Department of Electronics & Telecommunication Engineering

University of Moratuwa

EN3150 - Pattern Recognition

Learning from data and related challenges and linear models for regression

EN3150 Assignment 01

Name: **Dilshan N.L.**
Index No: **210129P**

Date - 2024.09.02

Contents

1 Data Pre-Processing

Max-abs scaling is the preferred scaling method for the given features.

Reason: Max-Abs Scaling is ideal for this scenario because it scales the feature values relative to their maximum absolute value while preserving zero values. This method ensures that the zero values remain unchanged, which is crucial for maintaining the structure of the data if zeroes are meaningful in the feature. In contrast, Standard Scaling and Min-Max Scaling would shift the zero values or alter their meaning, which could be undesirable if preserving the original structure is important.

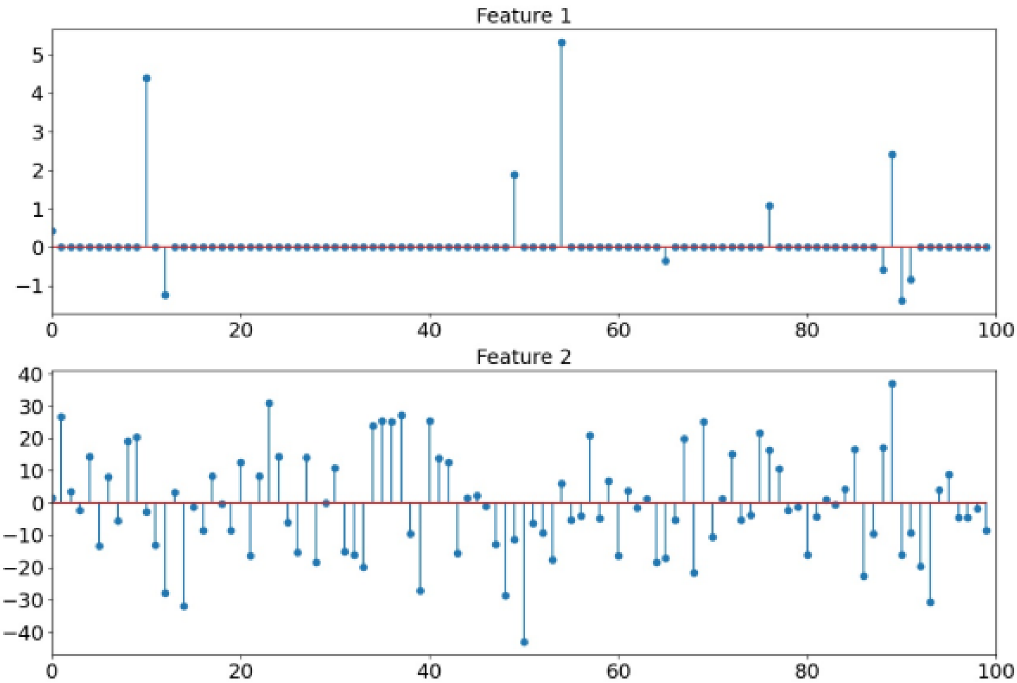


Figure 1: Feature values of a dataset.

2 Learning from Data

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

1. Generating Data Using Listing 1

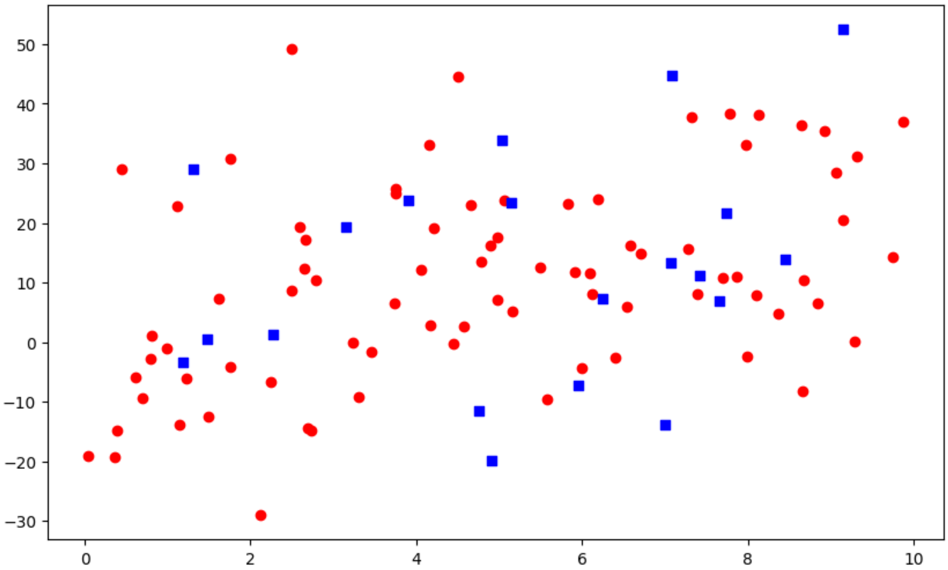
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# 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 + 2 * X + epsilon
Y = 3 + 2 * X + epsilon[:, np.newaxis]
```

2. Running Listing 2 and Observing Training and Testing Data

```
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()
```

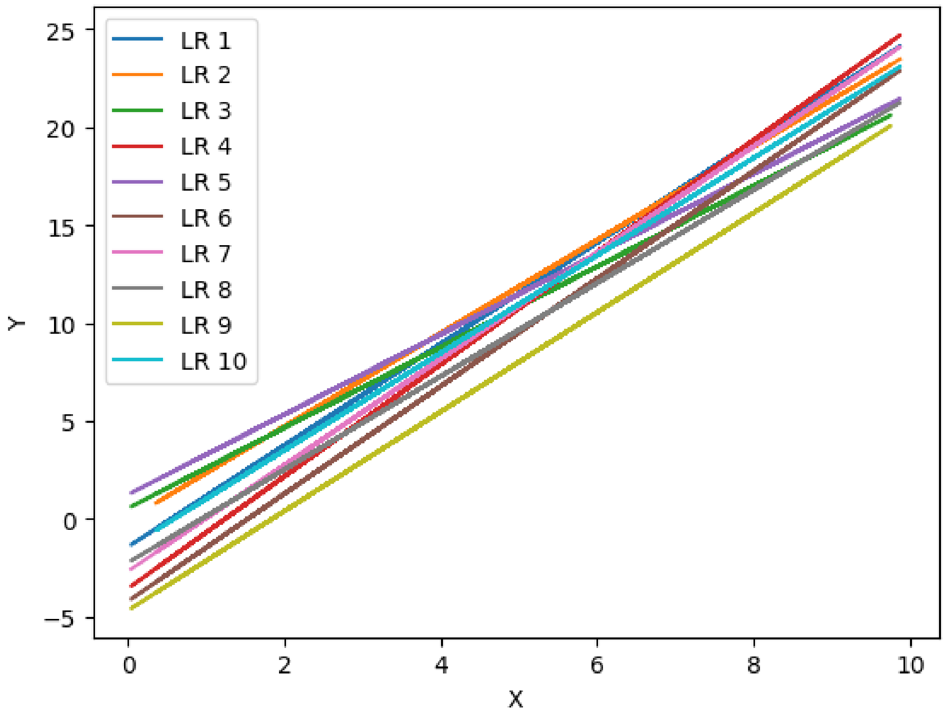
Observation:

- Each time we run the code, the training and testing data will be different. This is because the 'random_state' used in 'train_test_split' is generated using a random integer('r'), which changes on every run.
- **Reason**: The 'random_state' controls the shuffling of data before splitting into training and test sets. Since 'r' changes each time, the split is different in each run.



3. Fitting Linear Regression Model and Observing Different Instances

```
for i in range(10):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=np.random.
    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()
```

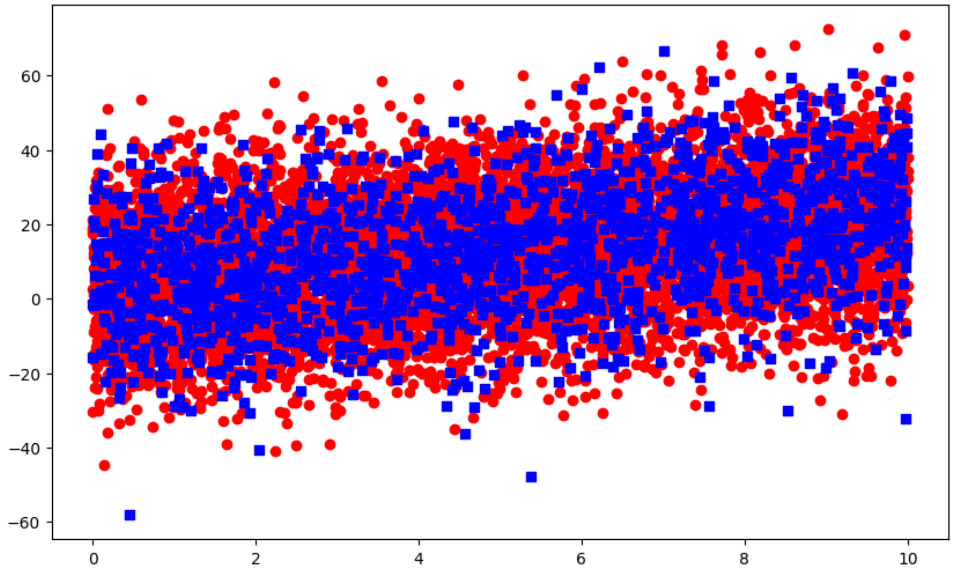


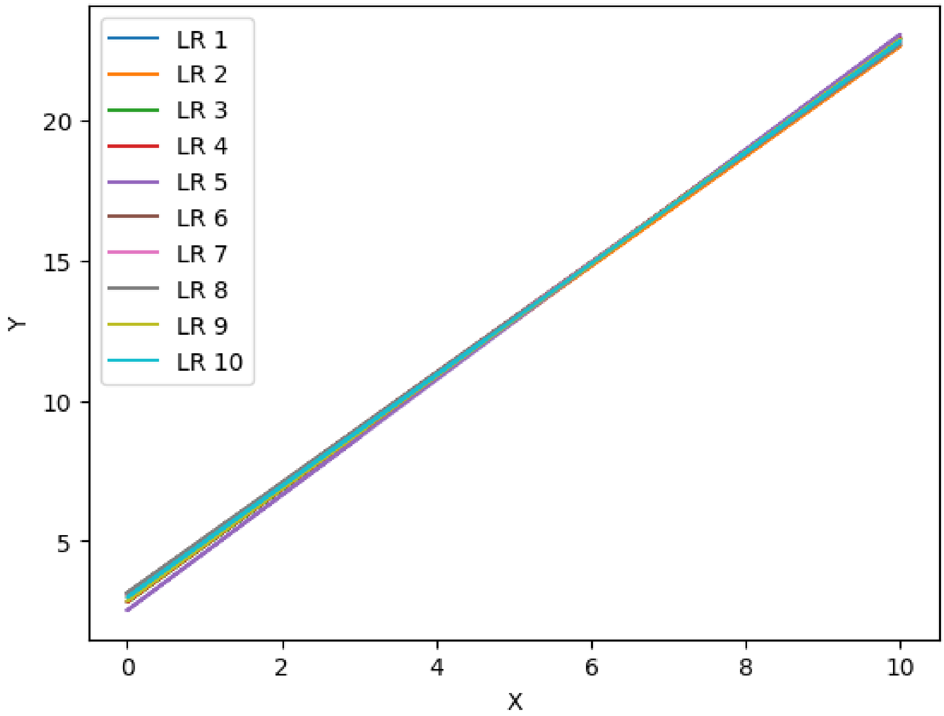
Observation:

- The linear regression model will vary slightly between each instance.
- Reason: Each time the data is split differently due to the changing 'random_state', the training data the model learns from is different. This leads to slight variations in the fitted model.

4. 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 + 2 * X + epsilon
Y = 3 + 2 * X + epsilon[:, np.newaxis]

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()

for i in range(10):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=np.random.
    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()
```





Observation:

- When the number of samples is increased to 10000, the linear regression model instances will exhibit much less variation compared to when there are only 100 samples.
- Reason:
  - Larger Dataset: With a larger dataset, the training data becomes more representative of the entire data distribution. Thus, even with different random splits, the model tends to converge towards a more consistent fit.
  - Reduced Impact of Random Variations: Random variations in the training data have less influence when there are more samples, leading to more stable models across different instances.

3 Linear regression on real world data

1. Loading the Dataset

```
# 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)
```

2. Independent and Dependent Variables

- Independent variables: These are the features in X.
- Dependent variables: These are the target values in y.

```
print(infrared_thermography_temperature.data)
```

```
{'ids':       SubjectID
0       161117-1
1       161117-2
2       161117-3
3       161117-4
4       161117-5
...          ...
1015  180425-05
1016  180425-06
1017  180502-01
1018  180507-01
1019  180514-01

[1020 rows x 1 columns], 'features':       Gender    Age                  Ethnicity  T_atm  Humidity
0       Male  41-50                      White   24.0      28.0       0.8
1     Female  31-40  Black or African-American   24.0      26.0       0.8
2     Female  21-30                      White   24.0      26.0       0.8
3     Female  21-30  Black or African-American   24.0      27.0       0.8
4       Male  18-20                      White   24.0      27.0       0.8
...      ...    ...                        ...    ...       ...       ...
1015  Female  21-25                      Asian   25.7      50.8       0.6
1016  Female  21-25                      White   25.7      50.8       0.6
1017  Female  18-20  Black or African-American   28.0      24.3       0.6
1018    Male  26-30            Hispanic/Latino   25.0      39.8       0.6
1019  Female  18-20                      White   23.8      45.6       0.6

      T_offset1  Max1R13_1  Max1L13_1  aveAllR13_1  ...  T_FHCC1  T_FHRC1  \
0        0.7025    35.0300    35.3775      34.4000  ...  33.5775  33.4775
1        0.7800    34.5500    34.5200      33.9300  ...  34.0325  34.0550
2        0.8625    35.6525    35.5175      34.2775  ...  34.9000  34.8275
3        0.9300    35.2225    35.6125      34.3850  ...  34.4400  34.4225
4        0.8950    35.5450    35.6650      34.9100  ...  35.0900  35.1600
...         ...        ...        ...          ...  ...      ...      ...
```

```
1015     1.2225    35.6425    35.6525      34.8575  ...  35.1075  35.3475
1016     1.4675    35.9825    35.7575      35.4275  ...  35.3100  35.2175
1017     0.1300    36.4075    36.3400      35.8700  ...  35.4350  35.2400
1018     1.2450    35.8150    35.5250      34.2950  ...  34.8400  35.0200
1019     0.8675    35.7075    35.5825      34.8875  ...  34.5475  34.6500

      T_FHLC1  T_FHBC1  T_FHTC1  T_FH_Max1  T_FHC_Max1   T_Max1    T_OR1  \
0     33.3725  33.4925  33.0025    34.5300     34.0075  35.6925  35.6350
1     33.6775  33.9700  34.0025    34.6825     34.6600  35.1750  35.0925
2     34.6475  34.8200  34.6700    35.3450     35.2225  35.9125  35.8600
3     34.6550  34.3025  34.9175    35.6025     35.3150  35.7200  34.9650
4     34.3975  34.6700  33.8275    35.4175     35.3725  35.8950  35.5875
...       ...      ...      ...        ...         ...      ...      ...
1015  35.4000  35.1375  35.2750    35.8525     35.7475  36.0675  35.6775
1016  35.2200  35.2075  35.0700    35.7650     35.5525  36.5000  36.4525
1017  35.2275  35.3675  35.3425    36.3750     35.7100  36.5350  35.9650
1018  34.9250  34.7150  34.5950    35.4150     35.3100  35.8600  35.4150
1019  34.6700  34.2150  34.7100    35.1525     35.1175  35.9725  35.8900

      T_OR_Max1
0       35.6525
1       35.1075
2       35.8850
3       34.9825
4       35.6175
...         ...
1015    35.7100
1016    36.4900
1017    35.9975
1018    35.4350
1019    35.9175

[1020 rows x 33 columns], 'targets':       aveOralF  aveOralM
0        36.85     36.59
1        37.00     37.19
2        37.20     37.34
3        36.85     37.09
4        36.80     37.04
...        ...       ...
1015     36.95     36.99
1016     37.25     37.19
1017     37.35     37.59
1018     37.15     37.29
1019     37.05     37.19

[1020 rows x 2 columns], 'original':       SubjectID  aveOralF  aveOralM  Gender    Age
0      161117-1     36.85     36.59    Male  41-50                    White
1      161117-2     37.00     37.19  Female  31-40  Black or African-American
2      161117-3     37.20     37.34  Female  21-30                    White
3      161117-4     36.85     37.09  Female  21-30  Black or African-American
4      161117-5     36.80     37.04    Male  18-20                    White
...         ...       ...       ...     ...    ...                      ...
1015  180425-05     36.95     36.99  Female  21-25                    Asian
1016  180425-06     37.25     37.19  Female  21-25                    White
1017  180502-01     37.35     37.59  Female  18-20  Black or African-American
1018  180507-01     37.15     37.29    Male  26-30          Hispanic/Latino
1019  180514-01     37.05     37.19  Female  18-20                    White
```

```
      T_atm  Humidity  Distance  T_offset1  ...  T_FHCC1  T_FHRC1  T_FHLC1  \
0      24.0      28.0       0.8     0.7025  ...  33.5775  33.4775  33.3725
1      24.0      26.0       0.8     0.7800  ...  34.0325  34.0550  33.6775
2      24.0      26.0       0.8     0.8625  ...  34.9000  34.8275  34.6475
3      24.0      27.0       0.8     0.9300  ...  34.4400  34.4225  34.6550
4      24.0      27.0       0.8     0.8950  ...  35.0900  35.1600  34.3975
...     ...       ...       ...        ...  ...      ...      ...      ...
1015   25.7      50.8       0.6     1.2225  ...  35.1075  35.3475  35.4000
1016   25.7      50.8       0.6     1.4675  ...  35.3100  35.2175  35.2200
1017   28.0      24.3       0.6     0.1300  ...  35.4350  35.2400  35.2275
1018   25.0      39.8       0.6     1.2450  ...  34.8400  35.0200  34.9250
1019   23.8      45.6       0.6     0.8675  ...  34.5475  34.6500  34.6700

      T_FHBC1  T_FHTC1  T_FH_Max1  T_FHC_Max1   T_Max1    T_OR1  T_OR_Max1
0     33.4925  33.0025    34.5300     34.0075  35.6925  35.6350    35.6525
1     33.9700  34.0025    34.6825     34.6600  35.1750  35.0925    35.1075
2     34.8200  34.6700    35.3450     35.2225  35.9125  35.8600    35.8850
3     34.3025  34.9175    35.6025     35.3150  35.7200  34.9650    34.9825
4     34.6700  33.8275    35.4175     35.3725  35.8950  35.5875    35.6175
...       ...      ...        ...         ...      ...      ...        ...
1015  35.1375  35.2750    35.8525     35.7475  36.0675  35.6775    35.7100
1016  35.2075  35.0700    35.7650     35.5525  36.5000  36.4525    36.4900
1017  35.3675  35.3425    36.3750     35.7100  36.5350  35.9650    35.9975
1018  34.7150  34.5950    35.4150     35.3100  35.8600  35.4150    35.4350
1019  34.2150  34.7100    35.1525     35.1175  35.9725  35.8900    35.9175

[1020 rows x 36 columns], 'headers': Index(['SubjectID', 'aveOralF', 'aveOralM', 'Gender', 'Age', 'E
       'T_atm', 'Humidity', 'Distance', 'T_offset1', 'Max1R13_1', 'Max1L13_1',
       'aveAllR13_1', 'aveAllL13_1', 'T_RC1', 'T_RC_Dry1', 'T_RC_Wet1',
       'T_RC_Max1', 'T_LC1', 'T_LC_Dry1', 'T_LC_Wet1', 'T_LC_Max1', 'RCC1',
       'LCC1', 'canthiMax1', 'canthi4Max1', 'T_FHCC1', 'T_FHRC1', 'T_FHLC1',
       'T_FHBC1', 'T_FHTC1', 'T_FH_Max1', 'T_FHC_Max1', 'T_Max1', 'T_OR1',
       'T_OR_Max1'],
      dtype='object')}
```

```python
print(f"Number of Independent Variables: {X.shape[1]}")
print(f"Number of Dependent Variables: {y.shape[1]}")
print(X,y)
```

```
Number of Independent Variables: 33
Number of Dependent Variables: 2
```

3. Is it possible to apply linear regression?

In this dataset, we have non-numeric data such as age ranges, sex, and other categorical variables. To apply linear regression to these types of data, they need to be converted into a numerical format. This can be achieved using the following methods:

1. **Label Encoding**: Assigns a unique integer to each category. This is suitable for ordinal data where categories have a meaningful order, such as 'low', 'medium', and 'high'.
2. **One-Hot Encoding**: Creates binary columns for each category, indicating the presence or absence of each category. This method is ideal for nominal data without an inherent order, such as 'sex' or 'ethnicity'.
3. **Ordinal Encoding**: Assigns integer values to categories based on their inherent order. This method is appropriate for ordinal variables where the sequence of categories carries significance, such as 'age ranges'.
4. **Binning**: Converts continuous variables into discrete categories or bins. This is useful for grouping continuous data, like 'age', into meaningful ranges.

By employing these encoding techniques, non-numeric data can be effectively transformed into a numerical format suitable for linear regression analysis.

4. Handling NaN/Missing Values

The provided code is not correct. Because we must remove both the X and y values corresponding to a missing value.

table.dropna() ensures that we remove rows with any missing values across the entire dataset, maintaining consistency and alignment. X.dropna() and y.dropna() separately might lead to mismatched data and additional complexity, especially when dealing with feature and target data.

```python
import pandas as pd

table = pd.concat([X, y], axis = 1)
# Count missing values for each column
missing_values_per_column = table.isnull().sum()
print("Missing values per column:")
print(missing_values_per_column)
# Count the total number of missing values in the DataFrame
total_missing_values = table.isnull().sum().sum()
print(f"Total number of missing values in the DataFrame: {total_missing_values}")
```

```
Missing values per column:
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_FHLC1         0
T_FHBC1         0
T_FHTC1         0
T_FH_Max1       0
T_FHC_Max1      0
T_Max1          0
T_OR1           0
T_OR_Max1       0
aveOralF        0
aveOralM        0
dtype: int64
```

```
Total number of missing values in the DataFrame: 2
```

```python
table = table.dropna()
```

```python
# Count missing values for each column
missing_values_per_column = table.isnull().sum()
print("Missing values per column:")
print(missing_values_per_column)
# Count the total number of missing values in the DataFrame
total_missing_values = table.isnull().sum().sum()
print(f"Total number of missing values in the DataFrame: {total_missing_values}")
```

```
Missing values per column:
Gender          0
Age             0
Ethnicity       0
T_atm           0
Humidity        0
Distance        0
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_FHLC1         0
T_FHBC1         0
T_FHTC1         0
T_FH_Max1       0
T_FHC_Max1      0
T_Max1          0
T_OR1           0
T_OR_Max1       0
aveOralF        0
aveOralM        0
dtype: int64
Total number of missing values in the DataFrame: 0
```

5. and 6. Selecting Features and Splitting Data

```python
# Selecting 'aveOralM' as the dependent variable
y = y[['aveOralM']]

# Selecting 'Age' and four other features based on preference
X = X[['Age', 'T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1']]

print(X,y)
```

```python
# Splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
        Age    T_OR1  T_OR_Max1  T_FHC_Max1  T_FH_Max1
0     41-50  35.6350    35.6525     34.0075    34.5300
1     31-40  35.0925    35.1075     34.6600    34.6825
2     21-30  35.8600    35.8850     35.2225    35.3450
3     21-30  34.9650    34.9825     35.3150    35.6025
4     18-20  35.5875    35.6175     35.3725    35.4175
...     ...      ...        ...         ...        ...
1015  21-25  35.6775    35.7100     35.7475    35.8525
1016  21-25  36.4525    36.4900     35.5525    35.7650
1017  18-20  35.9650    35.9975     35.7100    36.3750
1018  26-30  35.4150    35.4350     35.3100    35.4150
1019  18-20  35.8900    35.9175     35.1175    35.1525

[1020 rows x 5 columns]       aveOralM
0         36.59
1         37.19
2         37.34
3         37.09
4         37.04
...         ...
1015      36.99
1016      37.19
1017      37.59
1018      37.29
1019      37.19

[1020 rows x 1 columns]
```

7. Training a Linear Regression Model

```python
print(X.columns)
```

```
Index(['Age', 'T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1'], dtype='object')
```

```python
print(X.Age)
```

```
0       41-50
1       31-40
2       21-30
3       21-30
4       18-20
        ...
1015    21-25
1016    21-25
1017    18-20
1018    26-30
1019    18-20
Name: Age, Length: 1020, dtype: object
```

```python
def convert_age_range(age_range):
    """Converts the age range to a single average value"""
    if '>' in age_range:
        return int(age_range.replace('>', '').strip())
    lower, upper = map(int, age_range.split('-'))
    return (lower + upper) / 2
```

```
X.Age = X.Age.apply(convert_age_range)
print(X.Age)
```

```
0       45.5
1       35.5
2       25.5
3       25.5
4       19.0
        ...
1015    23.0
1016    23.0
1017    19.0
1018    28.0
1019    19.0
Name: Age, Length: 1020, dtype: float64
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_27752\2353021610.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
  X.Age = X.Age.apply(convert_age_range)
```

```
from sklearn.linear_model import LinearRegression

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)

# Coefficients corresponding to independent variables
coefficients = model.coef_
print(f"Estimated Coefficients: {coefficients}")
```

```
Estimated Coefficients: [[ 0.00113644  0.05647584  0.49937613 -0.08398371  0.36994022]]
```

8. Identifying the Most Contributing Variable

The variable with the highest absolute value in the coefficient array contributes the most:

```
import numpy as np

max_contributor_index = np.argmax(np.abs(coefficients))
most_contributing_feature = X.columns[max_contributor_index]
print(f"Most contributing feature: {most_contributing_feature}")
```

```
Most contributing feature: T_OR_Max1
```

9. Additional Feature Selection and Model Training

```
X = X[['T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1']]
print(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model.fit(X_train, y_train)
coefficients = model.coef_
print(f"Estimated Coefficients: {coefficients}")
```

```
        T_OR1  T_OR_Max1  T_FHC_Max1  T_FH_Max1
0     35.6350    35.6525     34.0075    34.5300
1     35.0925    35.1075     34.6600    34.6825
2     35.8600    35.8850     35.2225    35.3450
```

```
3     34.9650    34.9825     35.3150    35.6025
4     35.5875    35.6175     35.3725    35.4175
...       ...        ...         ...        ...
1015  35.6775    35.7100     35.7475    35.8525
1016  36.4525    36.4900     35.5525    35.7650
1017  35.9650    35.9975     35.7100    36.3750
1018  35.4150    35.4350     35.3100    35.4150
1019  35.8900    35.9175     35.1175    35.1525

[1020 rows x 4 columns]
Estimated Coefficients: [[ 0.09199696  0.4640698  -0.08733171  0.37088645]]
```

10. Calculating Statistical Measures

```python
from sklearn.metrics import mean_squared_error

# Residual sum of squares (RSS)
y_pred = model.predict(X_test)
RSS = np.sum(np.square(y_test - y_pred))

# Residual Standard Error (RSE)
N = len(y_test)
d = X_train.shape[1]
RSE = np.sqrt(RSS / (N - d - 1))

# Mean Squared Error (MSE)
MSE = mean_squared_error(y_test, y_pred)

# R-squared statistic
R_squared = model.score(X_test, y_test)

# Standard Error, t-statistic, p-value
import statsmodels.api as sm

X_train_with_const = sm.add_constant(X_train)
ols_model = sm.OLS(y_train, X_train_with_const).fit()
standard_errors = ols_model.bse
t_statistics = ols_model.tvalues
p_values = ols_model.pvalues

print(f"RSS: {RSS}")
print(f"RSE: {RSE}")
print(f"MSE: {MSE}")
print(f"R-squared: {R_squared}")
print(f"Standard Errors: {standard_errors}")
print(f"t-statistics: {t_statistics}")
print(f"p-values: {p_values}")
```

```
c:\Users\HP\AppData\Local\Programs\Python\Python311\Lib\site-packages\numpy\core\fromnumeric.py:86:
  return reduction(axis=axis, out=out, **passkwargs)
```

```
RSS: aveOralM    15.170504
dtype: float64
RSE: aveOralM    0.276104
dtype: float64
MSE: 0.07436521744807979
R-squared: 0.6468420800555861
Standard Errors: const        0.803926
T_OR1         0.883501
```

```
T_OR_Max1      0.882069
T_FHC_Max1     0.044464
T_FH_Max1      0.049258
dtype: float64
t-statistics: const         8.753146
T_OR1          0.104128
T_OR_Max1      0.526115
T_FHC_Max1    -1.964102
T_FH_Max1      7.529419
dtype: float64
p-values: const         1.191574e-17
T_OR1          9.170938e-01
T_OR_Max1      5.989521e-01
T_FHC_Max1     4.985945e-02
T_FH_Max1      1.358512e-13
dtype: float64
```

11. Significant and Insignificant features

In linear regression, we consider a feature significant if its p-value is less than 0.05. Conversely, if the p-value is greater than or equal to 0.05, we regard the feature as insignificant.

```python
significant_features = p_values[p_values < 0.05]
insignificant_features = p_values[p_values >= 0.05]

print(f"Significant Features: {significant_features}")
print(f"Insignificant Features: {insignificant_features}")
```

```
Significant Features: const         1.191574e-17
T_FHC_Max1     4.985945e-02
T_FH_Max1      1.358512e-13
dtype: float64
Insignificant Features: T_OR1        0.917094
T_OR_Max1    0.598952
dtype: float64
```

4 Performance evaluation of Linear regression

2. Residual Standard Error (RSE)

The Residual Standard Error (RSE):

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

N = Total number of data samples
d = The number of independent features

Model A:

$$\text{RSE}_A = \sqrt{\frac{9}{10000 - 2 - 1}} \approx \sqrt{\frac{9}{9997}} \approx 0.03$$

Model B:

$$\text{RSE}_B = \sqrt{\frac{2}{10000 - 4 - 1}} \approx \sqrt{\frac{2}{9995}} \approx 0.01$$

- Since Model B has a lower RSE, Model B fits more with the dataset.

3. R-squared (R^2)

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

Model A:

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

Model B:

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

- Model A has a higher R^2, indicating it explains more variance in the response variable.

4. Metrics Comparison

1. Scale Independence:

- **R^2:** R^2 is a unitless measure that indicates the proportion of variance in the dependent variable that is explained by the model. This makes it scale-independent, meaning it remains consistent regardless of the range or units of the data. This property allows for fair comparisons between models across different datasets or variables with varying scales.
- **RSE:** RSE is measured in the same units as the dependent variable, so its value can vary depending on the scale of the data. This means RSE's value is influenced by the range of the dataset, making it less straightforward to compare models if the datasets have different units or scales.

2. Normalized Benchmark for Model Performance:

- **R^2:** Since R^2 ranges from 0 to 1, it provides a normalized benchmark for evaluating how well a model explains the variance in the data. A higher R^2 value indicates better model performance, making it easier to assess and compare models directly.
- **RSE:** Although RSE indicates the average size of residuals, its absolute value can be influenced by the data's scale and units. This makes it harder to compare models across different datasets, as the RSE values are not normalized and can vary with the scale of the outcome variable.

5 Linear regression impact on outliers

2. What happens when $a \to 0$?

When a approaches 0, both modified loss functions $L_1(w)$ and $L_2(w)$ change their behavior significantly. Let's consider each function:

- **For $L_1(w)$:**

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

As a approaches 0, the term a^2 becomes negligible compared to r_i^2, so:

$$L_1(w) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\frac{r_i^2}{r_i^2} \right) = \frac{1}{N} \sum_{i=1}^{N} 1 = 1$$

This implies that $L_1(w)$ converges to 1 for every data point, making the loss function independent of the residuals r_i. Essentially, the influence of outliers becomes uniform across all data points.

- **For $L_2(w)$:**

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

As a approaches 0, $\frac{2|r_i|}{a}$ becomes very large, so $\exp\left(-\frac{2|r_i|}{a}\right)$ approaches 0. Thus:

$$L_2(w) \approx \frac{1}{N} \sum_{i=1}^{N} (1 - 0) = 1$$

Similar to $L_1(w)$, $L_2(w)$ also converges to 1 for all data points, meaning that all residuals are treated the same regardless of their size.

3. Minimizing the influence of data points with $|r_i| \geq 40$

To minimize the influence of data points with $|r_i| \geq 40$, we need to choose a value of a and a loss function that allows us to easily identify points where $|r_i| \geq 40$.

Thus, we should select a loss function and an a value such that the loss function increases significantly for $|r_i| \geq 40$ and remains relatively smaller for $|r_i| < 40$.

In my opinion, the L_1 loss function with $a = 25$ is a better choice for this purpose.

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the range of r_i values
r_i = np.linspace(-50, 50, 500)

# Define the values of a
a_values = [2.5, 25, 100]

# Define the L1 and L2 loss functions
def L1(r_i, a):
    return r_i**2 / (a**2 + r_i**2)

def L2(r_i, a):
    return  1 - np.exp(-2 * np.abs(r_i) / a)

# Plot L1 and L2 Loss Functions
plt.figure(figsize=(14, 7))
```

```
# Plot L1 Loss Function for different values of a
for a in a_values:
    plt.plot(r_i, L1(r_i, a), label=f'L1 Loss with a={a}')

# Plot L2 Loss Function for different values of a
for a in a_values:
    plt.plot(r_i, L2(r_i, a), linestyle='--', label=f'L2 Loss with a={a}')

# Add labels and legend
plt.xlabel('$r_i$')
plt.ylabel('Loss')
plt.title('Comparison of Loss Functions: $L_1$ and $L_2$')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```