Assignment 7.1 Classifications and Regression

Name: Jomarie Dupaya

Course and Section: CPE019, CPE32S3

Instructor: Engr. Roman Richard

Date Performed: 4/3/24

Date Submitted: 2/11/24

Classification

Dataset: Breast Cancer Wisconsin

Dataset Link: https://archive.ics.uci.edu/dataset/15/breast+cancer+wisconsin+original

Based on the article of the dataset the problem trying to solve is to classifify the findings of individuals that has abnomalities in the breast which can lead to breast cancer.

```
pip uninstall tensorflow
```

```
pip install tensorflow==2.12.0
```

```
!pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0
```

```
import pandas as pd
import numpy as np
```

```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
breast_cancer_wisconsin_original = fetch_ucirepo(id=15)

# data (as pandas dataframes)
X = breast_cancer_wisconsin_original.data.features
y = breast_cancer_wisconsin_original.data.targets

# metadata
print(breast_cancer_wisconsin_original.metadata)

# variable information
print(breast_cancer_wisconsin_original.variables)

# Combine X and y into one DataFrame
data = pd.concat([X, y], axis=1)
```

```
{'uci_id': 15, 'name': 'Breast Cancer Wisconsin (Original)', 'repository_url': 'https://archive.ics.uci.edu/dataset/15/breast+cancer+wi
                          name     role         type demographic  \
0           Sample_code_number       ID  Categorical        None
1              Clump_thickness  Feature      Integer        None
2       Uniformity_of_cell_size  Feature      Integer        None
```

```
3      Uniformity_of_cell_shape  Feature      Integer        None
4             Marginal_adhesion  Feature      Integer        None
5   Single_epithelial_cell_size  Feature      Integer        None
6                   Bare_nuclei  Feature      Integer        None
7              Bland_chromatin  Feature      Integer        None
8              Normal_nucleoli  Feature      Integer        None
9                      Mitoses  Feature      Integer        None
10                       Class   Target       Binary        None

                   description units missing_values
0                         None  None             no
1                         None  None             no
2                         None  None             no
3                         None  None             no
4                         None  None             no
5                         None  None             no
6                         None  None            yes
7                         None  None             no
8                         None  None             no
9                         None  None             no
10  2 = benign, 4 = malignant  None             no
```

Reamarks: The data has missing values in Bare_nuclei Feature which will part of the cleaning process in the EDA.

Cleaning and Exploratory Data Analysis

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from ucimlrepo import fetch_ucirepo


# Check for missing values in features
missing_values = X.isnull().sum()
print("Missing Values in Features:")
print(missing_values)

# Check for missing values in target
missing_values_target = y.isnull().sum()
print("Missing Values in Target:")
print(missing_values_target)
```

```
Missing Values in Features:
Clump_thickness                0
Uniformity_of_cell_size        0
Uniformity_of_cell_shape       0
Marginal_adhesion              0
Single_epithelial_cell_size    0
Bare_nuclei                   16
Bland_chromatin                0
Normal_nucleoli                0
Mitoses                        0
dtype: int64
Missing Values in Target:
Class    0
dtype: int64
```

```
# Fill missing values in features with mean value
X.fillna(X.mean(), inplace=True)
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 9 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Clump_thickness              699 non-null    int64
 1   Uniformity_of_cell_size      699 non-null    int64
 2   Uniformity_of_cell_shape     699 non-null    int64
 3   Marginal_adhesion            699 non-null    int64
 4   Single_epithelial_cell_size  699 non-null    int64
 5   Bare_nuclei                  699 non-null    float64
 6   Bland_chromatin              699 non-null    int64
```

```
 7   Normal_nucleoli              699 non-null    int64
 8   Mitoses                      699 non-null    int64
dtypes: float64(1), int64(8)
memory usage: 49.3 KB
<ipython-input-7-3eb05db537f0>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  X.fillna(X.mean(), inplace=True)
```

```
# Summary statistics of features
print("Summary Statistics of Features:")
print(X.describe())

# Summary statistics of target
print("Summary Statistics of Target:")
print(y.describe())
```

```
Summary Statistics of Features:
       Clump_thickness  Uniformity_of_cell_size  Uniformity_of_cell_shape  \
count       699.000000               699.000000                699.000000
mean          4.417740                 3.134478                  3.207439
std           2.815741                 3.051459                  2.971913
min           1.000000                 1.000000                  1.000000
25%           2.000000                 1.000000                  1.000000
50%           4.000000                 1.000000                  1.000000
75%           6.000000                 5.000000                  5.000000
max          10.000000                10.000000                 10.000000

       Marginal_adhesion  Single_epithelial_cell_size  Bare_nuclei  \
count         699.000000                   699.000000   699.000000
mean            2.806867                     3.216023     3.544656
std             2.855379                     2.214300     3.601852
min             1.000000                     1.000000     1.000000
25%             1.000000                     2.000000     1.000000
50%             1.000000                     2.000000     1.000000
75%             4.000000                     4.000000     5.000000
max            10.000000                    10.000000    10.000000

       Bland_chromatin  Normal_nucleoli     Mitoses
count       699.000000       699.000000  699.000000
mean          3.437768         2.866953    1.589413
std           2.438364         3.053634    1.715078
min           1.000000         1.000000    1.000000
25%           2.000000         1.000000    1.000000
50%           3.000000         1.000000    1.000000
75%           5.000000         4.000000    1.000000
max          10.000000        10.000000   10.000000
Summary Statistics of Target:
            Class
count  699.000000
mean     2.689557
std      0.951273
min      2.000000
25%      2.000000
50%      2.000000
75%      4.000000
max      4.000000
```

Remarks: The statistical data shows that there is a big gap between 75% and max values which means an outlier removal must be used.

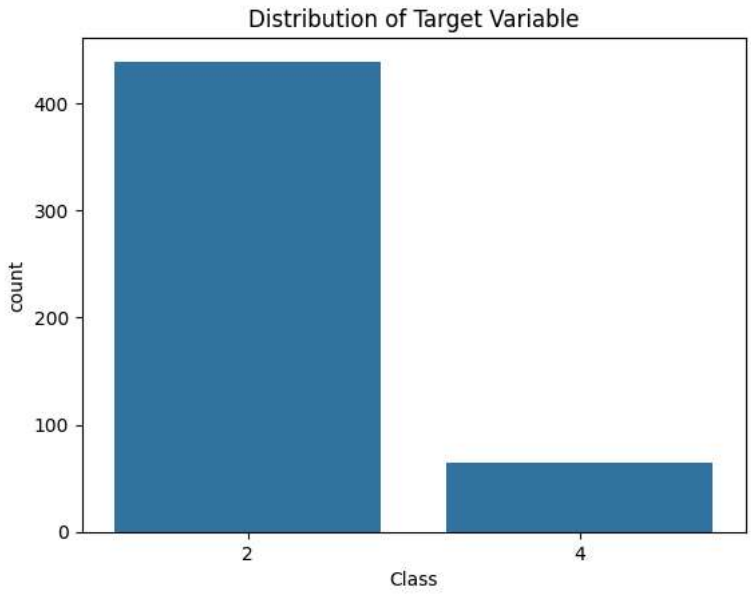
```
def remove_outliers(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[((df >= lower_bound) & (df <= upper_bound)).all(axis=1)]

# Remove outliers from features (X) and target (y)
X_cleaned = remove_outliers(X)
y_cleaned = y.loc[X_cleaned.index]
```

```
# Summary Statistics after removing outliers

print("Summary Statistics of Features:")
print(X_cleaned.describe())

print("Summary Statistics of Target:")
print(y_cleaned.describe())
```

```
Summary Statistics of Features:
       Clump_thickness  Uniformity_of_cell_size  Uniformity_of_cell_shape  \
count       504.000000               504.000000                504.000000
mean          3.422619                 1.803571                  1.976190
std           2.211924                 1.817495                  1.910375
min           1.000000                 1.000000                  1.000000
25%           1.000000                 1.000000                  1.000000
50%           3.000000                 1.000000                  1.000000
75%           5.000000                 2.000000                  2.000000
max          10.000000                10.000000                 10.000000

       Marginal_adhesion  Single_epithelial_cell_size  Bare_nuclei  \
count         504.000000                   504.000000   504.000000
mean            1.623016                     2.269841     2.208493
std             1.370212                     1.008218     2.633365
min             1.000000                     1.000000     1.000000
25%             1.000000                     2.000000     1.000000
50%             1.000000                     2.000000     1.000000
75%             1.250000                     2.000000     1.000000
max             8.000000                     7.000000    10.000000

       Bland_chromatin  Normal_nucleoli  Mitoses
count       504.000000        504.00000    504.0
mean          2.484127          1.52381      1.0
std           1.598877          1.40272      0.0
min           1.000000          1.00000      1.0
25%           1.000000          1.00000      1.0
50%           2.000000          1.00000      1.0
75%           3.000000          1.00000      1.0
max           9.000000          8.00000      1.0
Summary Statistics of Target:
            Class
count  504.000000
mean     2.257937
std      0.670996
min      2.000000
25%      2.000000
50%      2.000000
75%      2.000000
max      4.000000
```

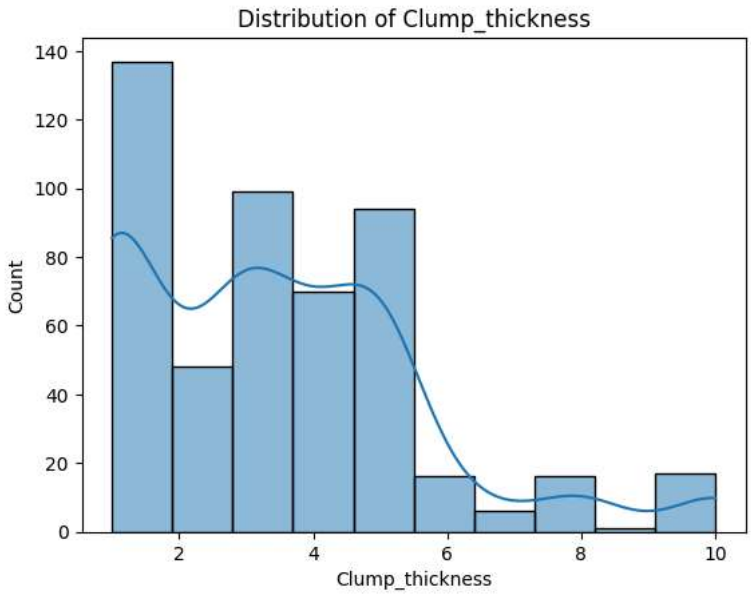
Remarks: After removing outliers in the data there are differences in the values but still a large gap is present in the statistical reading.

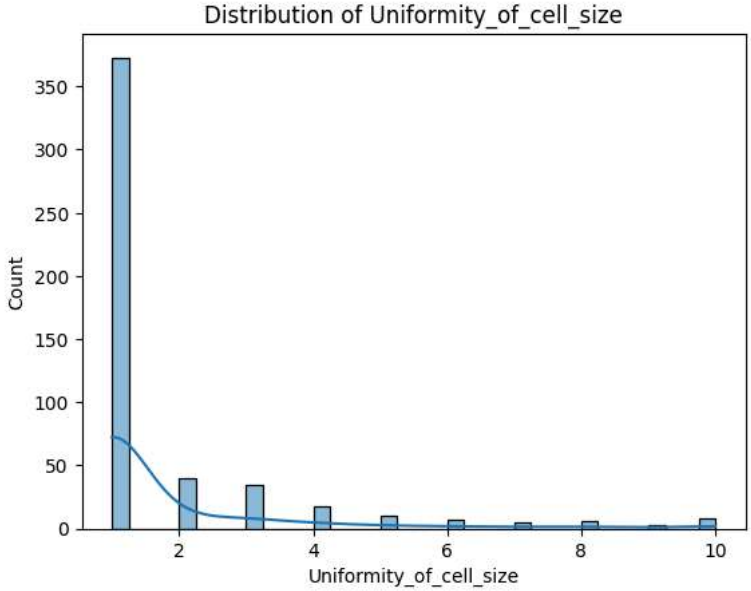
```
# Distribution of Target Variable
sns.countplot(x=y_cleaned.iloc[:, 0])
plt.title("Distribution of Target Variable")
plt.show()
```

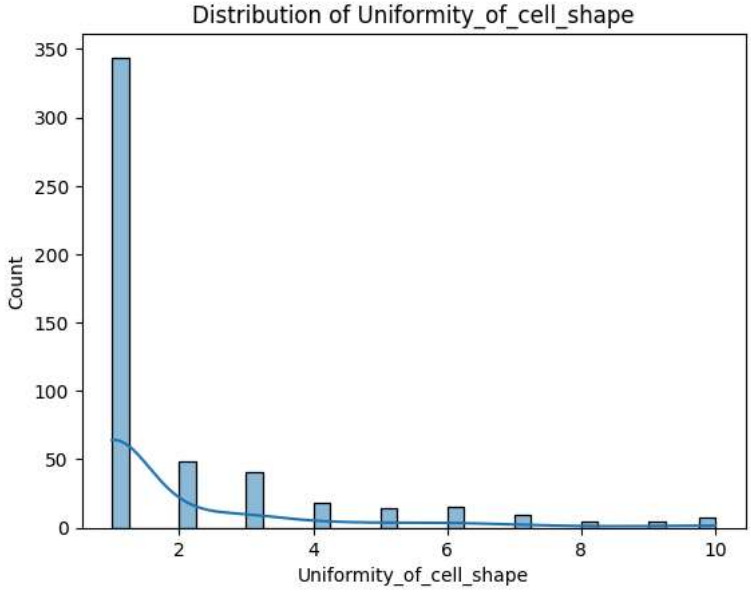


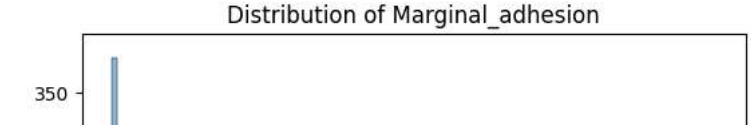
Remarks: Based on the distribution of target variable that most of the data are class 2 in breast cancer, which means those who are in class 2 may be in suspicion of an abnomalities, while class 4 are minimal.

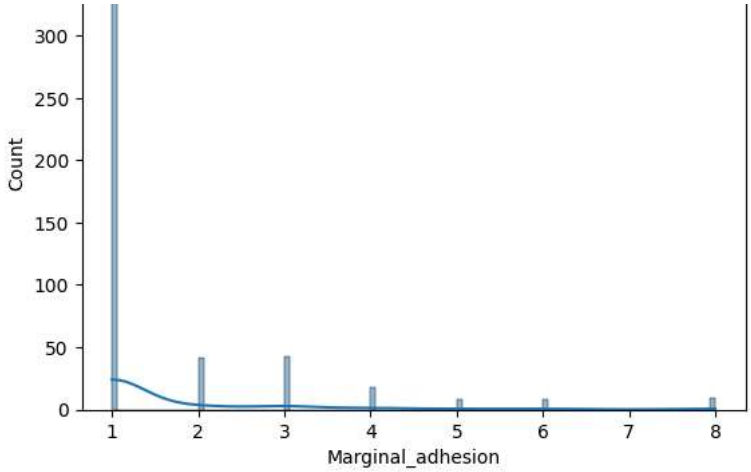
```
# Distribution of Features
for column in X_cleaned.columns:
    sns.histplot(x=X_cleaned[column], kde=True)
    plt.title(f"Distribution of {column}")
    plt.show()
```

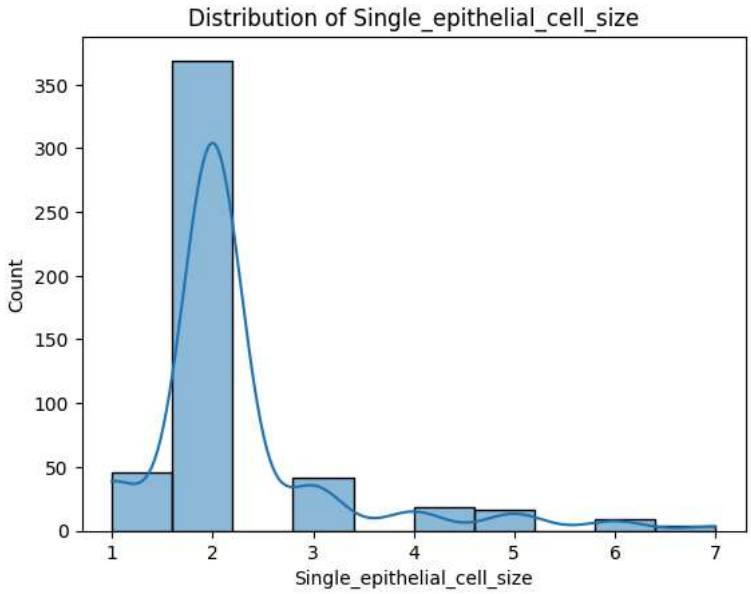

Distribution of Clump_thickness
Count
Clump_thickness
0
20
40
60
80
100
120
140
2
4
6
8
10

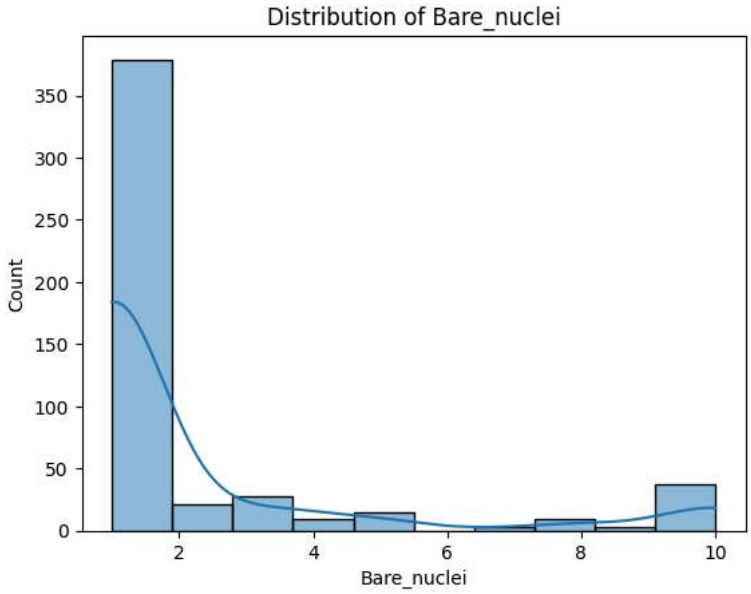

Distribution of Uniformity_of_cell_size
Count
Uniformity_of_cell_size
0
50
100
150
200
250
300
350
2
4
6
8
10

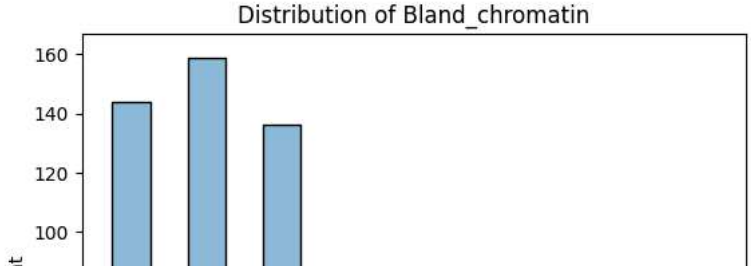

Distribution of Uniformity_of_cell_shape
Count
Uniformity_of_cell_shape
0
50
100
150
200
250
300
350
2
4
6
8
10

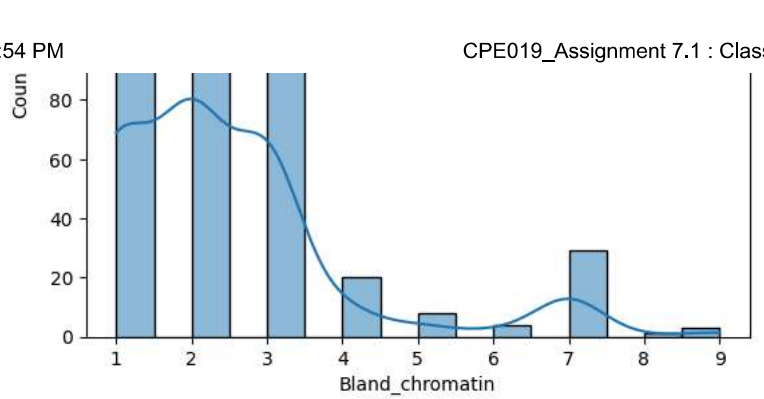

Distribution of Marginal_adhesion
350

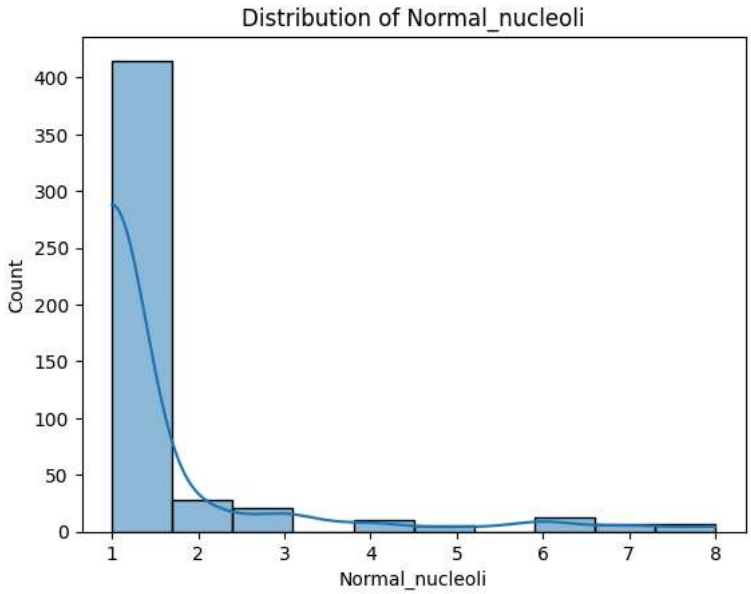

Count
Marginal_adhesion

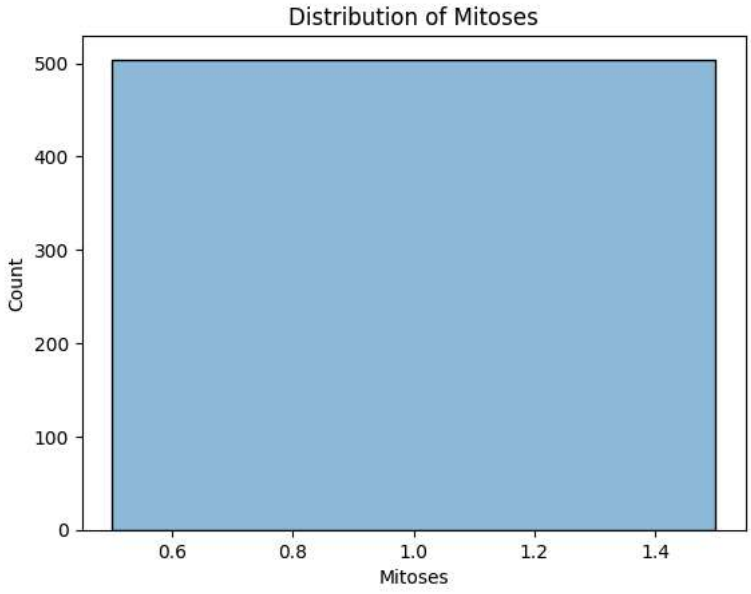

Distribution of Single_epithelial_cell_size
Count
Single_epithelial_cell_size


Distribution of Bare_nuclei
Count
Bare_nuclei


Distribution of Bland_chromatin

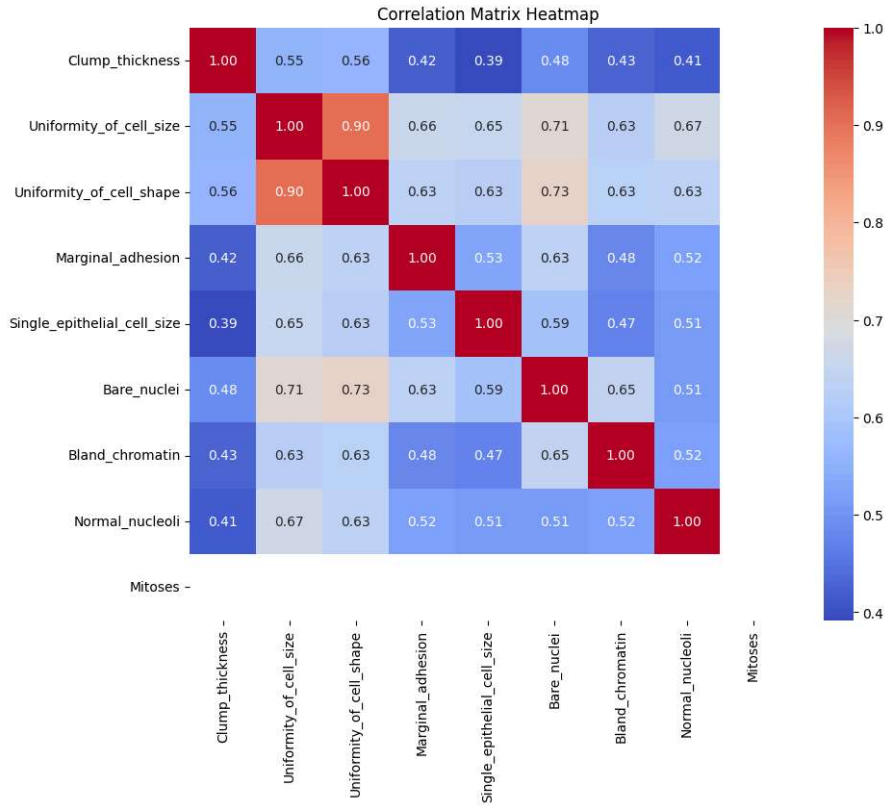






Remarks: Based on the distribution of the feature variables most of the data are in between 0-4 in read rating, which means that the data that surround the features are mostly likely in the class 2.

```
# Correlation Analysis
correlation_matrix = X_cleaned.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```




```
from sklearn.ensemble import RandomForestClassifier
import numpy as np

#Separate features and target variable
X = X_cleaned.drop(['Mitoses', 'Marginal_adhesion','Single_epithelial_cell_size', 'Normal_nucleoli','Clump_thickness','Bland_chromatin'], a
y = y_cleaned['Class']

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

#Feature importance
#Train a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

#Get feature importances
feature_importances = rf.feature_importances_

#Create DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

print("Feature importance:")
print(feature_importance_df)
```

```
Feature importance:
                    Feature  Importance
2              Bare_nuclei    0.373678
1  Uniformity_of_cell_shape    0.313804
0   Uniformity_of_cell_size    0.312518
```

Remarks: The highest positive correlation on this data is the Uniformity_of_cell_size and Uniformity_of_cell_shape, and Bare_nuclei.

```
print(X_cleaned.shape)
print(y_cleaned.shape)
```

```
(504, 9)
(504, 1)
```

Classification Modeling

```
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from keras.losses import CategoricalCrossentropy
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.losses import CategoricalCrossentropy
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold


def baseline_model():
    # Create model
    model = Sequential()
    model.add(Dense(5, input_dim=3, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model


estimator = KerasClassifier(build_fn=baseline_model, epochs=200, batch_size=5)
kfold = KFold(n_splits=10, shuffle=True)
results = cross_val_score(estimator, X_train, y_train, cv=kfold)
```

```
73/73 [==============================] - 0s 2ms/step - loss: 0.0864 - accuracy: 0.9669
Epoch 69/200
73/73 [==============================] - 0s 2ms/step - loss: 0.0853 - accuracy: 0.9669
Epoch 70/200
73/73 [==============================] - 0s 3ms/step - loss: 0.0856 - accuracy: 0.9669
Epoch 71/200
73/73 [==============================] - 0s 2ms/step - loss: 0.0857 - accuracy: 0.9669
Epoch 72/200
73/73 [==============================] - 0s 2ms/step - loss: 0.0863 - accuracy: 0.9669
Epoch 73/200
73/73 [==============================] - 0s 2ms/step - loss: 0.0858 - accuracy: 0.9669
Epoch 74/200
73/73 [==============================] - 0s 2ms/step - loss: 0.0848 - accuracy: 0.9669
Epoch 75/200
73/73 [==============================] - 0s 2ms/step - loss: 0.0846 - accuracy: 0.9669
Epoch 76/200
73/73 [==============================] - 0s 2ms/step - loss: 0.0847 - accuracy: 0.9669
```

```
mean_accuracy = results.mean() * 100
std_accuracy = results.std() * 100
print("Accuracy: %.2f%% (%.2f%%)" % (mean_accuracy, std_accuracy))
```

```
Accuracy: 96.77% (3.36%)
```

Remarks: Based on the models output and reapeted training data results shows a around 95% score on accuracy which means that the modeling is succeful in classifying the class of breast cancer according to the dataset.

Regression

Dataset: Concrete Compressive Strength

Dataset Link: https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength

The problem is to solve the strength of concrete through the features of a concrete and to solve it through a regression how long it can last on different data.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
concrete_compressive_strength = fetch_ucirepo(id=165)

# data (as pandas dataframes)
X2 = concrete_compressive_strength.data.features
y2 = concrete_compressive_strength.data.targets

# metadata
print(concrete_compressive_strength.metadata)

# variable information
print(concrete_compressive_strength.variables)
```

Cleaning and Exploratory Data Analysis

```
# Check for missing values in features
missing_values = X2.isnull().sum()
print("Missing Values in Features:")
print(missing_values)

# Check for missing values in target
missing_values_target = y2.isnull().sum()
print("Missing Values in Target:")
print(missing_values_target)
```

```
Missing Values in Features:
Cement               0
Blast Furnace Slag   0
Fly Ash              0
Water                0
Superplasticizer     0
Coarse Aggregate     0
Fine Aggregate       0
Age                  0
dtype: int64
Missing Values in Target:
Concrete compressive strength    0
dtype: int64
```

```
# Fill missing values in features with mean value
X2.fillna(X.mean(), inplace=True)
X2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 8 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  
dtypes: float64(7), int64(1)
memory usage: 64.5 KB
<ipython-input-88-b2d9f7625fbc>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  X2.fillna(X.mean(), inplace=True)
```

```
# Summary statistics of features
print("Summary Statistics of Features:")
print(X2.describe())

# Summary statistics of target
print("Summary Statistics of Target:")
print(y2.describe())
```

```
Summary Statistics of Features:
            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  
Summary Statistics of Target:
       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
```

```
def remove_outliers(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[((df >= lower_bound) & (df <= upper_bound)).all(axis=1)]

# Remove outliers from features (X) and target (y)
X_cleaned2 = remove_outliers(X2)
y_cleaned2 = y2.loc[X_cleaned2.index]

X_cleaned3 = remove_outliers(X_cleaned2)
y_cleaned3 = y_cleaned2.loc[X_cleaned2.index]

X_cleaned4 = remove_outliers(X_cleaned3)
y_cleaned4 = y_cleaned3.loc[X_cleaned3.index]

X_cleaned5 = remove_outliers(X_cleaned4)
y_cleaned5 = y_cleaned4.loc[X_cleaned4.index]
```

```
# Summary statistics of features
print("Summary Statistics of Features:")
print(X_cleaned5.describe())

# Summary statistics of target
print("Summary Statistics of Target:")
print(y_cleaned5.describe())
```

```
Summary Statistics of Features:
            Cement  Blast Furnace Slag     Fly Ash       Water  \
count  763.000000          763.000000  763.000000  763.000000
mean   271.342726           72.729096   62.334993  179.065793
std     97.839806           85.348462   64.250382   17.478538
min    108.300000            0.000000    0.000000  127.000000
25%    188.100000            0.000000    0.000000  164.900000
50%    255.000000           20.000000   77.000000  181.700000
75%    338.450000          144.450000  121.500000  192.000000
max    540.000000          342.100000  200.100000  228.000000

       Superplasticizer  Coarse Aggregate  Fine Aggregate         Age
count        763.000000        763.000000      763.000000  763.000000
mean           6.675491        971.101048      782.784273   23.142857
std            5.215753         77.448379       61.624854   15.493773
min            0.000000        801.000000      636.000000    1.000000
25%            0.000000        929.800000      746.700000    7.000000
50%            7.500000        968.000000      780.600000   28.000000
75%           10.650000       1028.400000      821.700000   28.000000
max           22.100000       1145.000000      925.700000   56.000000
Summary Statistics of Target:
       Concrete compressive strength
count                     763.000000
mean                       33.017025
std                        16.419163
min                         2.330000
25%                        20.945000
50%                        31.540000
75%                        42.080000
max                        81.750000
```

```
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X_normalized)
X_standardized = min_max_scaler.transform(X_normalized)
X_standardized_df = pd.DataFrame(X_standardized, columns=X_cleaned5.columns)
print(X_standardized_df.describe())
```

```
            Cement  Blast Furnace Slag     Fly Ash       Water  \
count  763.000000          763.000000  763.000000  763.000000
mean     0.377676            0.212596    0.311519    0.515503
std      0.226638            0.249484    0.321091    0.173055
```

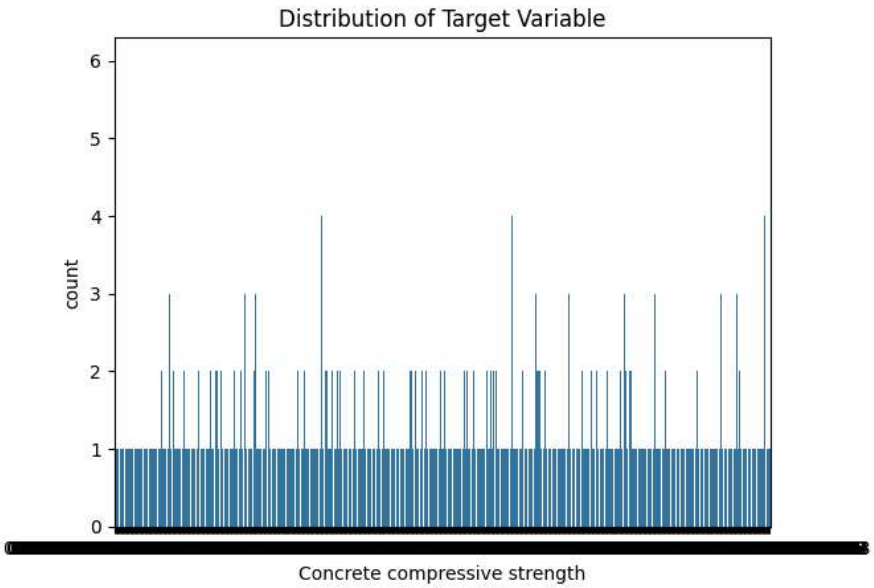
```
min      0.000000           0.000000    0.000000    0.000000
25%      0.184851           0.000000    0.000000    0.375248
50%      0.339819           0.058462    0.384808    0.541584
75%      0.533125           0.422245    0.607196    0.643564
max      1.000000           1.000000    1.000000    1.000000

       Superplasticizer  Coarse Aggregate  Fine Aggregate         Age
count        763.000000        763.000000      763.000000  763.000000
mean           0.302058          0.494480        0.506677    0.402597
std            0.236007          0.225141        0.212720    0.281705
min            0.000000          0.000000        0.000000    0.000000
25%            0.000000          0.374419        0.382119    0.109091
50%            0.339367          0.485465        0.499137    0.490909
75%            0.481900          0.661047        0.641008    0.490909
max            1.000000          1.000000        1.000000    1.000000
```

```
min_max_scaler.fit(y_cleaned5.values.reshape(-1, 1))
y_standardized = min_max_scaler.transform(y_cleaned5.values.reshape(-1, 1))
y_standardized_df = pd.DataFrame(y_standardized, columns=y_cleaned5.columns)
print(y_standardized_df.describe())
```

```
       Concrete compressive strength
count                     763.000000
mean                        0.386389
std                         0.206738
min                         0.000000
25%                         0.234387
50%                         0.367791
75%                         0.500504
max                         1.000000
```

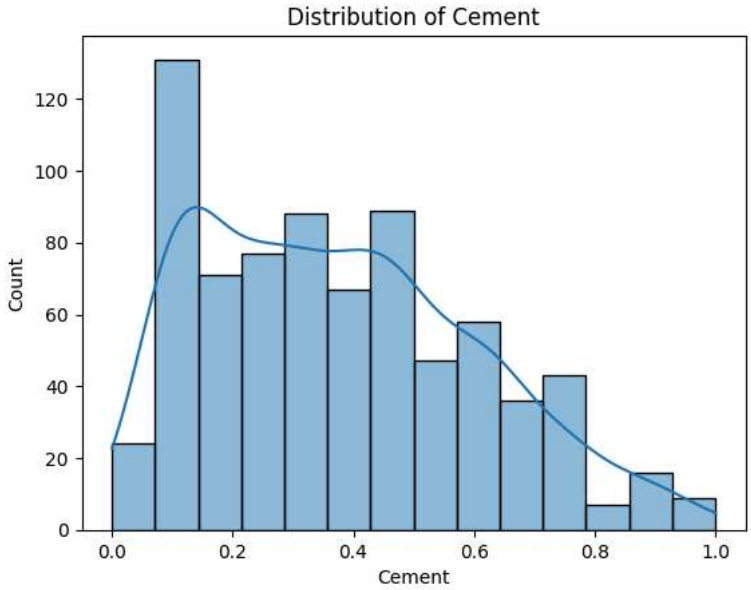
Remarks: Standardizing and normalizing the data to bring it close to 0 and 1

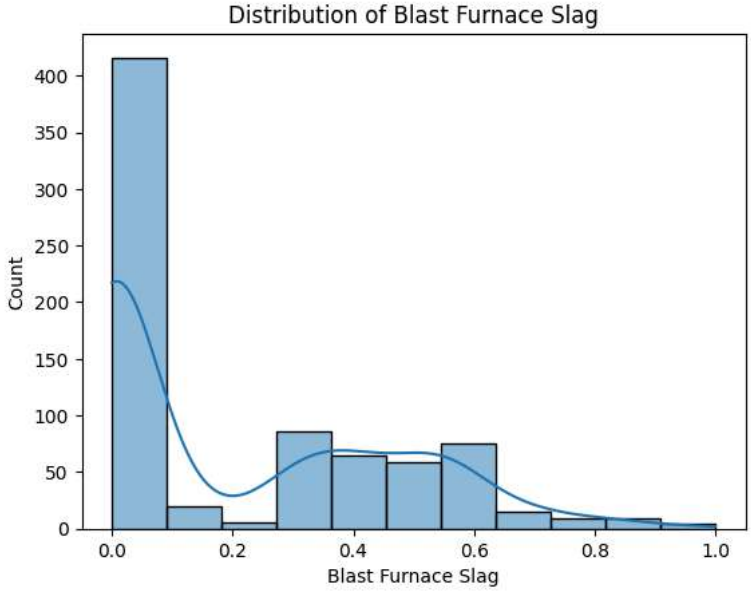
```
# Distribution of Target Variable
sns.countplot(x=y_standardized_df.iloc[:, 0])
plt.title("Distribution of Target Variable")
plt.show()
```

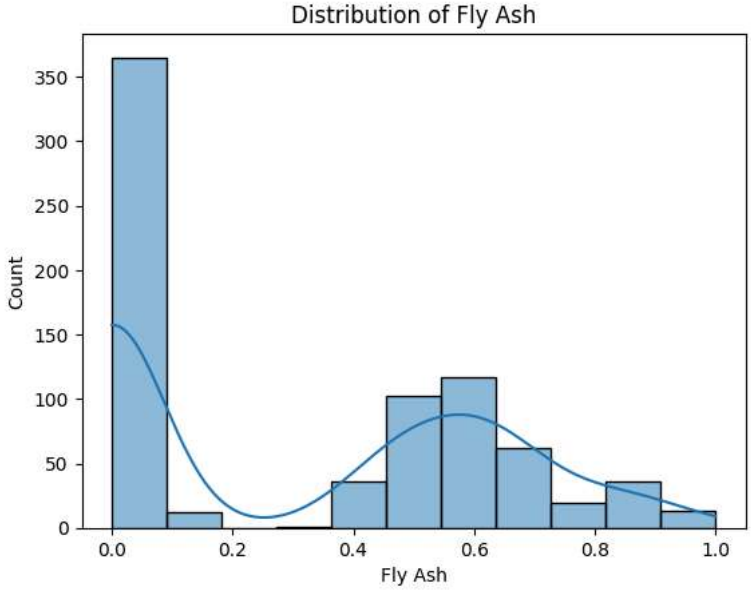


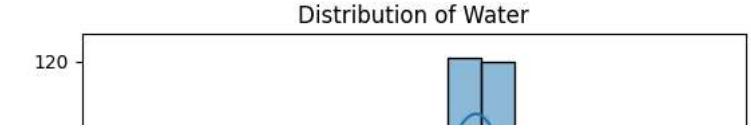
Remarks: The distribution of the target variable is somewhat almost equally divided to each other with some has higher counts than the other.

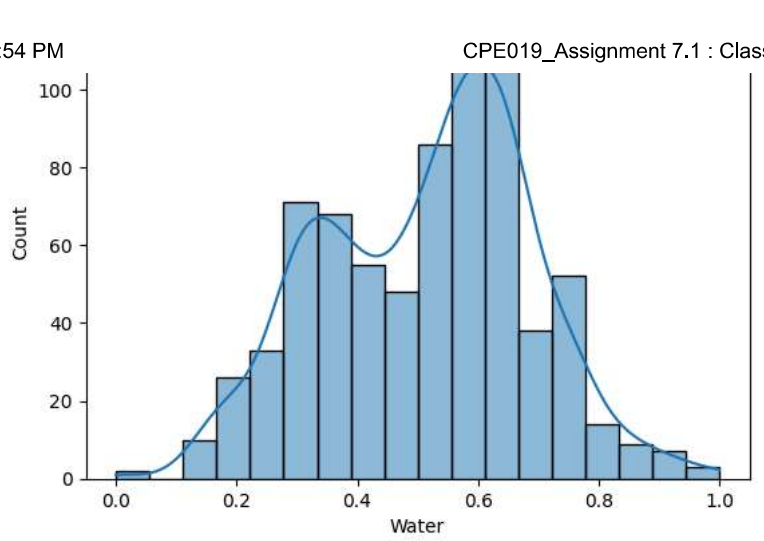
```
# Distribution of Features
for column in X_standardized_df.columns:
    sns.histplot(x=X_standardized_df[column], kde=True)
    plt.title(f"Distribution of {column}")
    plt.show()
```

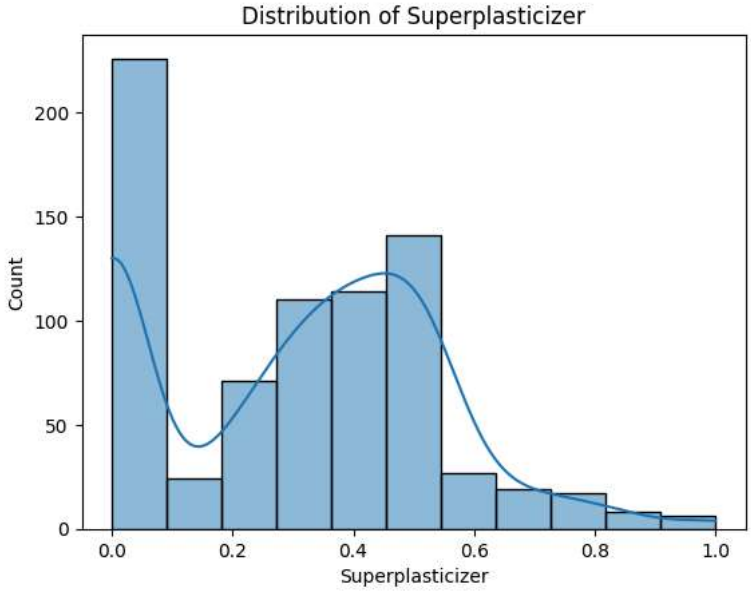

Distribution of Cement
Count
Cement
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0

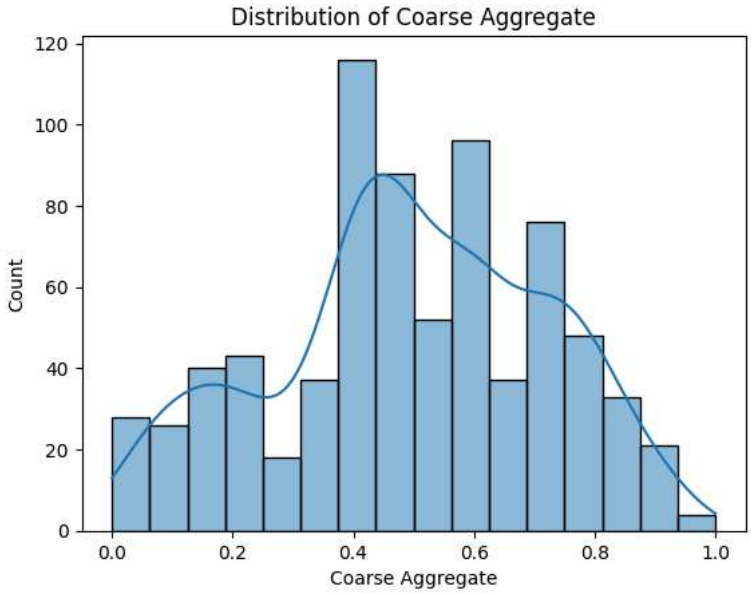

Distribution of Blast Furnace Slag
Count
Blast Furnace Slag
0
50
100
150
200
250
300
350
400
0.0
0.2
0.4
0.6
0.8
1.0

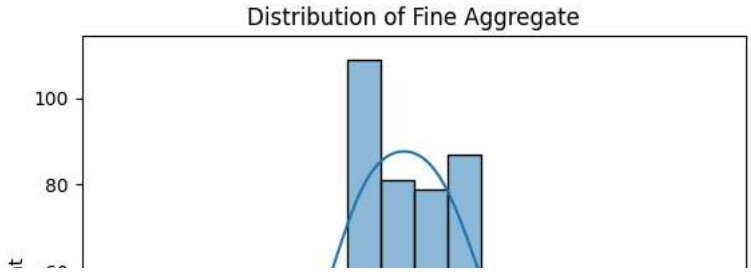

Distribution of Fly Ash
Count
Fly Ash
0
50
100
150
200
250
300
350
0.0
0.2
0.4
0.6
0.8
1.0

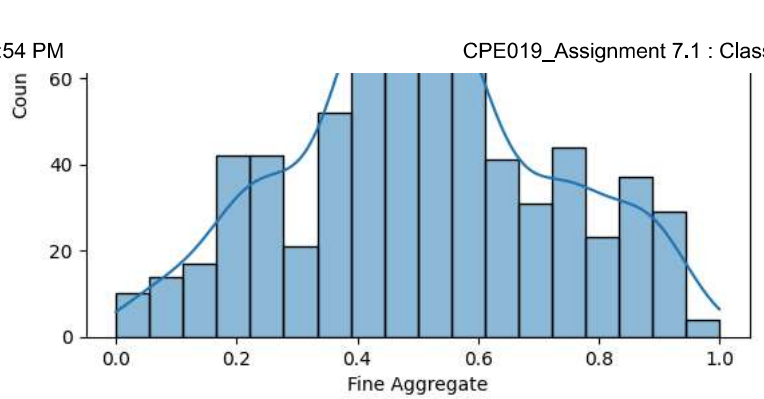

Distribution of Water
120

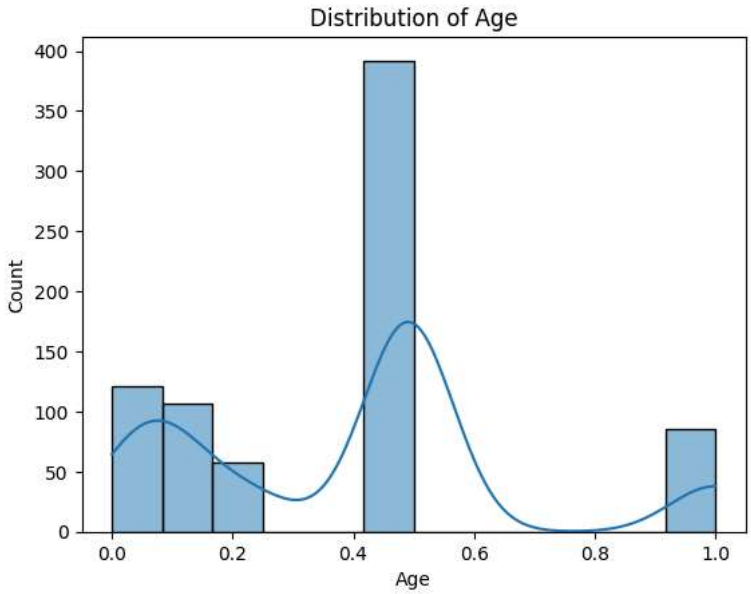

Count
Water


Distribution of Superplasticizer
Count
Superplasticizer


Distribution of Coarse Aggregate
Count
Coarse Aggregate

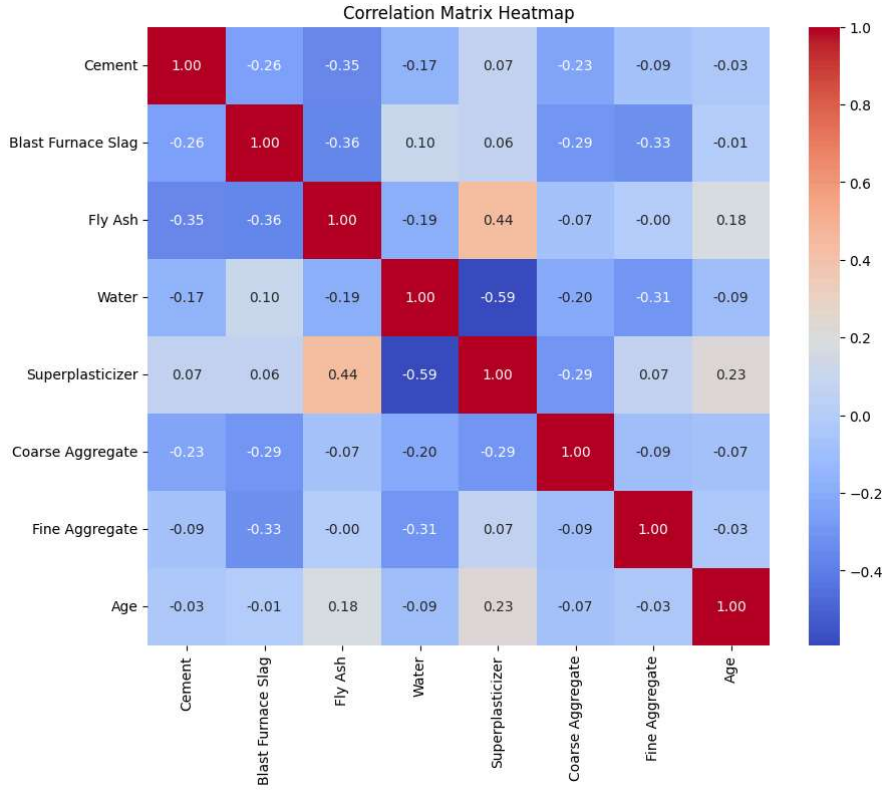

Distribution of Fine Aggregate



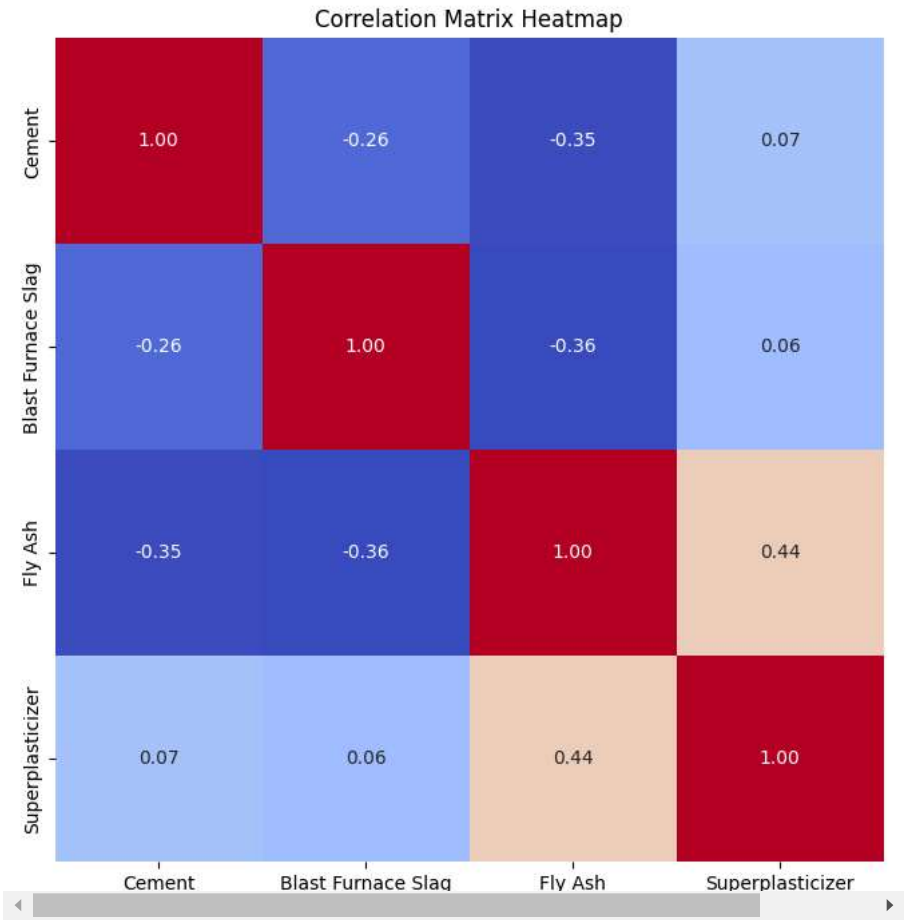


Remarks: The distribution of the features are in an almost bell curved as the data shows to have higher counts in the middle point of the X axis.

```
# Correlation Analysis
correlation_matrix = X_standardized_df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```




```
#Separate features and target variable
X = X_standardized_df.drop(['Coarse Aggregate', 'Fine Aggregate','Age','Water'],axis=1)
y = y_standardized_df['Concrete compressive strength']


# Correlation Analysis
correlation_matrix = X.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```



Remarks: Removal of columns that no correalation in the data.

Regression Modeling

```
def baseline_model():
    # create model
  model = Sequential()
  model.add(Dense(4, input_shape=(4,), kernel_initializer='normal', activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model


#evaluate model
estimator = KerasRegressor(model=baseline_model, epochs=100, batch_size=8)
kfold = KFold(n_splits=10)
results = cross_val_score(estimator, X, y, cv=kfold, scoring='neg_mean_squared_error')
```

```
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 21/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 22/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 23/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 24/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 25/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 26/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 27/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 28/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 29/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0446
Epoch 30/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 31/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 32/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 33/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 34/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0446
Epoch 35/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 36/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
Epoch 37/100
86/86 [==============================] - 0s 3ms/step - loss: 0.0446
Epoch 38/100
86/86 [==============================] - 0s 3ms/step - loss: 0.0445
Epoch 39/100
86/86 [==============================] - 0s 2ms/step - loss: 0.0445
```

```
print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Baseline: -0.02 (0.01) MSE
```

Remarks: The result reports the mean squared error including the average and standard deviation which the data can go through the unseen data after processing.

```
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=baseline_model, epochs=50, batch_size=8)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, y, cv=kfold, scoring='neg_mean_squared_error')
```

```
86/86 [==============================] - 0s 2ms/step - loss: 0.0235
Epoch 6/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0231
Epoch 7/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0228
Epoch 8/50
86/86 [==============================] - 0s 2ms/step - loss: 0.0226
Epoch 9/50
86/86 [==============================] - 0s 2ms/step - loss: 0.0225
Epoch 10/50
86/86 [==============================] - 0s 2ms/step - loss: 0.0226
Epoch 11/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0224
Epoch 12/50
86/86 [==============================] - 0s 2ms/step - loss: 0.0224
Epoch 13/50
86/86 [==============================] - 0s 2ms/step - loss: 0.0223
Epoch 14/50
86/86 [==============================] - 0s 2ms/step - loss: 0.0224
Epoch 15/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0223
Epoch 16/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0222
Epoch 17/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0223
Epoch 18/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0223
Epoch 19/50
86/86 [==============================] - 0s 3ms/step - loss: 0.0223
Epoch 20/50
86/86 [==============================] - 0s 2ms/step - loss: 0.0223
Epoch 21/50
```

```
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Standardized: -0.03 (0.02) MSE
```

Remarks: even after running the standardized data this shows that the training model can handle the data regression at an excellent results

```
def larger_model():
  # create model
  model = Sequential()
  model.add(Dense(4, input_shape=(4,), kernel_initializer='normal', activation='relu'))
  model.add(Dense(2, kernel_initializer='normal', activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))
  # Compile model
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model


# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=larger_model, epochs=50, batch_size=8)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, y, cv=kfold, scoring='neg_mean_squared_error')
```

```
86/86 [==============================] - 0s 2ms/step - loss: 0.0235
Epoch 5/50
```