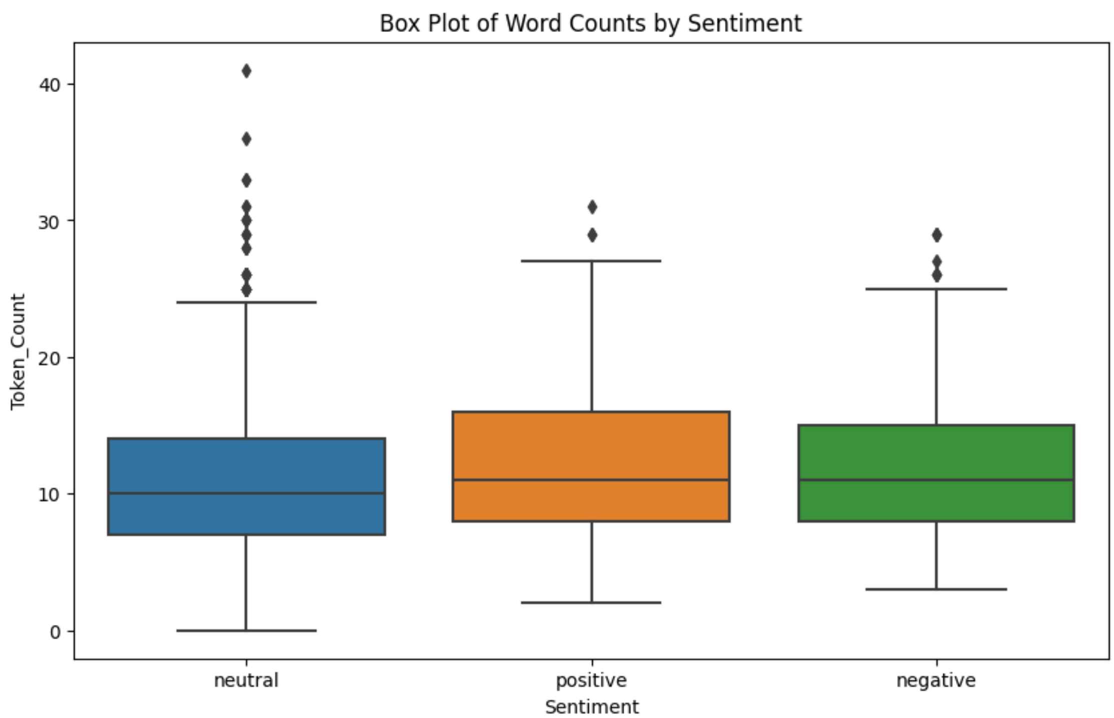
[4]:
```
import pandas as pd
import matplotlib.pyplot as plt

sentiment_counts = df['Sentiment'].value_counts()

# Plotting sentiment distribution
plt.figure(figsize=(8, 6))
sentiment_counts.plot(kind='bar')
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```



In this dataset the classes are quite imbalanced, with there being a much fewer number of negative and positive entries compared to neutral entries. This could affect our future analyses and we may want to address this by oversampling the underrepresented classes (negative and positive) or undersampling the overrepresented class (neutral).

Word Count Analysis: Create box plots for word/token counts per sentiment label. Discuss discrepancies.

[5]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df['Token_Count'] = df['Tokens'].apply(len)

# Create box plots
plt.figure(figsize=(10, 6))
sns.boxplot(x='Sentiment', y='Token_Count', data=df)
plt.title('Box Plot of Word Counts by Sentiment')
plt.show()
```



With this plot we can identify the distribution of sentence lengths within each sentiment category. We can see that negative and positive sentences tend to be slightly longer than neutral ones on average, however the longest sentences in the dataset are neutral but these are outliers in the distribution of word counts in this category.

Data Splitting and Model Development and Evaluation:

[6]:
```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Initialize vectorizers
count_vect = CountVectorizer()
tfidf_vect = TfidfVectorizer()

# Fit and transform the data
X_count = count_vect.fit_transform(df['Text'])
X_tfidf = tfidf_vect.fit_transform(df['Text'])
```

[7]:
```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, df['Sentiment'],
 ↪test_size=0.2, random_state=64, stratify=df['Sentiment'])

# Apply SMOTE to the training set to account for class imbalances
smote = SMOTE(random_state=64)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Initialize Naive Bayes, Random Forest, and SVM models
models = {
    "MultinomialNB": MultinomialNB(),
    "RandomForestClassifier": RandomForestClassifier(),
    "SVC": SVC()
}

# Train and evaluate models using the SMOTE-resampled data
for name, model in models.items():
    model.fit(X_train_smote, y_train_smote)
    y_pred = model.predict(X_test)
    print(f"Model: {name}")
    print(classification_report(y_test, y_pred))
    print("Confusion matrix:")
    print(confusion_matrix(y_test, y_pred))
```

```
Model: MultinomialNB
              precision    recall  f1-score   support

    negative       0.57      0.80      0.67        59
     neutral       0.91      0.90      0.91       272
    positive       0.79      0.65      0.71       111

    accuracy                           0.82       442
```

```
   macro avg       0.76      0.78      0.76       442
weighted avg       0.84      0.82      0.83       442

Confusion matrix:
[[ 47   6   6]
 [ 14 245  13]
 [ 21  18  72]]
Model: RandomForestClassifier
              precision    recall  f1-score   support

    negative       0.81      0.59      0.69        59
     neutral       0.84      1.00      0.91       272
    positive       0.87      0.59      0.70       111

    accuracy                           0.84       442
   macro avg       0.84      0.73      0.76       442
weighted avg       0.84      0.84      0.83       442

Confusion matrix:
[[ 35  14  10]
 [  1 271   0]
 [  7  39  65]]
Model: SVC
              precision    recall  f1-score   support

    negative       0.85      0.56      0.67        59
     neutral       0.82      0.99      0.90       272
    positive       0.92      0.62      0.74       111

    accuracy                           0.84       442
   macro avg       0.86      0.72      0.77       442
weighted avg       0.85      0.84      0.83       442

Confusion matrix:
[[ 33  21   5]
 [  1 270   1]
 [  5  37  69]]
```

Based on the evaluation results, MultinomialNB shows moderate effectiveness, with a high precision in the neutral category but significantly lower precision and recall in the negative and positive categories. This suggests difficulty in distinguishing between sentiment extremes, likely due to the model's simplicity and its assumption of feature independence.

RandomForestClassifier improves notably on accuracy and precision across all categories compared to MultinomialNB. Its ability to handle the imbalanced dataset is evident, particularly in the neutral category. However, its performance on the negative category, despite high precision, indicates a lower recall, suggesting some overfitting or difficulty in generalizing from limited negative examples.

SVC demonstrates the best balance between precision and recall across all categories, indicating a strong ability to generalize and effectively manage the feature space of the text data. Its performance is particularly noteworthy in handling negative sentiments more effectively than MultinomialNB and with comparable effectiveness to RandomForestClassifier but with better recall.

Considering aspects like overfitting, class imbalances, and model handling of the feature space, SVC stands out as the most effective model for this dataset. It shows a balanced approach to dealing with both majority and minority classes while maintaining high precision and recall. RandomForestClassifier also performs well, particularly in dealing with class imbalances but may need tuning to reduce potential overfitting. MultinomialNB, while useful for a baseline, struggles with the nuances of sentiment analysis, particularly in distinguishing between different sentiment polarities.

Question 2: Predicting Building Energy Efficiency

Data Preprocessing:

[8]:
```python
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

df2 = pd.read_excel('ENB2012_data.xlsx')

df2.head()
```

[8]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

[9]:
```python
df2.describe()
```

[9]:

	X1	X2	X3	X4	X5	X6 \
count	768.000000	768.000000	768.000000	768.000000	768.00000	768.000000
mean	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000
std	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763
min	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000
25%	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000
50%	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000
75%	0.830000	741.125000	343.000000	220.500000	7.00000	4.250000
max	0.980000	808.500000	416.500000	220.500000	7.00000	5.000000

	X7	X8	Y1	Y2
count	768.000000	768.00000	768.000000	768.000000
mean	0.234375	2.81250	22.307195	24.587760
std	0.133221	1.55096	10.090204	9.513306
min	0.000000	0.00000	6.010000	10.900000
25%	0.100000	1.75000	12.992500	15.620000
50%	0.250000	3.00000	18.950000	22.080000

```
75%       0.400000    4.00000    31.667500    33.132500
max       0.400000    5.00000    43.100000    48.030000
```

Close mean and median values for some variables suggest a symmetric distribution, while differences in others may indicate slight skewness.

[10]:
```
df2.info()
```

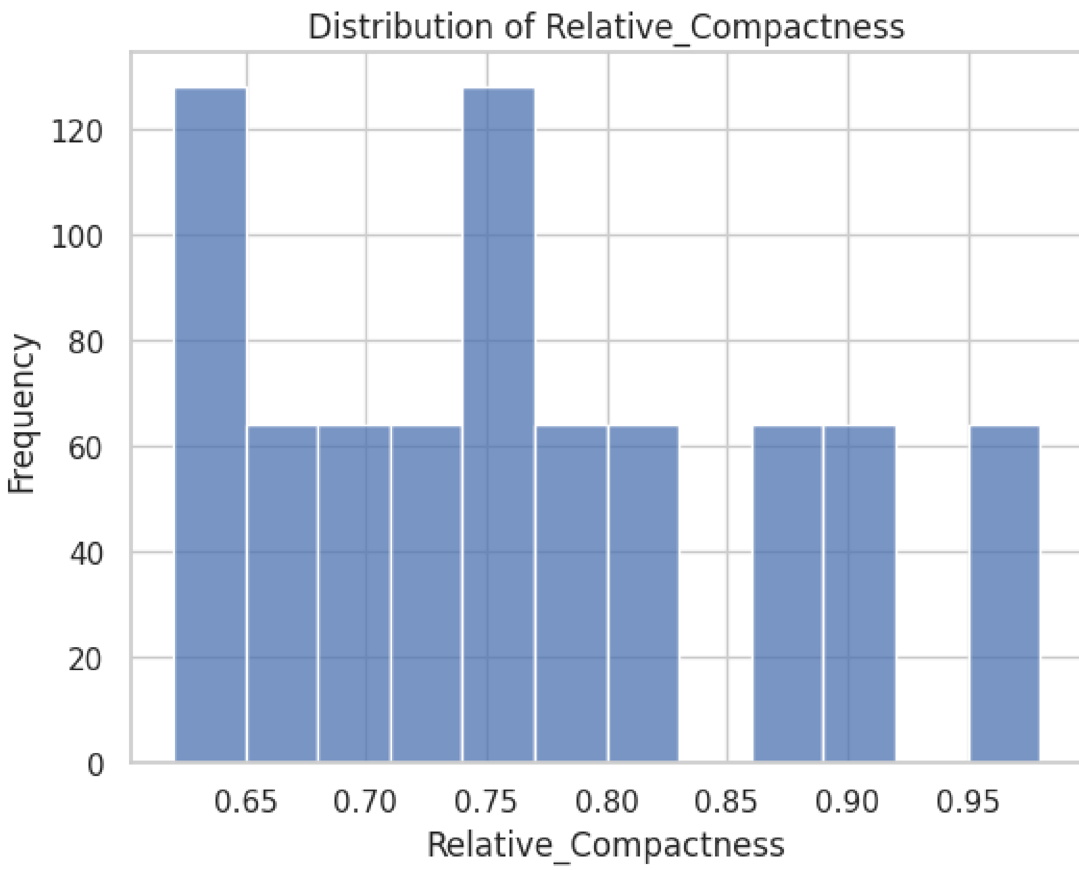
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      768 non-null    float64
 1   X2      768 non-null    float64
 2   X3      768 non-null    float64
 3   X4      768 non-null    float64
 4   X5      768 non-null    float64
 5   X6      768 non-null    int64
 6   X7      768 non-null    float64
 7   X8      768 non-null    int64
 8   Y1      768 non-null    float64
 9   Y2      768 non-null    float64
dtypes: float64(8), int64(2)
memory usage: 60.1 KB
```

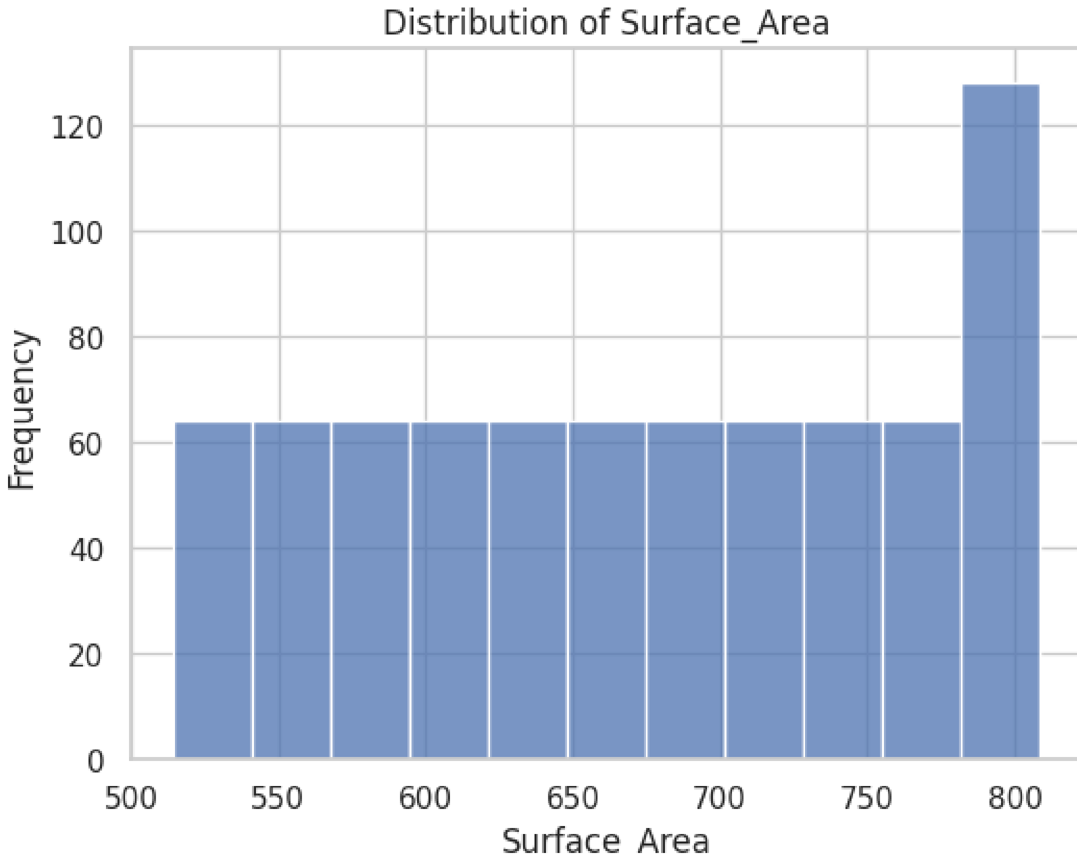
We observe no missing values.

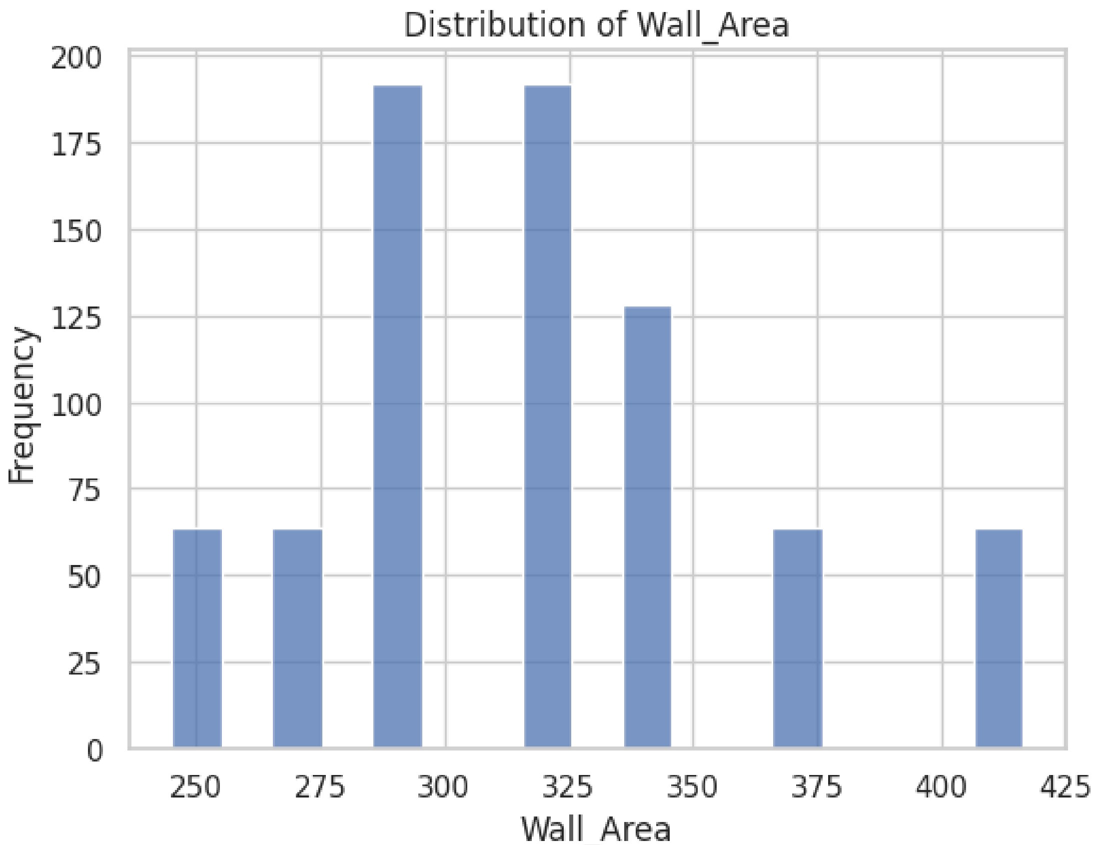
[21]:
```
# Rename columns
column_names = {
    'X1': 'Relative_Compactness',
    'X2': 'Surface_Area',
    'X3': 'Wall_Area',
    'X4': 'Roof_Area',
    'X5': 'Overall_Height',
    'X6': 'Orientation',
    'X7': 'Glazing_Area',
    'X8': 'Glazing_Area_Distribution',
    'Y1': 'Heating_Load',
    'Y2': 'Cooling_Load'
}

df2.rename(columns=column_names, inplace=True)
```

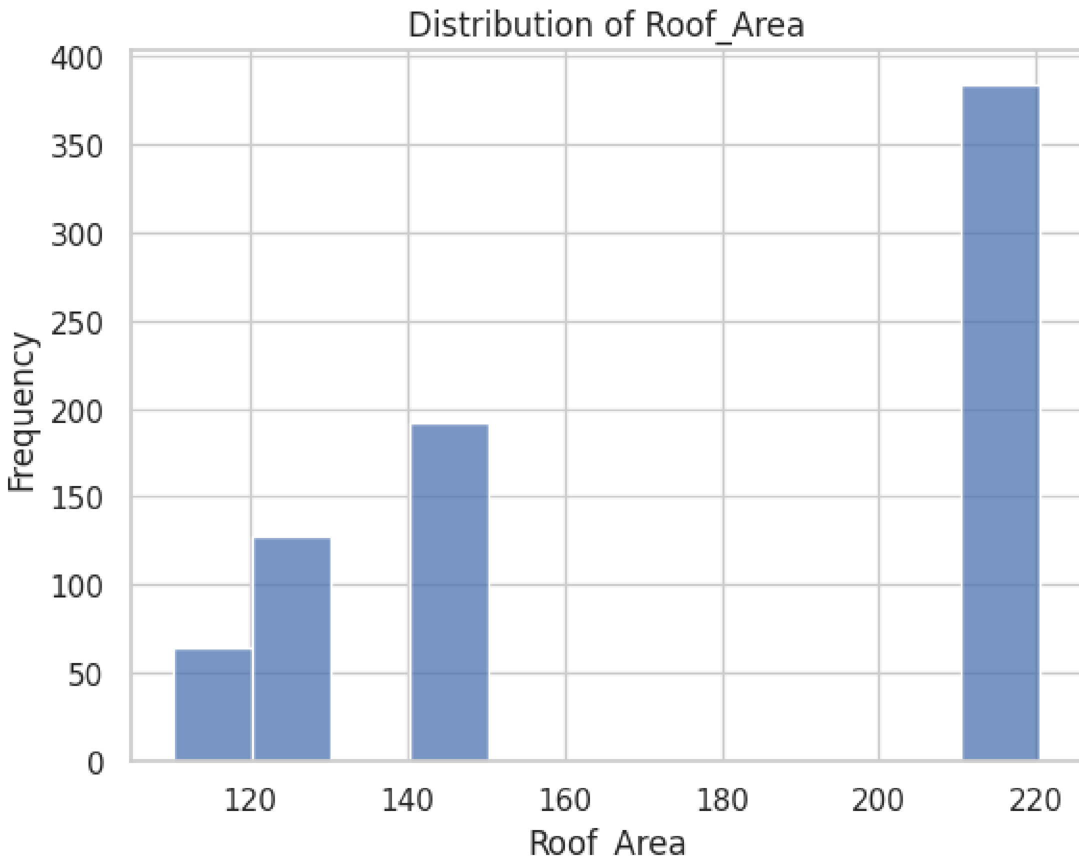
[22]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Set the style of the plots
sns.set(style='whitegrid')
```

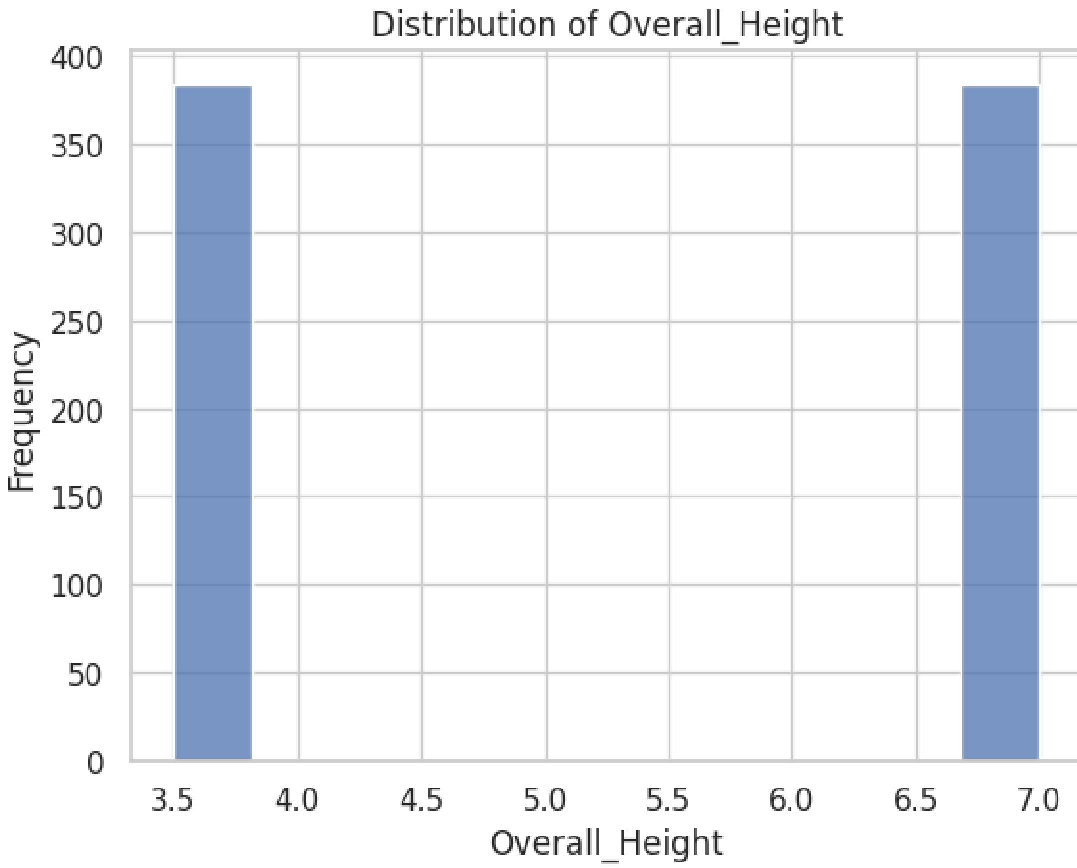
```python
# Plot distribution of each column
for column in df2.columns:
    sns.histplot(df2[column])
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

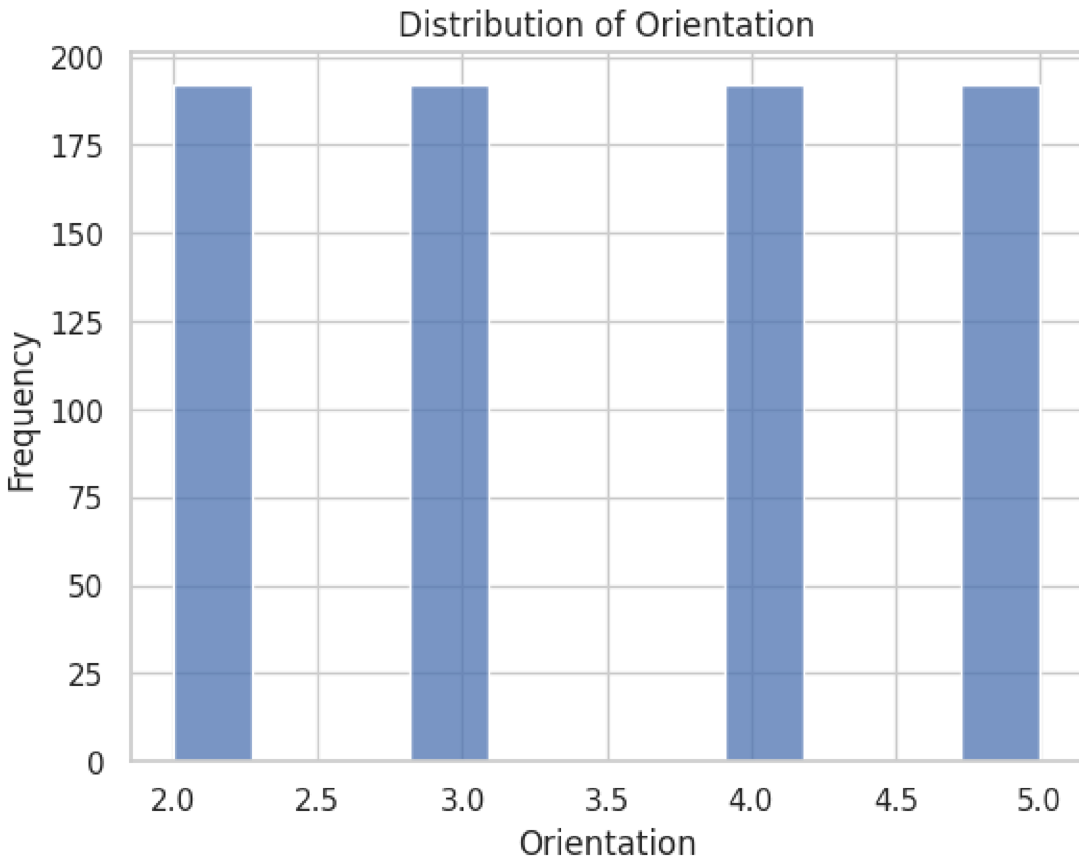



Distribution of Surface_Area
Frequency
0
20
40
60
80
100
120
500
550
600
650
700
750
800
Surface_Area

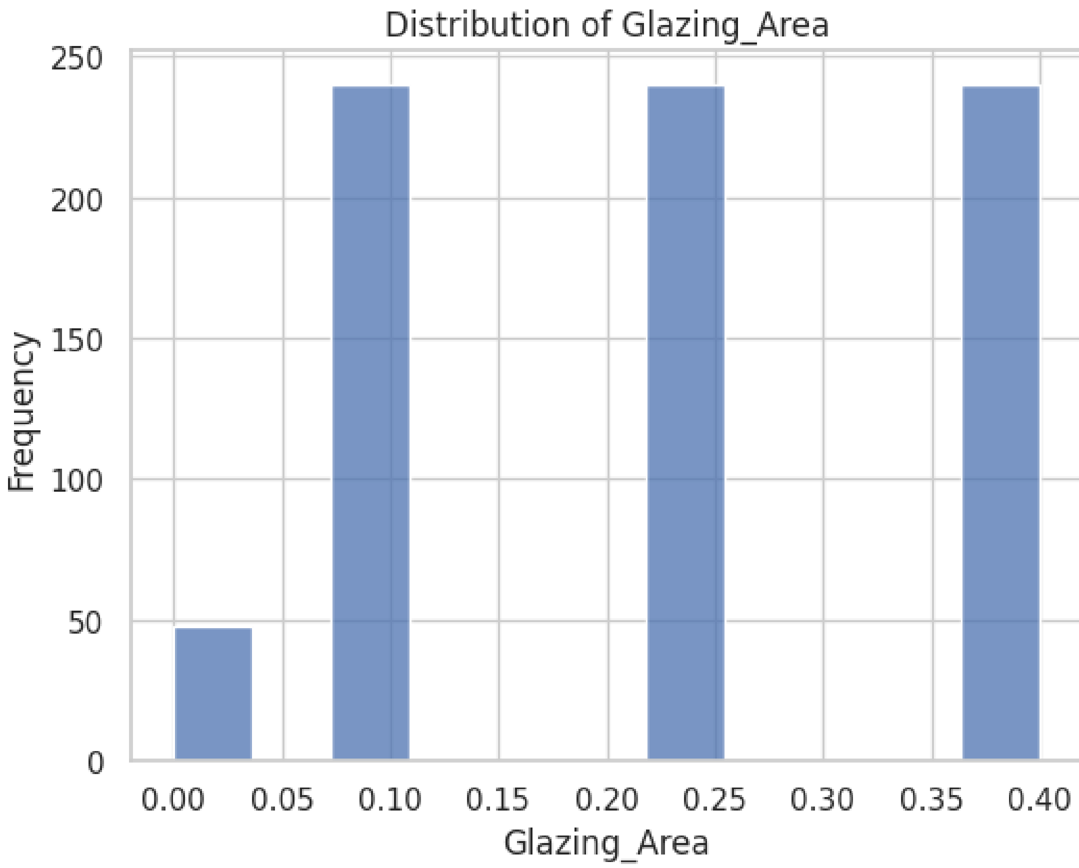

Distribution of Wall_Area
200
175
150
125
100
75
50
25
0
Frequency
250
275
300
325
350
375
400
425
Wall_Area

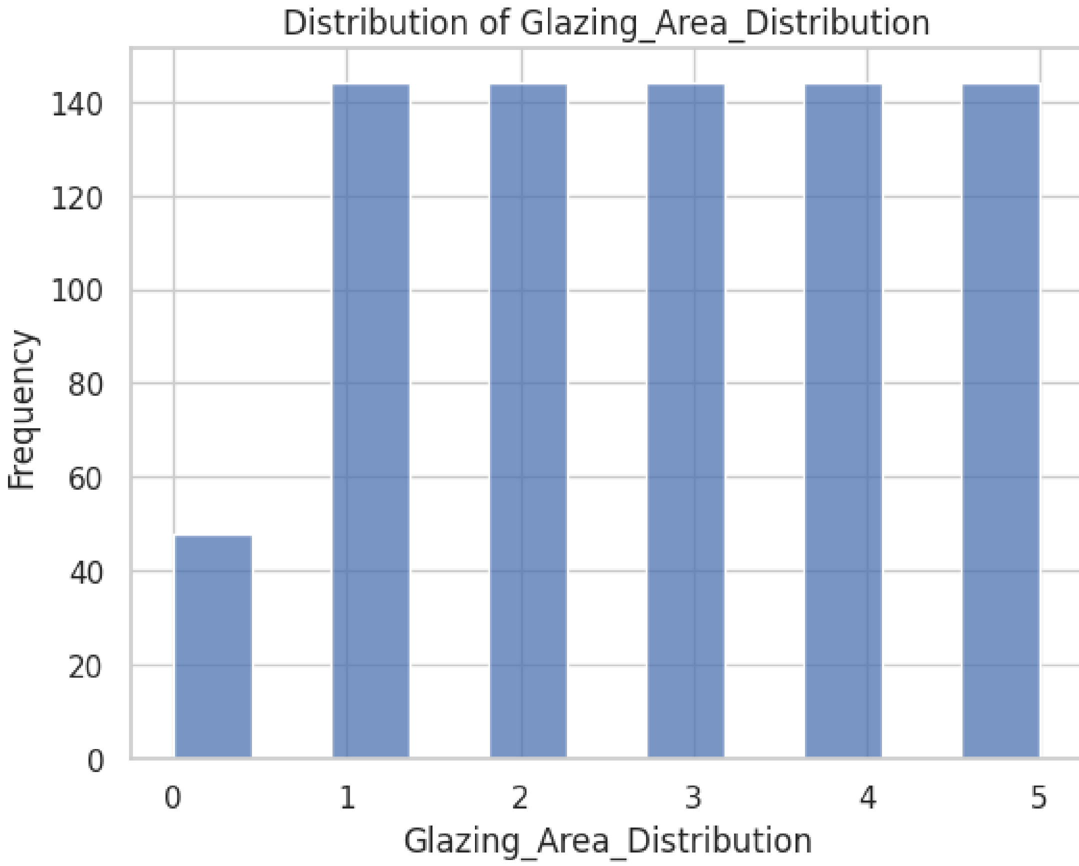

Distribution of Roof_Area
400
350
300
250
200
150
100
50
0
Frequency
120
140
160
180
200
220
Roof_Area

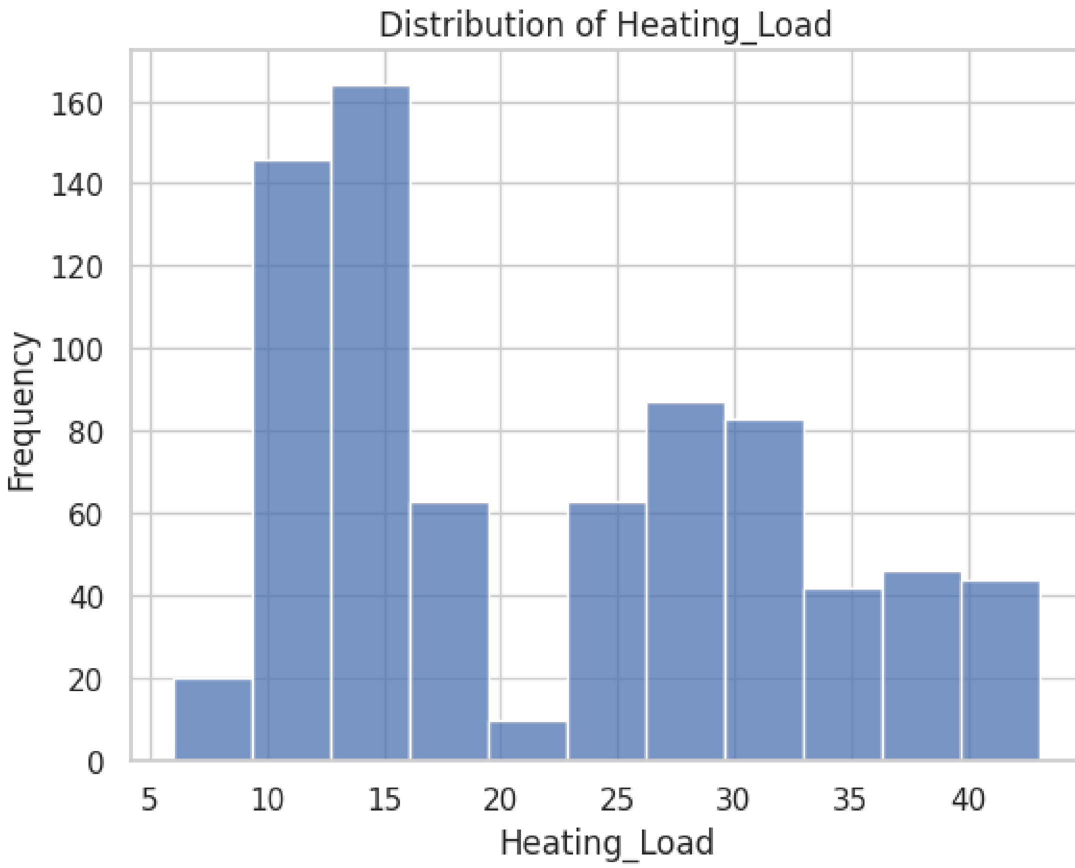

Distribution of Overall_Height
Frequency
Overall_Height
0
50
100
150
200
250
300
350
400
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0

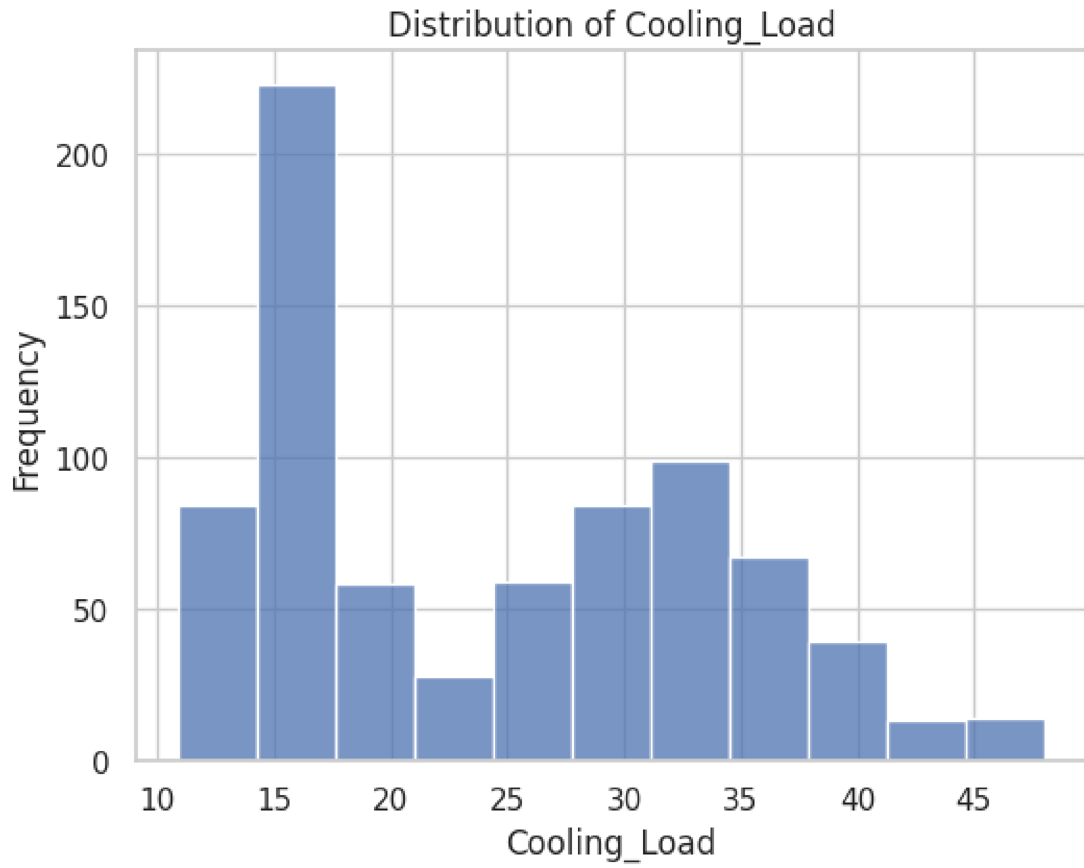


Distribution of Orientation
200
175
150
125
100
75
50
25
0
Frequency
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Orientation


Distribution of Glazing_Area
Frequency
0
50
100
150
200
250
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Glazing_Area


Distribution of Glazing_Area_Distribution
Frequency
0
20
40
60
80
100
120
140
0
1
2
3
4
5
Glazing_Area_Distribution


Distribution of Heating_Load
Frequency
0
20
40
60
80
100
120
140
160
5
10
15
20
25
30
35
40
Heating_Load



We see that indeed most variables have a relatively symmetric distribution, while others are slightly skewed. Given the varying scales of the continuous variables, it would be beneficial to normalize or standardize them. We'll also use one-hot encoding for Glazing_Area_Distribution due to its nominal nature.

[13]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

# Identifying categorical and continuous columns
continuous_cols = ['Relative_Compactness', 'Surface_Area', 'Wall_Area',␣
↪'Roof_Area', 'Overall_Height', 'Glazing_Area']
categorical_cols = ['Glazing_Area_Distribution']

# Defining the transformers for continuous and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('std_scaler', StandardScaler(), continuous_cols),
```

```
        ('onehot', OneHotEncoder(), categorical_cols)
    ])

# Applying the transformations
df_transformed = preprocessor.fit_transform(df2)
```

Model Development and Target Variable Analysis:

[14]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Define targets
y_heating = df2['Heating_Load']
y_cooling = df2['Cooling_Load']

# Splitting the dataset for Heating Load
X_train_h, X_test_h, y_train_h, y_test_h = train_test_split(df_transformed,␣
↪y_heating, test_size=0.2, random_state=42)

# Splitting the dataset for Cooling Load
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(df_transformed,␣
↪y_cooling, test_size=0.2, random_state=42)
```

Let's define a function that takes a model, training data, and testing data, then fits the model, makes predictions, and evaluates these predictions:

[15]:
```
def evaluate_model(model, X_train, y_train, X_test, y_test, name=''):
    # Fit the model
    model.fit(X_train, y_train)

    # Predictions
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    # Evaluation on Training set
    mse_train = mean_squared_error(y_train, y_pred_train)
    mae_train = mean_absolute_error(y_train, y_pred_train)
    r2_train = r2_score(y_train, y_pred_train)

    # Evaluation on Testing set
    mse_test = mean_squared_error(y_test, y_pred_test)
    mae_test = mean_absolute_error(y_test, y_pred_test)
    r2_test = r2_score(y_test, y_pred_test)

    # Print results
    print(f"{name} - Training set: RMSE={np.sqrt(mse_train)}, MAE={mae_train},␣
↪R²={r2_train}")
```

```
    print(f"{name} - Testing set: RMSE={np.sqrt(mse_test)}, MAE={mae_test},␣
↪R²={r2_test}\n")
```

[16]:
```
from sklearn.linear_model import LinearRegression

# Linear Regression doesn't require hyperparameter tuning
evaluate_model(LinearRegression(), X_train_h, y_train_h, X_test_h, y_test_h,␣
↪'Linear Regression - Heating Load')
evaluate_model(LinearRegression(), X_train_c, y_train_c, X_test_c, y_test_c,␣
↪'Linear Regression - Cooling Load')
```

```
Linear Regression - Heating Load - Training set: RMSE=2.777750197956874,
MAE=1.9884627043385459, R²=0.9235473934099114
Linear Regression - Heating Load - Testing set: RMSE=2.862593047906846,
MAE=2.06264761352539, R²=0.9213830034786246

Linear Regression - Cooling Load - Training set: RMSE=3.1792410392586437,
MAE=2.251422023959579, R²=0.8872665382481612
Linear Regression - Cooling Load - Testing set: RMSE=3.1084895300700874,
MAE=2.187161759215516, R²=0.8957155697026589
```

[17]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Setting up the hyperparameter grid
parameters_ridge = {'alpha': [0.01, 0.1, 1, 10, 100]}
ridge = GridSearchCV(Ridge(), parameters_ridge,␣
↪scoring='neg_mean_squared_error', cv=5)

# Evaluate Ridge Regression
evaluate_model(ridge, X_train_h, y_train_h, X_test_h, y_test_h, 'Ridge␣
↪Regression - Heating Load')
evaluate_model(ridge, X_train_c, y_train_c, X_test_c, y_test_c, 'Ridge␣
↪Regression - Cooling Load')
```

```
Ridge Regression - Heating Load - Training set: RMSE=2.7678047381973006,
MAE=1.989162794766509, R²=0.9240938753308721
Ridge Regression - Heating Load - Testing set: RMSE=2.865118837571915,
MAE=2.053250785353397, R²=0.9212442079198869

Ridge Regression - Cooling Load - Training set: RMSE=3.17557998490684,
MAE=2.255835287736234, R²=0.8875260251445081
Ridge Regression - Cooling Load - Testing set: RMSE=3.111981949461153,
MAE=2.182905771981858, R²=0.895481108845055
```

[18]:

```
from sklearn.linear_model import Lasso

# Hyperparameters for Lasso
parameters_lasso = {'alpha': [0.001, 0.01, 0.1, 1, 10]}

# GridSearchCV for Lasso
lasso = GridSearchCV(Lasso(), parameters_lasso, cv=5,␣
↪scoring='neg_mean_squared_error')

# Evaluate for Heating Load
evaluate_model(lasso, X_train_h, y_train_h, X_test_h, y_test_h, 'Lasso␣
↪Regression - Heating Load')
# Evaluate for Cooling Load
evaluate_model(lasso, X_train_c, y_train_c, X_test_c, y_test_c, 'Lasso␣
↪Regression - Cooling Load')
```

```
Lasso Regression - Heating Load - Training set: RMSE=2.767822537117594,
MAE=1.9890918167946123, R²=0.9240928990687909
Lasso Regression - Heating Load - Testing set: RMSE=2.864697998850492,
MAE=2.053461208979122, R²=0.9212673420756456

Lasso Regression - Cooling Load - Training set: RMSE=3.175582279228524,
MAE=2.2552160362173455, R²=0.8875258626220247
Lasso Regression - Cooling Load - Testing set: RMSE=3.111314795493689,
MAE=2.182321938630584, R²=0.8955259180499799
```

[19]:

```
from sklearn.linear_model import ElasticNet

# Hyperparameters for Elastic Net
parameters_elastic = {'alpha': [0.001, 0.01, 0.1, 1, 10], 'l1_ratio': [0.2, 0.
↪5, 0.8]}

# GridSearchCV for Elastic Net
elastic_net = GridSearchCV(ElasticNet(), parameters_elastic, cv=5,␣
↪scoring='neg_mean_squared_error')

# Evaluate for Heating Load
evaluate_model(elastic_net, X_train_h, y_train_h, X_test_h, y_test_h, 'Elastic␣
↪Net Regression - Heating Load')
# Evaluate for Cooling Load
evaluate_model(elastic_net, X_train_c, y_train_c, X_test_c, y_test_c, 'Elastic␣
↪Net Regression - Cooling Load')
```

```
Elastic Net Regression - Heating Load - Training set: RMSE=2.7678894754696053,
MAE=1.9889378840158642, R²=0.9240892274766884
Elastic Net Regression - Heating Load - Testing set: RMSE=2.865675890414485,
MAE=2.0541465568133286, R²=0.9212135806369199
```

```
Elastic Net Regression - Cooling Load - Training set: RMSE=3.175660555232848,
MAE=2.2556970062300112, R²=0.8875203177273904
Elastic Net Regression - Cooling Load - Testing set: RMSE=3.1123090366835786,
MAE=2.182922931382794, R²=0.8954591366158015
```

[20]:
```python
from sklearn.ensemble import RandomForestRegressor

# Hyperparameters for Random Forest
parameters_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# GridSearchCV for Random Forest
random_forest = GridSearchCV(RandomForestRegressor(random_state=42),␣
 ↪parameters_rf, cv=5, scoring='neg_mean_squared_error')

# Evaluate for Heating Load
evaluate_model(random_forest, X_train_h, y_train_h, X_test_h, y_test_h, 'Random␣
 ↪Forest Regression - Heating Load')
# Evaluate for Cooling Load
evaluate_model(random_forest, X_train_c, y_train_c, X_test_c, y_test_c, 'Random␣
 ↪Forest Regression - Cooling Load')
```

```
Random Forest Regression - Heating Load - Training set:
RMSE=0.33135195422392366, MAE=0.22162715137762076, R²=0.9989121099740298
Random Forest Regression - Heating Load - Testing set: RMSE=0.517465811534824,
MAE=0.3551063296515583, R²=0.9974310210427606

Random Forest Regression - Cooling Load - Training set: RMSE=1.504600435069373,
MAE=0.933010918224424, R²=0.9747507607457195
Random Forest Regression - Cooling Load - Testing set: RMSE=1.9702018406932347,
MAE=1.2541592449671717, R²=0.9581069388129628
```

Model Performance on Heating Load: - Random Forest Regression shows exceptionally high performance with both the lowest RMSE and MAE and the highest R^2 score on both training and testing sets. It demonstrates an almost perfect fit on the training set and maintains a high level of accuracy on the testing set, indicating strong generalization capabilities. - Linear Regression, Ridge, Lasso, and Elastic Net show similar performance metrics, with slight variations. Among these, Ridge and Lasso provide marginally better results on the testing set than Linear Regression and Elastic Net, suggesting a slight advantage of regularization. However, the differences are minimal, indicating that all these linear models have captured the underlying relationship in the data similarly. - The R^2 scores for linear models are notably high, exceeding 0.92 on the testing set, which signifies a strong predictive capability, although not as robust as Random Forest.

Model Performance on Cooling Load: - As with the Heating Load, Random Forest Regression outperforms the linear models by a significant margin on the Cooling Load, with much lower RMSE and MAE values and a higher R^2 score. This indicates that the relationship between features and the Cooling Load is more complex and nonlinear, which Random Forest is better equipped to model. - Among the linear models, the performance is again quite similar across Linear Regression, Ridge, Lasso, and Elastic Net. The regularization in Ridge, Lasso, and Elastic Net does not significantly outperform the basic Linear Regression model, as indicated by the very close RMSE, MAE, and R^2 scores on both the training and testing sets. - The R^2 scores for linear models on the Cooling Load are slightly lower than those for the Heating Load, indicating that these models find it slightly more challenging to predict the Cooling Load accurately. However, the scores are still high, suggesting good predictive capability.

Final Thoughts and Recommendations:

Random Forest clearly demonstrates the highest effectiveness for both targets, with significantly better performance metrics. This suggests that the features' relationship with both Heating Load and Cooling Load has nonlinear characteristics that Random Forest can capture more effectively than linear models. The linear models perform similarly across both targets, with slight variations that might not be practically significant. This indicates that for linear relationships or when a simpler model is required due to interpretability or computational efficiency, any of these models could be a reasonable choice. The generalization capability of Random Forest is particularly noteworthy, as evidenced by the minimal drop in R^2 from training to testing sets, especially for Heating Load.

For applications prioritizing accuracy and capable of handling more complex models, Random Forest Regression is the recommended choice for both Heating and Cooling Loads. In scenarios where model simplicity, interpretability, or computational efficiency is crucial, the choice among linear models might depend on slight preferences for regularization or ease of implementation, as their performance is closely matched. Ridge or Lasso could be slightly preferred due to their inherent regularization, which might offer some advantage in terms of model stability and prevention of overfitting.