Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN3150 Pattern Recognition

Assignment 1

J.Charles 210079K

1. Data pre-processing

Feature 1: Standard Scaling

Feature 1 consists primarily of values that are clustered close to zero, with a few significant outliers. To effectively manage this distribution, standard scaling is the ideal approach. Standard scaling, also known as Z-score normalization, adjusts the feature such that it has a mean of 0 and a standard deviation of 1. This is accomplished by subtracting the mean of the feature from each data point and dividing by the standard deviation. The major benefit of applying standard scaling to Feature 1 is that it normalizes the small variances around the mean without distorting the relative distances between data points. This ensures that all data points are treated uniformly, which is particularly important for machine learning algorithms that are sensitive to the variance of features, such as gradient-based optimization methods. At the same time, the impact of outliers is preserved, allowing the feature's full range of information to be utilized in the analysis.

Feature 2: Min-Max Scaling

Feature 2, on the other hand, exhibits a wide range of values, extending approximately from -40 to +30. Given this significant variability, min-max scaling is the most appropriate method. Min-max scaling transforms the feature's values into a specified range, typically [0, 1] or [-1, 1]. This process involves subtracting the minimum value of the feature from each data point and then dividing by the range (the difference between the maximum and minimum values). The key advantage of using min-max scaling for Feature 2 is that it compresses the values into a smaller, more manageable range, creating uniformity across features. This uniformity is crucial for algorithms that depend on distance metrics or require inputs to be within a specific range, such as K-Nearest Neighbors or neural networks. By applying min-max scaling, Feature 2 is brought into alignment with other features in the dataset, enabling more balanced and effective processing by machine learning models.

2. Learning from data

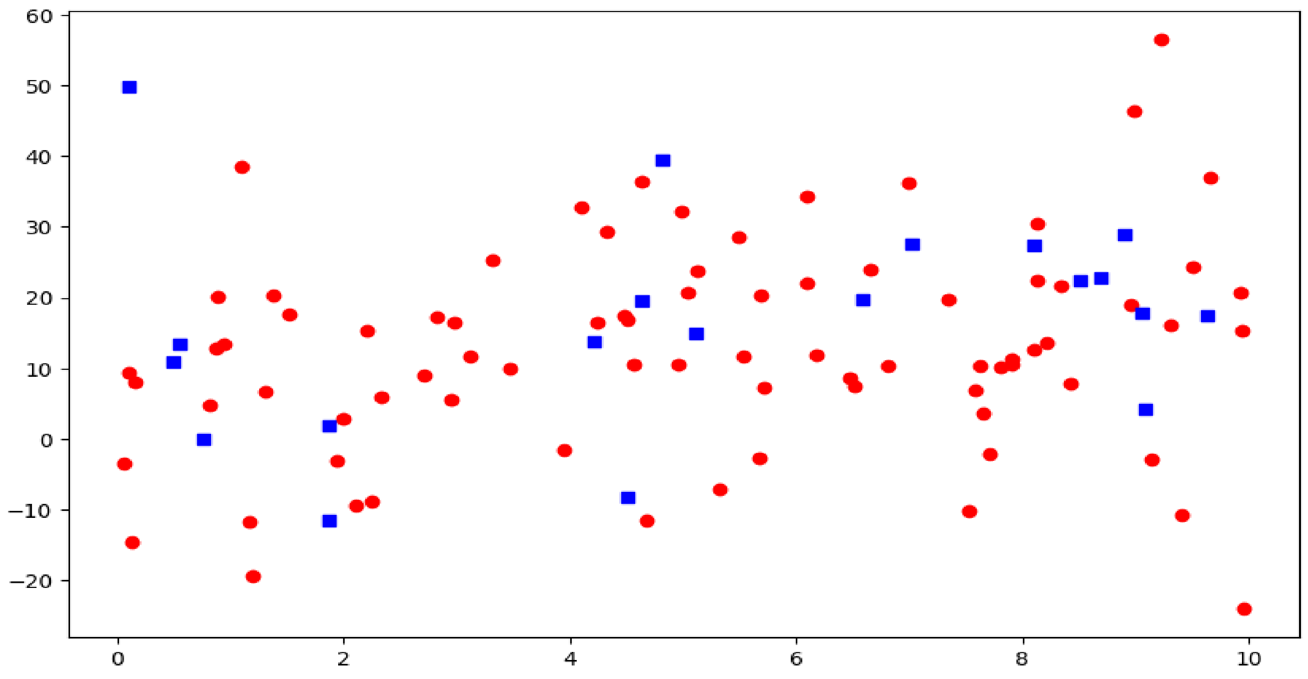
Data generation.

```
import numpy as np
import matplotlib . pyplot as plt
from sklearn . model_selection import train_test_split
from sklearn . linear_model import LinearRegression
# Generate 100 samples
n_samples = 100
# Generate X values ( uniformly distributed between 0 and 10)
X = 10 * np . random . rand ( n_samples , 1)
# Generate epsilon values ( normally distributed with mean 0 and standard
deviation 15)
epsilon = np . random . normal (0 , 15 , n_samples )
```

```
# Generate Y values using the model Y = 3 + 3X + epsilon
Y = 3 + 2 * X + epsilon [: , np . newaxis ]
```

Data visualization

```
r = np . random . randint (104)
# Split the data into training and test sets (80% train , 20% test )
X_train , X_test , Y_train , Y_test = train_test_split (X , Y , test_size =0.2 ,
random_state = r )
# Plot the data points
plt . figure ( figsize =(10 , 6) )
plt . scatter ( X_train , Y_train , alpha =1 , marker ='o', color ='red', label
='Training Data ')
plt . scatter ( X_test , Y_test , alpha =1 , marker ='s', color ='blue', label
='Testing Data ')
plt . show ()
```

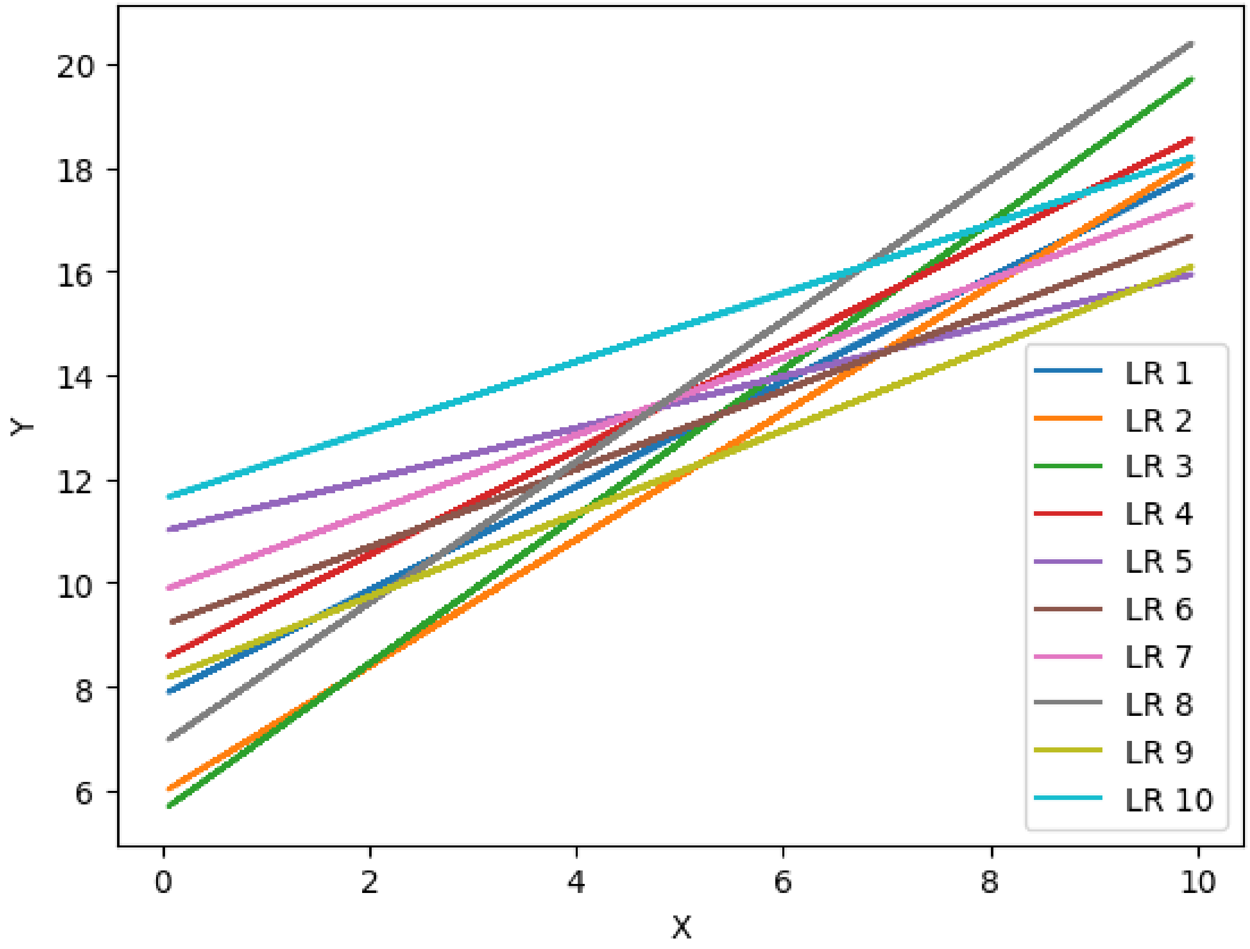


Observation: I find that the training and testing datasets change with each iteration of the code when I run it using the instructions in Listing 2. Every time, the particular data points in the testing and training sets are different from what was used in the last run.

Explanation: The code divides the data into training and testing sets using a random approach, which accounts for this unpredictability. The data is separated based on a randomly created variable in each run. Every time this random variable is executed, new training and testing datasets are produced, so that they are never the same.

One such solution is to utilize a set random seed to guarantee consistency across several runs. By maintaining the same random variable, the training and testing datasets would split equally after each run.

Linear regression.



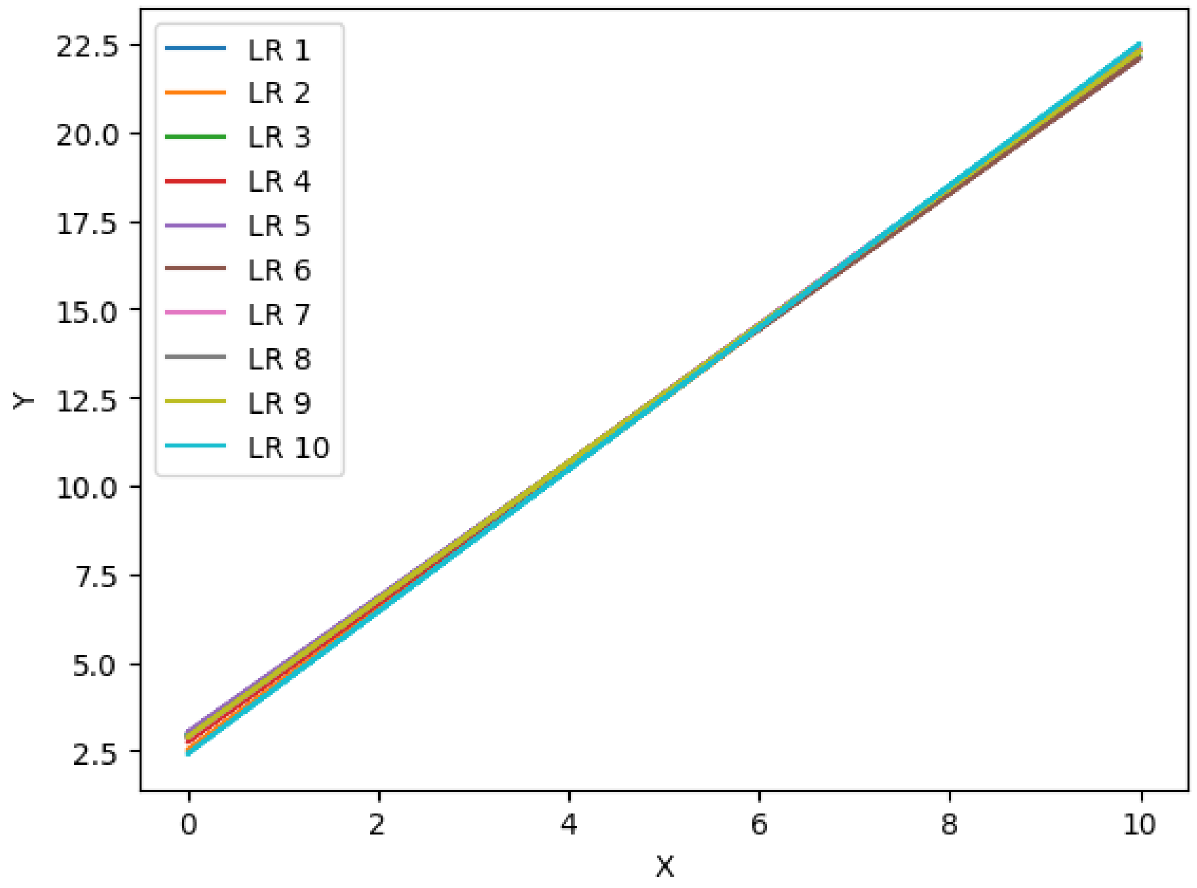
Explanation: The differences observed in linear regression models across various instances arise mainly from the variability introduced when the dataset is randomly split into training and testing sets. Each time this split occurs with a different random seed, the model is trained on a slightly different portion of the data. This leads to variations in how the model learns the relationships between the input features and the output variable.

As a result, the regression lines differ between instances because the model may overfit to certain subsets of the training data or respond to noise differently, depending on the specific data it was trained on. This underscores the sensitivity of linear regression to the particular data sample it receives during training.

2.3 Increasing the Number of Data Samples to 10,000

```
import numpy as np
import matplotlib . pyplot as plt
from sklearn . model_selection import train_test_split
from sklearn . linear_model import LinearRegression
# Generate 100 samples
n_samples = 10000
# Generate X values ( uniformly distributed between 0 and 10)
X = 10 * np . random . rand ( n_samples , 1)
# Generate epsilon values ( normally distributed with mean 0 and standard deviation
15)
epsilon = np . random . normal (0 , 15 , n_samples )
# Generate Y values using the model Y = 3 + 3X + epsilon
Y = 3 + 2 * X + epsilon [: , np . newaxis ]
```

```
for i in range(10):  # Plotting 10 different instances
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=np.random.randint(104))
    model = LinearRegression()
    model.fit(X_train, Y_train)
    Y_pred_train = model.predict(X_train)
    plt.plot(X_train, Y_pred_train, label=f'LR {i+1}')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



3 Linear regressions on real world data

3.1 Loading the data

```
Defaulting to user installation because normal site-packages is not writeable
Collecting ucimlrepo
Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in c:\program files\python312\lib\site-packages (from
ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in c:\program files\python312\lib\site-packages (from
ucimlrepo) (2024.7.4)
Requirement already satisfied: numpy>=1.26.0 in c:\program files\python312\lib\site-packages (from
pandas>=1.0.0->ucimlrepo) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\program files\python312\lib\site-packages (from
pandas>=1.0.0->ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\program files\python312\lib\site-packages (from
pandas>=1.0.0->ucimlrepo) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\program files\python312\lib\site-packages (from
pandas>=1.0.0->ucimlrepo) (2024.1)
Requirement already satisfied: six>=1.5 in c:\program files\python312\lib\site-packages (from python-
dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.16.0)
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

```
 # If package not installed, install it using pip install ucimlrepo
from ucimlrepo import fetch_ucirepo
# fetch dataset
infrared_thermography_temperature = fetch_ucirepo(id=925)
# data (as pandas dataframes)
X = infrared_thermography_temperature.data.features
y = infrared_thermography_temperature.data.targets
# metadata
print(infrared_thermography_temperature.metadata)
# variable information
print(infrared_thermography_temperature.variables)
```

3.2 Determine independent and dependent variables

Independent

```
print(X.shape[1])
```

33

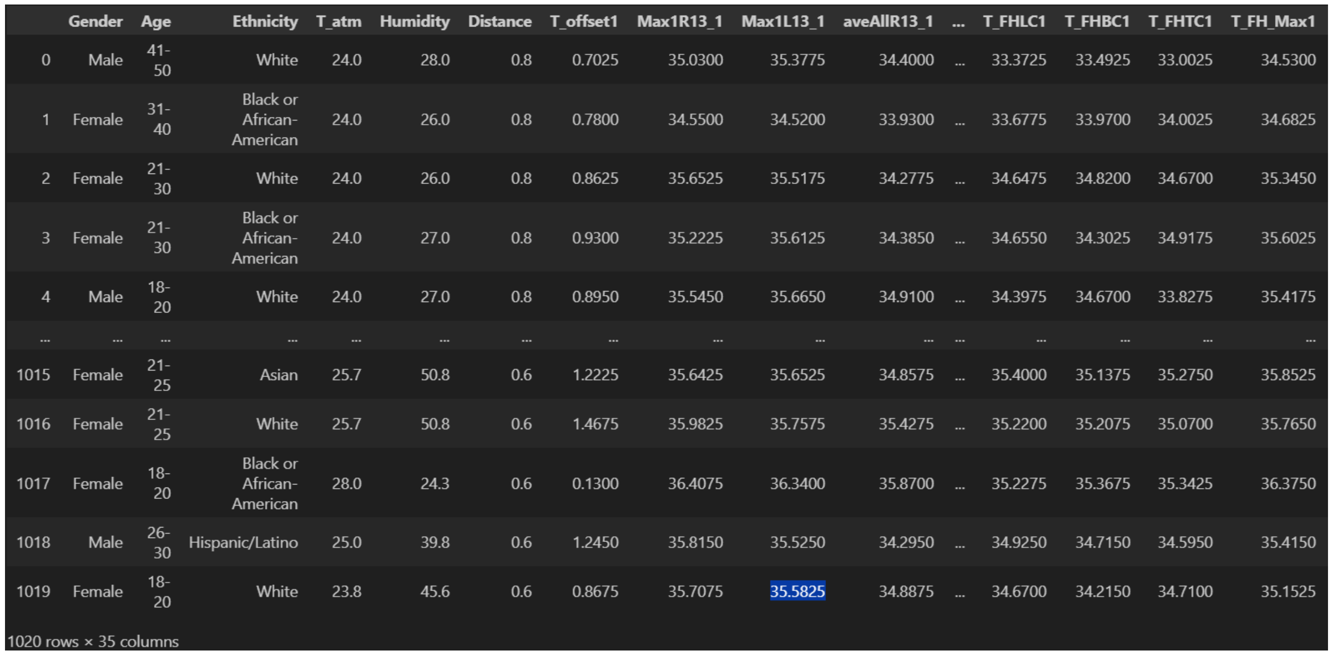
Dependent

```
print(y.shape[1])
```

2

3.3 Feasibility of Applying Linear Regression

```
import pandas as pd
#creating a dataframe including features and tragets(targets are the last 2
columns)
df = pd.concat([X, y],axis=1)
df
```



	Gender	Age	Ethnicity	T_atm	Humidity	Distance	T_offset1	Max1R13_1	Max1L13_1	aveAllR13_1	...	T_FHLC1	T_FHBC1	T_FHTC1	T_FH_Max1
0	Male	41-50	White	24.0	28.0	0.8	0.7025	35.0300	35.3775	34.4000	...	33.3725	33.4925	33.0025	34.5300
1	Female	31-40	Black or African-American	24.0	26.0	0.8	0.7800	34.5500	34.5200	33.9300	...	33.6775	33.9700	34.0025	34.6825
2	Female	21-30	White	24.0	26.0	0.8	0.8625	35.6525	35.5175	34.2775	...	34.6475	34.8200	34.6700	35.3450
3	Female	21-30	Black or African-American	24.0	27.0	0.8	0.9300	35.2225	35.6125	34.3850	...	34.6550	34.3025	34.9175	35.6025
4	Male	18-20	White	24.0	27.0	0.8	0.8950	35.5450	35.6650	34.9100	...	34.3975	34.6700	33.8275	35.4175
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1015	Female	21-25	Asian	25.7	50.8	0.6	1.2225	35.6425	35.6525	34.8575	...	35.4000	35.1375	35.2750	35.8525
1016	Female	21-25	White	25.7	50.8	0.6	1.4675	35.9825	35.7575	35.4275	...	35.2200	35.2075	35.0700	35.7650
1017	Female	18-20	Black or African-American	28.0	24.3	0.6	0.1300	36.4075	36.3400	35.8700	...	35.2275	35.3675	35.3425	36.3750
1018	Male	26-30	Hispanic/Latino	25.0	39.8	0.6	1.2450	35.8150	35.5250	34.2950	...	34.9250	34.7150	34.5950	35.4150
1019	Female	18-20	White	23.8	45.6	0.6	0.8675	35.7075	35.5825	34.8875	...	34.6700	34.2150	34.7100	35.1525

1020 rows × 35 columns

```
missing_counts = df.isnull().sum()
print(missing_counts)
```

```
Gender          0
Age             0
Ethnicity       0
T_atm           0
Humidity        0
Distance        2
T_offset1       0
Max1R13_1       0
Max1L13_1       0
aveAllR13_1     0
aveAllL13_1     0
T_RC1           0
T_RC_Dry1       0
T_RC_Wet1       0
T_RC_Max1       0
T_LC1           0
T_LC_Dry1       0
T_LC_Wet1       0
T_LC_Max1       0
RCC1            0
LCC1            0
canthiMax1      0
canthi4Max1     0
T_FHCC1         0
T_FHRC1         0
...
T_OR_Max1       0
aveOralF        0
aveOralM        0

dtype: int64
```

3.4 Is NaN can be used??

```
#We could calculate the ratio of Nan values without calculating the number of
values
nan_ratio_distance = df['Distance'].isna().sum() / df.shape[0]
print(f"NaN ratio for distance: {nan_ratio_distance}")
```

Given that the 'Distance' feature has only 2 missing values, the proportion of missing data is quite low, less than 0.1. In this scenario, it's reasonable to remove those 2 values. However, if the proportion of missing data were higher and closer to 1, other strategies would be needed to avoid significantly reducing the dataset's size. Some of these strategies include:

- Replacing the missing values with the mean of the available data
- Filling the missing values with zeros
- Using interpolation to estimate the missing values

These methods help in maintaining the completeness of the dataset while effectively handling the missing data.

3.5 Select "aveOralM" as the dependent feature. For the independent features, select 'Age' and four other features based on your preference.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

```
X = infrared_thermography_temperature.data.features
y = infrared_thermography_temperature.data.targets

y = df['aveOralM']
X = df[['Age', 'T_atm', 'Humidity', 'Distance', 'T_OR_Max1']]

df = pd.concat([X, y], axis=1).dropna()

X = df.drop(columns=y.name)  # Use y.name since y is a Series
y = df[y.name]

X = pd.get_dummies(X, drop_first=True)
```

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

model = LinearRegression()
model.fit(X_train, y_train)
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Combine the coefficients with their corresponding feature names
coefficients_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_
})

# Display the estimated coefficients
print(coefficients_df)
print(f"Intercept: {model.intercept_}")
```

```
Feature  Coefficient
0        T_atm    -0.033412
1     Humidity    -0.000427
2     Distance     0.002033
3    T_OR_Max1     0.735013
4    Age_21-25    -0.021580
5    Age_21-30     0.045786
6    Age_26-30    -0.049989
7    Age_31-40    -0.055768
8    Age_41-50    -0.128181
9    Age_51-60     0.127456
10     Age_>60    -0.048658

Intercept: 11.517385698018217
```

3.8. Which independent variable contributes highly for the dependent feature?

According to the selected data, it is T_OR_max1. This is because it has the largest weight in the coefficient matrix.

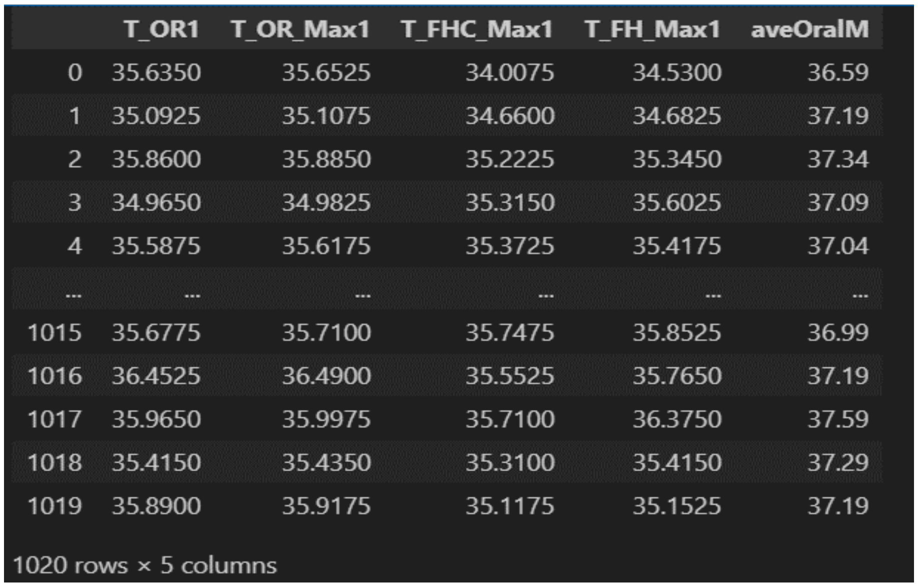
```
X = infrared_thermography_temperature.data.features
y = infrared_thermography_temperature.data.targets

y = df['aveOralM']
X = df[['T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1']]

df = pd.concat([X, y], axis=1).dropna()

X = df.drop(columns=y.name)  # Use y.name since y is a Series
```

```
y = df[y.name]

X = pd.get_dummies(X, drop_first=True)  # Age drop_first=True to reduce redundancy
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


model = LinearRegression()
model.fit(X_train, y_train)

# Combine the coefficients with their corresponding feature names
coefficients_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_
})

# Display the estimated coefficients
print(coefficients_df)
print(f"Intercept: {model.intercept_}")
```

```
      Feature  Coefficient
0    T_OR1         0.091997
1    T_OR_Max1      0.464070
2    T_FHC_Max1    -0.087332
3    T_FH_Max1      0.370886

Intercept: 7.036879763545969
```

3.9. Selecting 'T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1' features as independent features.

```
df_new = df[['T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1', 'aveOralM']]
df_new
```



	T_OR1	T_OR_Max1	T_FHC_Max1	T_FH_Max1	aveOralM
0	35.6350	35.6525	34.0075	34.5300	36.59
1	35.0925	35.1075	34.6600	34.6825	37.19
2	35.8600	35.8850	35.2225	35.3450	37.34
3	34.9650	34.9825	35.3150	35.6025	37.09
4	35.5875	35.6175	35.3725	35.4175	37.04
...	...	...	...	...	...
1015	35.6775	35.7100	35.7475	35.8525	36.99
1016	36.4525	36.4900	35.5525	35.7650	37.19
1017	35.9650	35.9975	35.7100	36.3750	37.59
1018	35.4150	35.4350	35.3100	35.4150	37.29
1019	35.8900	35.9175	35.1175	35.1525	37.19

1020 rows × 5 columns

3.10 Calculation

```python
from scipy import stats

# Assuming you have trained the model with X_train, X_test, y_train, y_test
y_pred = model.predict(X_test)

# 1. Residual Sum of Squares (RSS)
rss = np.sum((y_test - y_pred) ** 2)
print(f"Residual Sum of Squares (RSS): {rss}")

# 2. Residual Standard Error (RSE)
N = len(y_test)
d = X_train.shape[1]
rse = np.sqrt(rss / (N - d - 1))
print(f"Residual Standard Error (RSE): {rse}")

# 3. Mean Squared Error (MSE)
mse = np.mean((y_test - y_pred) ** 2)
print(f"Mean Squared Error (MSE): {mse}")

# 4. R-squared (R2)
tss = np.sum((y_test - np.mean(y_test)) ** 2)
r2 = 1 - (rss / tss)
print(f"R-squared (R2): {r2}")

# 5. Standard Error for each feature
X_with_intercept = np.column_stack([np.ones(X_test.shape[0]), X_test])
cov_matrix = np.linalg.inv(X_with_intercept.T @ X_with_intercept)
standard_errors = np.sqrt(np.diag(cov_matrix) * (rss / (N - d - 1)))
print(f"Standard Errors: {standard_errors[1:]}")  # Exclude intercept

# 6. t-statistic for each feature
t_stats = model.coef_ / standard_errors[1:]
print(f"t-statistics: {t_stats}")

# 7. p-value for each feature
p_values = [2 * (1 - stats.t.cdf(np.abs(t), df=N - d - 1)) for t in t_stats]
print(f"p-values: {p_values}")
```

```
Residual Sum of Squares (RSS): 15.170504359408238
Residual Standard Error (RSE): 0.2761044915394941
Mean Squared Error (MSE): 0.0743652174480796
R-squared (R2): 0.646842080055587
Standard Errors: [1.65762902 1.65275638 0.08367653 0.09237917]
t-statistics: [ 0.05549913  0.28078536 -1.04368222  4.01482765]
```

```
p-values:    [0.9557965077129815,    0.7791667407595964,    0.2978986235970691,
8.424166312304138e-05]
```

3.11 Will you be able to discard any features based on p-value?

Based on the p-values, some features can be excluded from the model. Typically, a p-value less than 0.05 indicates that a feature is statistically significant and plays an important role in predicting the dependent variable. In your situation, three features have p-values that are much higher than 0.05, indicating they are not statistically significant and could be removed from the model.

4. Performance evaluation of Linear regression

4.1 Compute Residual Standard Error (RSE) for Models A and B

The Residual Standard Error (RSE) is calculated using the formula:

$$RSE = \sqrt{\frac{SSE}{N - p}}$$

For Model A:

$$RSEA = \sqrt{\frac{9}{10000 - 3}} \approx \sqrt{\frac{9}{9997}} \approx \sqrt{0.0009003} \approx 0.03$$

For Model B:

$$RSEB = \sqrt{\frac{2}{10000 - 5}} \approx \sqrt{\frac{2}{9995}} \approx \sqrt{0.0002001} \approx 0.0141$$

Comparison Based on RSE: Model B has a lower RSE (0.0141) compared to Model A (0.03). This suggests that Model B has a better fit to the data since it has a smaller error on average.

4.2 Compute R-squared (R^2) for Models A and B

The R2 value is calculated using the formula:

$$R^2 = 1 - \frac{SSE}{TSS}$$

For Model A:

$$R_A^2 = 1 - \frac{9}{90} = 1 - 0.1 = 0.9$$

For Model B:

$$R_B^2 = 1 - \frac{2}{10} = 1 - 0.2 = 0.8$$

Comparison Based on R^2: Model A has a higher R^2 (0.9) compared to Model B (0.8), indicating that Model A explains more of the variance in the dependent variable than Model B

4.3 Which Performance Metric is Fairer for Comparing Two Models?

The choice between RSE and R^2 as a performance metric depends on what aspect of the models you want to evaluate:

- **RSE (Residual Standard Error)** provides an indication of the average size of the residuals (or errors) and is particularly useful when comparing models with different numbers of predictors. It gives a direct measure of how well the model fits the data, taking into account the actual differences between observed and predicted values.

- **R^2** measures how much of the variance in the dependent variable is explained by the model. However, R^2 does not consider the number of predictors used in the model, which can sometimes lead to misleading comparisons if the models have different complexities.

Generally, **RSE** might be considered a more fair metric when comparing models with varying complexities because it penalizes models with more parameters. This makes it a better choice for assessing which model fits the data more accurately while considering the model's complexity. On the other hand, **R^2** is still valuable for understanding how much of the variance in the outcome is explained by the model.

In your case, Model B has a lower RSE, indicating a better fit to the data, while Model A has a higher R^2, showing better explanatory power. If the goal is to determine which model fits the data better while considering model complexity, **RSE** might be the fairer metric to use.

5. Impact of Outliers on Linear Regression

5.1

Consider the following modified loss functions:

$$L1(w) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{r_i^2}{a^2 + r_i^2}\right) = \frac{1}{N}\sum_{i=1}^{N} L1_i$$

$$L2(w) = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \exp\left(-\frac{2|r_i|}{a}\right)\right) = \frac{1}{N}\sum_{i=1}^{N} L2_i$$

Where is the residual, is the predicted value, is the true value, and a is a hyper-parameter.

$\boldsymbol{Behavior\ as\ a\ \to 0}$

$\boldsymbol{For\ L1(w)}$:

$$\boldsymbol{L_1, i = \frac{r_i^2}{a^2 + r_i^2}}$$

As $a \to 0, a^2$ becomes negligible compared to r_i^2, so

$$L_1, i \approx \frac{r_i^2}{r_i^2} = 1$$

Thus, $L_1(w)$ treats all non-zero residuals similarly, making it less sensitive to the magnitude of outliers.

For $L_2(w)$:

$$\boldsymbol{L_2, i = 1 - exp\left(\frac{-2|r_i|}{a}\right)}$$

$As\ \ a \to 0, \boldsymbol{exp}\left(\frac{\boldsymbol{-2|r_i|}}{\boldsymbol{a}}\right) becomes\ very\ large\ for\ non-zero\ \boldsymbol{r_i}, so$

$$L_2, i \approx 1 - 0 = 1$$

Thus, $L_2(w)$ also behaves similarly to $L_1(w)$, giving a loss of 1 for large residuals, and thereby treating outliers less differently.

5.2

To minimize the influence of data points where $|r_i| \geq 40|r_i|$, we need a loss function that effectively reduces the impact of large residuals.

Choice of Function:

- For $L_1(w)$: As r_i becomes large $L_{1,i}$ levels off at 1, which effectively minimizes the influence of large residuals.
- For $L_2(w)$: Similarly, $L_{2,i}$ also levels off, but in a smoother manner. By adjusting the parameter, a, $L_2(w)$ allows for more controlled reduction of the impact from large residuals.

Value of a:

$$a \approx 40$$

Selecting a value of a close to 40 will help reduce the impact of residuals around 40 and above. This is because:

$$L_2, i \approx 1 - \exp\left(-\frac{2 \times 40}{40}\right) = 1 - \exp(-2) \approx 1 - 0.1353 = 0.8647$$

For residuals $|r_i| \geq 40|r_i|$, $L_{2,i}$ will approach 1, thereby significantly reducing their influence.

To effectively minimize the impact of large residuals, $L_2(w)$ with $a \approx 40$ is recommended, as it offers more precise control over how these residuals are penalized.

5.3

Both functions are effective in minimizing the impact of residuals $|r_i| \geq 40$. When comparing the two, the L2(w) function may be the better choice because of its exponential scaling, which aggressively reduces the influence of outliers. This is evident in Figure 3, where, for larger residuals, the loss value in L2 is lower compared to L1.

Regarding the value of a, suppose we want the loss to be capped at 1 when $|ri| = 40$. This would ensure that any residuals greater than 40 won't cause the loss to exceed 1. A lower value of a, in the range of 5 to 20, can achieve this by making $1 - \exp\left(\frac{-2|ri|}{a}\right) \approx 1$. Therefore, a value of a between 5 and 20 is suitable.