E11_Jupyter_Notebook._22B3914_22B3976_22B4217

November 13, 2023

22B3914 22B3976 22B4217

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

[ ]:
```
df_og=pd.read_csv('e11.csv')
control_parameters = ['c26', 'c27', 'c28', 'c29', 'c30', 'c31', 'c32',
'c33', 'c39', 'c139', 'c142', 'c143', 'c155', 'c156', 'c157', 'c158', 'c160',␣
 ↪'c161', 'c162', 'c163']
vibrations = [ 'c51', 'c52', 'c53', 'c54']
```

1 Data Cleaning

[ ]:
```
cols_removed = []
for column in df_og.columns:
    if df_og[column].isnull().all() and column not in vibrations and column not␣
 ↪in control_parameters:
        cols_removed.append(column)
        df_og = df_og.drop(column,axis=1)
print("Removed columns")
print(cols_removed)
```

```
Removed columns
['c199', 'c202', 'c204', 'c226', 'c229']
```

[ ]:
```
def replace_with_random(df):
    for col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        mask = df[col].isna()
        mean = df[col].mean()
        std = df[col].std()
        random_values = np.random.normal(mean, std, size=sum(mask))
        df[col][mask] = random_values
    return df
```

[ ]:
```
def IQR_Cleaning(df):
    iqr_factor = 1.5
    cols_to_be_dropped = []
    rows_outlier_count = np.zeros(1025, dtype=int)
    # Iterate through each column in the input DataFrame
    for column in df.columns:
        # Check if the column contains numeric data
        if pd.api.types.is_numeric_dtype(df[column]):
            # Calculate the first and third quartiles
            Q1 = df[column].quantile(0.25)
            Q3 = df[column].quantile(0.75)

            # Calculate the IQR for the column
            IQR = Q3 - Q1

            lower_bound = Q1 - iqr_factor * IQR
            upper_bound = Q3 + (2 * iqr_factor * IQR)

            upper_array = np.where(df[column]>=upper_bound)
            lower_array = []
            if upper_bound != lower_bound:
                lower_array = np.where(df[column]<=lower_bound)
            elif (column not in vibrations) and (column not in␣
 ↪control_parameters):
                cols_to_be_dropped.append(column)
            rows_outlier_count[upper_array] += 1
            rows_outlier_count[lower_array] += 1
        else:
            print("Error in column" + column)
    rows_to_be_removed = np.where(rows_outlier_count > 12)[0]
    print("Columns dropped are")
    print(cols_to_be_dropped)
    print(f"No of rows dropped are {len(rows_to_be_removed)}")
    df1= df.drop(rows_to_be_removed,axis=0)
    return df1.drop(cols_to_be_dropped,axis=1)
```

[ ]:
```
df_filled_missing = replace_with_random(df_og)
```

[ ]:
```
cleaned_df = IQR_Cleaning(df_filled_missing)
```

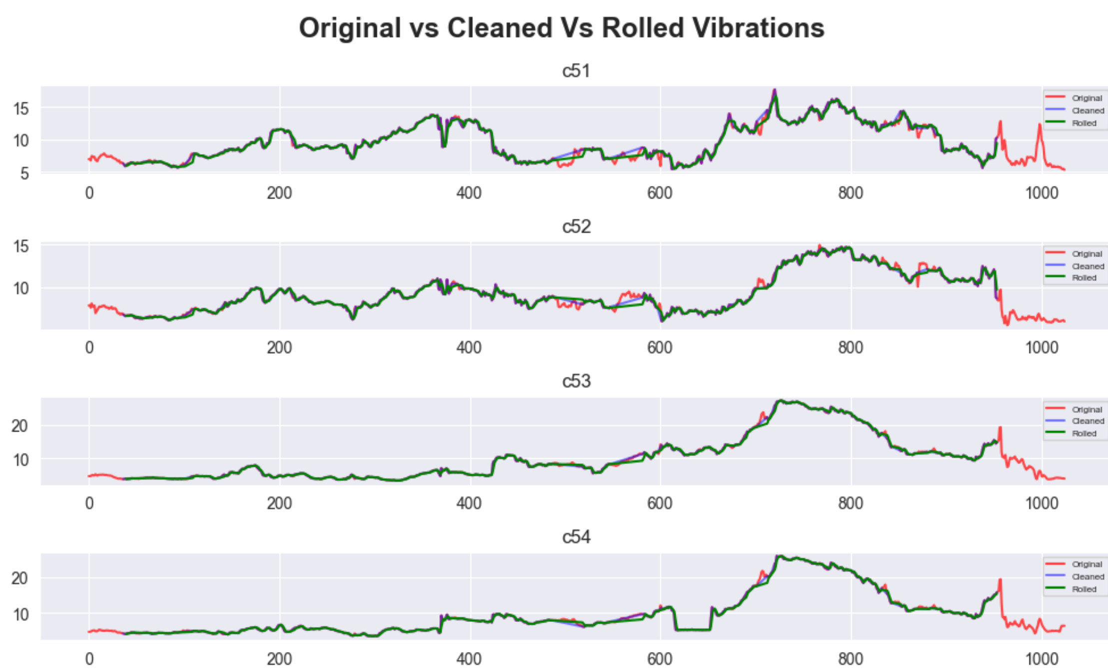
```
Columns dropped are
['c2', 'c82', 'c110', 'c168', 'c169', 'c170', 'c171']
No of rows dropped are 308
```

[ ]:
```
rolled_df = cleaned_df.copy()
for column in cleaned_df.columns:
    window_size = 3
```

```
    if pd.api.types.is_numeric_dtype(cleaned_df[column]):
        rolled_df[column] = cleaned_df[column].rolling(window=window_size).
 ↪mean()

rolled_df = rolled_df.dropna()
```

[ ]:
```
rolled_df.to_csv("IQRCleaned_Rolled_Data.csv")
```

[ ]:
```
fig_og_v, ax_og_v = plt.subplots(4, figsize = (10,6))

for i in range(4):
    ax_og_v[i].plot(np.
 ↪arange(len(df_og)),df_og[vibrations[i]],color='red',label='Original',alpha=0.
 ↪7)
    ax_og_v[i].plot(cleaned_df.
 ↪index,cleaned_df[vibrations[i]],color='blue',label='Cleaned',alpha=0.5)
    ax_og_v[i].plot(rolled_df.
 ↪index,rolled_df[vibrations[i]],color='green',label='Rolled',alpha=1)
    ax_og_v[i].set_title(vibrations[i])
    ax_og_v[i].legend(fontsize='xx-small')
fig_og_v.suptitle('Original vs Cleaned Vs Rolled Vibrations',fontsize =␣
 ↪'xx-large',
             weight = 'extra bold')
fig_og_v.tight_layout()
# plt.savefig('Original_vs_Cleaned_vs_Rolled_Vibrations.png', dpi = 300)
plt.show()
```



2 Q1

2.1 Trying MLR

[ ]:
```
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

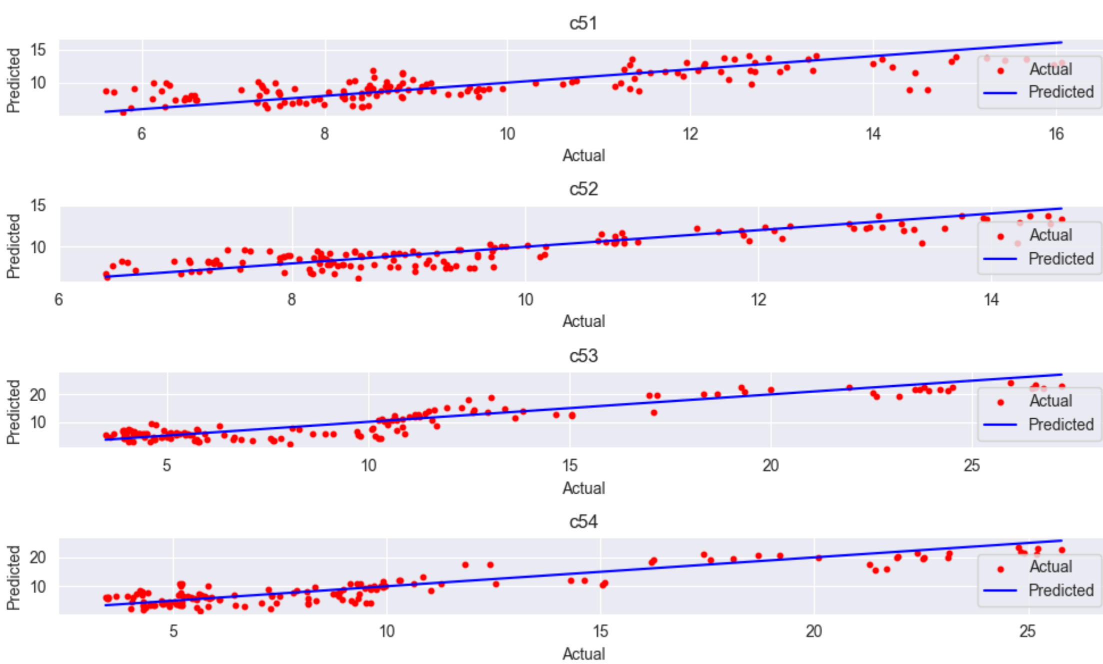
[ ]:
```
df = pd.read_csv('IQRCleaned_Rolled_Data.csv')
```

[ ]:
```
df_control = df[control_parameters]
df_vibrations = df[vibrations]
```

[ ]:
```
fig_mlr, ax_mlr = plt.subplots(4, figsize = (10,6))
for i in range(4):
    X= sm.add_constant(df_control)
    Y = df_vibrations[vibrations[i]]

    # Split the data into training and test sets (80% training, 20% testing)
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.
 ↪2,random_state=42)

    # Fit the linear regression model on the training data
    model = sm.OLS(Y_train, X_train).fit()
    # Print the model summary
    # print(model.summary())

    while max(model.pvalues) >= 0.05:
        parameter_to_be_removed = model.pvalues.idxmax()
        X = X.drop(labels = parameter_to_be_removed, axis = 1)
        # Split the data into training and test sets (80% training, 20% testing)
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.
 ↪2,random_state=42)

        # Fit the linear regression model on the training data
        model = sm.OLS(Y_train, X_train).fit()
        # Print the model summary
        # print(model.rsquared)
    y_predict = model.predict(X_test)
    col = i%2
    row = int(i/2)
    ax_mlr[i].set_title(vibrations[i])
    ax_mlr[i].scatter(Y_test,y_predict,color="red",label="Actual",s=10)
    ax_mlr[i].plot([min(Y_test), max(Y_test)], [min(Y_test),␣
 ↪max(Y_test)],color="blue",label="Predicted")
```

```
    ax_mlr[i].set_xlabel("Actual")
    ax_mlr[i].set_ylabel("Predicted")
    ax_mlr[i].legend()
    print(vibrations[i])
    print(f"F value is {model.fvalue}")
    print(f"R2 is {model.rsquared}")
    print(f"MSE is {model.mse_model}")

fig_mlr.tight_layout()
plt.savefig('MLR.png', dpi = 300)
```

```
c51
F value is 69.48231767314853
R2 is 0.6358887995649743
MSE is 181.4040019251702
c52
F value is 128.0012011132955
R2 is 0.7754458989191217
MSE is 143.19037758121053
c53
F value is 1003.3236350083954
R2 is 0.9643105534748677
MSE is 5749.587281750687
c54
F value is 783.5987867857114
R2 is 0.9575364434320357
MSE is 4594.1471982786525
```



2.2 Trying Decision Trees

```
[ ]: X = df[['c26','c27', 'c28', 'c29', 'c30', 'c31', 'c32',
     'c33', 'c39', 'c139', 'c142', 'c143', 'c155', 'c156', 'c157', 'c158', 'c160',␣
      ↪'c161', 'c162', 'c163']]
     y = df[['c51','c52','c53','c54']]
     Xt, Xtest, yt, ytest = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ]: from sklearn.model_selection import cross_val_score
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.metrics import mean_squared_error, r2_score

     decision_tree = DecisionTreeRegressor(random_state=42,min_samples_split=5)
     #min_samples_split=20
     #max_depth = 5
     decision_tree.fit(Xt, yt)
     decision_tree_predictions = decision_tree.predict(Xtest)

     mse = mean_squared_error(ytest, decision_tree_predictions)
     r2 = r2_score(ytest, decision_tree_predictions)
     print("Decision Tree Mean Squared Error:", mse)
     print("Decision Tree R^2:", r2)
```

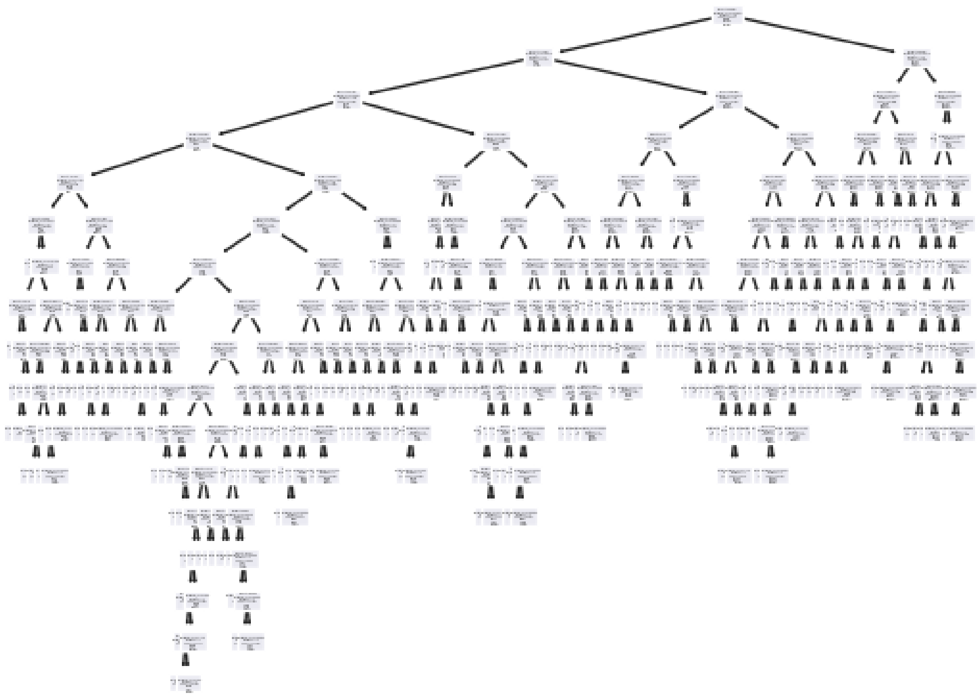
```
Decision Tree Mean Squared Error: 0.20282147885688384
Decision Tree R^2: 0.9833184652997977
```

```
[ ]: #Finding Cross Validation Scores
     k=5
     cross_val_scores = cross_val_score(decision_tree, X, y, cv=k,␣
      ↪scoring='neg_mean_squared_error')
     print("Cross-Validation Scores:")
     print(cross_val_scores)
     print(f"Mean Cross-Validation Scores: {cross_val_scores.mean()}")
```

```
Cross-Validation Scores:
[ -2.59819956  -4.81276039  -7.19307783 -19.3446615  -18.80061618]
Mean Cross-Validation Scores: -10.549863091282297
```

```
[ ]: from sklearn import tree
     tree.plot_tree(decision_tree)
     plt.title("Decision Tree Regression")
     plt.show()
     # plt.savefig('Decision_Tree.png', dpi = 1000)
```

Decision Tree Regression



3 Removing Multicollinearity

[ ]:
```
# #Removing Multicollinearity Using VIF
# #Not showing the compiled jupyer notebook as it took a long time and had no␣
↪display output
#
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data["Variable"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
↪shape[1])]
X_new = X.copy()
removed_multicollinear_columns = []
while max(vif_data['VIF'])>10:
    column_to_be_removed_index = vif_data['VIF'].idxmax()
    column_to_be_removed = vif_data['Variable'][column_to_be_removed_index]
    # print(f"Removing column {column_to_be_removed} with vif␣
↪{vif_data['VIF'][column_to_be_removed_index]}")
    removed_multicollinear_columns.append(column_to_be_removed)
    X_new = X_new.drop(column_to_be_removed,axis=1)
```

```
    vif_data = vif_data.drop(column_to_be_removed_index,axis=0)
df_nmcl = df.drop(removed_multicollinear_columns, axis = 1)
Other_Operating_Parameters = df_nmcl.columns
```

[ ]:
```
# Other_Operating_Parameters =␣
↪['c6'        ,'c14'        ,'c15'        ,'c36'        ,'c113'        ,'c147'        ,'c156

X_nmc = pd.
↪concat((df[Other_Operating_Parameters],df[control_parameters]),axis=1)
```

[ ]:
```
Xt, Xtest, yt, ytest = train_test_split(X_nmc, y, test_size=0.2,␣
↪random_state=42)
```

4 Random Forest Model

[ ]:
```
```

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

random_forest = RandomForestRegressor(n_estimators=100, random_state=37,␣
↪oob_score=True)
random_forest.fit(Xt, yt)
random_forest_predictions = random_forest.predict(Xtest)

mse = mean_squared_error(ytest, random_forest_predictions)
r2 = r2_score(ytest, random_forest_predictions)

print("Random Forest Mean Squared Error:", mse)
print("Random Forest R^2:", r2)
print("Out of the Bag Score", random_forest.oob_score_)
```

```
Random Forest Mean Squared Error: 0.28051746518925885
Random Forest R^2: 0.9855887918712677
Out of the Bag Score 0.9851407553842235
```

[ ]:
```
feature_importances = random_forest.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X_nmc.columns, 'Importance':␣
↪feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance',␣
↪ascending=False)

print("Feature Importance (Descending Order):")
print(feature_importance_df)
```

```
Feature Importance (Descending Order):
   Feature  Importance
```

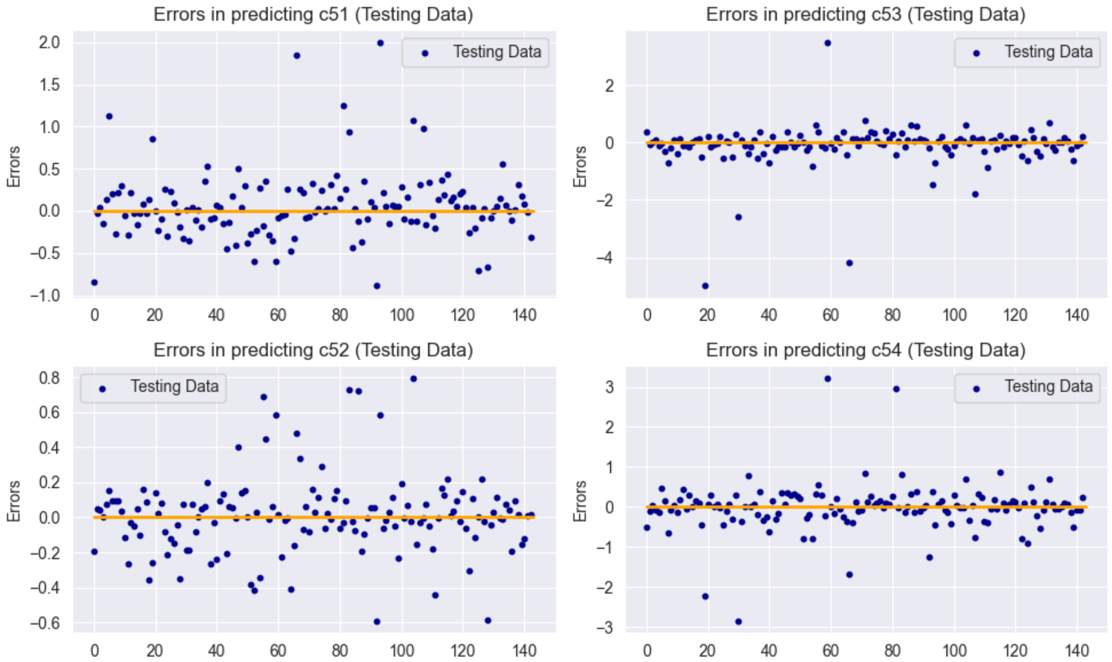
```
22     c155    0.733322
8      c185    0.124972
25     c158    0.023470
7      c184    0.020669
24     c157    0.019491
27     c161    0.015810
18      c39    0.012402
21     c143    0.005750
1       c14    0.005149
4      c113    0.004812
2       c15    0.003808
5      c147    0.003654
12      c28    0.003565
15      c31    0.003257
20     c142    0.003104
11      c27    0.002169
19     c139    0.002144
16      c32    0.002001
9      c208    0.001983
0        c6    0.001601
13      c29    0.001440
17      c33    0.001316
14      c30    0.001306
29     c163    0.000827
10      c26    0.000730
3       c36    0.000593
26     c160    0.000411
28     c162    0.000223
6      c156    0.000011
23     c156    0.000010
```

[ ]:
```
k=5
cross_val_scores = cross_val_score(random_forest, X_nmc, y, cv=k)
print("Cross-Validation Scores:")
print(cross_val_scores)
print(f"Mean Cross-Validation Scores: {cross_val_scores.mean()}")
```

```
Cross-Validation Scores:
[-3.96874135 -2.85360626 -0.59445025  0.63763677 -1.80841298]
Mean Cross-Validation Scores: -1.7175148137890246
```

[ ]:
```
vibrations = ['c51','c52','c53','c54']
fig_rf, ax_rf = plt.subplots(2,2, figsize = (10,6))

for i in range(4):
    col = i%2
    row = int(i/2)
    error = random_forest_predictions[:,i] - ytest[vibrations[i]]
```

```
    ax_rf[col][row].scatter(np.arange(len(error)),error, c='darkblue',␣
↪label='Testing Data',s=10)
    ax_rf[col][row].plot([0, len(error)], [0,0], '-', c='orange', linewidth=2)
    ax_rf[col][row].set_ylabel('Errors')
    ax_rf[col][row].set_title('Errors in predicting ' + vibrations[i] + '␣
↪(Testing Data)')
    ax_rf[col][row].legend()
fig_rf.tight_layout()
plt.savefig('Random_Forest.png', dpi = 300)
```



5 Q2

[ ]:
```
df = pd.read_csv('IQRCleaned_Rolled_Data.csv')
X = df[['c26', 'c27', 'c28', 'c29', 'c30', 'c31', 'c32', 'c33', 'c39', 'c139',␣
↪'c142', 'c143', 'c155', 'c156', 'c157', 'c158', 'c160', 'c161', 'c162',␣
↪'c163']]
y = df[['c51', 'c52', 'c53', 'c54']]
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=23)
random_forest = RandomForestRegressor(n_estimators=100, random_state=23)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)
r2_value = r2_score(y_test, predictions)
```

```
print("Random Forest Regressor R-squared Value:", r2_value)
```

```
Random Forest Regressor R-squared Value: 0.9791434317784855
```

[ ]:
```
def map_to_category(value):
    if value < 5:
        return 'Safe'
    elif 5 <= value < 10:
        return 'Moderate'
    elif 10 <= value < 20:
        return 'High'
    else:
        return 'Critical'


random_forest = RandomForestRegressor(n_estimators=100, random_state=10)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)
predicted_categories = pd.DataFrame(predictions, columns=['c51', 'c52', 'c53',␣
 ↪'c54']).applymap(map_to_category)
```

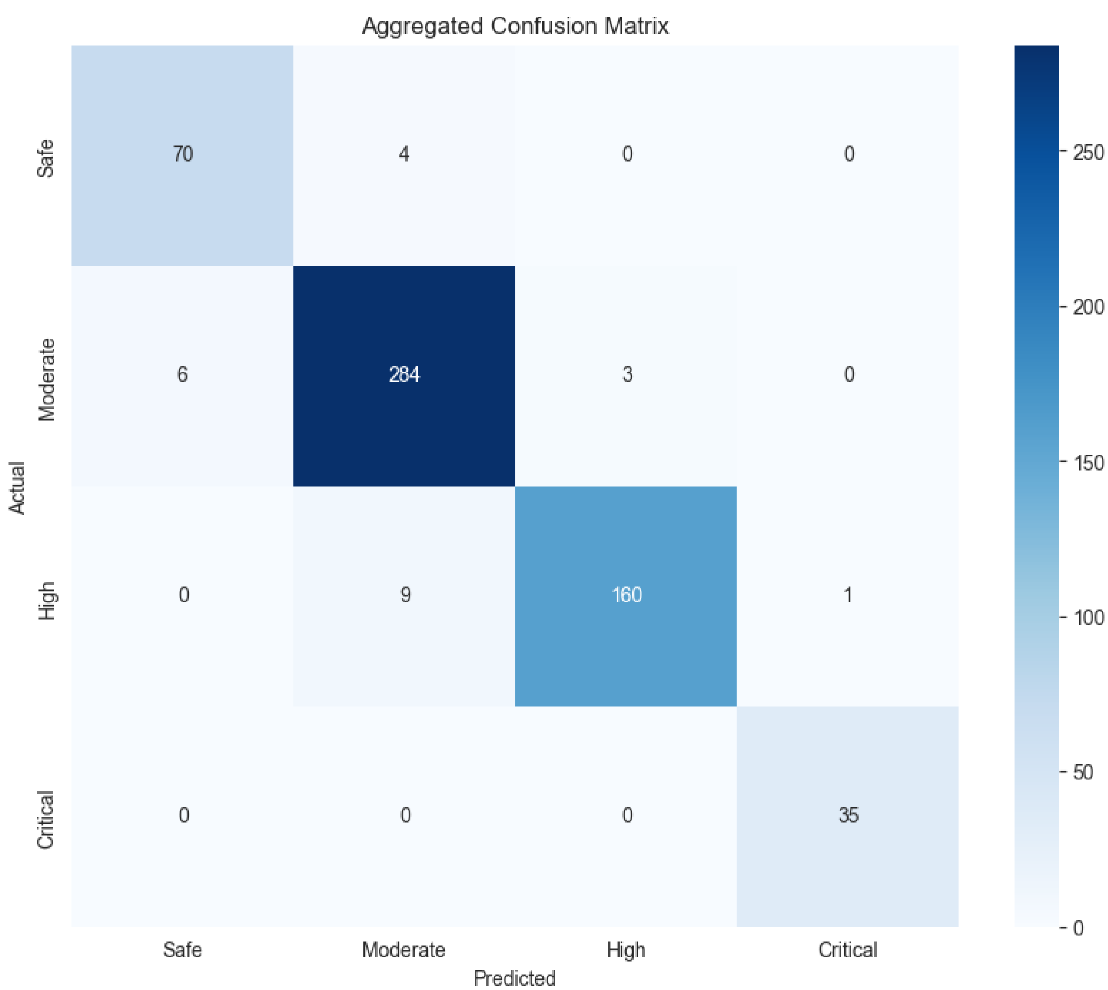
[ ]:
```
discrete_y = pd.DataFrame(columns=vibrations)
for column in y_test.columns:
    discrete_y[column] = y_test[column].apply(map_to_category)
y_final=discrete_y.reset_index()
y_final=y_final.drop('index',axis=1)
```

[ ]:
```
common_critical_positions = (y_final == 'Critical') & (predicted_categories ==␣
 ↪'Critical')
total_common_critical_positions = common_critical_positions.sum().sum()
critical_counts_df1 = y_final[vibrations].apply(lambda col: col.value_counts().
 ↪get('Critical', 0))
critical_counts_df2 = predicted_categories[vibrations].apply(lambda col: col.
 ↪value_counts().get('Critical', 0))
total_critical_df1 = critical_counts_df1.sum()
total_critical_df2 = critical_counts_df2.sum()
print("Total Critical in Data Given:", total_critical_df1)
print("Total Critical in Predicted Data:", total_critical_df2)
print("Total Number of 'Critical' correctly predicted:",␣
 ↪total_common_critical_positions)
```

```
Total Critical in Data Given: 35
Total Critical in Predicted Data: 36
Total Number of 'Critical' correctly predicted: 35
```

[ ]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

```python
target_columns = ['c51', 'c52', 'c53', 'c54']
# y_test = y_final[target_columns]
y_predicted = predicted_categories[target_columns]
total_cm = pd.DataFrame(0, index=['Safe', 'Moderate', 'High', 'Critical'],
 ↪columns=['Safe', 'Moderate', 'High', 'Critical'])
for true_col in target_columns:
    pred_col = true_col
    cm = confusion_matrix(y_final[true_col], y_predicted[pred_col],
 ↪labels=['Safe', 'Moderate', 'High', 'Critical'])
    total_cm += pd.DataFrame(cm, index=['Safe', 'Moderate', 'High',
 ↪'Critical'], columns=['Safe', 'Moderate', 'High', 'Critical'])
plt.figure(figsize=(10, 8))
sns.heatmap(total_cm, annot=True, fmt='d', cmap='Blues', cbar=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Aggregated Confusion Matrix')
plt.savefig('ConfusionMatrix.png')
```



```
[ ]: from sklearn.inspection import permutation_importance
import pandas as pd

# Assuming you already have a trained model 'random_forest' and testing data␣
↪'X_test', 'y_test'
perm_importance = permutation_importance(random_forest, X_test, y_test,␣
↪n_repeats=30, random_state=27)

# Create a dictionary with column names and their respective permutation␣
↪importance values
perm_importance_dict = dict(zip(X.columns, perm_importance.importances_mean))

# Convert the dictionary to a DataFrame for better visualization
perm_importance_df = pd.DataFrame(list(perm_importance_dict.items()),␣
↪columns=['Feature', 'Permutation_Importance'])
perm_importance_df = perm_importance_df.
↪sort_values(by='Permutation_Importance', ascending=False)

# Display the results
print("Permutation Importance:")
print(perm_importance_df[:12])
```

```
Permutation Importance:
   Feature  Permutation_Importance
12    c155                1.402603
17    c161                0.112344
15    c158                0.100374
11    c143                0.058490
14    c157                0.041183
8      c39                0.014445
2      c28                0.008648
9     c139                0.006805
7      c33                0.006330
4      c30                0.004580
10    c142                0.004462
5      c31                0.004194
```

6 Q3

```
[ ]: y = df['c241']
Xt, Xtest, yt, ytest = train_test_split(X_nmc, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
#Facts about the data
print(f"Mean of data is {y.mean()}")
print(f"Standard deviation of data is {y.std()}")
```

```
Mean of data is 2.1829525549869464
Standard deviation of data is 0.09463878558827112
```

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error

# Create a Random Forest Regressor
random_forest_regressor = RandomForestRegressor(n_estimators=100,␣
↪random_state=42)
random_forest_regressor.fit(Xt, yt)
predictions_rf = random_forest_regressor.predict(Xtest)
r2_value_rf = r2_score(ytest, predictions_rf)
mse_rf = mean_squared_error(ytest, predictions_rf)

print("Random Forest Regressor R-squared Value:", r2_value_rf)
print("MSE:", mse_rf)
```

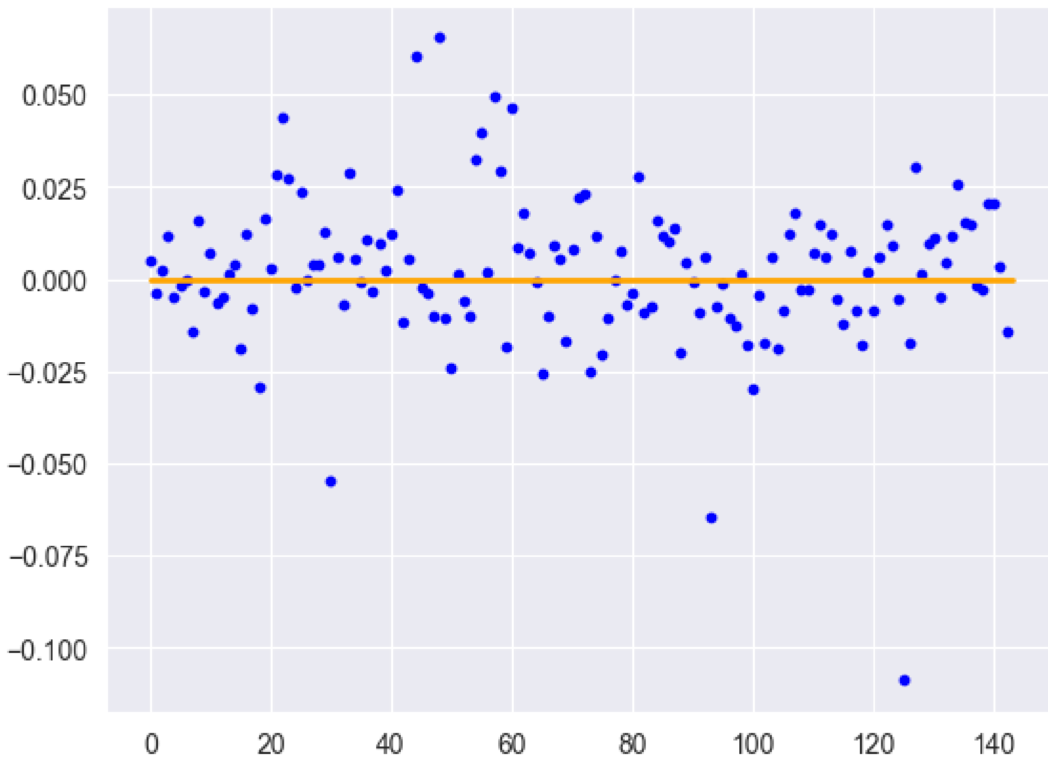
```
Random Forest Regressor R-squared Value: 0.9468748846621478
MSE: 0.0004165680759040543
```

[ ]:
```
from sklearn.model_selection import cross_val_score

k = 5  # You can choose the value of k
cross_val_scores = cross_val_score(random_forest_regressor, X_nmc, y, cv=k,␣
↪scoring='neg_mean_squared_error')

# Note: 'neg_mean_squared_error' is used because cross_val_score maximizes␣
↪scores, and mean squared error is a loss function to be minimized.

# Step 4: Print the cross-validation scores
print("Cross-Validation Scores:")
print(cross_val_scores)

mean_CV = round(sum(cross_val_scores )/len(cross_val_scores ), 3)
print(mean_CV)
```

```
Cross-Validation Scores:
[-0.00303376 -0.00099923 -0.00200107 -0.00375534 -0.00121473]
-0.002
```

[ ]:
```
import matplotlib.pyplot as plt
error_rf = predictions_rf - ytest
plt.scatter(np.arange(len(error_rf)),error_rf, color="blue",s=10)
plt.plot([0, len(error_rf)], [0,0], '-', c='orange', linewidth=2)
```

```
plt.title("Errors in predicting C241 - Testing Data",color = "white")
plt.ylabel("Error",color="white")
# plt.yscale(,color="white")
# plt.show()
plt.savefig("C241_RandomForest.png",dpi = 300)
# plt.plot()
```



7 Q4

[ ]:
```
from sklearn.inspection import permutation_importance
import pandas as pd

# Assuming you already have a trained model 'random_forest' and testing data␣
↪'X_test', 'y_test'
perm_importance = permutation_importance(random_forest_regressor, Xtest, ytest,␣
↪n_repeats=30, random_state=42)

# Create a dictionary with column names and their respective permutation␣
↪importance values
```

```python
perm_importance_dict = dict(zip(Xtest.columns, perm_importance.
 ↪importances_mean))

# Convert the dictionar y to a DataFrame for better visualization
perm_importance_df = pd.DataFrame(list(perm_importance_dict.items()),␣
 ↪columns=['Feature', 'Permutation_Importance'])
perm_importance_df = perm_importance_df.
 ↪sort_values(by='Permutation_Importance', ascending=False)

# Display the results
print("Permutation Importance:")
print(perm_importance_df[:12])
```

```
Permutation Importance:
   Feature  Permutation_Importance
21    c143                0.403974
20    c142                0.371647
19    c139                0.027010
24    c158                0.023043
2      c15                0.016513
12     c28                0.012566
4     c113                0.007517
15     c31                0.006730
0       c6                0.006314
14     c30                0.004965
26    c161                0.004673
18     c39                0.004428
```