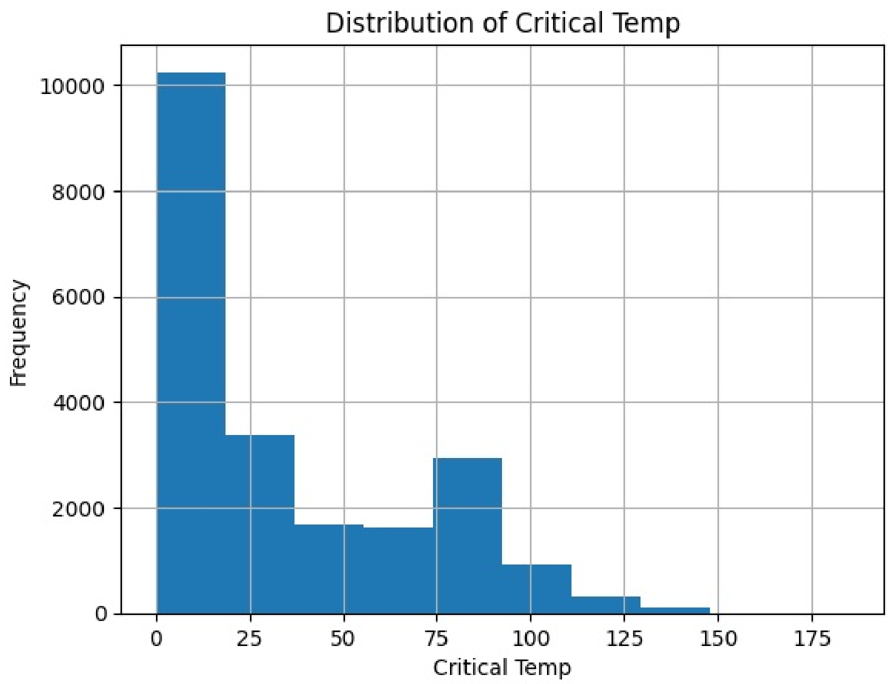
In [106…
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import precision_score, recall_score, f1_score, matthews_corrcoef, confusion_matrix, accur
from sklearn.metrics import explained_variance_score, mean_squared_error, max_error, mean_absolute_error
from scipy.stats import pearsonr
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

def printRegStatistics(truth, preds):
    print("The RVE is: ", explained_variance_score(truth, preds))
    print("The rmse is: ", mean_squared_error(truth, preds, squared=False))
    corr, pval = pearsonr(truth, preds)
    print("The Correlation Score is is: %6.4f (p-value=%e)\n"%(corr,pval))
    print("The Maximum Error is is: ", max_error(truth, preds))
    print("The Mean Absolute Error is: ", mean_absolute_error(truth, preds))
```

In [107…
```python
df = pd.read_csv("HA1-DatasetScaled.tsv", sep="\t")
```

Understanding dataset

In [111…
```python
import matplotlib.pyplot as plt
df['critical_temp'].hist()
plt.title('Distribution of Critical Temp')
plt.xlabel('Critical Temp')
plt.ylabel('Frequency')
plt.show()
```



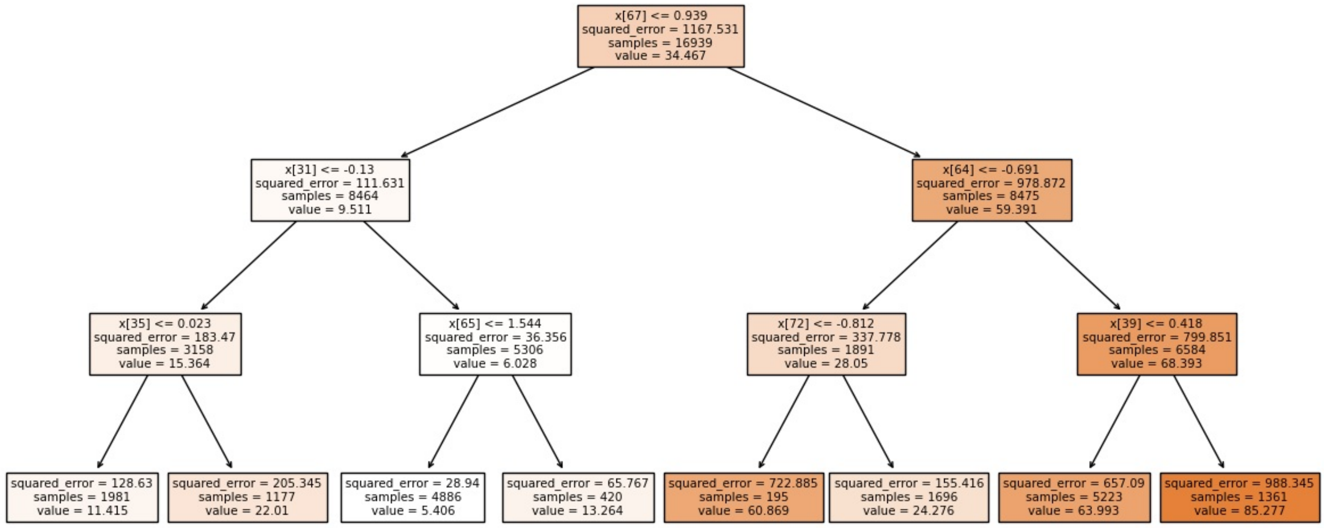
Data Processing

In [112…
```python
# Load the dataset
df = pd.read_csv("HA1-DatasetScaled.tsv", sep="\t")

# Separate features (X) and the target (y)
X = df.drop(columns=['critical_temp'])
y = df['critical_temp']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Objective 1 best regression model for "critical_temp"

In [113…
```python
# Regression model using Decision Tree

mdl = DecisionTreeRegressor(max_depth=3)
```

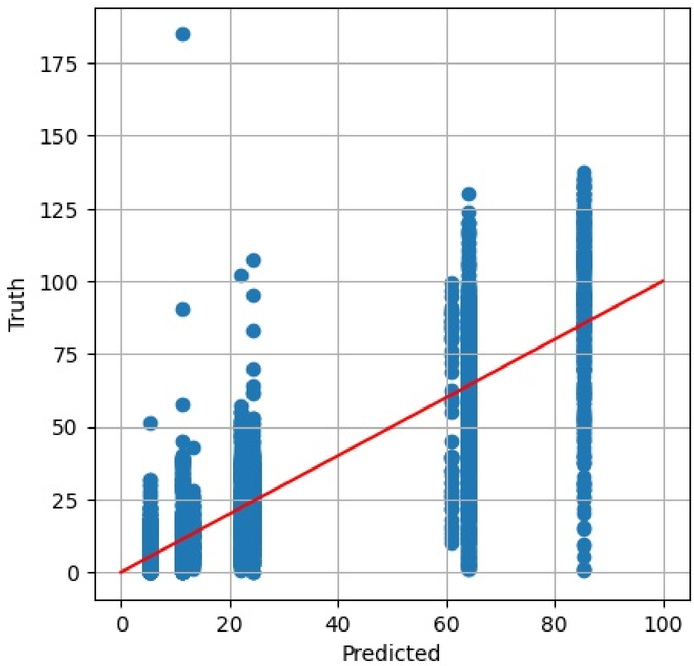
```
mdl.fit(X_train, y_train)


plt.figure(figsize=(15, 7))
plot_tree(mdl, filled=True)
plt.show()
```



In [114…

```
preds=mdl.predict(X_test)
explained_variance_score(y_test, preds)

print("The RVE is: ", explained_variance_score(y_test, preds))
print("The rmse is: ", mean_squared_error(y_test, preds, squared=False))
corr, pval=pearsonr(y_test, preds)
print("The Correlation Score is is: %6.4f (p-value=%e)\n"%(corr,pval))

print("The Maximum Error is is: ", max_error(y_test, preds))
print("The Mean Absolute Error is: ", mean_absolute_error(y_test, preds))
reds=mdl.predict(X_test)
plt.figure(figsize=(5,5))
plt.scatter(preds, y_test)
plt.plot((0, 100), (0,100), c="r")
plt.grid()
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

```
The RVE is:  0.707829542939332
The rmse is:  18.70295266891859
The Correlation Score is is: 0.8414 (p-value=0.000000e+00)

The Maximum Error is is:  173.5853332155477
The Mean Absolute Error is:  13.092702793498137
```



In [115…

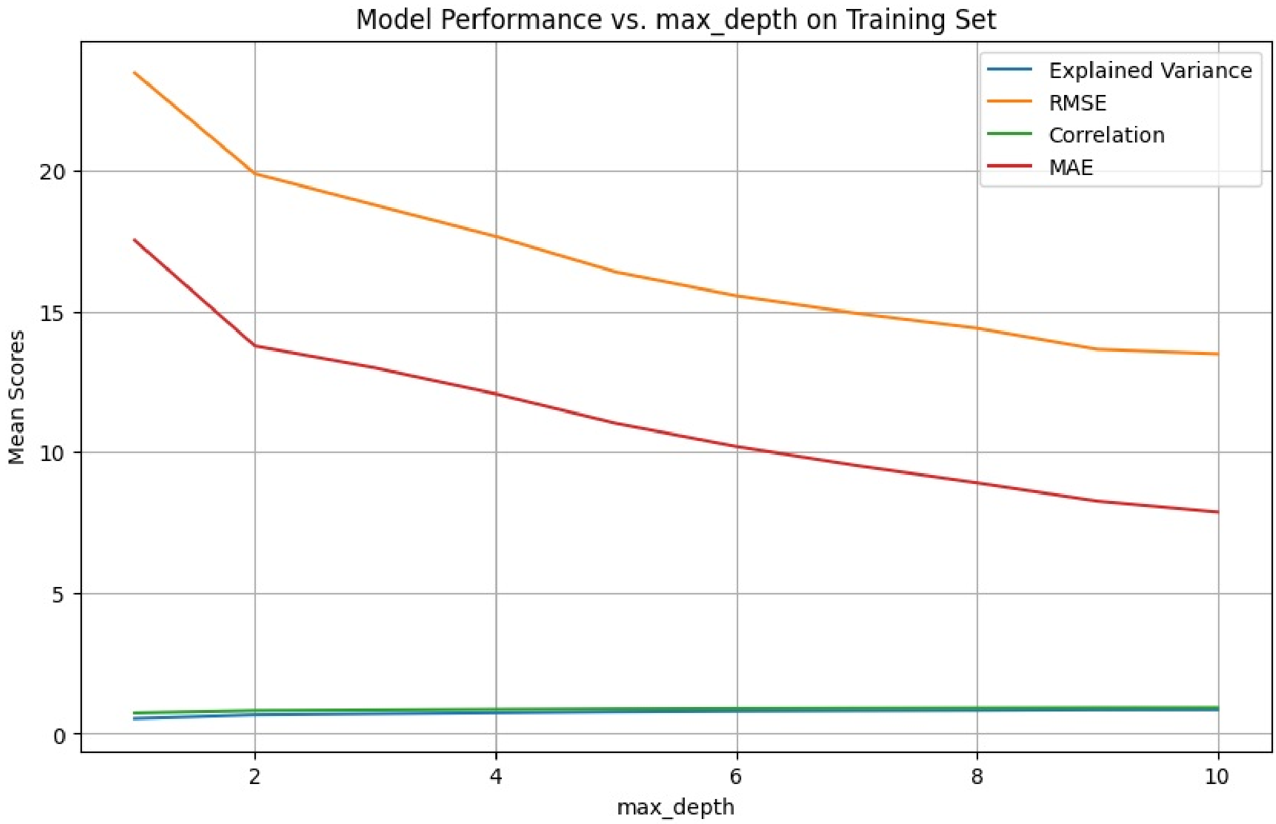
```
kf = KFold(n_splits=5, shuffle=True, random_state=22)
```

```python
explained_variances = []
rmse_scores = []
correlations = []
max_errors = []
mae_scores = []

for train_idx, test_idx in kf.split(X):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    mdl = DecisionTreeRegressor(max_depth=3)
    mdl.fit(X_train, y_train)

    preds = mdl.predict(X_test)

    explained_variances.append(explained_variance_score(y_test, preds))
    rmse_scores.append(mean_squared_error(y_test, preds, squared=False))
    corr, _ = pearsonr(y_test, preds)
    correlations.append(corr)
    max_errors.append(max_error(y_test, preds))
    mae_scores.append(mean_absolute_error(y_test, preds))

print("Mean Explained Variance: ", sum(explained_variances) / len(explained_variances))
print("Mean RMSE: ", sum(rmse_scores) / len(rmse_scores))
print("Mean Correlation Score: ", sum(correlations) / len(correlations))
print("Mean Max Error: ", sum(max_errors) / len(max_errors))
print("Mean MAE: ", sum(mae_scores) / len(mae_scores))
```

```
Mean Explained Variance:  0.7021983580109994
Mean RMSE:  18.69131270993887
Mean Correlation Score:  0.8380431124634333
Mean Max Error:  108.07777300892226
Mean MAE:  13.042101141667633
```

In [116…

```python
X_TR, X_IVS, y_TR, y_IVS = train_test_split(X, y, test_size=0.25, random_state=1337)

max_depth_values = range(1, 11)

mean_explained_variances = []
mean_rmse_scores = []
mean_correlations = []
mean_mae_scores = []

kf = KFold(n_splits=5, shuffle=True, random_state=1337)

for max_depth in max_depth_values:
    explained_variances = []
    rmse_scores = []
    correlations = []
    mae_scores = []

    for train_idx, test_idx in kf.split(X_train):
        X_train_fold, X_test_fold = X_train.iloc[train_idx], X_train.iloc[test_idx]
        y_train_fold, y_test_fold = y_train.iloc[train_idx], y_train.iloc[test_idx]

        mdl = DecisionTreeRegressor(max_depth=max_depth)
        mdl.fit(X_train_fold, y_train_fold)
        preds = mdl.predict(X_test_fold)

        explained_variances.append(explained_variance_score(y_test_fold, preds))
        rmse_scores.append(mean_squared_error(y_test_fold, preds, squared=False))
        corr, _ = pearsonr(y_test_fold, preds)
        correlations.append(corr)
        mae_scores.append(mean_absolute_error(y_test_fold, preds))

    mean_explained_variances.append(np.mean(explained_variances))
    mean_rmse_scores.append(np.mean(rmse_scores))
    mean_correlations.append(np.mean(correlations))
    mean_mae_scores.append(np.mean(mae_scores))

plt.figure(figsize=(10, 6))
plt.plot(max_depth_values, mean_explained_variances, label='Explained Variance')
plt.plot(max_depth_values, mean_rmse_scores, label='RMSE')
plt.plot(max_depth_values, mean_correlations, label='Correlation')
plt.plot(max_depth_values, mean_mae_scores, label='MAE')
plt.xlabel('max_depth')
plt.ylabel('Mean Scores')
plt.legend()
plt.title('Model Performance vs. max_depth on Training Set')
plt.grid(True)
plt.show()
```



In [117…

```python
kf = KFold(n_splits=5, shuffle=True)

explained_variances = []
rmse_scores = []
correlations = []
max_errors = []
mae_scores = []

for train_idx, test_idx in kf.split(X_TR):
    X_train, X_test = X_TR.iloc[train_idx], X_TR.iloc[test_idx]
    y_train, y_test = y_TR.iloc[train_idx], y_TR.iloc[test_idx]

    mdl = DecisionTreeRegressor(max_depth=10)
    mdl.fit(X_train, y_train)

    preds = mdl.predict(X_test)

    explained_variances.append(explained_variance_score(y_test, preds))
    rmse_scores.append(mean_squared_error(y_test, preds, squared=False))
    corr, _ = pearsonr(y_test, preds)
    correlations.append(corr)
    max_errors.append(max_error(y_test, preds))
    mae_scores.append(mean_absolute_error(y_test, preds))

print("Mean Explained Variance: ", sum(explained_variances) / len(explained_variances))
print("Mean RMSE: ", sum(rmse_scores) / len(rmse_scores))
print("Mean Correlation Score: ", sum(correlations) / len(correlations))
print("Mean Max Error: ", sum(max_errors) / len(max_errors))
print("Mean MAE: ", sum(mae_scores) / len(mae_scores))
```

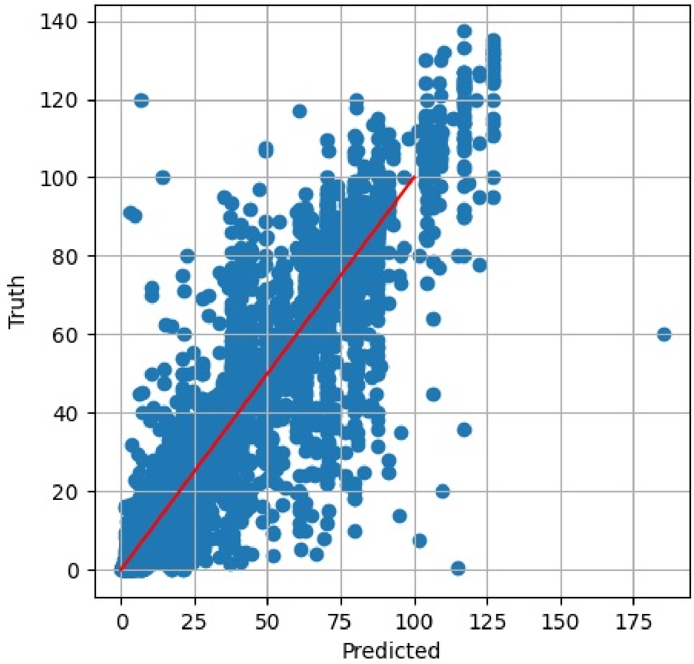
```
Mean Explained Variance:  0.8532852373692009
Mean RMSE:  13.162851824147154
Mean Correlation Score:  0.9242289019854206
Mean Max Error:  125.82211376674547
Mean MAE:  7.825935768633552
```

In [119…

```python
mdl = DecisionTreeRegressor(max_depth=10)
mdl.fit(X_TR, y_TR)

preds_IVS = mdl.predict(X_IVS)

explained_variance_IVS = explained_variance_score(y_IVS, preds_IVS)
rmse_IVS = mean_squared_error(y_IVS, preds_IVS, squared=False)
corr_IVS, _ = pearsonr(y_IVS, preds_IVS)
max_error_IVS = max_error(y_IVS, preds_IVS)
mae_IVS = mean_absolute_error(y_IVS, preds_IVS)

print("Explained Variance on IVS: ", explained_variance_IVS)
print("RMSE on IVS: ", rmse_IVS)
print("Correlation Score on IVS: ", corr_IVS)
print("Max Error on IVS: ", max_error_IVS)
print("MAE on IVS: ", mae_IVS)
```

```
plt.figure(figsize=(5, 5))
plt.scatter(preds_IVS, y_IVS)
plt.plot((0, 100), (0, 100), c="r")
plt.grid()
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

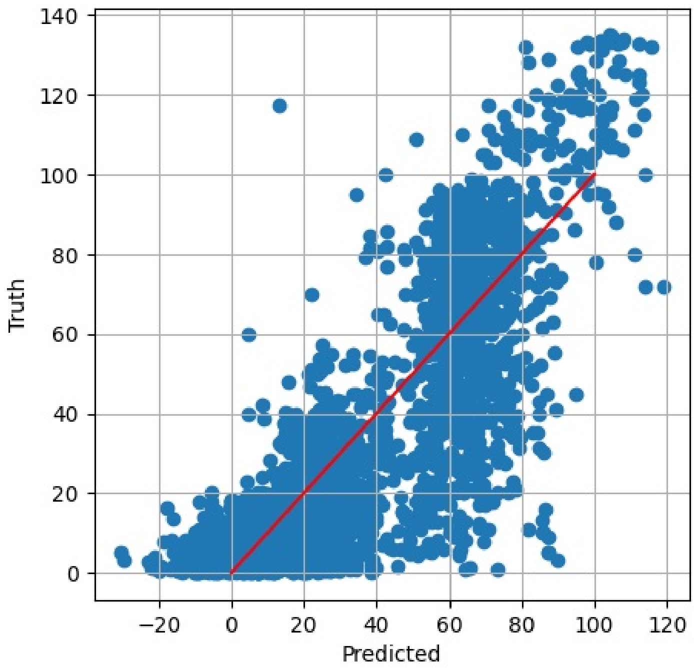
```
Explained Variance on IVS:  0.8610343821946201
RMSE on IVS:  12.6182763168542
Correlation Score on IVS:  0.9284560131250607
Max Error on IVS:  125.0
MAE on IVS:  7.418958073476574
```



Building and Evaluating Linear Regression Model

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

# Linear Regression model
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)

# Evaluating the Linear Regression model
linear_reg_preds = linear_reg.predict(X_test)

linear_reg_rve = explained_variance_score(y_test, linear_reg_preds)
linear_reg_rmse = mean_squared_error(y_test, linear_reg_preds, squared=False)
linear_reg_corr, _ = pearsonr(y_test, linear_reg_preds)
linear_reg_max_error = max_error(y_test, linear_reg_preds)
linear_reg_mae = mean_absolute_error(y_test, linear_reg_preds)


print("Linear Regression Model Metrics:")
print("RVE: ", linear_reg_rve)
print("RMSE: ", linear_reg_rmse)
print("Correlation Score: ", linear_reg_corr)
print("Max Error: ", linear_reg_max_error)
print("MAE: ", linear_reg_mae)

# predicted vs. actual values for Linear Regression
plt.figure(figsize=(5, 5))
plt.scatter(linear_reg_preds, y_test)
plt.plot((0, 100), (0, 100), c="r")
plt.grid()
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

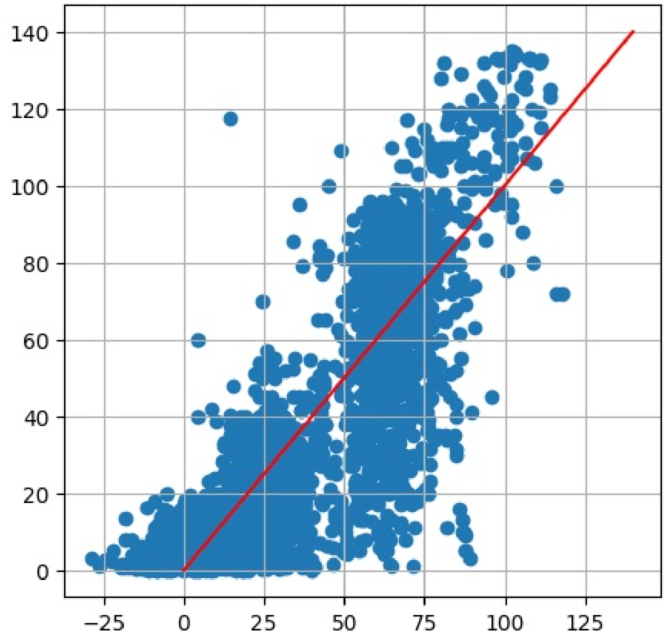
```
Linear Regression Model Metrics:
RVE:  0.7158158733285267
RMSE:  18.193608538894956
Correlation Score:  0.84651218226191
Max Error:  104.36223500698324
MAE:  13.773812503084889
```



In [ ]:
```
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.linear_model import Ridge, Lasso
from statsmodels.api import OLS, add_constant

rveCorrelation = 0
alphaMaior = 0
for i in range(5, 50, 5):
    ridge = Ridge(alpha=i, max_iter=9999999).fit(X_train, y_train)
    preds=ridge.predict(X_test)
    if (explained_variance_score(y_test, preds)>rveCorrelation):
        rveCorrelation = explained_variance_score(y_test, preds)
        alphaMaior = i
ridge = Ridge(alpha=alphaMaior, max_iter=9999999).fit(X_train, y_train)
```

In [ ]:
```
preds=ridge.predict(X_test)
print("The best result was with alpha = " + str(alphaMaior) + ":")
printRegStatistics(y_test, preds)
rr_rve = explained_variance_score(y_test, preds)
rr_rmse = mean_squared_error(y_test, preds, squared=False)
corr1, pval = pearsonr(y_test, preds)
rr_CS = corr1
rr_pval = pval
rr_ME = max_error(y_test, preds)
rr_MAE = mean_absolute_error(y_test, preds)

plt.figure(figsize=(5,5))
plt.scatter(preds, y_test)
plt.plot((0, 140), (0, 140), c="r")
plt.grid()
plt.show()
```

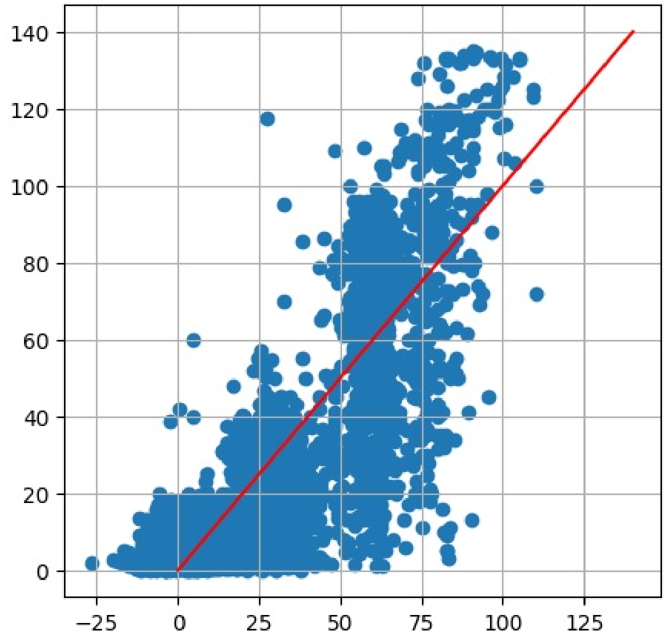
```
The best result was with alpha = 5:
The RVE is:  0.7146545501553065
The rmse is:  18.231207117136577
The Correlation Score is is: 0.8457 (p-value=0.000000e+00)

The Maximum Error is is:  103.04175599991952
The Mean Absolute Error is:  13.792493955210666
```



In [ ]:
```
rveCorrelation = 0
alphaMaior = 0

for i in range(2, 20, 2):
    L = Lasso(alpha=i/10, max_iter=9999999).fit(X_train, y_train)
    preds=L.predict(X_test)
    if (explained_variance_score(y_test, preds)>rveCorrelation):
        rveCorrelation = explained_variance_score(y_test, preds)
        alphaMaior = i/10

L = Lasso(alpha=alphaMaior, max_iter=9999999).fit(X_train, y_train)
```

In [ ]:
```
preds=L.predict(X_test)
print("O melhor resultado foi com alpha = " + str(alphaMaior) + ":")
printRegStatistics(y_test, preds)
lr_rve = explained_variance_score(y_test, preds)
lr_rmse = mean_squared_error(y_test, preds, squared=False)
corr2, pval = pearsonr(y_test, preds)
lr_CS = corr2
lr_pval = pval
lr_ME = max_error(y_test, preds)
lr_MAE = mean_absolute_error(y_test, preds)

plt.figure(figsize=(5,5))
plt.scatter(preds, y_test)
plt.plot((0, 140), (0, 140), c="r")
plt.grid()
plt.show()
```

```
O melhor resultado foi com alpha = 0.2:
The RVE is:  0.6784157671444373
The rmse is:  19.35049749812771
The Correlation Score is is: 0.8237 (p-value=0.000000e+00)

The Maximum Error is is:  90.12740155540378
The Mean Absolute Error is:  14.81665006525913
```



In [ ]:
```
# performance metrics for Linear Regression
linear_reg_rve = explained_variance_score(y_test, linear_reg_preds)
linear_reg_rmse = mean_squared_error(y_test, linear_reg_preds, squared=False)
linear_reg_corr, _ = pearsonr(y_test, linear_reg_preds)
linear_reg_max_error = max_error(y_test, linear_reg_preds)
linear_reg_mae = mean_absolute_error(y_test, linear_reg_preds)


# performance metrics for Linear Regression with Ridge
rr_rve = explained_variance_score(y_test, preds)
rr_rmse = mean_squared_error(y_test, preds, squared=False)
corr1, pval = pearsonr(y_test, preds)
rr_ME = max_error(y_test, preds)
rr_MAE = mean_absolute_error(y_test, preds)

# performance metrics for Linear Regression with Lasso
lr_rve = explained_variance_score(y_test, preds)
lr_rmse = mean_squared_error(y_test, preds, squared=False)
corr2, pval = pearsonr(y_test, preds)
lr_ME = max_error(y_test, preds)
lr_MAE = mean_absolute_error(y_test, preds)

#  calculate performance metrics
mdl_preds = mdl.predict(X_test)
mdl_rve = explained_variance_score(y_test, mdl_preds)
mdl_rmse = mean_squared_error(y_test, mdl_preds, squared=False)
mdl_corr, _ = pearsonr(y_test, mdl_preds)
mdl_max_error = max_error(y_test, mdl_preds)
mdl_mae = mean_absolute_error(y_test, mdl_preds)

# Compare the models using the metrics (rve)
if mdl_rve > linear_reg_rve:
    better_model = "Decision Tree Regression"
else:
    better_model = "Linear Regression"

# Print the results
print("Comparison of Models:")
print("Linear Regression Model Metrics:")
print("RVE: ", linear_reg_rve)
print("RMSE: ", linear_reg_rmse)
print("Correlation Score: ", linear_reg_corr)
print("Max Error: ", linear_reg_max_error)
print("MAE: ", linear_reg_mae)

print("\nLinear Regression Ridge:")
print("RVE: ", rr_rve)
print("RMSE: ", rr_rmse)
print("Correlation Score: ", corr1)
print("Max Error: ", rr_ME)
print("MAE: ", rr_MAE)
```

```
print("\nLinear Regression Lasso:")
print("RVE: ", lr_rve)
print("RMSE: ", lr_rmse)
print("Correlation Score: ", corr2)
print("Max Error: ", lr_ME)
print("MAE: ", lr_MAE)


print("\nDecision Tree Regression Model Metrics:")
print("RVE: ", mdl_rve)
print("RMSE: ", mdl_rmse)
print("Correlation Score: ", mdl_corr)
print("Max Error: ", mdl_max_error)
print("MAE: ", mdl_mae)

print("\nConclusion:")
print(f"{better_model} is the better model based on the explained variance (RVE) metric.")
```

```
Comparison of Models:
Linear Regression Model Metrics:
RVE:  0.7158158733285267
RMSE:  18.193608538894956
Correlation Score:  0.84651218226191
Max Error:  104.36223500698324
MAE:  13.773812503084889

Linear Regression Ridge:
RVE:  0.6784157671444373
RMSE:  19.35049749812771
Correlation Score:  0.8236611656635581
Max Error:  90.12740155540378
MAE:  14.81665006525913

Linear Regression Lasso:
RVE:  0.6784157671444373
RMSE:  19.35049749812771
Correlation Score:  0.8236611656635581
Max Error:  90.12740155540378
MAE:  14.81665006525913

Decision Tree Regression Model Metrics:
RVE:  0.9160400853256694
RMSE:  9.88466780587543
Correlation Score:  0.9570998437963399
Max Error:  68.80432781456955
MAE:  6.028217792918141

Conclusion:
Decision Tree Regression is the better model based on the explained variance (RVE) metric.
```

Objective 2 best binary classification model

In [ ]:

```
# Loading the dataset
df = pd.read_csv("HA1-DatasetScaled.tsv", sep="\t")

# Creating a binary target variable
df['binary_target'] = (df['critical_temp'] >= 80.0).astype(int)

# Spliting data into feature (X) and target (y)
X = df.drop(columns=['critical_temp', 'binary_target'])
y = df['binary_target']

# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=22)
```

In [ ]:

```
# Classification model using Logistic Regression
logistic_reg = LogisticRegression(max_iter=1000, random_state=22)
logistic_reg.fit(X_train, y_train)
logistic_reg_preds = logistic_reg.predict(X_test)

# Calculate performance metrics for Logistic Regression
logistic_reg_accuracy = accuracy_score(y_test, logistic_reg_preds)
logistic_reg_precision = precision_score(y_test, logistic_reg_preds)
logistic_reg_recall = recall_score(y_test, logistic_reg_preds)
logistic_reg_f1 = f1_score(y_test, logistic_reg_preds)
logistic_reg_mcc = matthews_corrcoef(y_test, logistic_reg_preds)
logistic_reg_confusion = confusion_matrix(y_test, logistic_reg_preds)

print("Comparison of Classification Models:")
print("Logistic Regression Model Metrics:")
print(f"Accuracy: %7.4f" % logistic_reg_accuracy)
print(f"Precision: %7.4f" % logistic_reg_precision)
print(f"Recall: %7.4f" % logistic_reg_recall)
print(f"F1 Score: %7.4f" % logistic_reg_f1)
print(f"Matthews Correlation Coefficient: %7.4f" % logistic_reg_mcc)
print()
```

```
print("Confusion Matrix:")
pd.DataFrame(confusion_matrix(y_test, logistic_reg_preds))
```

```
Comparison of Classification Models:
Logistic Regression Model Metrics:
Accuracy:  0.9011
Precision:  0.7108
Recall:  0.6425
F1 Score:  0.6749
Matthews Correlation Coefficient:  0.6179

Confusion Matrix:
```

Out[ ]:

	0	1
0	3381	177
1	242	435

In [ ]:

```
mdl = DecisionTreeClassifier(min_samples_leaf=5)
mdl.fit(X_train, y_train)
preds = mdl.predict(X_test)

print("The Precision is: %7.4f" % precision_score(y_test, preds))
print("The Recall is: %7.4f" % recall_score(y_test, preds))
print("The F1 score is: %7.4f" % f1_score(y_test, preds))
print("The Matthews correlation coefficient is: %7.4f" % matthews_corrcoef(y_test, preds))
print()
print("This is the Confusion Matrix")
pd.DataFrame(confusion_matrix(y_test, preds))
```

```
The Precision is:  0.7908
The Recall is:  0.7873
The F1 score is:  0.7890
The Matthews correlation coefficient is:  0.7490

This is the Confusion Matrix
```

Out[ ]:

	0	1
0	3417	141
1	144	533

In [ ]:

```
kf = KFold(n_splits=5, shuffle=True, random_state=23)
kf.get_n_splits(X)
TRUTH_nfold=None
PREDS_nfold=None
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    mdl = DecisionTreeClassifier()
    mdl.fit(X_train, y_train)
    preds = mdl.predict(X_test)
    if TRUTH_nfold is None:
        PREDS_nfold=preds
        TRUTH_nfold=y_test
    else:
        PREDS_nfold=np.hstack((PREDS_nfold, preds))
        TRUTH_nfold=np.hstack((TRUTH_nfold, y_test))

print("The Precision is: %7.4f" % precision_score(TRUTH_nfold, PREDS_nfold))
print("The Recall is: %7.4f" % recall_score(TRUTH_nfold, PREDS_nfold))
print("The F1 score is: %7.4f" % f1_score(TRUTH_nfold, PREDS_nfold))
print("The Matthews correlation coefficient is: %7.4f" % matthews_corrcoef(TRUTH_nfold, PREDS_nfold))
```

```
The Precision is:  0.7977
The Recall is:  0.7999
The F1 score is:  0.7988
The Matthews correlation coefficient is:  0.7582
```

In [ ]:

```
X_TR, X_IVS, y_TR, y_IVS = train_test_split(X, y, test_size=0.25, random_state=1337)

n_splits_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

best_n_splits = None
best_f1_score = 0.0

for n_splits in n_splits_values:
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=23)
    TRUTH_nfold = None
    PREDS_nfold = None

    for train_index, test_index in kf.split(X_TR):  # Use X_TR and y_TR for training
        X_train, X_test = X_TR.iloc[train_index], X_TR.iloc[test_index]
        y_train, y_test = y_TR.iloc[train_index], y_TR.iloc[test_index]

        mdl = DecisionTreeClassifier()
        mdl.fit(X_train, y_train)
```

```
        preds = mdl.predict(X_test)

        if TRUTH_nfold is None:
            PREDS_nfold = preds
            TRUTH_nfold = y_test
        else:
            PREDS_nfold = np.hstack((PREDS_nfold, preds))
            TRUTH_nfold = np.hstack((TRUTH_nfold, y_test))

    f1 = f1_score(TRUTH_nfold, PREDS_nfold)
    print(f"For n_splits = {n_splits}, F1 score is: {f1:.4f}")

    if f1 > best_f1_score:
        best_n_splits = n_splits
        best_f1_score = f1

print(f"The best n_splits is: {best_n_splits}")
```

```
For n_splits = 2, F1 score is: 0.7418
For n_splits = 3, F1 score is: 0.7656
For n_splits = 4, F1 score is: 0.7671
For n_splits = 5, F1 score is: 0.7743
For n_splits = 6, F1 score is: 0.7725
For n_splits = 7, F1 score is: 0.7786
For n_splits = 8, F1 score is: 0.7823
For n_splits = 9, F1 score is: 0.7819
For n_splits = 10, F1 score is: 0.7853
The best n_splits is: 10
```

In [ ]:

```
kf = KFold(n_splits=8, shuffle=True, random_state=23)
kf.get_n_splits(X)
TRUTH_nfold=None
PREDS_nfold=None
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    mdl = DecisionTreeClassifier()
    mdl.fit(X_train, y_train)
    preds = mdl.predict(X_test)
    if TRUTH_nfold is None:
        PREDS_nfold=preds
        TRUTH_nfold=y_test
    else:
        PREDS_nfold=np.hstack((PREDS_nfold, preds))
        TRUTH_nfold=np.hstack((TRUTH_nfold, y_test))

print("The Precision is: %7.4f" % precision_score(TRUTH_nfold, PREDS_nfold))
print("The Recall is: %7.4f" % recall_score(TRUTH_nfold, PREDS_nfold))
print("The F1 score is: %7.4f" % f1_score(TRUTH_nfold, PREDS_nfold))
print("The Matthews correlation coefficient is: %7.4f" % matthews_corrcoef(TRUTH_nfold, PREDS_nfold))
```

```
The Precision is:  0.8000
The Recall is:  0.8104
The F1 score is:  0.8052
The Matthews correlation coefficient is:  0.7656
```

In [ ]:

```
mdl = DecisionTreeClassifier()
mdl.fit(X_train, y_train)

preds_IVS = mdl.predict(X_IVS)

decision_tree_accuracy = accuracy_score(y_IVS, preds_IVS)
decision_tree_precision = precision_score(y_IVS, preds_IVS)
decision_tree_recall = recall_score(y_IVS, preds_IVS)
decision_tree_f1 = f1_score(y_IVS, preds_IVS)
decision_tree_mcc = matthews_corrcoef(y_IVS, preds_IVS)
decision_tree_confusion = confusion_matrix(y_IVS, preds_IVS)

print("\nDecision Tree Classification Model Metrics:")
print(f"Accuracy: %7.4f" % decision_tree_accuracy)
print(f"Precision: %7.4f" % decision_tree_precision)
print(f"Recall: %7.4f" % decision_tree_recall)
print(f"F1 Score: %7.4f" % decision_tree_f1)
print(f"Matthews Correlation Coefficient: %7.4f" % decision_tree_mcc)
print("\nConfusion Matrix:")
pd.DataFrame(confusion_matrix(y_IVS, preds_IVS))
```

```
Decision Tree Classification Model Metrics:
Accuracy:  0.9781
Precision:  0.9423
Recall:  0.9253
F1 Score:  0.9337
Matthews Correlation Coefficient:  0.9206

Confusion Matrix:
```

Out[ ]:

	0	1
0	4361	50
1	66	817

In [ ]:

```
# Comparing the models with accuracy metric
if logistic_reg_accuracy > decision_tree_accuracy:
    better_classification_model = "Logistic Regression"
else:
    better_classification_model = "Decision Tree Classification"
```

In [ ]:

```
print("\nConclusion:")
print(f"{better_classification_model} is the better classification model based on the accuracy metric.")

# Visualization of results with a bar chart
models = ["Logistic Regression", "Decision Tree Classification"]
accuracy_scores = [logistic_reg_accuracy, decision_tree_accuracy]

plt.bar(models, accuracy_scores, color=['green', 'red'])
plt.ylabel('Accuracy')
plt.title('Comparison of Classification Models')
plt.show()
```

```
Conclusion:
Decision Tree Classification is the better classification model based on the accuracy metric.
```