MODELING OF STRENGTH OF HIGH-PERFORMANCE CONCRETE USING *ARTIFICIAL NEURAL NETWORKS*

[1]:
```
# import Data Manipulation library
import pandas as pd
import numpy as np

#Import Data visualizatio library
import matplotlib.pyplot as plt
import seaborn as sns

#Import filter warning library
import warnings
warnings.filterwarnings('ignore')

#Import scikit Learn library
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler,␣
 ↪PowerTransformer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline, FunctionTransformer

import scipy.stats as stats

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn import set_config
set_config(display='diagram')

# import Deep Learning Library
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Dropout
# import keras_tuner as kt
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import load_model
```

2 Concrete Compressive Strength Dataset

2.1 Overview

- **Source**: UCI Machine Learning Repository
- **Dataset Type**: Regression
- **Number of Instances**: 1030
- **Number of Attributes**: 9 input variables + 1 target variable
- **Domain**: Civil Engineering, Material Science
- **Objective**: Predict the compressive strength of concrete based on its composition and age.

2.2 Attribute Information

Column	Description	Unit
Cement	Cement component	kg/m^3
Blast Furnace Slag	Blast furnace slag component	kg/m^3
Fly Ash	Fly ash component	kg/m^3
Water	Water component	kg/m^3
Superplasticizer	Superplasticizer additive	kg/m^3
Coarse Aggregate	Coarse aggregate component	kg/m^3
Fine Aggregate	Fine aggregate component	kg/m^3
Age	Age of concrete samples	Days
Compressive Strength	Target variable - Strength of concrete	MPa

2.3 Dataset Characteristics

- The dataset contains **continuous numerical features**.
- The target variable (compressive strength) is influenced by **composition and age**.
- No categorical features.

2.4 Applications

- Predicting the strength of concrete for **construction quality control**.
- Understanding the impact of different components on **material durability**.
- Optimizing concrete composition for **stronger and cost-effective construction**.

2.5 Source & Citation

Yeh, I-C. "Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks." *Cement and Concrete Research*, Vol. 28, No. 12, pp. 1797-1808, 1998.

---

[2]:
```
# Import Dataset Using Pandas Function

url = 'https://raw.githubusercontent.com/jadhavgaurav/
↪cement-composite-strength-prediction/refs/heads/main/concrete_data.csv'

df = pd.read_csv(url)
```

```
df.sample(frac = 1) # Shuffle Dataset
```

[2]:
```
     cement  blast_furnace_slag  fly_ash  water  superplasticizer  \
308   277.1                 0.0     97.4  160.6              11.8
595   186.2               124.1      0.0  185.7               0.0
248   238.1                 0.0     94.1  186.7               7.0
137   362.6               189.0      0.0  164.9              11.6
585   290.2               193.5      0.0  185.7               0.0
..      …                   …       …      …                 …
222   166.1                 0.0    163.3  176.5               4.5
673   212.0               141.3      0.0  203.5               0.0
636   300.0                 0.0      0.0  184.0               0.0
96    425.0               106.3      0.0  151.4              18.6
680   102.0               153.0      0.0  192.0               0.0

     coarse_aggregate  fine_aggregate   age  concrete_compressive_strength
308             973.9           875.6   100                          55.64
595            1083.4           764.3    28                          17.60
248             949.9           847.0   100                          44.30
137             944.7           755.8    28                          71.30
585             998.2           704.3    28                          33.04
..                …               …   …                            …
222            1058.6           780.1    56                          28.63
673             973.4           750.0     7                          15.03
636            1075.0           795.0    28                          26.85
96              936.0           803.7     7                          46.80
680             887.0           942.0    28                          17.28

[1030 rows x 9 columns]
```

[3]:
```
df.columns
```

[3]:
```
Index(['cement', 'blast_furnace_slag', 'fly_ash', 'water', 'superplasticizer',
       'coarse_aggregate', 'fine_aggregate ', 'age',
       'concrete_compressive_strength'],
      dtype='object')
```

[4]:
```
df.columns = df.columns.str.strip()
```

[5]:
```
# Checking Data information and Missing Values if any...

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
```

```
 0   cement                         1030 non-null   float64
 1   blast_furnace_slag             1030 non-null   float64
 2   fly_ash                        1030 non-null   float64
 3   water                          1030 non-null   float64
 4   superplasticizer               1030 non-null   float64
 5   coarse_aggregate               1030 non-null   float64
 6   fine_aggregate                 1030 non-null   float64
 7   age                            1030 non-null   int64
 8   concrete_compressive_strength  1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.6 KB
```

There are 8 Features and 1 Target Column. No null values found in the data

[6]:
```
# Checking Descriptive Stattistics
df.describe()
```

[6]:
```
            cement  blast_furnace_slag      fly_ash        water  \
count  1030.000000         1030.000000  1030.000000  1030.000000
mean    281.167864           73.895825    54.188350   181.567282
std     104.506364           86.279342    63.997004    21.354219
min     102.000000            0.000000     0.000000   121.800000
25%     192.375000            0.000000     0.000000   164.900000
50%     272.900000           22.000000     0.000000   185.000000
75%     350.000000          142.950000   118.300000   192.000000
max     540.000000          359.400000   200.100000   247.000000

       superplasticizer  coarse_aggregate  fine_aggregate          age  \
count       1030.000000       1030.000000     1030.000000  1030.000000
mean           6.204660        972.918932      773.580485    45.662136
std            5.973841         77.753954       80.175980    63.169912
min            0.000000        801.000000      594.000000     1.000000
25%            0.000000        932.000000      730.950000     7.000000
50%            6.400000        968.000000      779.500000    28.000000
75%           10.200000       1029.400000      824.000000    56.000000
max           32.200000       1145.000000      992.600000   365.000000

       concrete_compressive_strength
count                    1030.000000
mean                       35.817961
std                        16.705742
min                         2.330000
25%                        23.710000
50%                        34.445000
75%                        46.135000
max                        82.600000
```

[7]:
```python
# Univariate Analysis (Custom Function)

from collections import OrderedDict

stats = []
for i in df.columns:
    numerical_stats = OrderedDict({
        'Feature': i,
        'Maximum' : df[i].max(),
        'Minimum' : df[i].min(),
        'Mean' : df[i].mean(),
        'Median' : df[i].median(),
        '25%': df[i].quantile(0.25),
        '75%': df[i].quantile(0.75),
        'Standard Deviation': df[i].std(),
        'Variance': df[i].var(),
        'Skewness': df[i].skew(),
        'Kurtosis': df[i].kurt(),
        'IQR' : df[i].quantile(0.75) - df[i].quantile(0.25)
    })
    stats.append(numerical_stats)
report = pd.DataFrame(stats)
report
```

[7]:
```
                         Feature  Maximum  Minimum        Mean   Median  \
0                         cement    540.0   102.00  281.167864  272.900
1             blast_furnace_slag    359.4     0.00   73.895825   22.000
2                        fly_ash    200.1     0.00   54.188350    0.000
3                          water    247.0   121.80  181.567282  185.000
4               superplasticizer     32.2     0.00    6.204660    6.400
5               coarse_aggregate   1145.0   801.00  972.918932  968.000
6                 fine_aggregate    992.6   594.00  773.580485  779.500
7                            age    365.0     1.00   45.662136   28.000
8  concrete_compressive_strength     82.6     2.33   35.817961   34.445

       25%       75%  Standard Deviation     Variance  Skewness   Kurtosis  \
0  192.375   350.000          104.506364  10921.580220  0.509481  -0.520652
1    0.000   142.950           86.279342   7444.124812  0.800717  -0.508175
2    0.000   118.300           63.997004   4095.616541  0.537354  -1.328746
3  164.900   192.000           21.354219    456.002651  0.074628   0.122082
4    0.000    10.200            5.973841     35.686781  0.907203   1.411269
5  932.000  1029.400           77.753954   6045.677357 -0.040220  -0.599016
6  730.950   824.000           80.175980   6428.187792 -0.253010  -0.102177
7    7.000    56.000           63.169912   3990.437729  3.269177  12.168989
8   23.710    46.135           16.705742    279.081814  0.416977  -0.313725

       IQR
```

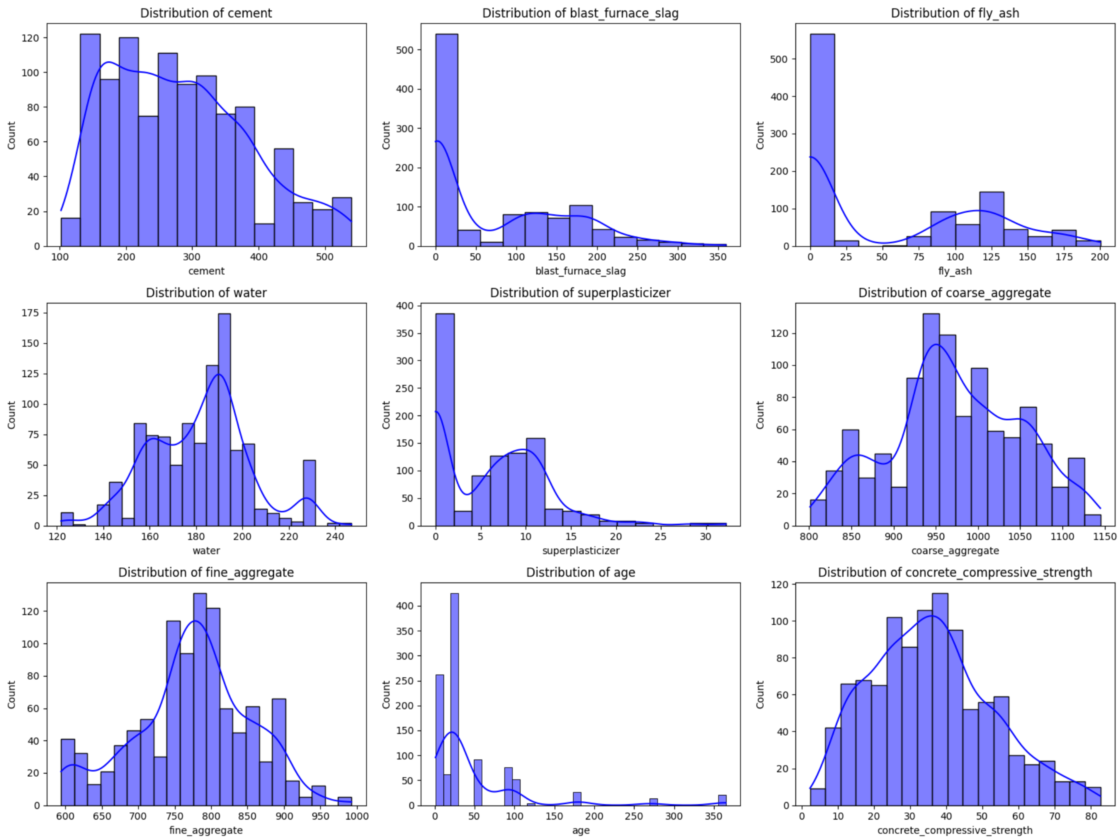
```
0    157.625
1    142.950
2    118.300
3     27.100
4     10.200
5     97.400
6     93.050
7     49.000
8     22.425
```

2.6 Based on above information, we find that the dataset is non normal distributed

2.7 Univariate Analysis

2.7.1 1. Histograms & KDE Plots

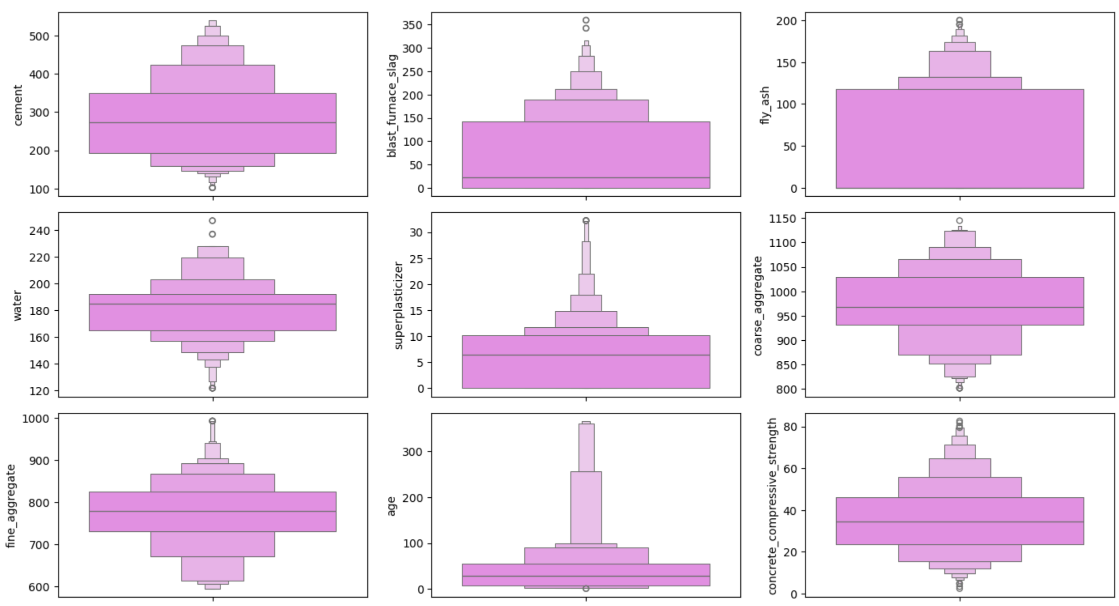
[8]:
```python
plt.figure(figsize=(16, 12))
for i, col in enumerate(df.columns):
    plt.subplot(3, 3, i+1)
    sns.histplot(df[col], kde=True, color='blue')
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()
```



- Cement, water, and aggregates follow near-normal distributions, meaning these - materials have consistent usage across different mixes.
- Blast furnace slag, fly ash, and superplasticizer are highly skewed, indicating - they are often absent or used in small amounts.
- Age distribution suggests that most concrete samples are tested at an early - stage, while long-term strength is measured less frequently.
- The strength distribution shows that most concrete samples achieve 30-50 MPa - compressive strength, which is a standard range for construction applications.

2.7.2 2. Box Plots for Outlier Detection

[9]:
```
# Boxplot for detecting outliers in the dataset
plt.figure(figsize=(14, 10))
plot = 0
for i in df.columns:
    plot += 1
    plt.subplot(4, 3, plot)
    sns.boxenplot(df[i], color='violet')
    plt.tight_layout()
plt.show()
```



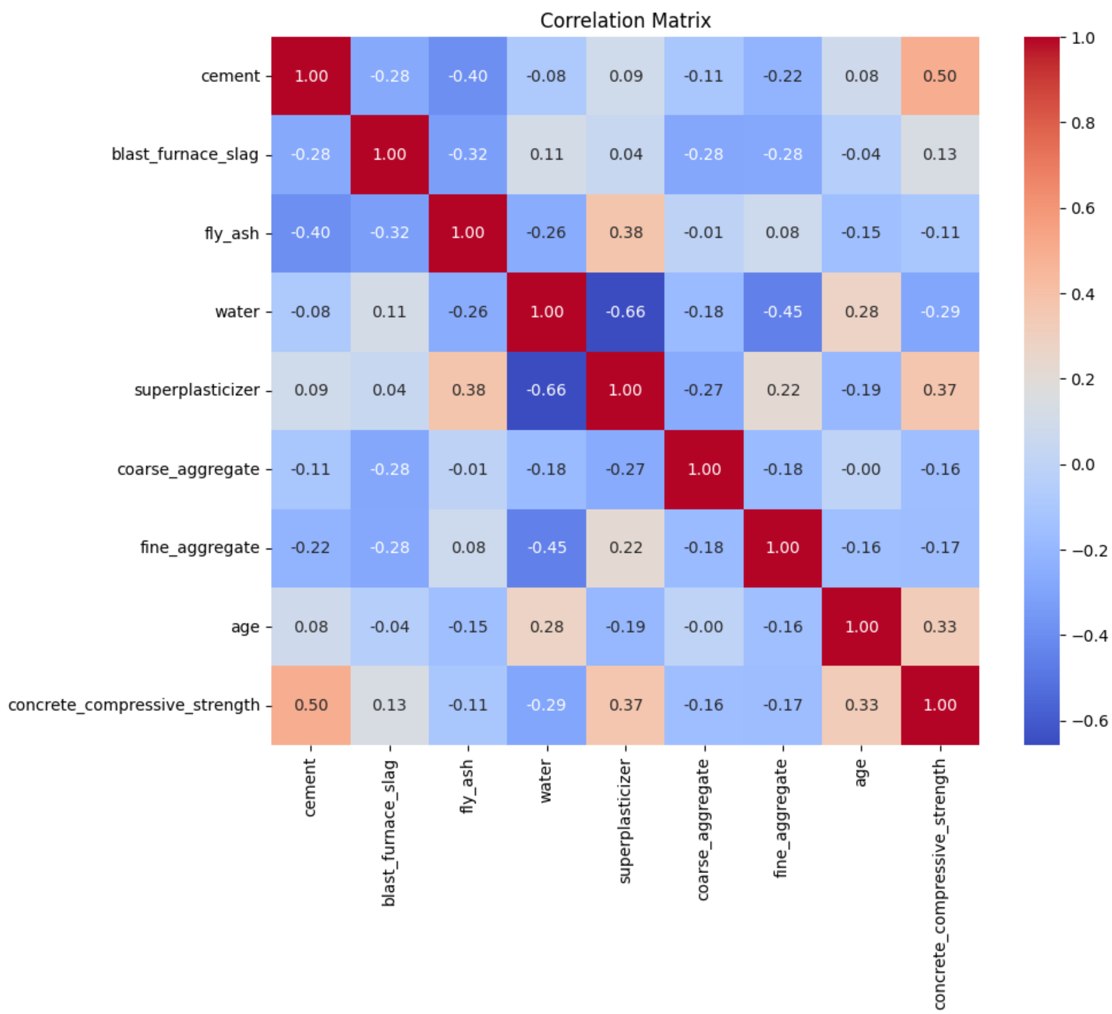
3 Bivariate Analysis

3.0.1 4.1. Correlation Matrix

[10]:
```
df.corr()['concrete_compressive_strength']
```

[10]:
```
cement                           0.497832
blast_furnace_slag               0.134829
fly_ash                         -0.105755
water                           -0.289633
superplasticizer                 0.366079
coarse_aggregate                -0.164935
fine_aggregate                  -0.167241
age                              0.328873
concrete_compressive_strength    1.000000
Name: concrete_compressive_strength, dtype: float64
```

[11]:
```
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title("Correlation Matrix")
plt.show()
```



1 Cement is the most critical factor (+0.50 correlation) – Higher cement content significantly increases concrete compressive strength.

2 Superplasticizer improves strength (+0.37 correlation) – It enhances workability while reducing water, leading to stronger concrete.

3 Age matters (+0.33 correlation) – As concrete cures over time, its strength increases.

4 Water negatively affects strength (-0.29 correlation) – Excess water weakens concrete, reducing durability.

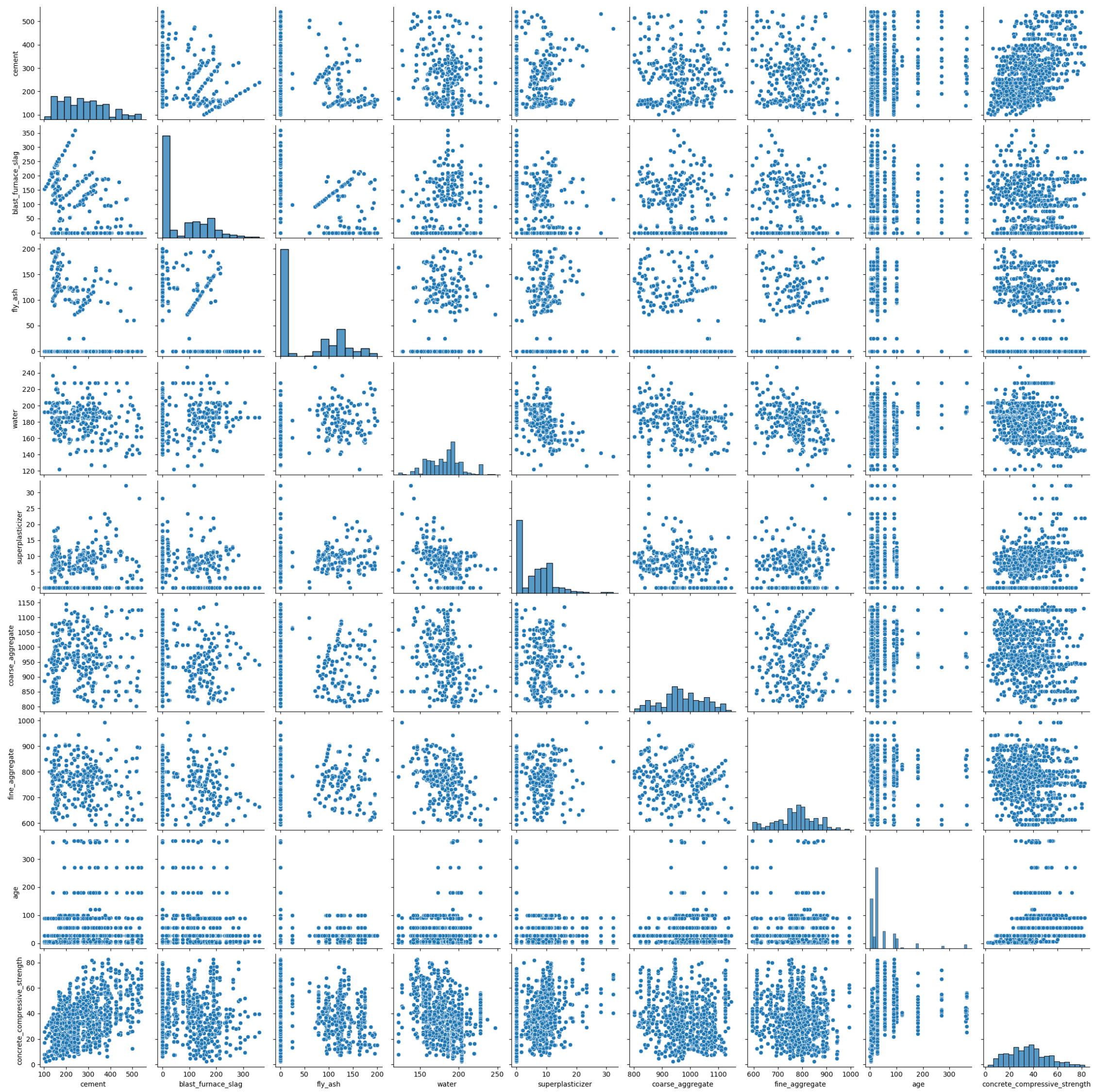
5 Aggregates (fine & coarse) have minimal direct impact (~-0.16 correlation) – While essential for structure, their contribution to compressive strength is not significant.

6 Fly ash and blast furnace slag have weak correlations (~0.13 to -0.11) – These materials don't drastically impact strength but may influence durability and workability.

7 Water and superplasticizer are strongly negatively correlated (-0.66) – More superplasticizer means less water is needed, leading to better strength.

3.0.2 2. Pair Plot

[12]:
```python
sns.pairplot(df, diag_kind='auto')
plt.show()
```



1 **Strong Positive Correlation:** Cement vs. Compressive Strength → As cement content increases, strength significantly improves. Age vs. Compressive Strength → Older concrete has higher strength due to curing effects.

2 **Negative Correlation:** Water vs. Compressive Strength → Higher water content weakens concrete, leading to lower strength. Water vs. Superplasticizer → More superplasticizer reduces water requirement, improving workability.

3 **Weak or No Significant Trends:** Coarse & Fine Aggregate vs. Compressive Strength → No clear impact; aggregates provide structure but don't directly boost strength. Fly Ash & Blast

Furnace Slag vs. Strength → Some contribution but not a strong determinant.

4 **Distribution Insights:** Some variables (like water and superplasticizer) have skewed distributions, indicating possible outliers or non-uniform data distribution. Compressive strength has a right-skewed distribution, meaning most values are lower, with fewer high-strength samples.

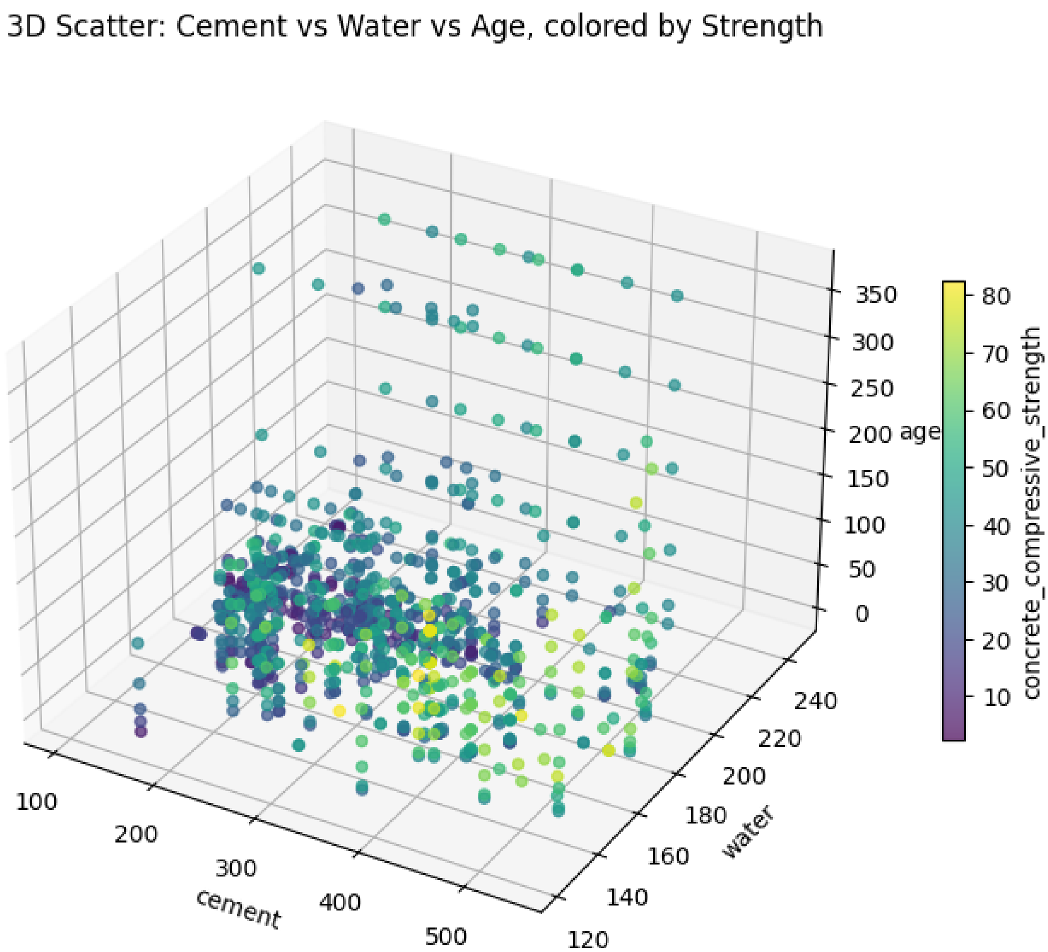
- Maximize cement & curing time for better strength.
- Reduce water & optimize superplasticizer for better workability and durability.
- Reevaluate the role of aggregates in mix design.

3.1 Multivariate Analysis / Deeper Outlier Inspection

3.1.1 1. 3D AXES PLOT Using a combination of features

3.1.2 For instance, we could see if certain features combined

[13]:
```
from mpl_toolkits.mplot3d import Axes3D
target_col = 'concrete_compressive_strength'

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Choose three features for demonstration
x_feat = 'cement'
y_feat = 'water'
z_feat = 'age'

scatter = ax.scatter(df[x_feat],
                     df[y_feat],
                     df[z_feat],
                     c=df[target_col],
                     cmap='viridis',
                     alpha=0.7)

ax.set_xlabel(x_feat)
ax.set_ylabel(y_feat)
ax.set_zlabel(z_feat)
cbar = fig.colorbar(scatter, ax=ax, shrink=0.5)
cbar.set_label(target_col)
plt.title("3D Scatter: Cement vs Water vs Age, colored by Strength")
plt.show()
```



1 Higher Cement = Higher Strength → More cement content generally results in stronger concrete. 2 Higher Age = Higher Strength → Older concrete (higher curing time) shows greater compressive strength. 3 Higher Water = Lower Strength → Excess water reduces strength, confirming the water-to-cement ratio's impact.

Conclusions for Optimization:

- Increase cement while keeping water in check for higher strength.
- Allow longer curing time to enhance strength.
- Balance water-to-cement ratio to prevent strength reduction.

[14]:
```
df.columns
```

[14]:
```
Index(['cement', 'blast_furnace_slag', 'fly_ash', 'water', 'superplasticizer',
       'coarse_aggregate', 'fine_aggregate', 'age',
       'concrete_compressive_strength'],
      dtype='object')
```

4 Feature Engineering

[15]:
```
# 1. Water-to-Cement Ratio (w/c Ratio)
# The ratio of water to cement significantly influences concrete strength.

df['water_cement_ratio'] = df['water'] / df['cement']
```

[ ]:
```
# 2. Concrete gains strength over time.
# This ratio helps normalize strength based on curing duration

df['strength_age_ratio'] = df['concrete_compressive_strength'] / df['age']
```

[17]:
```
new_features = ['water_cement_ratio', 'strength_age_ratio']
```

4.1 Variance Inflation Factor

[ ]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm  # Importing statsmodels

X = df.drop(columns='concrete_compressive_strength')  # Selecting only␣
↪independent variables

# Add a small constant to avoid division errors
X = X + 1e-10

# Compute VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
↪shape[1])]

# Display VIF values
print(vif_data)
```

```
              Feature         VIF
0              cement   48.913538
1  blast_furnace_slag    3.806776
2             fly_ash    4.239282
3               water  142.026838
4    superplasticizer    5.474348
5    coarse_aggregate   85.722070
6      fine_aggregate   79.466419
7                 age    2.044657
8  water_cement_ratio   45.564599
9  strength_age_ratio    2.700167
```

4.1.1 We will drop the features with High VIF(variance-inflation-factor) as they impact model performance

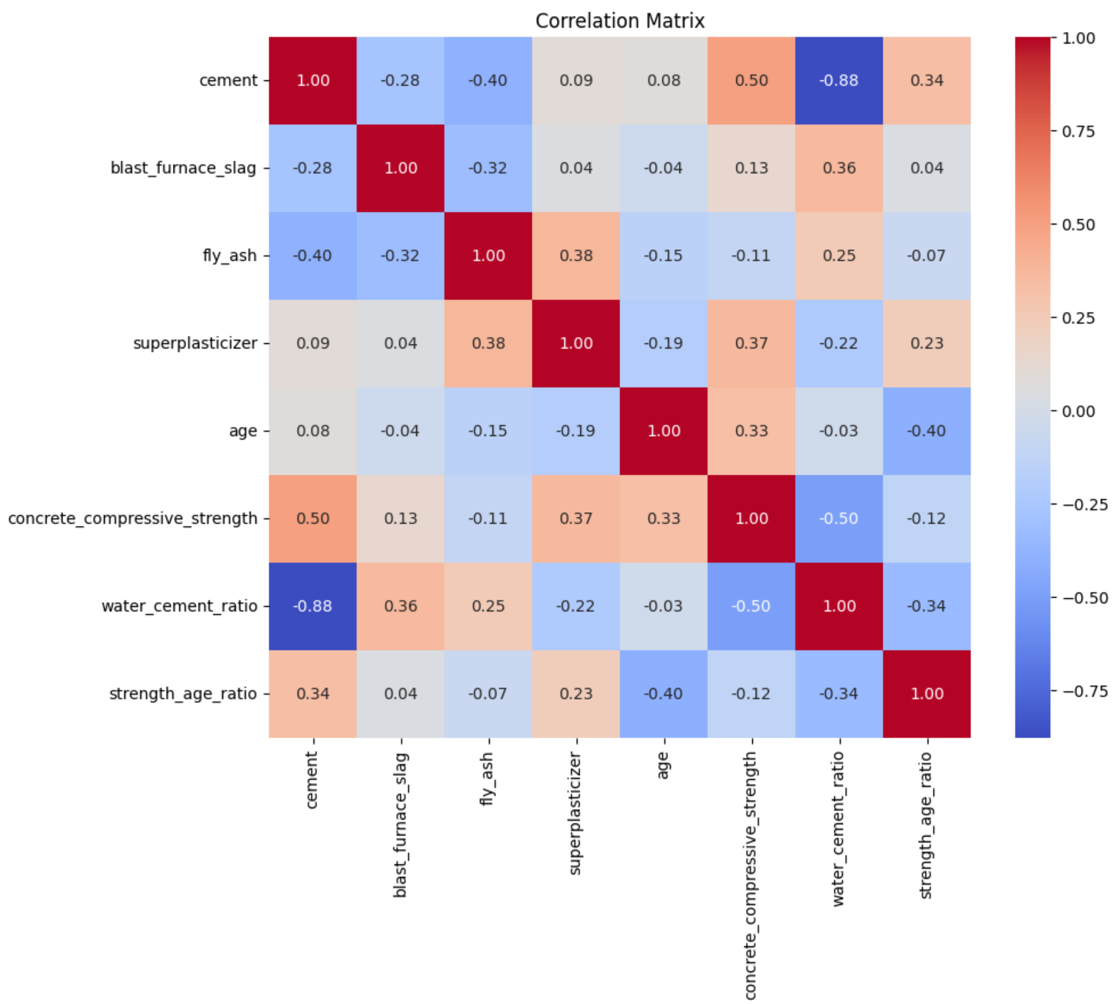
[19]:
```python
df.drop(columns=['water', 'coarse_aggregate', 'fine_aggregate'], inplace=True)
```

[ ]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm  # Importing statsmodels

X = df.drop(columns='concrete_compressive_strength')  # Selecting only␣
↪independent variables

# Add a small constant to avoid division errors
X = X + 1e-10

# Compute VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
↪shape[1])]

# Display VIF values
print(vif_data)
```

```
              Feature       VIF
0              cement  5.485113
1  blast_furnace_slag  3.190417
2             fly_ash  3.723656
3    superplasticizer  3.376431
4                 age  1.975182
5  water_cement_ratio  6.661283
6  strength_age_ratio  2.661419
```

VIF for all the features is less than 10, So the features will lead to good model performance

[21]:
```python
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title("Correlation Matrix")
plt.show()
```



Correlation Analysis: - **Strong Positive Correlations:**

- Cement vs. Concrete Compressive Strength (0.50): More cement leads to stronger concrete.
- Superplasticizer vs. Concrete Compressive Strength (0.37): Superplasticizers improve concrete strength.
- Age vs. Concrete Compressive Strength (0.33): Strength increases with curing time.
- **Strong Negative Correlations:**
- Water-Cement Ratio vs. Concrete Compressive Strength (-0.50): More water weakens the concrete.
- Water-Cement Ratio vs. Cement (-0.88): Higher cement reduces the water-cement ratio, indicating an inverse relationship.
- Strength-Age Ratio vs. Age (-0.40): As concrete ages, the ratio of strength gain per unit time decreases.

5 Handling Outliers

[ ]:
```python
# Outlier Detection using IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Detecting outliers
outliers = ((df < lower_bound) | (df > upper_bound)).sum()

# Display number of outliers per feature
print("Number of outliers per feature:\n", outliers)
```

```
Number of outliers per feature:
 cement                            0
blast_furnace_slag               2
fly_ash                          0
superplasticizer                10
age                             59
concrete_compressive_strength    4
water_cement_ratio              18
strength_age_ratio             129
dtype: int64
```

[ ]:
```python
def remove_outliers_iqr(df, columns):
    """
    Removes outliers based on IQR method for specified columns.

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        columns (list): List of column names to check for outliers.

    Returns:
        pd.DataFrame: DataFrame with outliers removed.
    """
    df_clean = df.copy()
    for col in columns:
        Q1 = df[col].quantile(0.25)  # First Quartile (25th percentile)
        Q3 = df[col].quantile(0.75)  # Third Quartile (75th percentile)
        IQR = Q3 - Q1                # Interquartile Range
        lower_bound = Q1 - 1.5 * IQR  # Lower bound
        upper_bound = Q3 + 1.5 * IQR  # Upper bound

        # Filter out outliers
```

```
        df_clean = df_clean[(df_clean[col] >= lower_bound) & (df_clean[col] <=␣
↪upper_bound)]

    return df_clean

# Remove outliers from specific columns
columns_to_check = ['strength_age_ratio', 'age', 'water_cement_ratio']  #␣
↪Replace with actual column names , o - 'water_cement_ratio'
df_cleaned = remove_outliers_iqr(df, columns_to_check)

# Display number of rows before and after removing outliers
print(f"Original dataset size: {df.shape[0]}")
print(f"Cleaned dataset size: {df_cleaned.shape[0]}")
```

```
Original dataset size: 1030
Cleaned dataset size: 824
```

[24]:
```
# Checking Column Names
df_cleaned.columns
```

[24]:
```
Index(['cement', 'blast_furnace_slag', 'fly_ash', 'superplasticizer', 'age',
       'concrete_compressive_strength', 'water_cement_ratio',
       'strength_age_ratio'],
      dtype='object')
```

[25]:
```
# Split Data into X and y
X = df_cleaned.drop(columns = ['concrete_compressive_strength']) # Independent␣
↪Features
y = df_cleaned['concrete_compressive_strength'] # target Variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

X_train.shape, X.shape, y_train.shape, y.shape
```

[25]:
```
((659, 7), (824, 7), (659,), (824,))
```

[26]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[27]:
```
X_train.max(), X_train.min()
```

[27]:
```
(4.587450263729816, -1.7508681903823078)
```

[ ]:
```
# import RandomForest
from sklearn.ensemble import RandomForestRegressor
```

```
model_rf = RandomForestRegressor()
model_rf.fit(X_train, y_train)
```

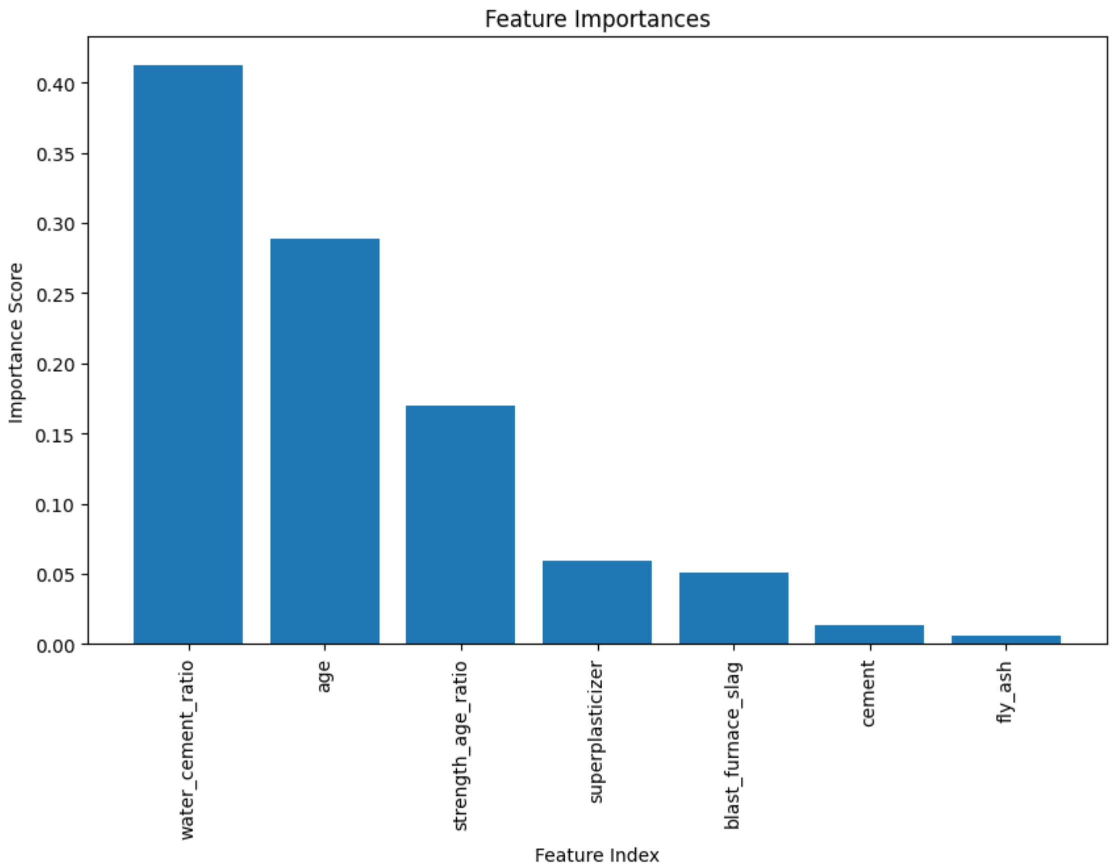
[ ]: RandomForestRegressor()

[29]:
```
kf = 5

# Cross-validation scores on Training Data
train_r2_score = np.mean(cross_val_score(model_rf, X_train, y_train, cv=kf,␣
  ↪scoring='r2'))
train_mae  = np.mean(cross_val_score(model_rf, X_train, y_train, cv=kf,␣
  ↪scoring='neg_mean_absolute_error')) * -1
train_mse = np.mean(cross_val_score(model_rf, X_train, y_train, cv=kf,␣
  ↪scoring='neg_mean_squared_error')) * -1
train_rmse = np.sqrt(train_mse)

# Evaluate on Test Data
y_pred_test = model_rf.predict(X_test)
test_r2_score = r2_score(y_test, y_pred_test)
test_mae = mean_absolute_error(y_test, y_pred_test)
test_mse = mean_squared_error(y_test, y_pred_test)
test_rmse = np.sqrt(test_mse)

# Print Evaluation Metrics
print('Evaluation for Random Forest:')
print('Train R2 Score  :', round(train_r2_score, 3))
print('Test R2 Score   :', round(test_r2_score, 3))
print('Train MAE       :', round(train_mae, 3))
print('Test MAE        :', round(test_mae, 3))
print('Train MSE       :', round(train_mse, 3))
print('Test MSE        :', round(test_mse, 3))
print('Train RMSE      :', round(train_rmse, 3))
print('Test RMSE       :', round(test_rmse, 3))
```

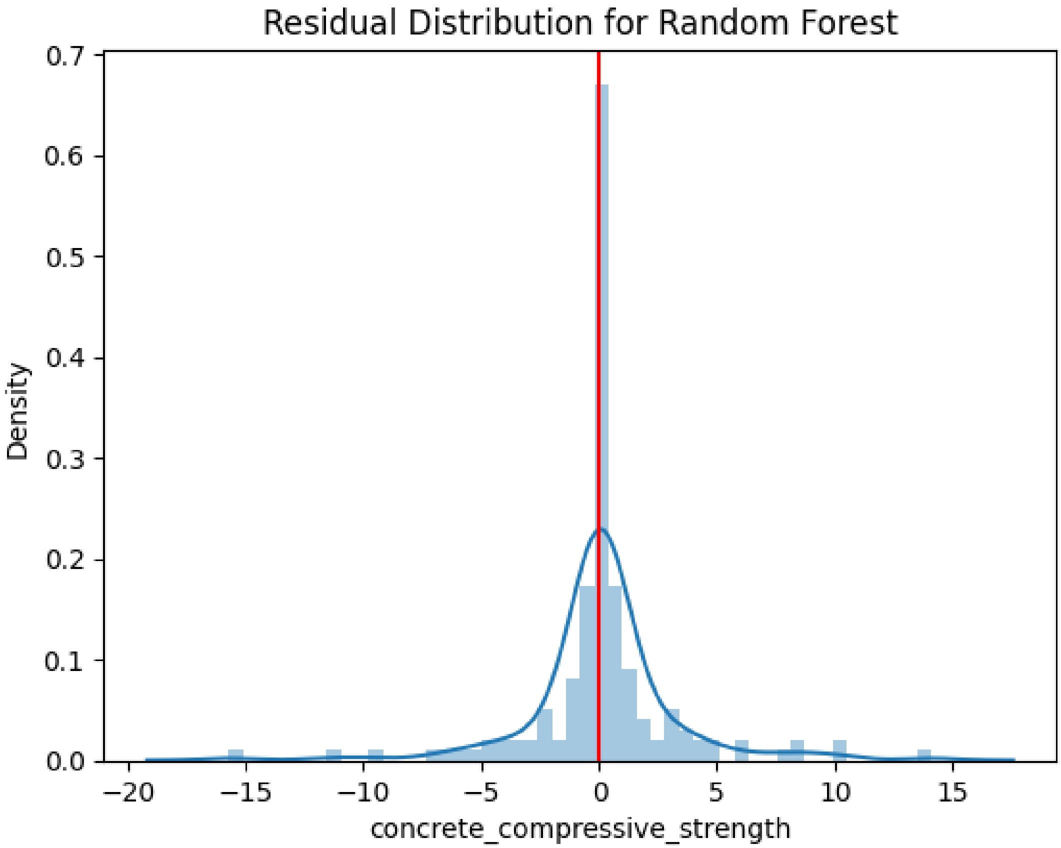
```
Evaluation for Random Forest:
Train R2 Score  : 0.961
Test R2 Score   : 0.967
Train MAE       : 1.694
Test MAE        : 1.7
Train MSE       : 12.05
Test MSE        : 10.207
Train RMSE      : 3.471
Test RMSE       : 3.195
```

[30]:
```
# Get feature importances
importances = model_rf.feature_importances_
indices = np.argsort(importances)[::-1]
```

```python
# Plot feature importances
plt.figure(figsize=(10,6))
plt.title("Feature Importances")
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), X.columns[indices], rotation=90)
plt.xlabel("Feature Index")
plt.ylabel("Importance Score")
plt.show()
```

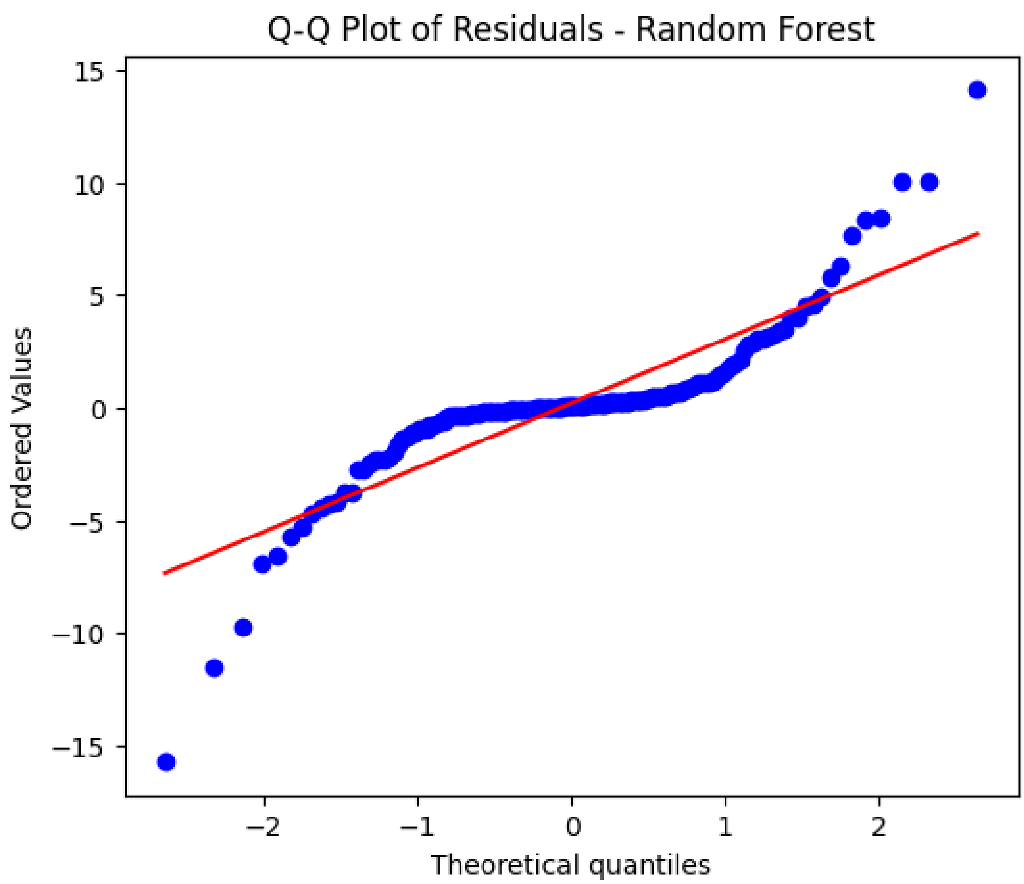


[31]:
```python
residuals = y_test - y_pred_test
sns.distplot(residuals)
plt.axvline(0,color = 'red')
plt.title('Residual Distribution for Random Forest')
plt.show()
```



[32]:

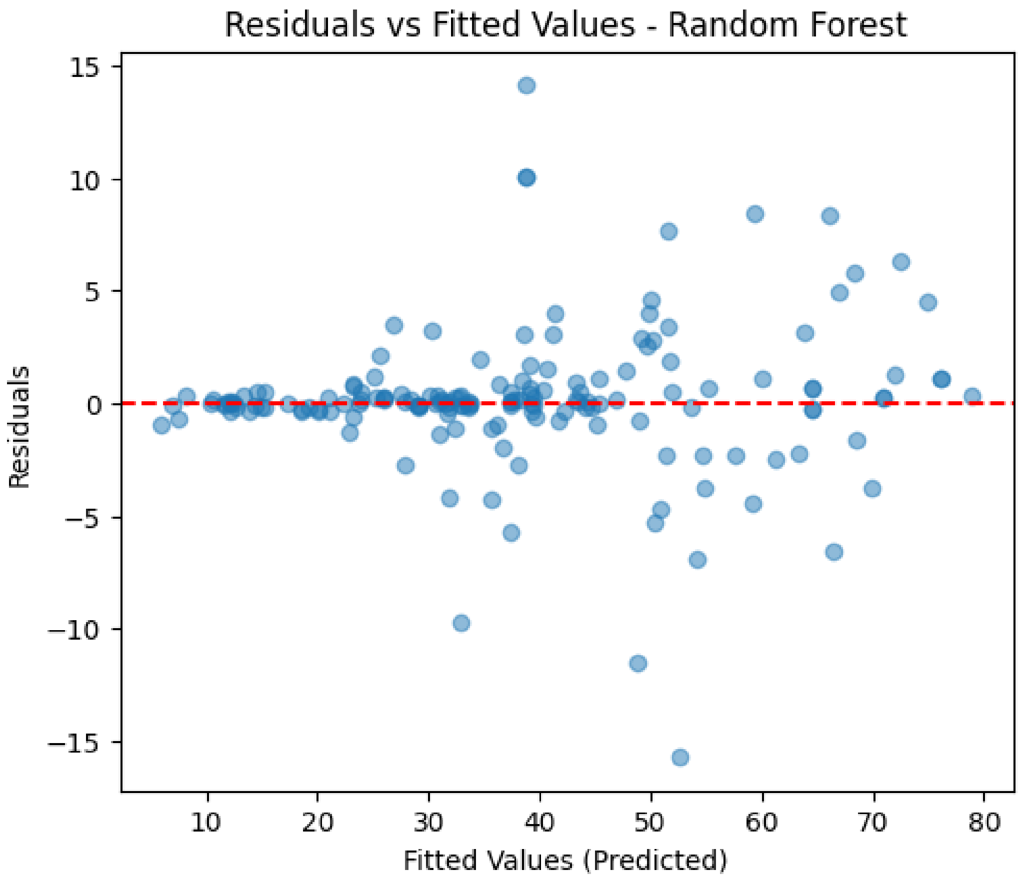
```
import scipy.stats as stats

# Q-Q Plot for Normality Check
plt.figure(figsize=(6,5))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals - Random Forest")
plt.show()
```



[33]:
```
y_pred_test.shape, residuals.shape
```

[33]: ((165,), (165,))

[34]:
```
# Residuals vs. Fitted Plot
plt.figure(figsize=(6,5))
plt.scatter(y_pred_test, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Fitted Values (Predicted)")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values - Random Forest")
plt.show()
```



[34]:

```

```

6 Artifical Neural Network Model training

[35]:

```
# ANN Model
model_ann = keras.Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),  # Prevents overfitting
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.3),
    # Dense(16, activation='relu'),
    # Dropout(0.2),
    Dense(1)  # Output Layer
])

# Compile the model
```

```python
model_ann.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the Model
history = model_ann.fit(X_train, y_train, epochs=200, batch_size=16,␣
 ↪validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/200
42/42                 3s 8ms/step - loss:
1649.6469 - mae: 36.2625 - val_loss: 1203.4313 - val_mae: 29.9895
Epoch 2/200
42/42                 0s 5ms/step - loss:
787.5009 - mae: 22.9982 - val_loss: 170.4703 - val_mae: 10.4065
Epoch 3/200
42/42                 0s 5ms/step - loss:
213.1496 - mae: 11.5776 - val_loss: 130.9749 - val_mae: 8.7979
Epoch 4/200
42/42                 0s 4ms/step - loss:
208.0865 - mae: 11.0986 - val_loss: 108.8363 - val_mae: 8.0259
Epoch 5/200
42/42                 0s 5ms/step - loss:
181.3646 - mae: 10.3372 - val_loss: 106.8592 - val_mae: 7.8952
Epoch 6/200
42/42                 0s 6ms/step - loss:
185.1751 - mae: 10.4054 - val_loss: 89.1844 - val_mae: 7.2653
Epoch 7/200
42/42                 1s 6ms/step - loss:
147.4144 - mae: 9.4222 - val_loss: 77.8729 - val_mae: 6.7187
Epoch 8/200
42/42                 0s 6ms/step - loss:
164.0083 - mae: 9.9297 - val_loss: 80.2472 - val_mae: 6.8788
Epoch 9/200
42/42                 0s 7ms/step - loss:
129.9151 - mae: 8.8536 - val_loss: 70.1980 - val_mae: 6.2938
Epoch 10/200
42/42                 0s 6ms/step - loss:
133.6063 - mae: 8.8434 - val_loss: 74.6259 - val_mae: 6.3777
Epoch 11/200
42/42                 1s 5ms/step - loss:
158.0594 - mae: 9.4625 - val_loss: 60.4006 - val_mae: 5.9369
Epoch 12/200
42/42                 0s 5ms/step - loss:
139.8902 - mae: 9.1418 - val_loss: 61.7309 - val_mae: 5.8299
Epoch 13/200
42/42                 0s 5ms/step - loss:
119.5192 - mae: 8.3840 - val_loss: 56.7572 - val_mae: 5.6685
Epoch 14/200
42/42                 0s 4ms/step - loss:
127.2956 - mae: 8.3707 - val_loss: 68.3145 - val_mae: 5.9780
```

```
Epoch 15/200
42/42            0s 5ms/step - loss:
119.1622 - mae: 7.9655 - val_loss: 63.0620 - val_mae: 5.7990
Epoch 16/200
42/42            0s 4ms/step - loss:
118.9127 - mae: 8.3521 - val_loss: 60.4515 - val_mae: 5.6835
Epoch 17/200
42/42            0s 4ms/step - loss:
123.0084 - mae: 8.6982 - val_loss: 53.4377 - val_mae: 5.3284
Epoch 18/200
42/42            0s 4ms/step - loss:
120.5001 - mae: 8.2872 - val_loss: 59.3571 - val_mae: 5.4623
Epoch 19/200
42/42            0s 4ms/step - loss:
94.8632 - mae: 7.3079 - val_loss: 51.6695 - val_mae: 5.1889
Epoch 20/200
42/42            0s 4ms/step - loss:
119.8536 - mae: 8.3453 - val_loss: 44.8041 - val_mae: 4.9086
Epoch 21/200
42/42            0s 5ms/step - loss:
121.2234 - mae: 8.0474 - val_loss: 44.2157 - val_mae: 4.8590
Epoch 22/200
42/42            0s 4ms/step - loss:
121.7825 - mae: 8.2419 - val_loss: 47.2915 - val_mae: 4.9222
Epoch 23/200
42/42            0s 4ms/step - loss:
121.3078 - mae: 8.0962 - val_loss: 47.5813 - val_mae: 4.9658
Epoch 24/200
42/42            0s 5ms/step - loss:
118.2699 - mae: 8.2481 - val_loss: 47.4033 - val_mae: 4.9906
Epoch 25/200
42/42            0s 4ms/step - loss:
96.2204 - mae: 7.4086 - val_loss: 46.4587 - val_mae: 4.8359
Epoch 26/200
42/42            0s 4ms/step - loss:
90.9818 - mae: 7.3093 - val_loss: 39.6873 - val_mae: 4.6394
Epoch 27/200
42/42            0s 4ms/step - loss:
111.9650 - mae: 7.9736 - val_loss: 37.2914 - val_mae: 4.5612
Epoch 28/200
42/42            0s 4ms/step - loss:
119.2628 - mae: 8.1673 - val_loss: 39.8498 - val_mae: 4.5973
Epoch 29/200
42/42            0s 4ms/step - loss:
101.7653 - mae: 7.4749 - val_loss: 38.4178 - val_mae: 4.5910
Epoch 30/200
42/42            0s 4ms/step - loss:
122.3588 - mae: 8.2436 - val_loss: 37.4348 - val_mae: 4.4529
```

```
Epoch 31/200
42/42            0s 4ms/step - loss:
99.7867 - mae: 7.6053 - val_loss: 36.1217 - val_mae: 4.3283
Epoch 32/200
42/42            0s 4ms/step - loss:
86.4446 - mae: 7.2263 - val_loss: 51.1253 - val_mae: 5.0385
Epoch 33/200
42/42            0s 4ms/step - loss:
78.3875 - mae: 6.7517 - val_loss: 38.2006 - val_mae: 4.5538
Epoch 34/200
42/42            0s 4ms/step - loss:
84.2104 - mae: 6.9922 - val_loss: 35.8154 - val_mae: 4.3036
Epoch 35/200
42/42            0s 4ms/step - loss:
80.4902 - mae: 6.9525 - val_loss: 44.9426 - val_mae: 4.8098
Epoch 36/200
42/42            0s 4ms/step - loss:
88.8582 - mae: 7.3062 - val_loss: 35.3624 - val_mae: 4.3172
Epoch 37/200
42/42            0s 4ms/step - loss:
102.3971 - mae: 7.5558 - val_loss: 45.8774 - val_mae: 4.7953
Epoch 38/200
42/42            0s 4ms/step - loss:
83.6069 - mae: 6.7047 - val_loss: 27.9649 - val_mae: 3.9393
Epoch 39/200
42/42            0s 4ms/step - loss:
84.4804 - mae: 7.1464 - val_loss: 27.8204 - val_mae: 3.9945
Epoch 40/200
42/42            0s 4ms/step - loss:
83.6259 - mae: 6.9500 - val_loss: 40.5558 - val_mae: 4.6430
Epoch 41/200
42/42            0s 4ms/step - loss:
86.7592 - mae: 6.8392 - val_loss: 32.2992 - val_mae: 4.0796
Epoch 42/200
42/42            0s 5ms/step - loss:
88.6301 - mae: 7.0953 - val_loss: 26.0636 - val_mae: 3.7422
Epoch 43/200
42/42            0s 4ms/step - loss:
82.4691 - mae: 6.9627 - val_loss: 30.0919 - val_mae: 3.9325
Epoch 44/200
42/42            0s 5ms/step - loss:
83.7667 - mae: 6.9608 - val_loss: 39.7043 - val_mae: 4.5051
Epoch 45/200
42/42            0s 5ms/step - loss:
92.3082 - mae: 7.3600 - val_loss: 25.7192 - val_mae: 3.6558
Epoch 46/200
42/42            0s 4ms/step - loss:
80.9147 - mae: 6.9106 - val_loss: 25.9125 - val_mae: 3.8016
```

```
Epoch 47/200
42/42            0s 6ms/step - loss:
98.8363 - mae: 7.5245 - val_loss: 26.8648 - val_mae: 4.0910
Epoch 48/200
42/42            0s 6ms/step - loss:
85.1437 - mae: 6.8477 - val_loss: 30.9909 - val_mae: 3.9263
Epoch 49/200
42/42            0s 7ms/step - loss:
71.4541 - mae: 6.3304 - val_loss: 23.2443 - val_mae: 3.6536
Epoch 50/200
42/42            1s 6ms/step - loss:
85.8813 - mae: 6.7934 - val_loss: 23.0339 - val_mae: 3.6015
Epoch 51/200
42/42            0s 7ms/step - loss:
83.2443 - mae: 7.1351 - val_loss: 28.4066 - val_mae: 3.7720
Epoch 52/200
42/42            0s 9ms/step - loss:
90.6339 - mae: 7.1831 - val_loss: 26.3663 - val_mae: 3.6871
Epoch 53/200
42/42            0s 4ms/step - loss:
80.2616 - mae: 6.7715 - val_loss: 27.6305 - val_mae: 3.7982
Epoch 54/200
42/42            0s 4ms/step - loss:
88.5153 - mae: 7.1623 - val_loss: 46.9522 - val_mae: 5.2049
Epoch 55/200
42/42            0s 4ms/step - loss:
73.7979 - mae: 6.3687 - val_loss: 27.3501 - val_mae: 3.6684
Epoch 56/200
42/42            0s 4ms/step - loss:
73.8392 - mae: 6.7575 - val_loss: 19.8651 - val_mae: 3.1849
Epoch 57/200
42/42            0s 4ms/step - loss:
78.6826 - mae: 6.5981 - val_loss: 19.5903 - val_mae: 3.2824
Epoch 58/200
42/42            0s 4ms/step - loss:
68.0455 - mae: 6.4289 - val_loss: 29.7221 - val_mae: 3.9961
Epoch 59/200
42/42            0s 5ms/step - loss:
82.2636 - mae: 6.8809 - val_loss: 28.8358 - val_mae: 3.8553
Epoch 60/200
42/42            0s 4ms/step - loss:
75.9778 - mae: 6.6111 - val_loss: 35.7315 - val_mae: 4.2260
Epoch 61/200
42/42            0s 4ms/step - loss:
76.2767 - mae: 6.6236 - val_loss: 28.7934 - val_mae: 3.9830
Epoch 62/200
42/42            0s 4ms/step - loss:
66.3485 - mae: 6.1157 - val_loss: 17.7755 - val_mae: 3.2952
```

```
Epoch 63/200
42/42            0s 5ms/step - loss:
81.2376 - mae: 6.8342 - val_loss: 28.5250 - val_mae: 3.8142
Epoch 64/200
42/42            0s 4ms/step - loss:
78.8702 - mae: 6.3969 - val_loss: 17.3097 - val_mae: 3.1080
Epoch 65/200
42/42            0s 4ms/step - loss:
67.3644 - mae: 6.2331 - val_loss: 15.9181 - val_mae: 2.9703
Epoch 66/200
42/42            0s 4ms/step - loss:
76.4548 - mae: 6.1455 - val_loss: 16.7331 - val_mae: 3.0321
Epoch 67/200
42/42            0s 4ms/step - loss:
79.7503 - mae: 6.4898 - val_loss: 23.6799 - val_mae: 3.6428
Epoch 68/200
42/42            0s 4ms/step - loss:
78.6433 - mae: 6.8001 - val_loss: 27.4343 - val_mae: 3.8423
Epoch 69/200
42/42            0s 4ms/step - loss:
81.2435 - mae: 6.6063 - val_loss: 22.2488 - val_mae: 3.4574
Epoch 70/200
42/42            0s 4ms/step - loss:
77.4525 - mae: 6.6430 - val_loss: 21.8632 - val_mae: 3.3436
Epoch 71/200
42/42            0s 4ms/step - loss:
72.3154 - mae: 6.4125 - val_loss: 17.5335 - val_mae: 2.9333
Epoch 72/200
42/42            0s 5ms/step - loss:
71.4160 - mae: 6.5065 - val_loss: 14.0476 - val_mae: 2.7236
Epoch 73/200
42/42            0s 4ms/step - loss:
76.8624 - mae: 6.4873 - val_loss: 25.6790 - val_mae: 3.7316
Epoch 74/200
42/42            0s 4ms/step - loss:
64.1373 - mae: 6.0288 - val_loss: 23.2715 - val_mae: 3.5323
Epoch 75/200
42/42            0s 4ms/step - loss:
69.9320 - mae: 6.0982 - val_loss: 14.7773 - val_mae: 2.7491
Epoch 76/200
42/42            0s 4ms/step - loss:
72.3867 - mae: 6.3506 - val_loss: 16.4610 - val_mae: 2.8798
Epoch 77/200
42/42            0s 4ms/step - loss:
84.3228 - mae: 6.6112 - val_loss: 17.1191 - val_mae: 2.9539
Epoch 78/200
42/42            0s 4ms/step - loss:
78.3755 - mae: 6.4787 - val_loss: 19.5273 - val_mae: 3.0835
```

```
Epoch 79/200
42/42           0s 4ms/step - loss:
66.5712 - mae: 6.0602 - val_loss: 15.1688 - val_mae: 2.6900
Epoch 80/200
42/42           0s 4ms/step - loss:
63.7008 - mae: 6.0312 - val_loss: 22.1789 - val_mae: 3.3124
Epoch 81/200
42/42           0s 4ms/step - loss:
71.1647 - mae: 6.4927 - val_loss: 19.2937 - val_mae: 3.1376
Epoch 82/200
42/42           0s 5ms/step - loss:
81.1119 - mae: 6.6275 - val_loss: 17.7599 - val_mae: 2.9236
Epoch 83/200
42/42           0s 4ms/step - loss:
79.2786 - mae: 6.5446 - val_loss: 21.5492 - val_mae: 3.3193
Epoch 84/200
42/42           0s 4ms/step - loss:
72.2872 - mae: 6.4472 - val_loss: 13.9670 - val_mae: 2.5962
Epoch 85/200
42/42           0s 5ms/step - loss:
63.9936 - mae: 5.9217 - val_loss: 14.0461 - val_mae: 2.5992
Epoch 86/200
42/42           0s 4ms/step - loss:
61.2843 - mae: 5.9638 - val_loss: 14.5200 - val_mae: 2.7303
Epoch 87/200
42/42           0s 4ms/step - loss:
68.9972 - mae: 6.0455 - val_loss: 17.7336 - val_mae: 3.0890
Epoch 88/200
42/42           0s 5ms/step - loss:
67.8744 - mae: 6.0838 - val_loss: 14.0969 - val_mae: 2.7796
Epoch 89/200
42/42           0s 6ms/step - loss:
70.7714 - mae: 6.0818 - val_loss: 14.5174 - val_mae: 2.6286
Epoch 90/200
42/42           0s 6ms/step - loss:
62.9362 - mae: 6.1876 - val_loss: 13.0735 - val_mae: 2.5167
Epoch 91/200
42/42           0s 8ms/step - loss:
87.1346 - mae: 6.7461 - val_loss: 22.9289 - val_mae: 3.4457
Epoch 92/200
42/42           0s 6ms/step - loss:
67.3080 - mae: 6.0810 - val_loss: 22.6117 - val_mae: 3.3947
Epoch 93/200
42/42           0s 6ms/step - loss:
59.4691 - mae: 5.6449 - val_loss: 17.1309 - val_mae: 2.8337
Epoch 94/200
42/42           0s 7ms/step - loss:
70.5772 - mae: 6.3456 - val_loss: 20.9261 - val_mae: 3.1591
```

```
Epoch 95/200
42/42            1s 5ms/step - loss:
67.2252 - mae: 6.0752 - val_loss: 21.2991 - val_mae: 3.3259
Epoch 96/200
42/42            0s 4ms/step - loss:
67.4645 - mae: 6.0425 - val_loss: 26.4374 - val_mae: 3.6724
Epoch 97/200
42/42            0s 4ms/step - loss:
60.0798 - mae: 5.7460 - val_loss: 18.1885 - val_mae: 3.0393
Epoch 98/200
42/42            0s 4ms/step - loss:
67.6190 - mae: 5.9927 - val_loss: 19.6796 - val_mae: 3.1261
Epoch 99/200
42/42            0s 4ms/step - loss:
62.8841 - mae: 5.9847 - val_loss: 14.0040 - val_mae: 2.5897
Epoch 100/200
42/42            0s 5ms/step - loss:
63.2127 - mae: 6.0520 - val_loss: 18.9799 - val_mae: 3.1649
Epoch 101/200
42/42            0s 4ms/step - loss:
73.9971 - mae: 6.5710 - val_loss: 16.7084 - val_mae: 2.8355
Epoch 102/200
42/42            0s 5ms/step - loss:
57.3942 - mae: 5.6211 - val_loss: 15.6497 - val_mae: 2.7350
Epoch 103/200
42/42            0s 5ms/step - loss:
54.8638 - mae: 5.6315 - val_loss: 11.3095 - val_mae: 2.4437
Epoch 104/200
42/42            0s 4ms/step - loss:
68.4367 - mae: 6.2397 - val_loss: 11.4214 - val_mae: 2.4369
Epoch 105/200
42/42            0s 4ms/step - loss:
72.2972 - mae: 6.1803 - val_loss: 10.5020 - val_mae: 2.4408
Epoch 106/200
42/42            0s 4ms/step - loss:
70.2834 - mae: 6.0315 - val_loss: 11.5022 - val_mae: 2.4030
Epoch 107/200
42/42            0s 4ms/step - loss:
57.7341 - mae: 5.7205 - val_loss: 14.7936 - val_mae: 2.7662
Epoch 108/200
42/42            0s 4ms/step - loss:
73.7290 - mae: 6.1561 - val_loss: 18.9262 - val_mae: 3.0533
Epoch 109/200
42/42            0s 4ms/step - loss:
65.7199 - mae: 6.1928 - val_loss: 17.8156 - val_mae: 2.9687
Epoch 110/200
42/42            0s 5ms/step - loss:
58.5135 - mae: 5.8168 - val_loss: 10.1274 - val_mae: 2.4218
```

```
Epoch 111/200
42/42             0s 4ms/step - loss:
70.6850 - mae: 6.2496 - val_loss: 11.7715 - val_mae: 2.5830
Epoch 112/200
42/42             0s 4ms/step - loss:
71.8022 - mae: 6.4245 - val_loss: 14.9565 - val_mae: 2.6209
Epoch 113/200
42/42             0s 5ms/step - loss:
65.9999 - mae: 5.9062 - val_loss: 38.5073 - val_mae: 4.8713
Epoch 114/200
42/42             0s 4ms/step - loss:
64.9567 - mae: 6.0094 - val_loss: 17.5764 - val_mae: 2.9658
Epoch 115/200
42/42             0s 4ms/step - loss:
58.2262 - mae: 5.6143 - val_loss: 16.5551 - val_mae: 2.9651
Epoch 116/200
42/42             0s 4ms/step - loss:
54.7419 - mae: 5.4962 - val_loss: 12.7820 - val_mae: 2.4255
Epoch 117/200
42/42             0s 4ms/step - loss:
69.5326 - mae: 6.0236 - val_loss: 20.2200 - val_mae: 3.1873
Epoch 118/200
42/42             0s 4ms/step - loss:
58.9968 - mae: 5.9029 - val_loss: 15.6507 - val_mae: 2.7016
Epoch 119/200
42/42             0s 4ms/step - loss:
68.3340 - mae: 5.8868 - val_loss: 15.9574 - val_mae: 2.6885
Epoch 120/200
42/42             0s 4ms/step - loss:
57.8412 - mae: 5.5597 - val_loss: 16.3639 - val_mae: 2.8817
Epoch 121/200
42/42             0s 4ms/step - loss:
62.6629 - mae: 5.9731 - val_loss: 20.7501 - val_mae: 3.3830
Epoch 122/200
42/42             0s 4ms/step - loss:
66.8111 - mae: 5.9735 - val_loss: 23.7255 - val_mae: 3.3961
Epoch 123/200
42/42             0s 4ms/step - loss:
62.0845 - mae: 5.6987 - val_loss: 14.5193 - val_mae: 2.6549
Epoch 124/200
42/42             0s 4ms/step - loss:
61.8428 - mae: 5.8323 - val_loss: 13.4469 - val_mae: 2.4754
Epoch 125/200
42/42             0s 5ms/step - loss:
63.4526 - mae: 5.8997 - val_loss: 15.9109 - val_mae: 2.8267
Epoch 126/200
42/42             0s 4ms/step - loss:
59.3974 - mae: 5.7632 - val_loss: 15.1684 - val_mae: 2.6776
```

```
Epoch 127/200
42/42           0s 5ms/step - loss:
60.0911 - mae: 5.7666 - val_loss: 19.3171 - val_mae: 3.0419
Epoch 128/200
42/42           0s 5ms/step - loss:
59.5922 - mae: 5.7810 - val_loss: 10.0237 - val_mae: 2.2148
Epoch 129/200
42/42           0s 5ms/step - loss:
62.6644 - mae: 5.8468 - val_loss: 16.7148 - val_mae: 2.7636
Epoch 130/200
42/42           0s 4ms/step - loss:
66.1332 - mae: 5.9632 - val_loss: 10.3891 - val_mae: 2.4304
Epoch 131/200
42/42           0s 7ms/step - loss:
67.0498 - mae: 6.1088 - val_loss: 13.4851 - val_mae: 2.5011
Epoch 132/200
42/42           0s 6ms/step - loss:
61.8302 - mae: 5.8984 - val_loss: 11.7556 - val_mae: 2.4719
Epoch 133/200
42/42           0s 6ms/step - loss:
54.3754 - mae: 5.4148 - val_loss: 11.0165 - val_mae: 2.2346
Epoch 134/200
42/42           0s 6ms/step - loss:
57.7066 - mae: 5.8301 - val_loss: 25.8402 - val_mae: 3.9250
Epoch 135/200
42/42           0s 6ms/step - loss:
51.1878 - mae: 5.3579 - val_loss: 11.1044 - val_mae: 2.2840
Epoch 136/200
42/42           0s 7ms/step - loss:
58.8646 - mae: 5.6433 - val_loss: 10.9256 - val_mae: 2.5307
Epoch 137/200
42/42           1s 5ms/step - loss:
58.0381 - mae: 5.6276 - val_loss: 13.0445 - val_mae: 2.5498
Epoch 138/200
42/42           0s 4ms/step - loss:
62.2540 - mae: 5.8087 - val_loss: 15.8566 - val_mae: 2.7537
Epoch 139/200
42/42           0s 4ms/step - loss:
66.7593 - mae: 6.0094 - val_loss: 13.1349 - val_mae: 2.4935
Epoch 140/200
42/42           0s 4ms/step - loss:
51.7195 - mae: 5.3765 - val_loss: 10.5297 - val_mae: 2.3040
Epoch 141/200
42/42           0s 4ms/step - loss:
71.8236 - mae: 6.1481 - val_loss: 10.4723 - val_mae: 2.2937
Epoch 142/200
42/42           0s 4ms/step - loss:
57.3522 - mae: 5.6379 - val_loss: 17.3009 - val_mae: 2.7770
```

```
Epoch 143/200
42/42            0s 4ms/step - loss:
56.9771 - mae: 5.6790 - val_loss: 27.8020 - val_mae: 3.7830
Epoch 144/200
42/42            0s 4ms/step - loss:
57.7847 - mae: 5.5915 - val_loss: 12.3886 - val_mae: 2.2960
Epoch 145/200
42/42            0s 4ms/step - loss:
55.1104 - mae: 5.6914 - val_loss: 17.9700 - val_mae: 2.8802
Epoch 146/200
42/42            0s 5ms/step - loss:
53.7105 - mae: 5.3240 - val_loss: 14.1698 - val_mae: 2.5000
Epoch 147/200
42/42            0s 4ms/step - loss:
66.9862 - mae: 6.0208 - val_loss: 9.5347 - val_mae: 2.1889
Epoch 148/200
42/42            0s 5ms/step - loss:
61.5784 - mae: 5.7110 - val_loss: 10.2554 - val_mae: 2.1307
Epoch 149/200
42/42            0s 4ms/step - loss:
60.9515 - mae: 5.8299 - val_loss: 13.6972 - val_mae: 2.5560
Epoch 150/200
42/42            0s 4ms/step - loss:
60.5653 - mae: 5.6187 - val_loss: 17.9053 - val_mae: 3.0621
Epoch 151/200
42/42            0s 4ms/step - loss:
54.8043 - mae: 5.4907 - val_loss: 29.9045 - val_mae: 4.2492
Epoch 152/200
42/42            0s 5ms/step - loss:
56.5576 - mae: 5.4436 - val_loss: 11.7855 - val_mae: 2.4075
Epoch 153/200
42/42            0s 4ms/step - loss:
54.3198 - mae: 5.3404 - val_loss: 19.5024 - val_mae: 3.1821
Epoch 154/200
42/42            0s 4ms/step - loss:
57.5668 - mae: 5.6359 - val_loss: 11.8282 - val_mae: 2.5176
Epoch 155/200
42/42            0s 5ms/step - loss:
58.6299 - mae: 5.8012 - val_loss: 11.7564 - val_mae: 2.3157
Epoch 156/200
42/42            0s 4ms/step - loss:
55.4285 - mae: 5.6276 - val_loss: 13.6254 - val_mae: 2.8797
Epoch 157/200
42/42            0s 4ms/step - loss:
68.7689 - mae: 6.2659 - val_loss: 18.0361 - val_mae: 2.9901
Epoch 158/200
42/42            0s 5ms/step - loss:
54.9901 - mae: 5.7067 - val_loss: 10.6259 - val_mae: 2.2084
```

```
Epoch 159/200
42/42            0s 5ms/step - loss:
57.5381 - mae: 5.4842 - val_loss: 13.7849 - val_mae: 2.5712
Epoch 160/200
42/42            0s 4ms/step - loss:
50.6104 - mae: 5.2754 - val_loss: 10.0559 - val_mae: 2.1482
Epoch 161/200
42/42            0s 4ms/step - loss:
59.0958 - mae: 5.8707 - val_loss: 22.5411 - val_mae: 3.5472
Epoch 162/200
42/42            0s 4ms/step - loss:
53.8695 - mae: 5.3340 - val_loss: 18.1055 - val_mae: 2.9878
Epoch 163/200
42/42            0s 4ms/step - loss:
53.3871 - mae: 5.3666 - val_loss: 15.4584 - val_mae: 2.6020
Epoch 164/200
42/42            0s 4ms/step - loss:
59.2192 - mae: 5.4818 - val_loss: 11.1453 - val_mae: 2.3869
Epoch 165/200
42/42            0s 5ms/step - loss:
61.1713 - mae: 6.0385 - val_loss: 11.7585 - val_mae: 2.2477
Epoch 166/200
42/42            0s 4ms/step - loss:
55.7333 - mae: 5.7287 - val_loss: 9.8739 - val_mae: 2.1734
Epoch 167/200
42/42            0s 4ms/step - loss:
54.2413 - mae: 5.4709 - val_loss: 10.3709 - val_mae: 2.0830
Epoch 168/200
42/42            0s 5ms/step - loss:
62.1040 - mae: 5.8695 - val_loss: 14.9524 - val_mae: 2.7968
Epoch 169/200
42/42            0s 5ms/step - loss:
56.6964 - mae: 5.5675 - val_loss: 13.5969 - val_mae: 2.5038
Epoch 170/200
42/42            0s 4ms/step - loss:
58.0255 - mae: 5.5860 - val_loss: 18.5247 - val_mae: 2.9300
Epoch 171/200
42/42            0s 4ms/step - loss:
68.0808 - mae: 6.0044 - val_loss: 13.8596 - val_mae: 2.5591
Epoch 172/200
42/42            0s 5ms/step - loss:
61.5190 - mae: 5.6590 - val_loss: 14.3675 - val_mae: 2.8854
Epoch 173/200
42/42            0s 4ms/step - loss:
53.7417 - mae: 5.5281 - val_loss: 15.8698 - val_mae: 2.8292
Epoch 174/200
42/42            0s 4ms/step - loss:
64.1871 - mae: 5.7317 - val_loss: 13.9377 - val_mae: 2.4685
```

```
Epoch 175/200
42/42            0s 4ms/step - loss:
50.4265 - mae: 5.1895 - val_loss: 14.9119 - val_mae: 2.7262
Epoch 176/200
42/42            0s 4ms/step - loss:
54.4403 - mae: 5.5623 - val_loss: 10.3396 - val_mae: 2.3533
Epoch 177/200
42/42            0s 7ms/step - loss:
58.1030 - mae: 5.6402 - val_loss: 11.7340 - val_mae: 2.3907
Epoch 178/200
42/42            1s 6ms/step - loss:
51.0114 - mae: 5.2973 - val_loss: 16.2613 - val_mae: 2.9109
Epoch 179/200
42/42            0s 6ms/step - loss:
60.9916 - mae: 5.8484 - val_loss: 14.2392 - val_mae: 2.4767
Epoch 180/200
42/42            0s 6ms/step - loss:
60.1616 - mae: 5.6377 - val_loss: 12.3786 - val_mae: 2.3905
Epoch 181/200
42/42            0s 6ms/step - loss:
56.4566 - mae: 5.6093 - val_loss: 20.4080 - val_mae: 3.2862
Epoch 182/200
42/42            1s 5ms/step - loss:
43.0219 - mae: 4.8167 - val_loss: 14.8601 - val_mae: 2.6249
Epoch 183/200
42/42            0s 5ms/step - loss:
53.2450 - mae: 5.3212 - val_loss: 13.3820 - val_mae: 2.3777
Epoch 184/200
42/42            0s 4ms/step - loss:
57.2685 - mae: 5.4286 - val_loss: 26.8758 - val_mae: 3.9136
Epoch 185/200
42/42            0s 4ms/step - loss:
60.2118 - mae: 5.7138 - val_loss: 21.3524 - val_mae: 3.4115
Epoch 186/200
42/42            0s 4ms/step - loss:
62.1373 - mae: 5.5246 - val_loss: 20.1299 - val_mae: 3.2096
Epoch 187/200
42/42            0s 5ms/step - loss:
61.1864 - mae: 5.7345 - val_loss: 10.0273 - val_mae: 2.1250
Epoch 188/200
42/42            0s 4ms/step - loss:
49.5820 - mae: 5.2678 - val_loss: 12.9555 - val_mae: 2.4428
Epoch 189/200
42/42            0s 4ms/step - loss:
56.5535 - mae: 5.3817 - val_loss: 11.2813 - val_mae: 2.3767
Epoch 190/200
42/42            0s 5ms/step - loss:
45.3114 - mae: 4.9759 - val_loss: 13.0155 - val_mae: 2.3773
```

```
Epoch 191/200
42/42          0s 4ms/step - loss:
51.2838 - mae: 5.3532 - val_loss: 14.5429 - val_mae: 2.5229
Epoch 192/200
42/42          0s 4ms/step - loss:
60.6232 - mae: 5.6528 - val_loss: 18.2446 - val_mae: 3.2450
Epoch 193/200
42/42          0s 4ms/step - loss:
51.6932 - mae: 5.4692 - val_loss: 8.9967 - val_mae: 2.0445
Epoch 194/200
42/42          0s 4ms/step - loss:
52.0826 - mae: 5.2178 - val_loss: 11.4901 - val_mae: 2.3971
Epoch 195/200
42/42          0s 4ms/step - loss:
57.3508 - mae: 5.4522 - val_loss: 22.8238 - val_mae: 3.3859
Epoch 196/200
42/42          0s 4ms/step - loss:
50.5809 - mae: 5.4181 - val_loss: 9.1222 - val_mae: 1.9961
Epoch 197/200
42/42          0s 4ms/step - loss:
54.8630 - mae: 5.3375 - val_loss: 15.6138 - val_mae: 2.8808
Epoch 198/200
42/42          0s 4ms/step - loss:
49.4376 - mae: 5.0345 - val_loss: 14.2664 - val_mae: 2.4058
Epoch 199/200
42/42          0s 5ms/step - loss:
54.7832 - mae: 5.5337 - val_loss: 8.6881 - val_mae: 1.9961
Epoch 200/200
42/42          0s 4ms/step - loss:
48.4725 - mae: 5.2411 - val_loss: 10.0784 - val_mae: 2.2446
```

[36]:

```
model_ann.summary()
```

```
Model: "sequential"


 Layer (type)                       Output Shape
↪Param #

 dense (Dense)                      (None, 128)
↪1,024

 dropout (Dropout)                  (None, 128)
↪  0

 dense_1 (Dense)                    (None, 64)
↪8,256
```

```
dropout_1 (Dropout)                   (None, 64)                          ␣
↪  0

dense_2 (Dense)                       (None, 32)                        ␣
↪2,080

dropout_2 (Dropout)                   (None, 32)                          ␣
↪  0

dense_3 (Dense)                       (None, 1)                           ␣
↪ 33
```

Total params: 34,181 (133.52 KB)

Trainable params: 11,393 (44.50 KB)

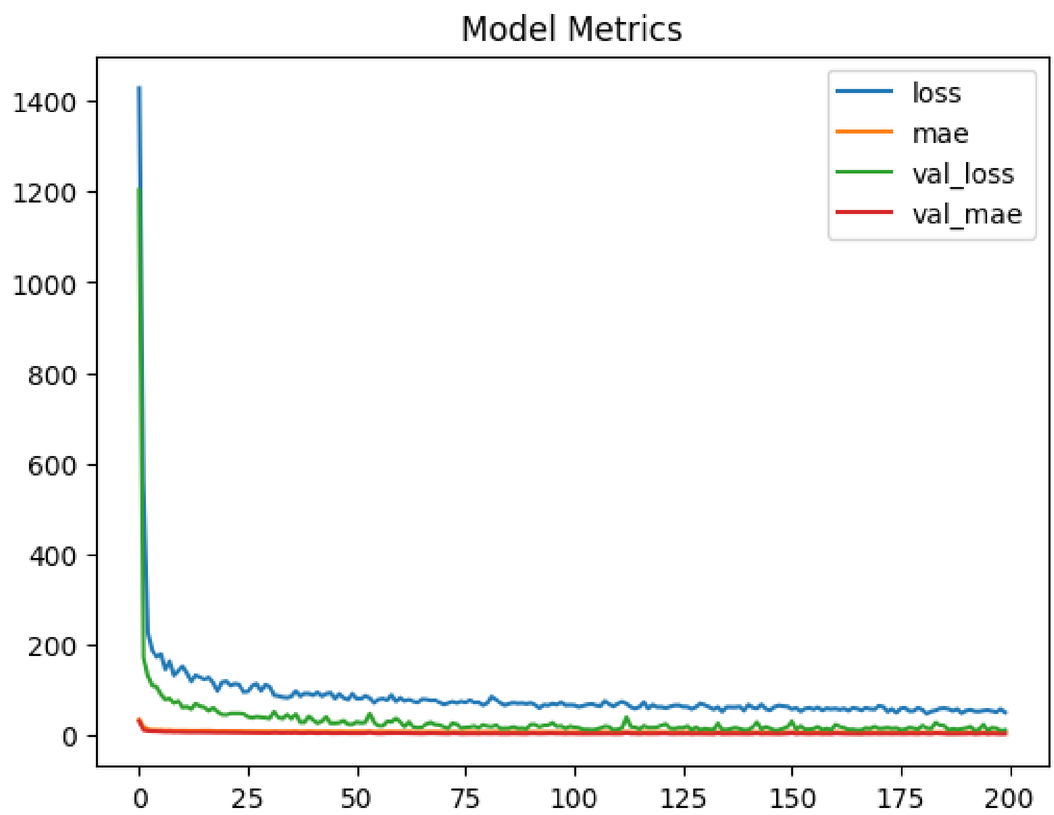
Non-trainable params: 0 (0.00 B)

Optimizer params: 22,788 (89.02 KB)

6.1 Model Training Curves

[37]:

```
hist = model_ann.history.history
hist = pd.DataFrame(hist)
hist.plot()
plt.title('Model Metrics')
plt.show()
```



[38]:
```
y_pred_train = model_ann.predict(X_train)
y_pred_test = model_ann.predict(X_test)
```

```
21/21                 0s 5ms/step
6/6                 0s 5ms/step
```

6.2 ANN Model Evaluation

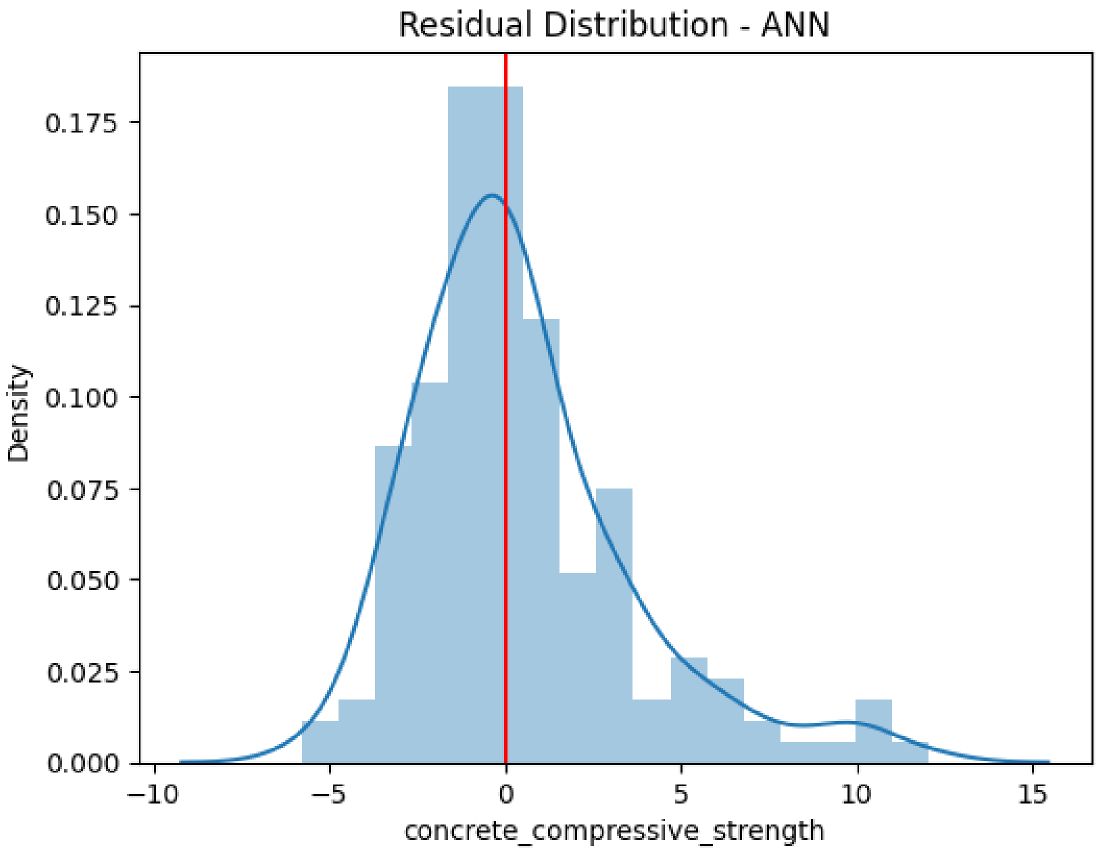
[39]:
```
# Evaluate on Train Data
y_pred_train = model_ann.predict(X_train)
train_r2_score = r2_score(y_train, y_pred_train)
train_mae = mean_absolute_error(y_train, y_pred_train)
train_mse = mean_squared_error(y_train, y_pred_train)
train_rmse = np.sqrt(train_mse)

# Evaluate on Test Data
y_pred_test = model_ann.predict(X_test)
test_r2_score = r2_score(y_test, y_pred_test)
test_mae = mean_absolute_error(y_test, y_pred_test)
test_mse = mean_squared_error(y_test, y_pred_test)
test_rmse = np.sqrt(test_mse)
```

```python
# Print Evaluation Metrics
print('Evaluation for Artificial Neural Network:')
print('Train R2 Score  :', round(train_r2_score, 3))
print('Test R2 Score   :', round(test_r2_score, 3))
print('Train MAE       :', round(train_mae, 3))
print('Test MAE        :', round(test_mae, 3))
print('Train MSE       :', round(train_mse, 3))
print('Test MSE        :', round(test_mse, 3))
print('Train RMSE      :', round(train_rmse, 3))
print('Test RMSE       :', round(test_rmse, 3))
```

```
21/21             0s 2ms/step
6/6             0s 5ms/step
Evaluation for Artificial Neural Network:
Train R2 Score  : 0.976
Test R2 Score   : 0.968
Train MAE       : 1.921
Test MAE        : 2.245
Train MSE       : 7.046
Test MSE        : 10.078
Train RMSE      : 2.654
Test RMSE       : 3.175
```

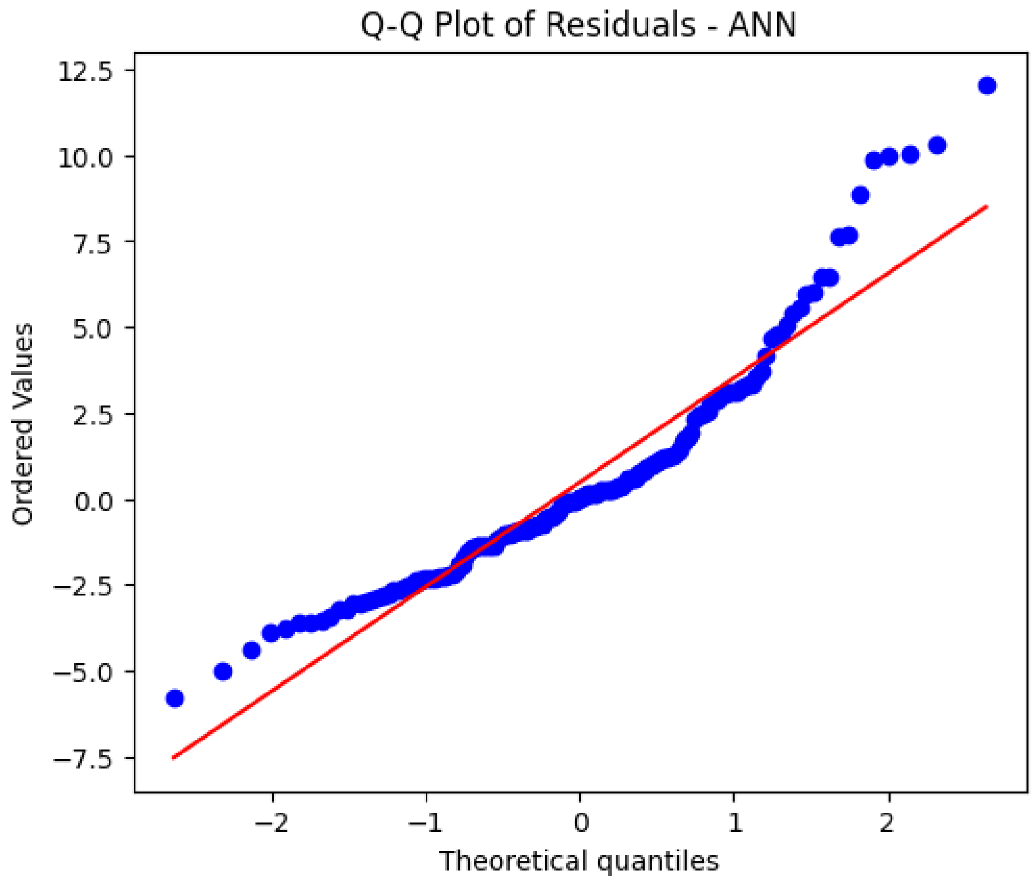
[40]:

```python
residuals = y_test - y_pred_test.flatten()
sns.distplot(residuals)
plt.axvline(0,color = 'red')
plt.title('Residual Distribution - ANN')
plt.show()
```

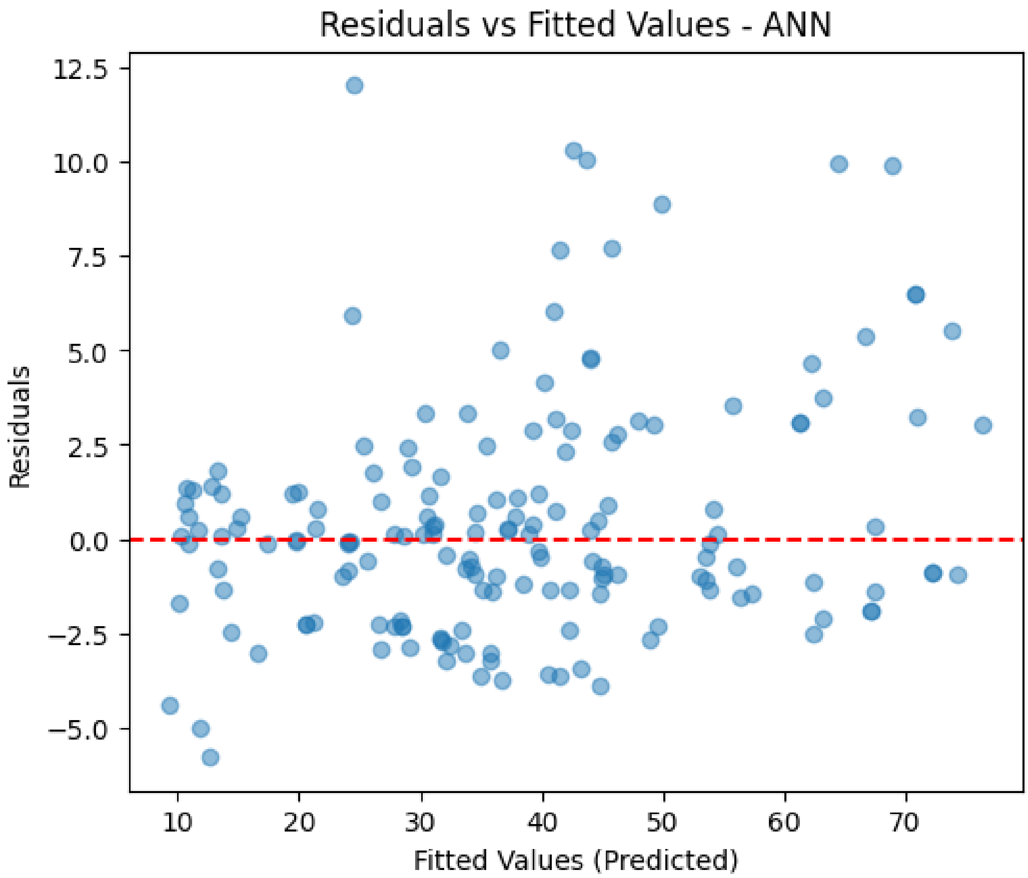


- Normally Distributed Residuals - The histogram and KDE plot show a bell-shaped curve, indicating that residuals are approximately normally distributed.
- Centered Around Zero - The red vertical line at zero suggests that the model has low bias, as the residuals are symmetrically distributed around zero.
- No Major Skewness - The distribution is balanced, meaning no major over-prediction or under-prediction trends.
- Low Residual Variability - Most residuals are within a narrow range, indicating good model performance with minimal errors.
- **Conclusion: The ANN model is well-fitted, showing low bias and normally distributed residuals, making it reliable for predictions.**

[41]:
```
# Q-Q Plot for Normality Check
plt.figure(figsize=(6,5))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals - ANN")
plt.show()
```



[42]:
```python
# Residuals vs. Fitted Plot
plt.figure(figsize=(6,5))
plt.scatter(y_pred_test, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Fitted Values (Predicted)")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values - ANN")
plt.show()
```

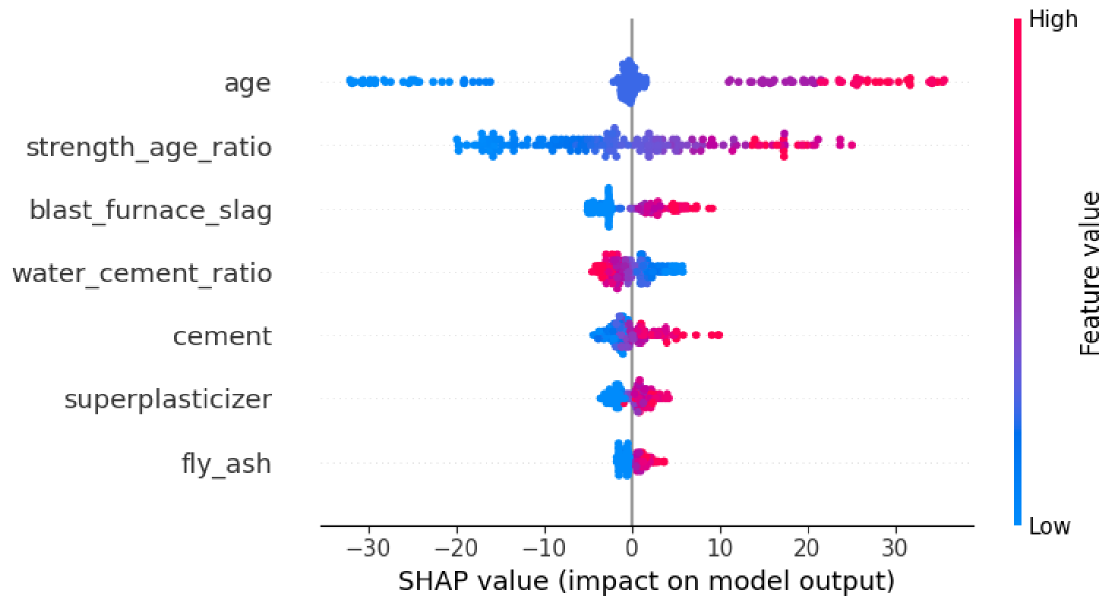


- No Clear Pattern - Residuals are randomly scattered around the red dashed line (zero), suggesting that the model captures the relationship well and there is no major systematic error.
- Homoscedasticity (Constant Variance) - The spread of residuals appears relatively uniform across different predicted values, indicating that heteroscedasticity is not a major issue.
- Some Outliers - A few residuals are noticeably large, suggesting some instances where predictions deviate significantly from actual values.
- Good Model Fit - Since there is no strong trend, the ANN model appears to be making unbiased predictions.
- **Conclusion: The ANN model performs well with no major bias or heteroscedasticity issues, but further fine-tuning may help address the few large residuals.**

[43]:
```
import shap

# Create SHAP explainer
explainer = shap.Explainer(model_ann, X_train)
shap_values = explainer(X_test)
```

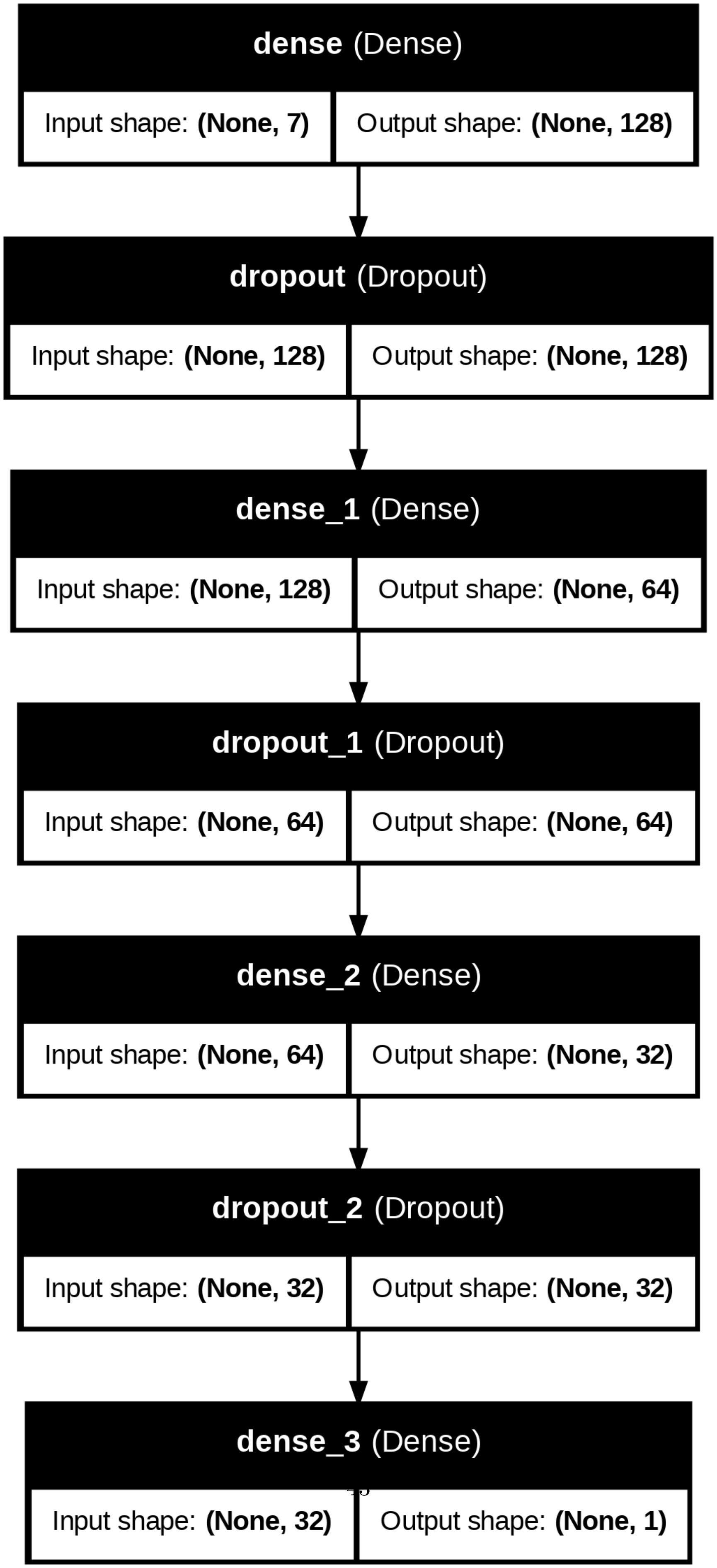
```
# Summary Plot
shap.summary_plot(shap_values, X_test, feature_names=X.columns)
```

```
ExactExplainer explainer: 166it [00:14,  8.93it/s]
```



[44]:
```
import os
os.environ["PATH"] += os.pathsep + "C:/Program Files/Graphviz/bin"

plot_model(model_ann, to_file="model_ann.png", show_shapes=True,␣
↪show_layer_names=True)
```

[44]:


dense (Dense)
Input shape: (None, 7)
Output shape: (None, 128)
dropout (Dropout)
Input shape: (None, 128)
Output shape: (None, 128)
dense_1 (Dense)
Input shape: (None, 128)
Output shape: (None, 64)
dropout_1 (Dropout)
Input shape: (None, 64)
Output shape: (None, 64)
dense_2 (Dense)
Input shape: (None, 64)
Output shape: (None, 32)
dropout_2 (Dropout)
Input shape: (None, 32)
Output shape: (None, 32)
dense_3 (Dense)
Input shape: (None, 32)
Output shape: (None, 1)

[45]:
```
import keras_tuner as kt
from tensorflow import keras
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential

def build_model(hp):
    model = Sequential()

    # Input Layer
    model.add(Dense(units=hp.Int('units_input', min_value=128, max_value=256,␣
↪step=32),
                    activation='relu', input_shape=(X_train.shape[1],)))

    # Hidden Layers (variable number)
    for i in range(hp.Int('num_layers', 1, 4)):  # Choose 1 to 3 hidden layers
        model.add(Dense(units=hp.Int(f'units_{i}', min_value=32, max_value=128,␣
↪step=32),
                        activation='relu'))
        model.add(Dropout(rate=hp.Float(f'dropout_{i}', min_value=0.2,␣
↪max_value=0.5, step=0.1)))

    # Output Layer
    model.add(Dense(1))

    # Compile model
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=hp.Float('learning_rate',␣
↪1e-4, 1e-2, sampling='LOG')),
        loss='mse',
        metrics=['mae']
    )

    return model
```

[53]:
```
tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=30,  # Number of models to test
    executions_per_trial=2,  # Number of times to train each model
    directory='kerastuner_results',
    project_name='ann_regression_tuning2'
)

# Start tuning
```

```
tuner.search(X_train, y_train, epochs=100, batch_size=16,␣
 ↪validation_data=(X_test, y_test))

# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

# Print best hyperparameters
print(f"""
Optimal number of layers: {best_hps.get('num_layers')}
Optimal neurons in input layer: {best_hps.get('units_input')}
Optimal learning rate: {best_hps.get('learning_rate')}
""")

best_hps.values
```

```
Trial 30 Complete [00h 01m 06s]
val_loss: 5.968364477157593

Best val_loss So Far: 4.311817646026611
Total elapsed time: 00h 33m 12s

Optimal number of layers: 1
Optimal neurons in input layer: 192
Optimal learning rate: 0.006429326855520711
```

[53]:
```
{'units_input': 192,
 'num_layers': 1,
 'units_0': 32,
 'dropout_0': 0.30000000000000004,
 'learning_rate': 0.006429326855520711,
 'units_1': 32,
 'dropout_1': 0.4,
 'units_2': 64,
 'dropout_2': 0.4,
 'units_3': 32,
 'dropout_3': 0.30000000000000004}
```

[59]:
```
# Get the best model and train it
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit(X_train, y_train, epochs=100, batch_size=16,␣
 ↪validation_data=(X_test, y_test), )
```

```
Epoch 1/100
42/42                 4s 18ms/step -
loss: 1140.3872 - mae: 28.7515 - val_loss: 153.5228 - val_mae: 9.6051
Epoch 2/100
42/42                 1s 13ms/step -
```

```
loss: 177.0464 - mae: 10.5177 - val_loss: 83.9548 - val_mae: 7.0434
Epoch 3/100
42/42            1s 12ms/step -
loss: 132.4118 - mae: 8.8269 - val_loss: 82.1858 - val_mae: 6.9521
Epoch 4/100
42/42            1s 9ms/step - loss:
122.6491 - mae: 8.4628 - val_loss: 74.3555 - val_mae: 6.4625
Epoch 5/100
42/42            1s 14ms/step -
loss: 121.5968 - mae: 8.5082 - val_loss: 53.5355 - val_mae: 5.3705
Epoch 6/100
42/42            0s 10ms/step -
loss: 123.4532 - mae: 8.2702 - val_loss: 61.3203 - val_mae: 5.7829
Epoch 7/100
42/42            1s 8ms/step - loss:
110.0974 - mae: 7.9444 - val_loss: 63.7772 - val_mae: 5.9503
Epoch 8/100
42/42            0s 9ms/step - loss:
104.6056 - mae: 7.6253 - val_loss: 41.9428 - val_mae: 5.0642
Epoch 9/100
42/42            1s 18ms/step -
loss: 104.7735 - mae: 7.7707 - val_loss: 39.6273 - val_mae: 4.5252
Epoch 10/100
42/42            1s 11ms/step -
loss: 93.5835 - mae: 7.0018 - val_loss: 40.5801 - val_mae: 4.5590
Epoch 11/100
42/42            1s 8ms/step - loss:
78.0807 - mae: 6.5828 - val_loss: 43.6940 - val_mae: 4.7329
Epoch 12/100
42/42            0s 8ms/step - loss:
91.5886 - mae: 6.9644 - val_loss: 26.3667 - val_mae: 3.7428
Epoch 13/100
42/42            0s 10ms/step -
loss: 67.0813 - mae: 6.2165 - val_loss: 31.9826 - val_mae: 4.2156
Epoch 14/100
42/42            1s 8ms/step - loss:
80.4271 - mae: 6.7153 - val_loss: 29.4338 - val_mae: 3.8250
Epoch 15/100
42/42            1s 11ms/step -
loss: 66.8782 - mae: 6.2829 - val_loss: 23.8813 - val_mae: 3.5755
Epoch 16/100
42/42            0s 4ms/step - loss:
62.8730 - mae: 5.9358 - val_loss: 31.6972 - val_mae: 4.3184
Epoch 17/100
42/42            0s 4ms/step - loss:
70.7983 - mae: 6.1954 - val_loss: 18.7610 - val_mae: 3.3780
Epoch 18/100
42/42            0s 4ms/step - loss:
```

```
70.2049 - mae: 6.2262 - val_loss: 37.4461 - val_mae: 4.4073
Epoch 19/100
42/42            0s 4ms/step - loss:
65.3117 - mae: 6.1360 - val_loss: 14.2983 - val_mae: 2.9057
Epoch 20/100
42/42            0s 4ms/step - loss:
68.1251 - mae: 6.3552 - val_loss: 19.5839 - val_mae: 3.2955
Epoch 21/100
42/42            0s 4ms/step - loss:
77.7653 - mae: 6.4032 - val_loss: 19.3602 - val_mae: 3.3195
Epoch 22/100
42/42            0s 4ms/step - loss:
66.9531 - mae: 6.1103 - val_loss: 17.2743 - val_mae: 3.1164
Epoch 23/100
42/42            0s 4ms/step - loss:
64.5844 - mae: 5.9426 - val_loss: 18.1294 - val_mae: 3.1152
Epoch 24/100
42/42            0s 4ms/step - loss:
55.2300 - mae: 5.5125 - val_loss: 19.4254 - val_mae: 3.1205
Epoch 25/100
42/42            0s 4ms/step - loss:
52.0183 - mae: 5.3476 - val_loss: 16.2803 - val_mae: 2.9240
Epoch 26/100
42/42            0s 4ms/step - loss:
63.6620 - mae: 5.9912 - val_loss: 16.6757 - val_mae: 3.0596
Epoch 27/100
42/42            0s 4ms/step - loss:
51.3756 - mae: 5.3857 - val_loss: 10.0968 - val_mae: 2.3079
Epoch 28/100
42/42            0s 4ms/step - loss:
62.0486 - mae: 5.5963 - val_loss: 19.4267 - val_mae: 3.3467
Epoch 29/100
42/42            0s 5ms/step - loss:
62.0964 - mae: 5.9673 - val_loss: 17.2144 - val_mae: 3.0736
Epoch 30/100
42/42            0s 4ms/step - loss:
65.3282 - mae: 5.8335 - val_loss: 13.8930 - val_mae: 2.7018
Epoch 31/100
42/42            0s 4ms/step - loss:
52.6248 - mae: 5.3934 - val_loss: 15.4756 - val_mae: 3.1042
Epoch 32/100
42/42            0s 4ms/step - loss:
53.8277 - mae: 5.5393 - val_loss: 14.4522 - val_mae: 2.8799
Epoch 33/100
42/42            0s 4ms/step - loss:
65.7157 - mae: 6.0289 - val_loss: 15.8446 - val_mae: 2.9793
Epoch 34/100
42/42            0s 4ms/step - loss:
```

```
61.8831 - mae: 5.9300 - val_loss: 9.0487 - val_mae: 2.2900
Epoch 35/100
42/42            0s 4ms/step - loss:
77.7580 - mae: 6.3940 - val_loss: 10.5013 - val_mae: 2.3825
Epoch 36/100
42/42            0s 4ms/step - loss:
48.5608 - mae: 5.2589 - val_loss: 32.1779 - val_mae: 4.2887
Epoch 37/100
42/42            0s 4ms/step - loss:
65.8172 - mae: 5.8681 - val_loss: 14.0618 - val_mae: 2.8755
Epoch 38/100
42/42            0s 4ms/step - loss:
57.1698 - mae: 5.6040 - val_loss: 9.3794 - val_mae: 2.4852
Epoch 39/100
42/42            0s 4ms/step - loss:
48.5015 - mae: 4.9358 - val_loss: 12.7587 - val_mae: 2.8618
Epoch 40/100
42/42            0s 4ms/step - loss:
61.5512 - mae: 5.6855 - val_loss: 36.0267 - val_mae: 4.6053
Epoch 41/100
42/42            0s 5ms/step - loss:
51.6164 - mae: 5.3792 - val_loss: 20.9802 - val_mae: 3.5199
Epoch 42/100
42/42            0s 4ms/step - loss:
53.0713 - mae: 5.5089 - val_loss: 8.0636 - val_mae: 2.1484
Epoch 43/100
42/42            0s 4ms/step - loss:
53.2652 - mae: 5.2829 - val_loss: 18.2405 - val_mae: 3.0347
Epoch 44/100
42/42            0s 4ms/step - loss:
60.0899 - mae: 5.8355 - val_loss: 16.5757 - val_mae: 3.1416
Epoch 45/100
42/42            0s 5ms/step - loss:
59.4189 - mae: 5.4479 - val_loss: 19.4143 - val_mae: 3.3523
Epoch 46/100
42/42            0s 5ms/step - loss:
58.8867 - mae: 5.6585 - val_loss: 12.9718 - val_mae: 2.7203
Epoch 47/100
42/42            0s 6ms/step - loss:
52.5994 - mae: 5.2220 - val_loss: 15.9813 - val_mae: 2.9560
Epoch 48/100
42/42            0s 6ms/step - loss:
45.6431 - mae: 4.9816 - val_loss: 21.9387 - val_mae: 3.6879
Epoch 49/100
42/42            0s 6ms/step - loss:
58.2007 - mae: 5.4408 - val_loss: 16.2337 - val_mae: 2.9796
Epoch 50/100
42/42            0s 6ms/step - loss:
```

```
51.4543 - mae: 5.2130 - val_loss: 21.8572 - val_mae: 3.7375
Epoch 51/100
42/42            0s 6ms/step - loss:
48.1592 - mae: 5.1369 - val_loss: 10.1213 - val_mae: 2.3612
Epoch 52/100
42/42            0s 7ms/step - loss:
59.7134 - mae: 5.4877 - val_loss: 13.6132 - val_mae: 2.7497
Epoch 53/100
42/42            0s 6ms/step - loss:
41.0626 - mae: 4.7797 - val_loss: 7.8548 - val_mae: 2.2052
Epoch 54/100
42/42            0s 5ms/step - loss:
52.3918 - mae: 5.1559 - val_loss: 9.2891 - val_mae: 2.3893
Epoch 55/100
42/42            0s 4ms/step - loss:
46.9841 - mae: 4.9809 - val_loss: 32.8707 - val_mae: 4.2594
Epoch 56/100
42/42            0s 4ms/step - loss:
58.7921 - mae: 5.4791 - val_loss: 5.7762 - val_mae: 1.8033
Epoch 57/100
42/42            0s 4ms/step - loss:
49.2388 - mae: 5.3016 - val_loss: 6.6485 - val_mae: 1.7217
Epoch 58/100
42/42            0s 5ms/step - loss:
50.3010 - mae: 5.0633 - val_loss: 7.3905 - val_mae: 2.0460
Epoch 59/100
42/42            0s 4ms/step - loss:
53.3014 - mae: 5.3963 - val_loss: 18.6761 - val_mae: 3.6854
Epoch 60/100
42/42            0s 4ms/step - loss:
54.4323 - mae: 5.4649 - val_loss: 16.4600 - val_mae: 3.1458
Epoch 61/100
42/42            0s 4ms/step - loss:
57.5684 - mae: 5.3978 - val_loss: 9.3536 - val_mae: 2.4291
Epoch 62/100
42/42            0s 4ms/step - loss:
54.9717 - mae: 5.4937 - val_loss: 11.4963 - val_mae: 2.5265
Epoch 63/100
42/42            0s 4ms/step - loss:
53.4326 - mae: 5.4357 - val_loss: 23.7183 - val_mae: 3.9118
Epoch 64/100
42/42            0s 4ms/step - loss:
55.0264 - mae: 5.2269 - val_loss: 9.1158 - val_mae: 2.2605
Epoch 65/100
42/42            0s 4ms/step - loss:
51.6082 - mae: 5.1793 - val_loss: 12.0231 - val_mae: 2.5499
Epoch 66/100
42/42            0s 4ms/step - loss:
```

```
32.4400 - mae: 4.2719 - val_loss: 12.8467 - val_mae: 2.6236
Epoch 67/100
42/42            0s 5ms/step - loss:
53.5478 - mae: 5.4128 - val_loss: 15.0341 - val_mae: 2.7167
Epoch 68/100
42/42            0s 5ms/step - loss:
38.2546 - mae: 4.6298 - val_loss: 16.2439 - val_mae: 3.1931
Epoch 69/100
42/42            0s 4ms/step - loss:
45.1196 - mae: 4.8254 - val_loss: 6.5949 - val_mae: 2.0144
Epoch 70/100
42/42            0s 4ms/step - loss:
44.0238 - mae: 4.9810 - val_loss: 12.5271 - val_mae: 2.3751
Epoch 71/100
42/42            0s 4ms/step - loss:
39.1347 - mae: 4.4750 - val_loss: 10.8629 - val_mae: 2.4988
Epoch 72/100
42/42            0s 4ms/step - loss:
48.8489 - mae: 4.9734 - val_loss: 15.3864 - val_mae: 3.0292
Epoch 73/100
42/42            0s 4ms/step - loss:
60.8370 - mae: 5.5104 - val_loss: 14.8781 - val_mae: 2.9737
Epoch 74/100
42/42            0s 4ms/step - loss:
37.0672 - mae: 4.2654 - val_loss: 6.9606 - val_mae: 1.8883
Epoch 75/100
42/42            0s 4ms/step - loss:
47.9664 - mae: 5.3789 - val_loss: 7.4384 - val_mae: 1.9043
Epoch 76/100
42/42            0s 4ms/step - loss:
55.0990 - mae: 5.2230 - val_loss: 6.4201 - val_mae: 1.8882
Epoch 77/100
42/42            0s 4ms/step - loss:
45.8243 - mae: 4.8707 - val_loss: 5.2178 - val_mae: 1.6340
Epoch 78/100
42/42            0s 4ms/step - loss:
50.0685 - mae: 5.2059 - val_loss: 5.7715 - val_mae: 1.8522
Epoch 79/100
42/42            0s 4ms/step - loss:
40.1446 - mae: 4.4024 - val_loss: 15.0311 - val_mae: 2.9027
Epoch 80/100
42/42            0s 4ms/step - loss:
44.4193 - mae: 4.9095 - val_loss: 9.7755 - val_mae: 2.3661
Epoch 81/100
42/42            0s 4ms/step - loss:
45.3255 - mae: 4.8411 - val_loss: 7.5279 - val_mae: 1.9439
Epoch 82/100
42/42            0s 4ms/step - loss:
```

```
50.7376 - mae: 5.0179 - val_loss: 13.6730 - val_mae: 2.7752
Epoch 83/100
42/42            0s 4ms/step - loss:
39.0503 - mae: 4.7996 - val_loss: 6.5622 - val_mae: 1.8476
Epoch 84/100
42/42            0s 4ms/step - loss:
40.0482 - mae: 4.7532 - val_loss: 5.6668 - val_mae: 1.8434
Epoch 85/100
42/42            0s 5ms/step - loss:
41.6730 - mae: 4.6625 - val_loss: 16.0337 - val_mae: 2.8506
Epoch 86/100
42/42            0s 4ms/step - loss:
33.7005 - mae: 4.2641 - val_loss: 7.5485 - val_mae: 2.1111
Epoch 87/100
42/42            0s 4ms/step - loss:
41.2889 - mae: 4.5711 - val_loss: 7.1518 - val_mae: 1.9432
Epoch 88/100
42/42            0s 4ms/step - loss:
39.5629 - mae: 4.4043 - val_loss: 7.2493 - val_mae: 2.0287
Epoch 89/100
42/42            0s 4ms/step - loss:
46.1625 - mae: 4.9313 - val_loss: 10.7442 - val_mae: 2.4276
Epoch 90/100
42/42            0s 4ms/step - loss:
48.5440 - mae: 4.9769 - val_loss: 5.1998 - val_mae: 1.6170
Epoch 91/100
42/42            0s 6ms/step - loss:
43.3740 - mae: 4.5926 - val_loss: 6.3004 - val_mae: 1.9615
Epoch 92/100
42/42            0s 8ms/step - loss:
41.8730 - mae: 4.6367 - val_loss: 8.4443 - val_mae: 2.0040
Epoch 93/100
42/42            1s 6ms/step - loss:
48.2741 - mae: 5.1057 - val_loss: 7.2652 - val_mae: 1.9161
Epoch 94/100
42/42            0s 6ms/step - loss:
37.9389 - mae: 4.5747 - val_loss: 14.1463 - val_mae: 2.8409
Epoch 95/100
42/42            0s 6ms/step - loss:
45.6903 - mae: 4.8833 - val_loss: 23.6716 - val_mae: 3.8797
Epoch 96/100
42/42            1s 4ms/step - loss:
46.8454 - mae: 4.8197 - val_loss: 12.1877 - val_mae: 2.5957
Epoch 97/100
42/42            0s 4ms/step - loss:
36.5840 - mae: 4.2910 - val_loss: 9.6243 - val_mae: 2.4078
Epoch 98/100
42/42            0s 5ms/step - loss:
```

```
45.8780 - mae: 4.8286 - val_loss: 6.0390 - val_mae: 1.7709
Epoch 99/100
42/42                  0s 4ms/step - loss:
31.0111 - mae: 3.9985 - val_loss: 9.6239 - val_mae: 2.3134
Epoch 100/100
42/42                  0s 4ms/step - loss:
48.4039 - mae: 5.1236 - val_loss: 11.5878 - val_mae: 2.5846
```

[60]:

```
loss, mae = best_model.evaluate(X_train, y_train)
print(f"Best Model Performance Train - MSE: {loss}, MAE: {mae}")

val_loss, val_mae = best_model.evaluate(X_test, y_test)
print(f"Best Model Performance Test - MSE: {val_loss}, MAE: {val_mae}")
```

```
21/21                  0s 3ms/step - loss:
9.0370 - mae: 2.3032
Best Model Performance Train - MSE: 8.980831146240234, MAE: 2.2771213054656982
6/6                0s 6ms/step - loss:
12.9785 - mae: 2.7672
Best Model Performance Test - MSE: 11.58780288696289, MAE: 2.584590196609497
```

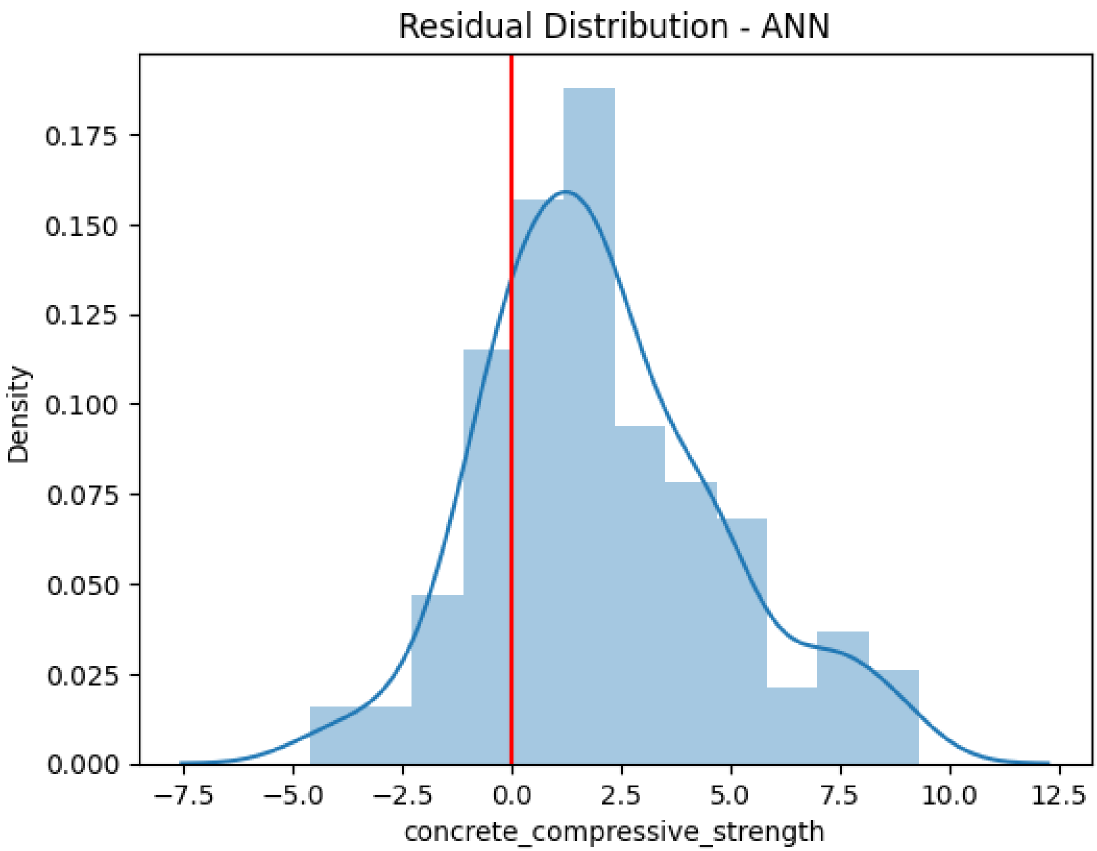
[61]:

```
# Evaluate on Train Data
y_pred_train = best_model.predict(X_train)
train_r2_score = r2_score(y_train, y_pred_train)
train_mae = mean_absolute_error(y_train, y_pred_train)
train_mse = mean_squared_error(y_train, y_pred_train)
train_rmse = np.sqrt(train_mse)

# Evaluate on Test Data
y_pred_test = best_model.predict(X_test)
test_r2_score = r2_score(y_test, y_pred_test)
test_mae = mean_absolute_error(y_test, y_pred_test)
test_mse = mean_squared_error(y_test, y_pred_test)
test_rmse = np.sqrt(test_mse)

# Print Evaluation Metrics
print('Evaluation for Artificial Neural Network:')
print('Train R2 Score   :', round(train_r2_score, 3))
print('Test R2 Score    :', round(test_r2_score, 3))
print('Train MAE        :', round(train_mae, 3))
print('Test MAE         :', round(test_mae, 3))
print('Train MSE        :', round(train_mse, 3))
print('Test MSE         :', round(test_mse, 3))
print('Train RMSE       :', round(train_rmse, 3))
print('Test RMSE        :', round(test_rmse, 3))
```

```
21/21                  0s 4ms/step
6/6                0s 5ms/step
Evaluation for Artificial Neural Network:
```

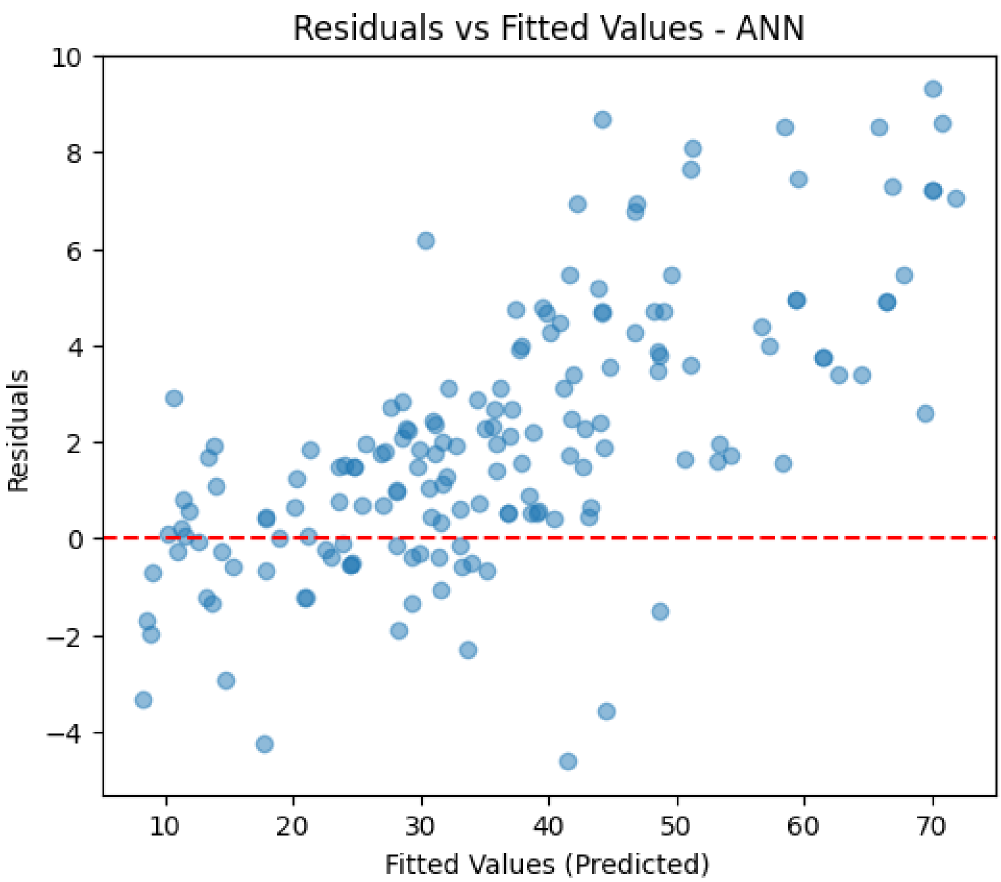
```
Train R2 Score  : 0.969
Test R2 Score   : 0.963
Train MAE       : 2.277
Test MAE        : 2.585
Train MSE       : 8.981
Test MSE        : 11.588
Train RMSE      : 2.997
Test RMSE       : 3.404
```

[62]:
```python
residuals = y_test - y_pred_test.flatten()
sns.distplot(residuals)
plt.axvline(0,color = 'red')
plt.title('Residual Distribution - ANN')
plt.show()
```



[ ]:
```python
# Residuals vs. Fitted Plot
plt.figure(figsize=(6,5))
plt.scatter(y_pred_test, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Fitted Values (Predicted)")
```

```
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values - ANN")
plt.show()
```



7 Save Trained model and preprocessor

[ ]:
```
import joblib

# Save only the fitted scaler
joblib.dump(scaler, "scaler.pkl")
print("Scaler saved as scaler.pkl")

# Save the trained model
best_model.save("final_model.keras")
print("Trained model saved as final_model.keras")
```

```
Scaler saved as scaler.pkl
Trained model saved as final_model.keras
```

[52]:

```

```