```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

#Importing Train and Test path
train_path = 'FLIR_groups1and2_train.csv'
test_path = 'FLIR_groups1and2_test.csv'
train_data = pd.read_csv(train_path)
test_data = pd.read_csv(test_path)

# Dataset Cleaning - Train

train_data.head(3)

#Check for null vales in train dataset
null_counts = train_data.isnull().sum()

columns_to_drop = null_counts[null_counts == len(train_data)].index
data = train_data.drop(columns=columns_to_drop)

columns = data.columns.tolist()
constant_features = columns[-7:]

constant_feature_start_index = len(columns) - 7
first_constant_feature = data.columns[0]

round_indices = {f"Round {i+1}": None for i in range(4)}
final = pd.DataFrame()
for col in columns[1:]:
    for round_name in round_indices.keys():
        if round_name in col and round_indices[round_name] is None:
            round_indices[round_name] = columns.index(col)
            break

round_dfs_t = {}
for i in range(1, 5):
    round_name = f"Round {i}"
    if i < 4:
        next_round_start = round_indices[f"Round {i+1}"]
    else:
        next_round_start = constant_feature_start_index
    round_columns = columns[round_indices[round_name]:next_round_start]

    round_dfs_t[round_name] = data[round_columns]


for round_name, df_round in round_dfs_t.items():
    df_round = df_round.iloc[1:].reset_index(drop=True)
    new_header = df_round.iloc[0]
    df_round = df_round[1:]
    df_round.columns = new_header
    df_round.reset_index(drop=True, inplace=True)
    round_dfs_t[round_name] = df_round

first = data[first_constant_feature].iloc[2:].reset_index(drop=True)
new_header = data[first_constant_feature].iloc[1]
first.columns = new_header
first.reset_index(drop=True, inplace=True)

last = data[columns[-7:]].iloc[2:].reset_index(drop=True)
new_header = data[columns[-7:]].iloc[1]
last.columns = new_header
last.reset_index(drop=True, inplace=True)

#Modication of header in the datset
dfs = list(round_dfs_t.values())
```

```
appended_df = pd.concat(dfs, axis=1)
final = pd.concat([first, appended_df, last], axis=1)
train_df = final.rename(columns={'Unnamed: 0': 'SubjectID'})

from sklearn.impute import KNNImputer
import pandas as pd

exclude = ['SubjectID', 'Gender', 'Ethnicity', 'Age']
df_to_exclude = train_df[exclude]
df_to_impute = train_df.drop(exclude, axis=1)

# Initialize the KNN imputer with the specified number of neighbors
imputer = KNNImputer(n_neighbors=5)
imputer.fit(df_to_impute)
# Apply the imputer to transform the data, filling in missing values based on the KNN strategy
imputed_data = imputer.transform(df_to_impute)
df_imputed = pd.DataFrame(imputed_data, columns=df_to_impute.columns)

train_df = pd.concat([df_to_exclude, df_imputed], axis=1)

train_df.shape

# Dataset Cleaning - Test

test_data.head(3)

#Check for null vales in test dataset
null_counts = test_data.isnull().sum()

columns_to_drop = null_counts[null_counts == len(test_data)].index
data = test_data.drop(columns=columns_to_drop)

columns = data.columns.tolist()
constant_features = columns[-7:]

constant_feature_start_index = len(columns) - 7
first_constant_feature = data.columns[0]

round_indices = {f"Round {i+1}": None for i in range(4)}
final = pd.DataFrame()
for col in columns[1:]:
    for round_name in round_indices.keys():
        if round_name in col and round_indices[round_name] is None:
            round_indices[round_name] = columns.index(col)
            break

round_dfs = {}
for i in range(1, 5):
    round_name = f"Round {i}"
    if i < 4:
        next_round_start = round_indices[f"Round {i+1}"]
    else:
        next_round_start = constant_feature_start_index
    round_columns = columns[round_indices[round_name]:next_round_start]

    round_dfs[round_name] = data[round_columns]


for round_name, df_round in round_dfs.items():
    df_round = df_round.iloc[1:].reset_index(drop=True)
    new_header = df_round.iloc[0]
    df_round = df_round[1:]
    df_round.columns = new_header
    df_round.reset_index(drop=True, inplace=True)
    round_dfs[round_name] = df_round

first = data[first_constant_feature].iloc[2:].reset_index(drop=True)
```

```
new_header = data[first_constant_feature].iloc[1]
first.columns = new_header
first.reset_index(drop=True, inplace=True)

last = data[columns[-7:]].iloc[2:].reset_index(drop=True)
new_header = data[columns[-7:]].iloc[1]
last.columns = new_header
last.reset_index(drop=True, inplace=True)


from sklearn.impute import KNNImputer
import pandas as pd

exclude = ['SubjectID', 'Gender', 'Ethnicity', 'Age']
df_to_exclude = test_df[exclude]
df_to_impute = test_df.drop(exclude, axis=1)

# Initialize the KNN imputer with the specified number of neighbors
imputer = KNNImputer(n_neighbors=5)
imputer.fit(df_to_impute)
# Apply the imputer to transform the data, filling in missing values based on the KNN strategy
imputed_data = imputer.transform(df_to_impute)
df_imputed = pd.DataFrame(imputed_data, columns=df_to_impute.columns)

test_df = pd.concat([df_to_exclude, df_imputed], axis=1)

# Data Visulaization

train_df.shape

test_df.shape

print("Null values in Train: ", train_df.isnull().sum().sum())
print("Null values in Test: ", test_df.isnull().sum().sum())

#Boxplot Visulaization based on Gender
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x='Gender', y='aveOralM', data=train_df)
plt.title('Distribution of Oral Temp by Gender')
plt.xlabel('Gender')
plt.ylabel('Oral Temp')

plt.show()

#Histogram plot for Gender
sns.histplot(data=train_df, x='Gender', discrete=True, shrink=0.8)
plt.title('Number of Subjects by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt

#Boxplot for Ethnicity in Datasets
sns.boxplot(x='Ethnicity', y='aveOralM', data=train_df)
plt.title('Distribution of Oral Temp by Ethnicity')
plt.xlabel('Ethnicity')
plt.ylabel('Oral Temp')

plt.xticks(rotation=45, ha='right', fontsize=8)

plt.show()

#Histograms for Ethnincty in Dataset
```

```
sns.histplot(data=train_df, x='Ethnicity', discrete=True, shrink=0.8)
plt.title('Number of Subjects by Ethnicity')
plt.xlabel('Ethnicity')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right', fontsize=7)
plt.show()

r_1 = [col for col in train_df.columns if col.endswith('_1')]
r_2 = [col for col in train_df.columns if col.endswith('_2')]
r_3 = [col for col in train_df.columns if col.endswith('_3')]
r_4 = [col for col in train_df.columns if col.endswith('_4')]

#Scatterplot to identify the Linearity between various features and target variable (round 1)
target_column = 'aveOralM'
fig, axes = plt.subplots(6, 5, figsize=(20, 24))
axes = axes.flatten()

# Iterate through each column (except the target column)
for i, column in enumerate(train_df[r_1]):
    # Scatter plot between current column and target column
    axes[i].scatter(train_df[column], train_df[target_column], alpha=0.5)

    # Set labels and title
    axes[i].set_xlabel(column)
    axes[i].set_ylabel(target_column)
    axes[i].set_title(f'{column} vs {target_column}')

plt.title('Round 1')
plt.tight_layout()
plt.show()

#Scatterplot to identify the Linearity between various features and target variable (round 2)
target_column = 'aveOralM'
fig, axes = plt.subplots(6, 5, figsize=(20, 24))
axes = axes.flatten()

# Iterate through each column (except the target column)
for i, column in enumerate(train_df[r_2]):
    # Scatter plot between current column and target column
    axes[i].scatter(train_df[column], train_df[target_column], alpha=0.5)

    # Set labels and title
    axes[i].set_xlabel(column)
    axes[i].set_ylabel(target_column)
    axes[i].set_title(f'{column} vs {target_column}')

plt.title('Round 2')
plt.tight_layout()
plt.show()

#Scatterplot to identify the Linearity between various features and target variable (round 3)
target_column = 'aveOralM'
fig, axes = plt.subplots(6, 5, figsize=(20, 24))
axes = axes.flatten()

# Iterate through each column (except the target column)
for i, column in enumerate(train_df[r_3]):
    # Scatter plot between current column and target column
    axes[i].scatter(train_df[column], train_df[target_column], alpha=0.5)

    # Set labels and title
    axes[i].set_xlabel(column)
    axes[i].set_ylabel(target_column)
    axes[i].set_title(f'{column} vs {target_column}')

plt.title('Round 3')
plt.tight_layout()
```

```
plt.show()

#Scatterplot to identify the Linearity between various features and target variable (round 4)
target_column = 'aveOralM'
fig, axes = plt.subplots(6, 5, figsize=(20, 24))
axes = axes.flatten()

# Iterate through each column (except the target column)
for i, column in enumerate(train_df[r_4]):
    # Scatter plot between current column and target column
    axes[i].scatter(train_df[column], train_df[target_column], alpha=0.5)

    # Set labels and title
    axes[i].set_xlabel(column)
    axes[i].set_ylabel(target_column)
    axes[i].set_title(f'{column} vs {target_column}')

plt.title('Round 4')
plt.tight_layout()
plt.show()

# Data Preprocessing - Train

from sklearn.preprocessing import OneHotEncoder
import pandas as pd

#One Hot Encoding to replace Genders to numerical features
encoder = OneHotEncoder(sparse_output=False, drop='if_binary')
gender_encoded = encoder.fit_transform(train_df[['Gender']])
train_df.drop('Gender', axis=1, inplace=True)
gender_encoded_df = pd.DataFrame(gender_encoded, index=train_df.index, columns=['Gender'])
train_df = pd.concat([train_df, gender_encoded_df], axis=1)

from sklearn.preprocessing import OneHotEncoder
import pandas as pd
import numpy as np

#One Hot Encoding to replace Ethnicity to numerical features
encoder = OneHotEncoder(sparse_output=False, categories=[['White', 'Black or African-
American', 'Asian', 'Multiracial', 'Hispanic/Latino', 'American Indian or Alaskan Native']],
handle_unknown='ignore')

ethnicity_mapping = {
    'White': 1,
    'Black or African-American': 2,
    'Asian': 3,
    'Multiracial': 4,
    'Hispanic/Latino': 5,
    'American Indian or Alaskan Native': 6

}

inverse_ethnicity_mapping = {v: k for k, v in ethnicity_mapping.items()}

#Drop Ethnicity Column and replace with mapped name for ethnicity declared in One Hot Encoding
if 'Ethnicity' in train_df.columns:
    ethnicity_encoded = encoder.fit_transform(train_df[['Ethnicity']])
    ethnicity_encoded_df = pd.DataFrame(ethnicity_encoded,
columns=encoder.get_feature_names_out(['Ethnicity']))
    train_df.drop('Ethnicity', axis=1, inplace=True)
    train_df['Ethnicity'] = np.argmax(ethnicity_encoded, axis=1)


#One Hot Encoding for Age
encoder = OneHotEncoder(sparse_output=False, categories='auto')

if 'Age' in train_df.columns:
```

```
    age_reshaped = train_df['Age'].values.reshape(-1, 1)
    age_encoded = encoder.fit_transform(age_reshaped)
    train_df.drop('Age', axis=1, inplace=True)
    train_df['Age'] = np.argmax(age_encoded, axis=1) + 1

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
T_LC_{round_number}
r = [1, 2, 3, 4]
for round_number in r:
    temperature_column = f'T_LC_{round_number}'

    if 'Gender' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            train_df[f'{temperature_column}_{stat.capitalize()}'] =
train_df['Gender'].map(stat_dict)

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
canthiMax_{round_number}
for round_number in r:
    temperature_column = f'canthiMax_{round_number}'

    if 'Gender' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            train_df[f'{temperature_column}_{stat.capitalize()}'] =
train_df['Gender'].map(stat_dict)

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
T_FHC_Max_{round_number}
for round_number in r:
    temperature_column = f'T_FHC_Max_{round_number}'

    if 'Gender' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            train_df[f'{temperature_column}_{stat.capitalize()}'] =
train_df['Gender'].map(stat_dict)

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
T_FHCC_{round_number}
for round_number in r:
    temperature_column = f'T_FHCC_{round_number}'

    if 'Gender' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            train_df[f'{temperature_column}_{stat.capitalize()}'] =
train_df['Gender'].map(stat_dict)

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on ethnicity
, canthiMax_{round_number}
for round_number in r:
    temperature_column = f'canthiMax_{round_number}'

    if 'Ethnicity' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
```

```
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            train_df = pd.merge(train_df, temp_df, on='Ethnicity', how='left')

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on ethnicity
, T_LC_{round_number}
for round_number in r:
    temperature_column = f'T_LC_{round_number}'

    if 'Ethnicity' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            train_df = pd.merge(train_df, temp_df, on='Ethnicity', how='left')


#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on ethnicity,
T_FHCC_{round_number}
for round_number in r:
    temperature_column = f'T_FHCC_{round_number}'

    if 'Ethnicity' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            train_df = pd.merge(train_df, temp_df, on='Ethnicity', how='left')


#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on ethnicity
, T_FHC_Max_{round_number}
for round_number in r:
    temperature_column = f'T_FHC_Max_{round_number}'

    if 'Ethnicity' in train_df.columns and temperature_column in train_df.columns:
        grouped_stats = train_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            train_df = pd.merge(train_df, temp_df, on='Ethnicity', how='left')


#Apply Standard Scaler features excluding 'aveOralM', 'SubjectID'
from sklearn.preprocessing import StandardScaler

features = train_df.drop(columns=['aveOralM', 'SubjectID'])
target = train_df[['aveOralM', 'SubjectID']]
scaler = StandardScaler()
scaler.fit(features)
scaled_features = scaler.transform(features)
train_df.loc[:, features.columns] = scaled_features
```

```
train_df.shape

# Feature Engineering

import pandas as pd
import numpy as np

#Pearson Coefficient to determine the correlation between features and aveOralM
output_column = 'aveOralM'

if output_column in train_df.columns:
    correlation_matrix = train_df.corr(method='pearson', numeric_only=True)
    if output_column in correlation_matrix.columns:
        abs_correlation_with_output =
correlation_matrix[output_column].abs().drop(output_column)
        psorted_features = abs_correlation_with_output.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
psorted_features.plot(kind='bar')
plt.title('Absolute Correlation with {}'.format(output_column))
plt.xticks(range(len(psorted_features)), psorted_features.index, fontsize = 1)
plt.xlabel('Features')
plt.ylabel('Absolute Correlation')
plt.show()

#Threshold to extract most correlated features
pearson = psorted_features[psorted_features > 0.2]

pearson = pearson.index.tolist()

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
import numpy as np

#Cross Validation Score  using neagtive mean squared error to determine the correlation between
features and aveOralM
output_column = 'aveOralM'
feature_performance = {}

features = [col for col in train_df.columns if col not in [output_column, 'SubjectID']]

for feature in features:
    X = train_df[[feature]].values.reshape(-1, 1)
    y = train_df[output_column].values

    model = LinearRegression()
    scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
    feature_performance[feature] = np.mean(scores)

top_features = sorted(feature_performance, key=feature_performance.get, reverse=True)

performance_scores = [feature_performance[feature] for feature in top_features]

plt.figure(figsize=(8, 6))
plt.bar(top_features, performance_scores)
plt.xlabel('Features')
plt.ylabel('Negative MSE')
plt.title('Feature Performance')
plt.show()

#Threshold for Cross Validation using Negative mean squared
linear = [feature for feature in top_features if feature_performance[feature] > -0.21]

from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
```

```
output_column = 'aveOralM'

X = train_df.select_dtypes(include=[np.number]).drop(columns=[output_column])
y = train_df[output_column]

model = LinearRegression()

#Using Forward Sequential Feature selector to determine highly correlated features
sfs = SequentialFeatureSelector(model,
                                n_features_to_select=100,
                                direction='forward',
                                scoring='neg_mean_squared_error')

sfs.fit(X, y)
selected_features = X.columns[sfs.get_support()]

#Listing all important featured by taking common features from Pearson, Croo Validation and
Sequential Feature Selector
important_features = list(set(pearson) & set(linear) & set(selected_features))

len(important_features)

# important_features = ['T_RC_Wet_3',
#  'T_FHC_Max_2',
#  'RCC_2',
#  'T_FH_Max_3',
#  'T_RC_3',
#  'T_FHBC_4',
#  'T_RC_Max_1',
#  'T_RC_Dry_1',
#  'T_FHLC_2',
#  'T_FHLC_4',
#  'T_Max_1',
#  'Max1R13_2',
#  'T_RC_Max_4',
#  'T_OR_Max_1',
#  'T_FHTC_2',
#  'T_LC_1',
#  'canthi4Max_2',
#  'aveAllL13_2',
#  'T_OR_1',
#  'canthiMax_1',
#  'T_FHTC_1',
#  'T_RC_4',
#  'T_RC_Dry_3',
#  'RCC_1',
#  'Max1L13_1',
#  'T_FHC_Max_4',
#  'LCC_4',
#  'T_LC_Dry_1',
#  'T_FH_Max_2',
#  'T_LC_Dry_2',
#  'T_Max_4',
#  'RCC_3',
#  'T_RC_Wet_1',
#  'T_FHCC_2',
#  'T_LC_Max_1',
#  'LCC_2',
#  'T_OR_4',
#  'T_RC_Max_2',
#  'Max1R13_1',
#  'aveAllR13_2',
#  'T_FHLC_3',
#  'T_OR_Max_4',
#  'T_Max_3',
#  'aveAllR13_3',
```

```
#   'Max1L13_3',
#   'aveAllR13_1']

# Data Preprocessing - Test

from sklearn.preprocessing import OneHotEncoder
import pandas as pd

#One Hot Encoding to replace Genders to numerical features
encoder = OneHotEncoder(sparse_output=False, drop='if_binary')
gender_encoded = encoder.fit_transform(test_df[['Gender']])
test_df.drop('Gender', axis=1, inplace=True)
gender_encoded_df = pd.DataFrame(gender_encoded, index=test_df.index, columns=['Gender'])
test_df = pd.concat([test_df, gender_encoded_df], axis=1)

from sklearn.preprocessing import OneHotEncoder
import pandas as pd
import numpy as np

#One Hot Encoding to replace Ethnicity to numerical features
encoder = OneHotEncoder(sparse_output=False, categories=[['White', 'Black or African-
American', 'Asian', 'Multiracial', 'Hispanic/Latino', 'American Indian or Alaskan Native']],
handle_unknown='ignore')

ethnicity_mapping = {
    'White': 1,
    'Black or African-American': 2,
    'Asian': 3,
    'Multiracial': 4,
    'Hispanic/Latino': 5,
    'American Indian or Alaskan Native': 6
}

#Drop Ethnicity Column and replace with mapped name for ethnicity declared in One Hot Encoding}
inverse_ethnicity_mapping = {v: k for k, v in ethnicity_mapping.items()}

if 'Ethnicity' in test_df.columns:
    ethnicity_encoded = encoder.fit_transform(test_df[['Ethnicity']])
    ethnicity_encoded_df = pd.DataFrame(ethnicity_encoded,
columns=encoder.get_feature_names_out(['Ethnicity']))
    test_df.drop('Ethnicity', axis=1, inplace=True)
    test_df['Ethnicity'] = np.argmax(ethnicity_encoded, axis=1)


#One Hot Encoding for Age
encoder = OneHotEncoder(sparse_output=False, categories='auto')

if 'Age' in test_df.columns:
    age_reshaped = test_df['Age'].values.reshape(-1, 1)
    age_encoded = encoder.fit_transform(age_reshaped)
    test_df.drop('Age', axis=1, inplace=True)
    test_df['Age'] = np.argmax(age_encoded, axis=1) + 1


#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
T_LC_{round_number}
r = [1, 2, 3, 4]
for round_number in r:
    temperature_column = f'T_LC_{round_number}'

    if 'Gender' in test_df.columns and temperature_column in test_df.columns:
        grouped_stats = test_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            test_df[f'{temperature_column}_{stat.capitalize()}'] =
test_df['Gender'].map(stat_dict)
```

```
#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
canthiMax_{round_number}
for round_number in r:
    temperature_column = f'canthiMax_{round_number}'

    if 'Gender' in test_df.columns and temperature_column in test_df.columns:
        grouped_stats = test_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            test_df[f'{temperature_column}_{stat.capitalize()}'] =
test_df['Gender'].map(stat_dict)


#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
T_FHC_Max_{round_number}
for round_number in r:
    temperature_column = f'T_FHC_Max_{round_number}'

    if 'Gender' in test_df.columns and temperature_column in test_df.columns:
        grouped_stats = test_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            test_df[f'{temperature_column}_{stat.capitalize()}'] =
test_df['Gender'].map(stat_dict)


#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on gender ,
T_FHCC_{round_number}
for round_number in r:
    temperature_column = f'T_FHCC_{round_number}'

    if 'Gender' in test_df.columns and temperature_column in test_df.columns:
        grouped_stats = test_df.groupby('Gender')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        for stat in ['mean', 'min', 'max', 'median']:
            stat_dict = dict(zip(grouped_stats['Gender'], grouped_stats[stat]))
            test_df[f'{temperature_column}_{stat.capitalize()}'] =
test_df['Gender'].map(stat_dict)

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on Ethnicity
, canthiMax_{round_number}

for round_number in r:
    temperature_column = f'canthiMax_{round_number}'

    if 'Ethnicity' in test_df.columns and temperature_column in test_df.columns:
        grouped_stats = test_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            test_df = pd.merge(test_df, temp_df, on='Ethnicity', how='left')


#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on Ethnicity
, T_LC_{round_number}
for round_number in r:
    temperature_column = f'T_LC_{round_number}'

    if 'Ethnicity' in test_df.columns and temperature_column in test_df.columns:
```

```
        grouped_stats = test_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            test_df = pd.merge(test_df, temp_df, on='Ethnicity', how='left')

#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on Ethnicity
, T_FHCC_{round_number}
for round_number in r:
    temperature_column = f'T_FHCC_{round_number}'

    if 'Ethnicity' in test_df.columns and temperature_column in test_df.columns:
        grouped_stats = test_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            test_df = pd.merge(test_df, temp_df, on='Ethnicity', how='left')


#Add Mean, Min, Max , Median for Important temperatures stated in the paper based on Ethnicity
, T_FHC_Max_{round_number}
for round_number in r:
    temperature_column = f'T_FHC_Max_{round_number}'

    if 'Ethnicity' in test_df.columns and temperature_column in test_df.columns:
        grouped_stats = test_df.groupby('Ethnicity')[temperature_column].agg(['mean', 'min',
'max', 'median']).reset_index()
        rename_stats = {stat: f'{temperature_column}_{stat.capitalize()}' for stat in ['mean',
'min', 'max', 'median']}
        grouped_stats = grouped_stats.rename(columns=rename_stats)
        for stat in ['mean', 'min', 'max', 'median']:
            new_col_name = f'{temperature_column}_{stat.capitalize()}'
            temp_df = grouped_stats[['Ethnicity', new_col_name]]
            test_df = pd.merge(test_df, temp_df, on='Ethnicity', how='left')


#Apply Standard Scaler features excluding 'aveOralM', 'SubjectID'
from sklearn.preprocessing import StandardScaler

test_features = test_df.drop(columns=['aveOralM', 'SubjectID'])
test_target = test_df[['aveOralM', 'SubjectID']]

scaled_test_features = scaler.transform(test_features)
test_df.loc[:, test_features.columns] = scaled_test_features

# Model - Linear Regression

#Correlated features combined with SubjectId and aveOralM
columns = ['SubjectID'] + important_features + ['aveOralM']
train_df = train_df[columns]
test_df = test_df[columns]

# Linear Regression

# Refernce : https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
```

```
from sklearn.metrics import mean_squared_error

X = train_df.drop(columns=['SubjectID', 'aveOralM'])
y = train_df['aveOralM']
n_splits = 5

#Declare Cross Validation with 5 Folds
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

#Sklearn Linear Regression model
model = LinearRegression()

# Initialize lists for scores
train_rmse_scores = []
val_rmse_scores = []

train_mae_scores = []
val_mae_scores = []

for train_index, val_index in kf.split(X):
    X_train_fold, X_val_fold = X.iloc[train_index], X.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    model.fit(X_train_fold, y_train_fold)

    y_pred_train = model.predict(X_train_fold)
    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train_fold, y_pred_train)))
    train_mae_scores.append(mean_absolute_error(y_train_fold, y_pred_train))

    y_pred_val = model.predict(X_val_fold)
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val_fold, y_pred_val)))
    val_mae_scores.append(mean_absolute_error(y_val_fold, y_pred_val))


print("Avg Train RMSE: ", np.mean(train_rmse_scores))
print("Avg Train MAE: ", np.mean(train_mae_scores))

print("\nAvg Val RMSE: ", np.mean(val_rmse_scores))
print("Avg Val MAE: ", np.mean(val_mae_scores))

from sklearn.metrics import r2_score, mean_absolute_error

X_test = test_df.drop(columns=['SubjectID', 'aveOralM'])
y_test = test_df['aveOralM']

#Predicting test datas with trained model
y_pred = model.predict(X_test)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))


# Model - Trivial System

# Define a TrivialSystem class that acts as a simple regression baseline
class TrivialSystem:
    def __init__(self):
        self.mean_value = None

    # Fit method calculates the mean of the target values in the training set
    def fit(self, X_train, y_train):
        self.mean_value = y_train.mean()

    # Predict method returns a list of the mean value
```

```
    def predict(self, input_data):
        return [self.mean_value] * len(input_data)


kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

#Trivial System declared with class
model = TrivialSystem()

# Initialize lists for scores
train_rmse_scores = []
val_rmse_scores = []

train_mae_scores = []
val_mae_scores = []

#Declared Cross Validation that does 5 Folds
for train_index, val_index in kf.split(X):
    X_train_fold, X_val_fold = X.iloc[train_index], X.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    model.fit(X_train_fold, y_train_fold)

    y_pred_train = model.predict(X_train_fold)
    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train_fold, y_pred_train)))
    train_mae_scores.append(mean_absolute_error(y_train_fold, y_pred_train))

    y_pred_val = model.predict(X_val_fold)
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val_fold, y_pred_val)))
    val_mae_scores.append(mean_absolute_error(y_val_fold, y_pred_val))


print("Avg Train RMSE: ", np.mean(train_rmse_scores))
print("Avg Train MAE: ", np.mean(train_mae_scores))

print("\nAvg Val RMSE: ", np.mean(val_rmse_scores))
print("Avg Val MAE: ", np.mean(val_mae_scores))

from sklearn.metrics import r2_score, mean_absolute_error

X_test = test_df.drop(columns=['SubjectID', 'aveOralM'])
y_test = test_df['aveOralM']

#Predicting test datas with trained model
y_pred = model.predict(X_test)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))


# Model - 1NN

#Reference : https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

from sklearn.neighbors import KNeighborsRegressor

kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

#Sklearn Kneighbours Regressor model
model = KNeighborsRegressor(n_neighbors=1)

# Initialize lists for scores
train_rmse_scores = []
```

```
val_rmse_scores = []

train_mae_scores = []
val_mae_scores = []

#Declared Cross Validation that does 5 Folds
for train_index, val_index in kf.split(X):
    X_train_fold, X_val_fold = X.iloc[train_index], X.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    model.fit(X_train_fold, y_train_fold)

    y_pred_train = model.predict(X_train_fold)
    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train_fold, y_pred_train)))
    train_mae_scores.append(mean_absolute_error(y_train_fold, y_pred_train))

    y_pred_val = model.predict(X_val_fold)
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val_fold, y_pred_val)))
    val_mae_scores.append(mean_absolute_error(y_val_fold, y_pred_val))



print("Avg Train RMSE: ", np.mean(train_rmse_scores))
print("Avg Train MAE: ", np.mean(train_mae_scores))

print("\nAvg Val RMSE: ", np.mean(val_rmse_scores))
print("Avg Val MAE: ", np.mean(val_mae_scores))

from sklearn.metrics import r2_score, mean_absolute_error

X_test = test_df.drop(columns=['SubjectID', 'aveOralM'])
y_test = test_df['aveOralM']

#Predicting test datas with trained model
y_pred = model.predict(X_test)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))


# Model - SVR - Linear

#Reference : https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

from sklearn.svm import SVR

kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

#Sklearn SVR Model
model = SVR(kernel='linear')

# Initialize lists for scores
train_rmse_scores = []
val_rmse_scores = []

train_mae_scores = []
val_mae_scores = []

#Declared Cross Validation that does 5 Folds
for train_index, val_index in kf.split(X):
    X_train_fold, X_val_fold = X.iloc[train_index], X.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    model.fit(X_train_fold, y_train_fold)
```

```
    y_pred_train = model.predict(X_train_fold)
    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train_fold, y_pred_train)))
    train_mae_scores.append(mean_absolute_error(y_train_fold, y_pred_train))

    y_pred_val = model.predict(X_val_fold)
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val_fold, y_pred_val)))
    val_mae_scores.append(mean_absolute_error(y_val_fold, y_pred_val))


print("Avg Train RMSE: ", np.mean(train_rmse_scores))
print("Avg Train MAE: ", np.mean(train_mae_scores))

print("\nAvg Val RMSE: ", np.mean(val_rmse_scores))
print("Avg Val MAE: ", np.mean(val_mae_scores))

from sklearn.metrics import r2_score, mean_absolute_error

X_test = test_df.drop(columns=['SubjectID', 'aveOralM'])
y_test = test_df['aveOralM']

#Predicting test datas with trained model
y_pred = model.predict(X_test)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))

# Model - SVR - RBF

#Reference : https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

from sklearn.svm import SVR

kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

#Sklearn SVR Model
model = SVR(kernel='rbf')

# Initialize lists for scores
train_rmse_scores = []
val_rmse_scores = []

train_mae_scores = []
val_mae_scores = []

#Declared Cross Validation that does 5 Folds
for train_index, val_index in kf.split(X):
    X_train_fold, X_val_fold = X.iloc[train_index], X.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    model.fit(X_train_fold, y_train_fold)

    y_pred_train = model.predict(X_train_fold)
    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train_fold, y_pred_train)))
    train_mae_scores.append(mean_absolute_error(y_train_fold, y_pred_train))

    y_pred_val = model.predict(X_val_fold)
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val_fold, y_pred_val)))
    val_mae_scores.append(mean_absolute_error(y_val_fold, y_pred_val))


print("Avg Train RMSE: ", np.mean(train_rmse_scores))
print("Avg Train MAE: ", np.mean(train_mae_scores))
```

```
print("\nAvg Val RMSE: ", np.mean(val_rmse_scores))
print("Avg Val MAE: ", np.mean(val_mae_scores))

from sklearn.metrics import r2_score, mean_absolute_error

X_test = test_df.drop(columns=['SubjectID', 'aveOralM'])
y_test = test_df['aveOralM']

#Predicting test datas with trained model
y_pred = model.predict(X_test)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))

# Model - Polynomial Regression

#Reference: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)
X_poly = pd.DataFrame(poly.fit_transform(X))

kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

#Sklearn model Linear Regression
model = LinearRegression()

# Initialize lists for scores
train_rmse_scores = []
val_rmse_scores = []

train_mae_scores = []
val_mae_scores = []

#Declared Cross Validation that does 5 Folds
for train_index, val_index in kf.split(X_poly):
    X_train_fold, X_val_fold = X_poly.iloc[train_index], X_poly.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    model.fit(X_train_fold, y_train_fold)

    y_pred_train = model.predict(X_train_fold)
    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train_fold, y_pred_train)))
    train_mae_scores.append(mean_absolute_error(y_train_fold, y_pred_train))

    y_pred_val = model.predict(X_val_fold)
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val_fold, y_pred_val)))
    val_mae_scores.append(mean_absolute_error(y_val_fold, y_pred_val))

print("Avg Train RMSE: ", np.mean(train_rmse_scores))
print("Avg Train MAE: ", np.mean(train_mae_scores))

print("\nAvg Val RMSE: ", np.mean(val_rmse_scores))
print("Avg Val MAE: ", np.mean(val_mae_scores))

from sklearn.metrics import r2_score, mean_absolute_error

X_test = test_df.drop(columns=['SubjectID', 'aveOralM'])
y_test = test_df['aveOralM']

Y_poly = pd.DataFrame(poly.fit_transform(X_test))
```

```
#Predicting test datas with trained model
y_pred = model.predict(Y_poly)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))

# Model - Non- Linear Transformation through RBF and Regression

# Calculate gamma by dividing M by a constant
def cal_gamma(M):
    g = M / 32
    return g

# Define an RBF kernel that computes the Gaussian similarity between two points
def rbf_kernel(x, mu, gamma):
    return np.exp(-gamma * np.linalg.norm(x - mu)**2)

# Build and fit a linear regression model using the given transformed training data
def Linear_Reg(X_train_transformed, y_train):
    model = LinearRegression()
    model.fit(X_train_transformed, y_train)
    return model

#Reference : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")

def cross_validation_with_kmeans(X, y, K_values, gammas, n_folds=5):
    best_gamma = None
    best_K = None
    best_rmse = float('inf')
    kf = KFold(n_splits=n_folds)

    for K in K_values:
        best_rmse_K = float('inf')
        for gamma in gammas:
            g_d = 0

            # Initialize lists for scores
            fold_rmse = []
            fold_rmse1 = []
            fold_mae = []
            fold_mae1 = []

            #Declared Cross Validation that does 5 Folds
            for train_index, val_index in kf.split(X):
                X_train, X_val = X.iloc[train_index], X.iloc[val_index]
                y_train, y_val = y.iloc[train_index], y.iloc[val_index]

                kmeans = KMeans(n_clusters=K, init='random')
                kmeans.fit(X_train)
                centers = kmeans.cluster_centers_

                if g_d == 0:
                  gamma = (K**0.06)/32 * gamma
                  g_d += 1

                X_train_transformed = np.array([[rbf_kernel(x, mu, gamma) for mu in centers]
for x in X_train.values])
                X_val_transformed = np.array([[rbf_kernel(x, mu, gamma) for mu in centers] for
x in X_val.values])
```

```
                model = Linear_Reg(X_train_transformed, y_train)
                y_pred_val = model.predict(X_val_transformed)
                y_pred_train = model.predict(X_train_transformed)

                rmse = np.sqrt(mean_squared_error(y_val, y_pred_val))
                rmse1 = np.sqrt(mean_squared_error(y_train, y_pred_train))

                mae =  mean_absolute_error(y_val, y_pred_val)
                mae1 = mean_absolute_error(y_train, y_pred_train)


                fold_rmse.append(rmse)
                fold_rmse1.append(rmse1)

                fold_mae.append(mae)
                fold_mae1.append(mae1)

            avg_rmse = np.mean(fold_rmse)
            avg_rmse1 = np.mean(fold_rmse1)
            avg_mae = np.mean(fold_mae)
            avg_mae1 = np.mean(fold_mae1)

            if avg_rmse < best_rmse:
                best_rmse = avg_rmse
                b = avg_rmse1
                c = avg_mae
                d = avg_mae1
                best_gamma = gamma
                best_K = K

    return best_K, best_gamma, best_rmse, b, c, d

gammas = [0.01, 0.1, 1, 10]
K = range(5, 101, 5)
best_K, best_gamma, val_rmse, train_rmse, val_mae, train_mae = cross_validation_with_kmeans(X,
y, K, gammas)

print(val_rmse, train_rmse, val_mae, train_mae)

print(best_K, best_gamma, )

X_train = X
y_train = y

#Declare Kmeans
kmeans = KMeans(n_clusters=best_K, init='random')
kmeans.fit(X_train)

#Declare kmeand cluster centres
centers = kmeans.cluster_centers_

X_train_transformed = np.array([[rbf_kernel(x, mu, best_gamma) for mu in centers] for x in
X_train.values])
model3 = Linear_Reg(X_train_transformed, y_train)

X_test_transformed = np.array([[rbf_kernel(x, mu, best_gamma) for mu in centers] for x in
X_test.values])

#Predicting test datas with trained model
y_pred = model3.predict(X_test_transformed)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))
```

```
# Model - Random Forest Regressor + Hyperparameter Tuning

#Reference : https://scikit-
learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
import numpy as np

#Finding Best Parameters fro our model using RandomizedDearchCV
param_dist = {
    'n_estimators': np.random.randint(50, 1000, size=100),
    'max_depth': np.random.randint(3, 15, size=100),
    'min_samples_split': np.random.randint(2, 20, size=100),
    'min_samples_leaf': np.random.randint(1, 10, size=100),
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]
}

#Sklearn Random Forest Regressor model
rf = RandomForestRegressor(random_state=42)

random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist, n_iter=20,
                                   scoring='neg_mean_squared_error', cv=5, random_state=42)

random_search.fit(X, y)

print("Best Hyperparameters:", random_search.best_params_)
print("Best Score:", -random_search.best_score_)

kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
best_params = random_search.best_params_
#Sklearn Random Forest Regressor model with best parameters
model = RandomForestRegressor(**best_params)

# Initialize lists for scores
train_rmse_scores = []
val_rmse_scores = []

train_mae_scores = []
val_mae_scores = []

#Declared Cross Validation that does 5 Folds
for train_index, val_index in kf.split(X):
    X_train_fold, X_val_fold = X.iloc[train_index], X.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    model.fit(X_train_fold, y_train_fold)

    y_pred_train = model.predict(X_train_fold)
    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train_fold, y_pred_train)))
    train_mae_scores.append(mean_absolute_error(y_train_fold, y_pred_train))

    y_pred_val = model.predict(X_val_fold)
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val_fold, y_pred_val)))
    val_mae_scores.append(mean_absolute_error(y_val_fold, y_pred_val))

print("Avg Train RMSE: ", np.mean(train_rmse_scores))
print("Avg Train MAE: ", np.mean(train_mae_scores))

print("\nAvg Val RMSE: ", np.mean(val_rmse_scores))
print("Avg Val MAE: ", np.mean(val_mae_scores))

from sklearn.metrics import r2_score, mean_absolute_error

X_test = test_df.drop(columns=['SubjectID', 'aveOralM'])
```

```
y_test = test_df['aveOralM']

#Predicting test datas with trained model
y_pred = model.predict(X_test)

#Finding metrics using Sklearn.metrics
print("Test MSE: ", mean_squared_error(y_test, y_pred))
print("Test RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE: ", mean_absolute_error(y_test, y_pred))
print("Test R^2: ", r2_score(y_test, y_pred))
```