Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN3150-Pattern Recognition

Assignment 01: Learning from data and related challenges and linear models for regression

Rajapaksha I.P.D.D. - 210503H

Contents

1. Datapre-processing

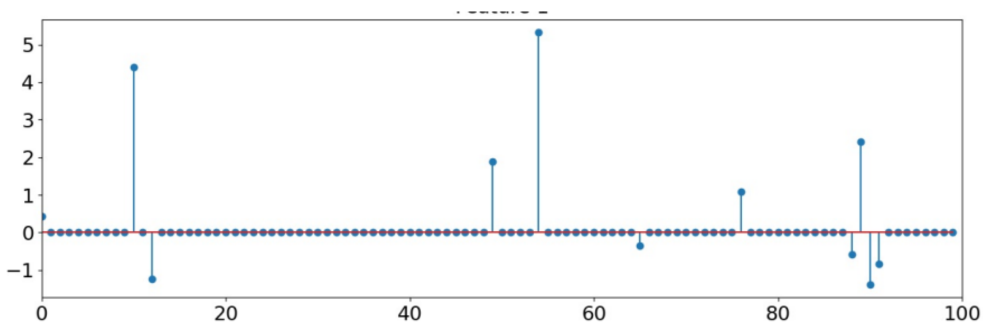


Figure 1: Feature 1 values of a dataset.

The values are mostly clustered around zero, with some occasional outliers reaching values between -1 and 5.Standard Scaling is suitable here because it will transform the feature to have a mean of 0 and a standard deviation of 1. This method handles outliers relatively well, preventing them from dominating the scaled values.

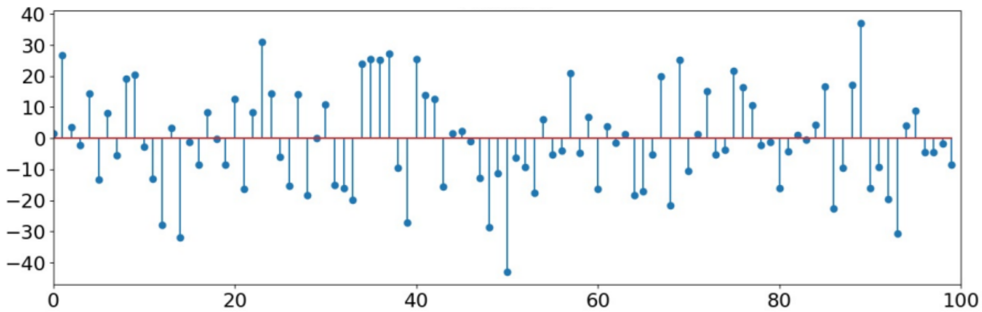


Figure 2: Feature 1 values of a dataset.

The feature values vary widely, ranging from approximately -40 to +40. The distribution is more spread out compared to Feature 1.Min-Max Scaling is suitable for Feature 2 because it scales the values to a [0, 1] range. This method is effective for maintaining the relationships between data points while minimizing the impact of outliers. It keeps the values within a defined range, preserving their relative distances.

2. Learning from data

2.1 Question 2 : Reason to training and testing data is different in each run

```
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()
```

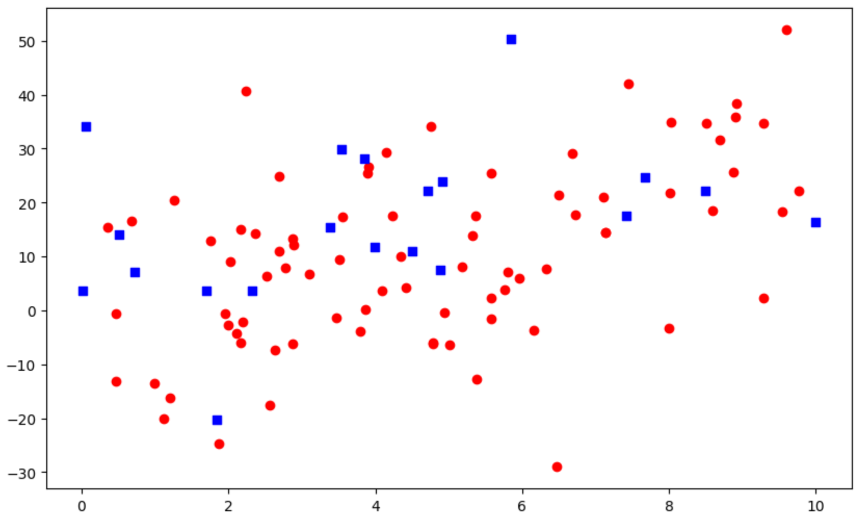


Figure 3: First Run

The reason the training and testing datasets differ in each run is due to the use of the `np.random.randint(104)` function, which generates a random integer each time it's called. This integer acts as a random seed for the `train_test_split` function, determining how the data is shuffled and then divided into training and testing sets.
When you set a specific value for the random state in the train-test data splitting, it guarantees that the same data points will be included in the training and testing sets every time the code is run, ensuring consistency. However, in this case, we are using `r=np.random.randint(104)` to generate a different random state for each run. Because `r` changes randomly every time the code is executed, it results in different data points being selected for training and testing, which is why we observe different scatter plots each time we run the code.

Extra part to compare difference

```python
import numpy as np
import matplotlib.pyplot as plt
#from sklearn.model_selection import train_test_split

# Generate four different random states
random_states = [np.random.randint(104) for _ in range(4)]

# Create a 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Loop over the random states and axes to create the plots
for i, ax in enumerate(axes.flat):
    # Split the data using the current random state
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
        random_state=random_states[i])

    # Plot the data points
    ax.scatter(X_train, Y_train, alpha=1, marker='o', color='red', label='Training
        Data')
    ax.scatter(X_test, Y_test, alpha=1, marker='s', color='blue', label='Testing Data
        ')

    # Add title and legend to each subplot
    ax.set_title(f'Random State: {random_states[i]}')
    ax.legend()

# Display the grid of images
plt.tight_layout()
plt.show()
```

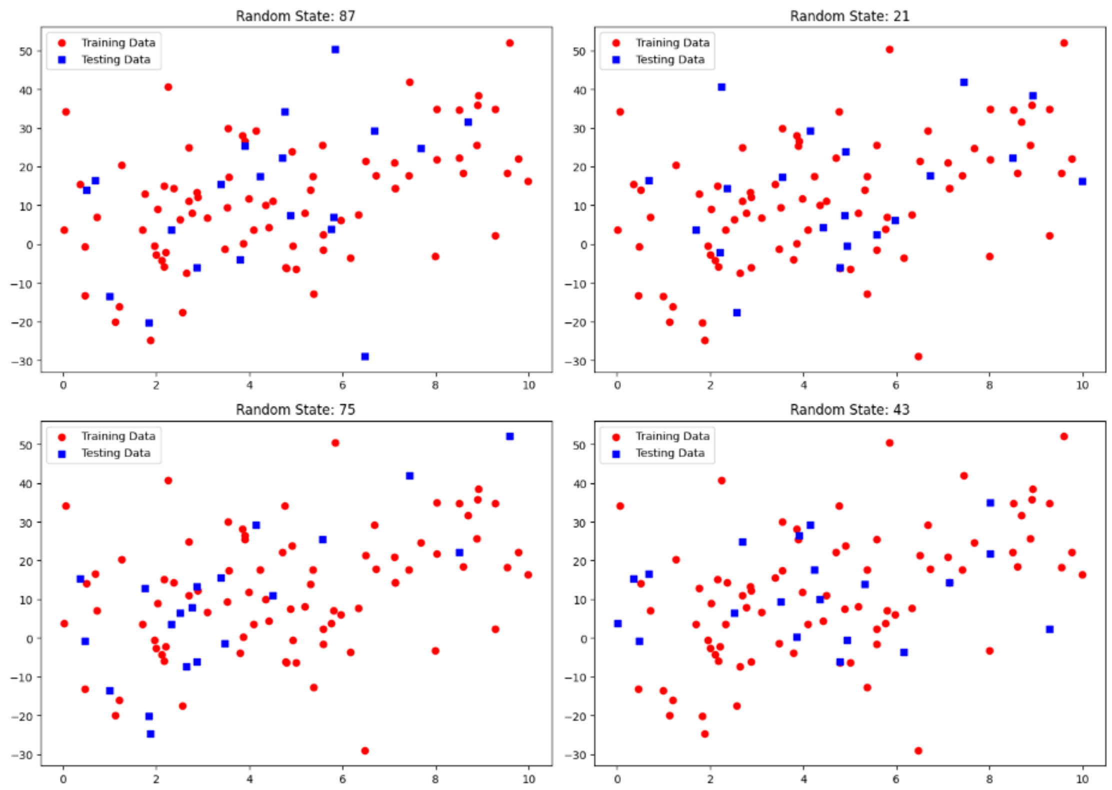


Figure 4: Code Output

When state change testing and training data set will change.But overall same data set was choosen for both.

2.2 Question 3 : Linear regression model is different from one instance to other

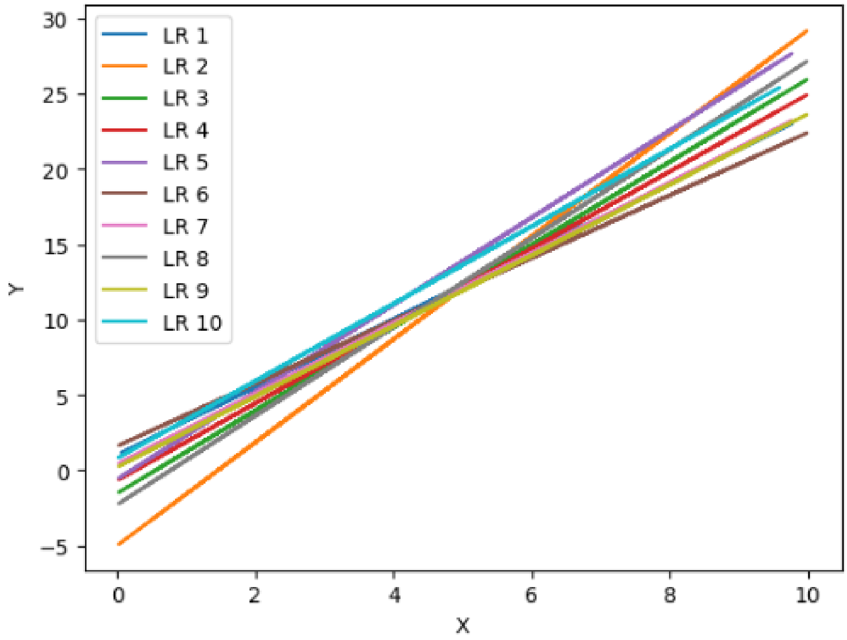


Figure 5: inear regression model

The linear regression model differs from one instance to another because the training dataset changes each time due to the random splitting of data using a randomly generated random state. Each time the `train_test_split` function is called with a new random state, different training data is selected. Since the model is trained on different data sets in each iteration, the model parameters (such as slope and intercept) change, resulting in different linear regression models for each iteration.

2.3 Question 4 : Increase the number of data samples to 10,000

```
# Generate 10,000 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()
```

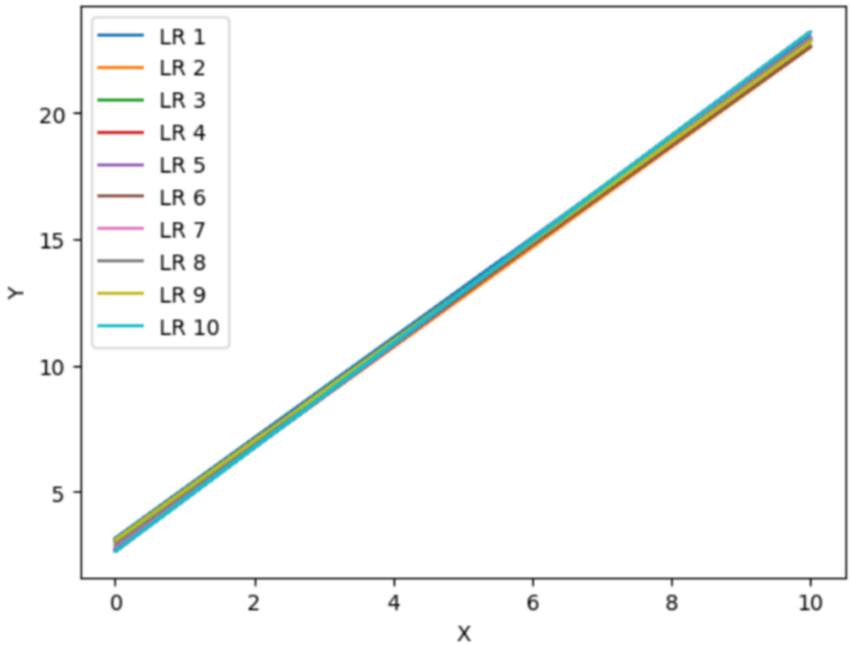


Figure 6: linear regression model with `n_sample`=10000

When the number of data samples is increased to 10,000, the variation in the linear regression models across different iterations will generally decrease. This happens because larger datasets provide more information, leading to more stable and consistent model training. With 10,000 samples, the training data is more representative of the overall data distribution, reducing the impact of random variations caused by different random states in the data splitting process. Consequently, the model parameters (such as slope and intercept) become more stable, resulting in similar linear regression lines across different iterations. In contrast, with only 100 samples, the random splits lead to greater variability in the training data, causing more fluctuations in the model parameters and the resulting regression lines.(When we increase the sample size population mean will equal to sample mean)

3. Linear regression on real world data

3.1 Question 1 : Load the dataset

Installing missing packge

```
!pip install ucimlrepo
```

3.2 Question 2 : Numer of independent variables and dependent variables in the data set

```
print(X.shape[1])#to cheak number of independent varibles
print(y.shape[1])#to cheak number of dependent variables
```

Number of independent variables = 33
Number of independent variables = 2

3.3 Question 3 : possibility to apply linear regression and steps before applying linear regression

```
print(infrared_thermography_temperature.metadata)
print(infrared_thermography_temperature.variables)
```

	name	role	type	demographic
0	SubjectID	ID	Categorical	None
1	aveOralF	Target	Continuous	None
2	aveOralM	Target	Continuous	None
3	Gender	Feature	Categorical	Gender
4	Age	Feature	Categorical	Age
5	Ethnicity	Feature	Categorical	Ethnicity
6	T_atm	Feature	Continuous	None
7	Humidity	Feature	Continuous	None
8	Distance	Feature	Continuous	None
9	T_offset1	Feature	Continuous	None

Figure 7: Output

It is not possible to apply linear regression directly on this dataset because it contains categorical data, such as 'Gender', 'Age', and 'Ethnicity'. Linear regression requires numerical input, so categorical data must be converted into a numerical form to proceed.

Steps:
1.One-Hot Encoding: Convert the categorical features ('Gender', 'Age', 'Ethnicity') into numerical values using one-hot encoding. This will create binary columns for each category, making them suitable for linear regression.
2.Feature Scaling: After encoding, features are scaled appropriately.
3.Handling Missing Values: Check for any missing values in the dataset remove them or replce them with average values.
4.Check for Multicollinearity: Cheak highly correlated features. If present, combine or remove correlated features to avoid redundancy and improve model performance.

3.4 Question 4 : NaN/missing values and correct approach

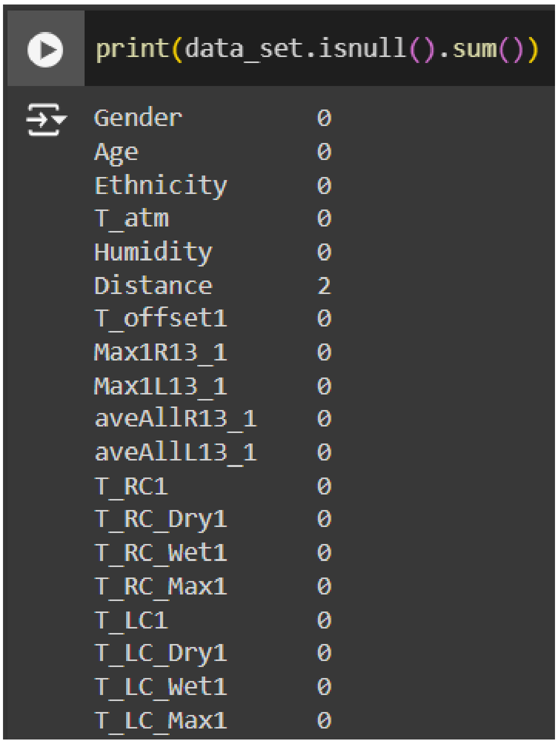
The provided code snippet attempts to handle missing values by dropping them separately from X and y:

```
# Drop rows with missing values from both X and y
X = X.dropna()
y = y.dropna()
```

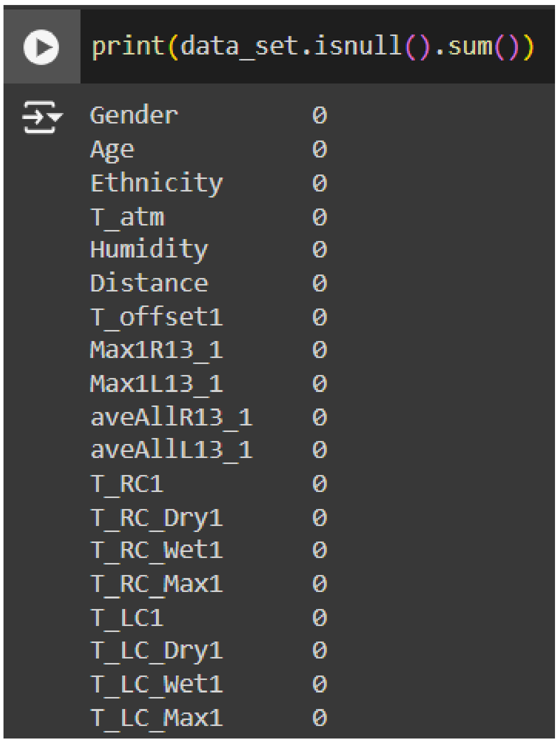
The provided method is wong because it removes missing values from X and y independently. This separate handling can cause the features and target labels to become misaligned. In a dataset, each row in the feature set X should correspond directly to a row in the target variable y. If rows are removed from X without removing the corresponding rows from y, or vice versa, the pairing of data points with their respective labels is disrupted.

To correctly handle the missing values, we first need to inspect the dataset to understand how many missing values are present. If only a few values are missing, it might be acceptable to drop the corresponding rows. **Corrected code:**

```
import pandas as pd
data_set = pd.concat([X, y], axis=1)
data_set = data_set.dropna(subset=['Distance'])
```



(a) before run the code



(b) Caption for second image

Figure 8: after run the code

3.5 Question 5 : Select 'aveOralM' as the dependent feature and 'T_atm', 'Humidity', 'Distance', 'T_FH_Max1', 'age' as independent variables

```
# One-hot encode the 'Age' categorical variable
age_encoded = pd.get_dummies(data_set['Age'], prefix='Age')

# Add the one-hot encoded columns back to the dataset and drop the original 'Age'
    column
data_set = pd.concat([data_set, age_encoded], axis=1)
data_set = data_set.drop('Age', axis=1)

# Select the dependent and independent variables
dependent_feature = data_set['aveOralM']
independent_features = data_set[['T_atm', 'Humidity', 'Distance', 'T_FH_Max1'] + list
    (age_encoded.columns)]

# Combine into a new DataFrame for modeling
model_data = pd.concat([independent_features, dependent_feature], axis=1)

# Display the prepared dataset
print("Prepared dataset:")
independent_features.head()
```

Prepared dataset:

	T_atm	Humidity	Distance	T_FH_Max1	Age_18-20	Age_21-25	Age_21-30	Age_26-30	Age_31-40	Age_41-50	Age_51-60	Age_>60
0	24.0	28.0	0.8	34.5300	False	False	False	False	False	True	False	False
1	24.0	26.0	0.8	34.6825	False	False	False	False	True	False	False	False
2	24.0	26.0	0.8	35.3450	False	False	True	False	False	False	False	False
3	24.0	27.0	0.8	35.6025	False	False	True	False	False	False	False	False
4	24.0	27.0	0.8	35.4175	True	False	False	False	False	False	False	False

Figure 9: Output

```
dependent_feature.head()
```

	aveOralM
0	36.59
1	37.19
2	37.34
3	37.09
4	37.04

dtype: float64

Figure 10: Output

3.6 Question 6 : Split the data into training and testing sets

```
X_train,X_test, y_train, y_test = train_test_split(independent_features,
    dependent_feature, test_size=0.2, random_state=42)
```

3.7 Question 7 : Train a linear regression model and estimate the coefficient corresponds to independent variables.

```
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

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

# Make predictions using the testing set
y_pred = model.predict(X_test)

# Creating the DataFrame with features and their coefficients
coeff_df = pd.DataFrame({'Feature': X_train.columns, 'Coefficient': model.coef_})

# Install the tabulate module
!pip install tabulate

# Import the tabulate function
from tabulate import tabulate

# Displaying the DataFrame as a table
print("Intercept:", model.intercept_)
print("Estimated features vs coefficients:")
print(tabulate(coeff_df, headers='keys', tablefmt='pretty'))
```

```
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (0.9.0)
Intercept: 13.238798996804732
Estimated features vs coefficients:
+----+-----------+-----------------------+
|    |  Feature  |      Coefficient      |
+----+-----------+-----------------------+
| 0  |   T_atm   | -0.055535893239226766 |
| 1  | Humidity  | 0.003422392005267033  |
| 2  | Distance  | 0.0026393537872089093 |
| 3  | T_FH_Max1 |  0.7092038079633628   |
| 4  | Age_18-20 | -0.09521023788460654  |
| 5  | Age_21-25 | -0.10911840778186253  |
| 6  | Age_21-30 | -0.03372839990148858  |
| 7  | Age_26-30 | -0.08394109824533079  |
| 8  | Age_31-40 | -0.13698172540270473  |
| 9  | Age_41-50 |  0.21089662330066358  |
| 10 | Age_51-60 | -0.08176010339345889  |
| 11 |  Age_>60  |  0.32984334930878834  |
+----+-----------+-----------------------+
```

Figure 11: Output

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'k--', lw=2, color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values')
plt.show()
```

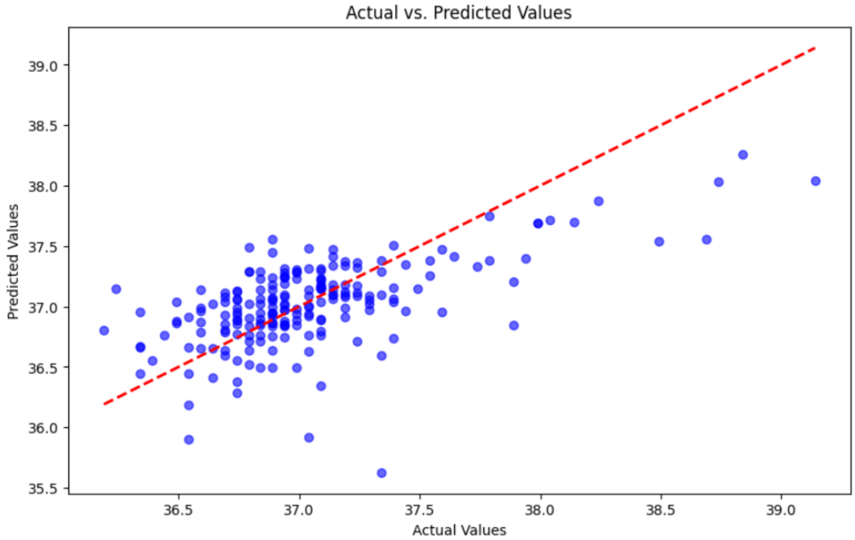


Figure 12: Output

3.8 Question 8 : Highly contributes independent variable.

```
# Plotting the coefficients
!pip install seaborn # install seaborn
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.barplot(x='Coefficient', y='Feature', data=coeff_df, palette='viridis')
plt.title('Feature Coefficients in Linear Regression Model')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()
```

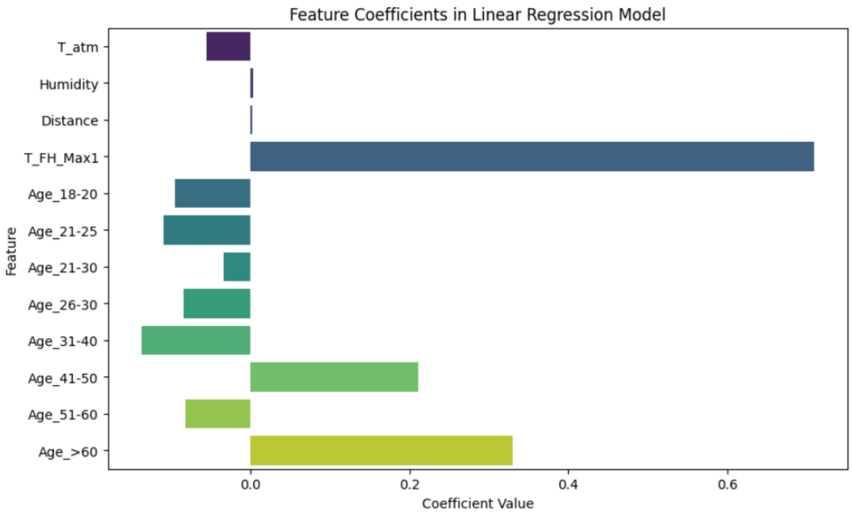


Figure 13: Output

By looking at the coefficient we can see `T_FH_Max1` got the maximum value which means `T_FH_Max1` contributes highly for the dependent feature

3.9 Question 9 : Select 'T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1' features as independent features

```python
# Select the dependent and independent variables
dependent_feature_n = data_set['aveOralM']
independent_features_n = data_set[['T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1'] ]
# Combine into a new DataFrame for modeling
model_data_n = pd.concat([independent_features_n, dependent_feature_n], axis=1)

# Display the prepared dataset
print("Prepared dataset:")
#independent_features_n.head()

X_train,X_test, y_train, y_test = train_test_split(independent_features_n,
    dependent_feature_n, test_size=0.2, random_state=42)

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

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

# Make predictions using the testing set
y_pred = model.predict(X_test)

# Creating the DataFrame with features and their coefficients
coeff_df = pd.DataFrame({'Feature': X_train.columns, 'Coefficient': model.coef_})

# Displaying the DataFrame as a table
print("Intercept:", model.intercept_)
print("Estimated features vs coefficients:")
print(tabulate(coeff_df, headers='keys', tablefmt='pretty'))
```

```
Prepared dataset:
Intercept: 6.79355629984887
Estimated features vs coefficients:
+---+------------+----------------------+
|   |  Feature   |     Coefficient      |
+---+------------+----------------------+
| 0 |   T_OR1    | 0.20545776323994563  |
| 1 | T_OR_Max1  | 0.34819684316002775  |
| 2 | T_FHC_Max1 | -0.08371846705362093 |
| 3 | T_FH_Max1  |  0.376564342065323   |
+---+------------+----------------------+
```

Figure 14: Output

```python
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'k--', lw=2, color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values')
plt.show()
```

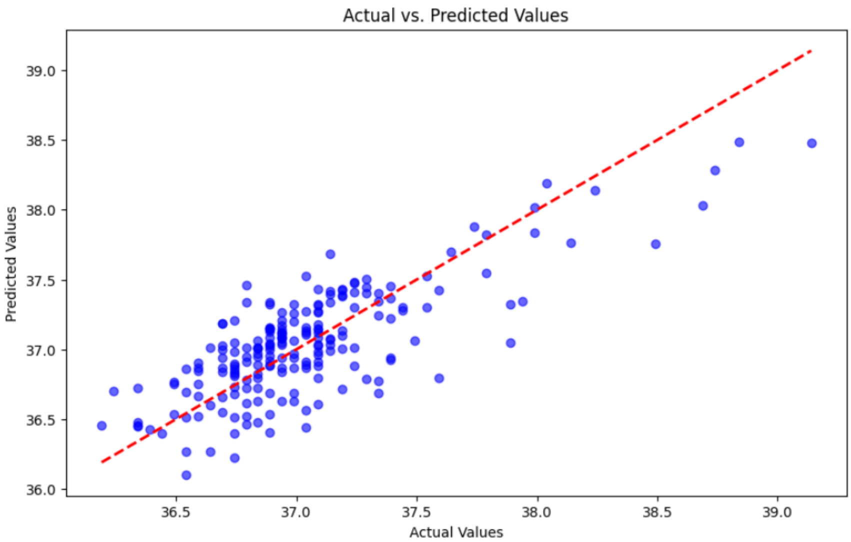


Figure 15: Output

3.10 Question 9 : Calculations

```python
yhat=model.predict(X_train)
d = independent_features_n.shape[1]
print('d:',d)

# Residual Sum of Sqaures (RSS)
RSS = np.sum((yhat - y_train)**2)
print('RSS=', RSS)

N=len(y_train)
#print('Number of Datapoints=',N)

# Residual Standard Error (RSE)
RSE = np.sqrt(1/(N-d-1)*RSS)
print('RSE=', RSE)

# Mean squre error(MSE)
#predictions = lm.predict(X_train)

newX = pd.DataFrame({"Constant":np.ones(len(X_train))}).join(pd.DataFrame(X_train))
MSE = (sum((y_train-yhat)**2))/(len(newX)-len(newX.columns))
print('MSE=', MSE)

# Total Sum of Squares (TSS)
TSS = np.sum((y_train- np.mean(y_train))**2)
print('TSS=', TSS)

```

```python
# R2

R2 = (TSS - RSS)/TSS
print(' R   (from direct calculations)=', R2)

# Calculation of R2 using sklearn
R2 = model.score(X_train,y_train)

print(' R   (from sklearn module)=', R2)
```

RSS= 77.97449082857881
RSE= 0.31045739777400483
MSE= 0.09638379583260669
TSS= 223.63911855036855
R^2 (from direct calculations)= 0.6513378726673116
R^2 (from sklearn module)= 0.6513378726673116

Calculate the standard error for each feature , t-statistic for each feature and p-value for each feature

```python
from scipy.stats import t
import numpy as np
import scipy.stats as stats
#samples size
SN = len(X_train)

features = ['T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1']
w_0 = model.intercept_
w_1 = model.coef_

NF = len(features)

#Calculate Standard error
standard_error = []
for feature in features:
  SE2 = RSS/(SN-NF-1) / np.sum((X_train[feature] - np.mean(X_train[feature]))**2)
  standard_error.append(np.sqrt(SE2))

#calculate t values
t_values = []
for i in range(NF):
  t_values.append(w_1[i]/standard_error[i])

#calculate p-values
p_values = []
for i in range(NF):
  p_values.append(2 * (1 - stats.t.cdf(abs(t_values[i]), SN-NF-1)))


# Create a DataFrame to display the values in a table
results_df = pd.DataFrame({
    'Feature': features,
    'Standard Error': standard_error,
    't-value': t_values,
    'p-value': p_values
})

# Print the table
print(results_df)

# Save the table to a CSV file (optional)
results_df.to_csv('model_statistics.csv', index=False)
```

	Feature	Standard Error	t-value	p-value
0	T_OR1	0.019226	10.686208	0.00000
1	T_OR_Max1	0.019219	18.117535	0.00000
2	T_FHC_Max1	0.018810	-4.450810	0.00001
3	T_FH_Max1	0.020771	18.129689	0.00000

Figure 16: Output

3.11 Question 11

The p-values are very low, which shows there's strong evidence against the null hypothesis. This means we can confidently reject the null hypothesis.

4. Performance evaluation of Linear regression

4.1 Question 2 : Residual standard error (RSE) for models A and B

The Residual Standard Error (RSE) formula is given by:

$$RSE = \sqrt{\frac{RSS}{N-d-1}}$$

For Model A:

$$RSE_A = \sqrt{\frac{9}{10^4-2-1}} = 0.03$$

For Model B:

$$RSE_B = \sqrt{\frac{2}{10^4-4-1}} = 0.0141$$

Since RSE_B is less than RSE_A, Model B shows a better fit compared to Model A.

4.2 Question 3 : R-squared (R2) for models A and B

The R^2 metric is computed as:

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

For Model A:

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

For Model B:

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

Based on the R^2 values, since R_A^2 is closer to 1 than R_B^2, Model A performs better in terms of variance explained by the model.

4.3 Question 4 : Between RSE and R-squared which performance metric is more fair

The R^2 value is generally preferred when comparing models because it is independent of the scale of y, making it a more consistent measure across different datasets.

5. Impact of Outliers on Linear Regression

5.1 Question2 : when a $\to$ 0

Let's define the loss functions $L_1(\omega)$ and $L_2(\omega)$:

$$L_1(\omega) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{r_i^2}{a^2 + r_i^2}\right)$$

As $a \to 0$:

$$\lim_{a\to 0} L_1(\omega) = \frac{1}{N}\sum_{i=1}^{N} 1 = 1$$

Now, for $L_2(\omega)$:

$$L_2(\omega) = \frac{1}{N}\sum_{i=1}^{N}\left(1 - e^{-\frac{2|r_i|}{a}}\right)$$

As $a \to 0$:

$$\lim_{a\to 0} L_2(\omega) = \frac{1}{N}\sum_{i=1}^{N} 1 = 1$$

Both $L_1(\omega)$ and $L_2(\omega)$ tend to 1 as a approaches 0, meaning they both become constant under this condition.

5.2 Question 3 : To minimize the influence of data points

When reducing the effect of outliers, such as for residuals $|r_i| \geq 40$, the loss function should be chosen based on how quickly it reduces the influence of these large residuals.
Comparing $L_1(\omega)$ and $L_2(\omega)$:
For $L_1(\omega)$:

$$L_1(\omega) = \frac{1}{N}\sum_{i=1}^{N}\frac{r_i^2}{a^2 + r_i^2}$$

This function diminishes the impact of outliers gradually. When the residuals are small, $L_1(\omega)$ behaves similarly to a squared loss. However, as the residuals grow, the function saturates, reducing the influence of larger residuals but not as quickly as $L_2(\omega)$.
For $L_2(\omega)$:

$$L_2(\omega) = \frac{1}{N}\sum_{i=1}^{N}\left(1 - e^{-\frac{2|r_i|}{a}}\right)$$

This loss function reduces the impact of large residuals more rapidly due to its exponential term. For smaller residuals, it behaves similarly to a linear loss function, but for larger residuals, it reaches saturation much faster.
Conclusion: Based on the plot, it is evident that $L_2(\omega)$ approaches saturation faster than $L_1(\omega)$, making it more suitable for handling outliers. By choosing $a = 40$, the loss function ensures that residuals with $|r_i| \geq 40$ have minimal impact on the model.