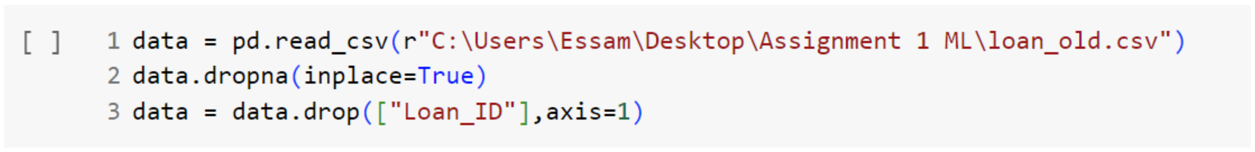
Assignment #1

Name	ID
Essam Alaa Nady	20216064
Ziyad Alfikie	20216043
Mariam Behairy	20216094
Mahitab Yasser	20216082
Ahmed El Tokhy	20216014

Introduction:

In this assignment, we are given a company dataset to automate the Loan Eligibility Process in a real time scenario related to customer's detail provided while applying application for home loan forms.

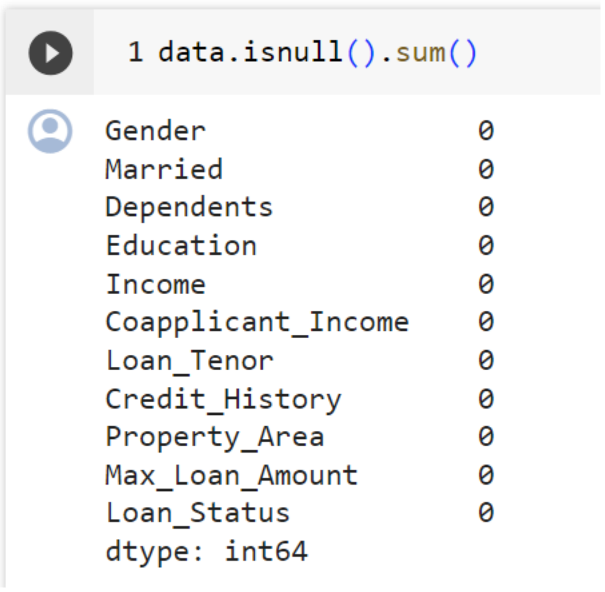
Loading dataset:



```
[ ]  data = pd.read_csv(r"C:\Users\Essam\Desktop\Assignment 1 ML\loan_old.csv")
     data.dropna(inplace=True)
     data = data.drop(["Loan_ID"],axis=1)
```

Data Analysis:

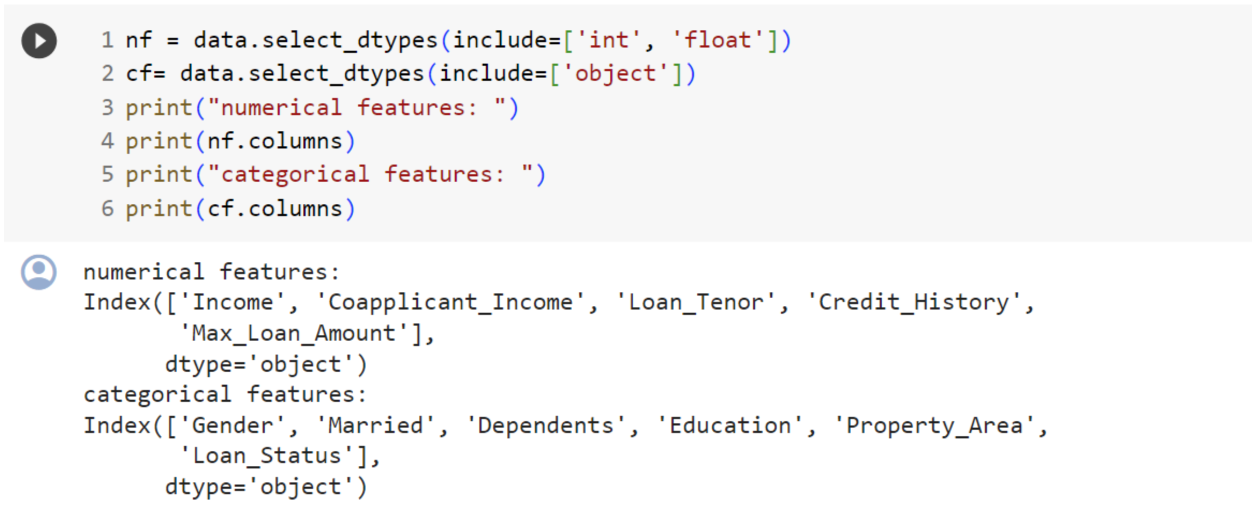
1- Checking whether there are missing values:



```
data.isnull().sum()
```

```
Gender                0
Married               0
Dependents            0
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor            0
Credit_History        0
Property_Area         0
Max_Loan_Amount       0
Loan_Status           0
dtype: int64
```

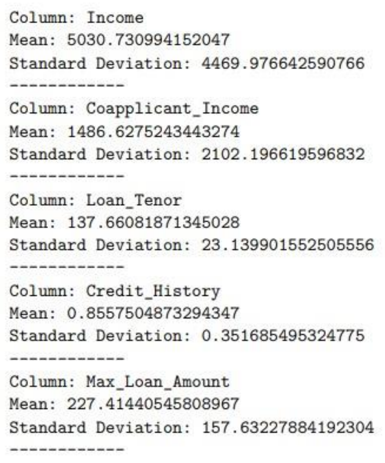
2- Checking the type of each feature (categorical or numerical):



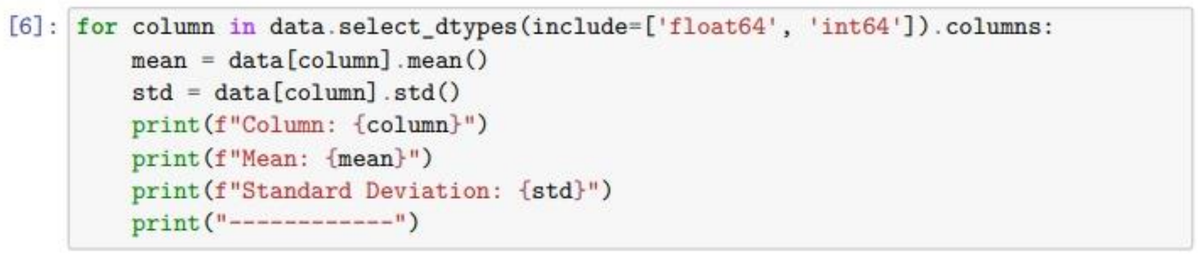
```python
nf = data.select_dtypes(include=['int', 'float'])
cf= data.select_dtypes(include=['object'])
print("numerical features: ")
print(nf.columns)
print("categorical features: ")
print(cf.columns)
```

```
numerical features:
Index(['Income', 'Coapplicant_Income', 'Loan_Tenor', 'Credit_History',
       'Max_Loan_Amount'],
      dtype='object')
categorical features:
Index(['Gender', 'Married', 'Dependents', 'Education', 'Property_Area',
       'Loan_Status'],
      dtype='object')
```

3- Checking whether numerical features have the same scale:



```
Column: Income
Mean: 5030.730994152047
Standard Deviation: 4469.976642590766
------------
Column: Coapplicant_Income
Mean: 1486.6275243443274
Standard Deviation: 2102.196619596832
------------
Column: Loan_Tenor
Mean: 137.66081871345028
Standard Deviation: 23.139901552505556
------------
Column: Credit_History
Mean: 0.8557504873294347
Standard Deviation: 0.351685495324775
------------
Column: Max_Loan_Amount
Mean: 227.41440545808967
Standard Deviation: 157.63227884192304
------------
```



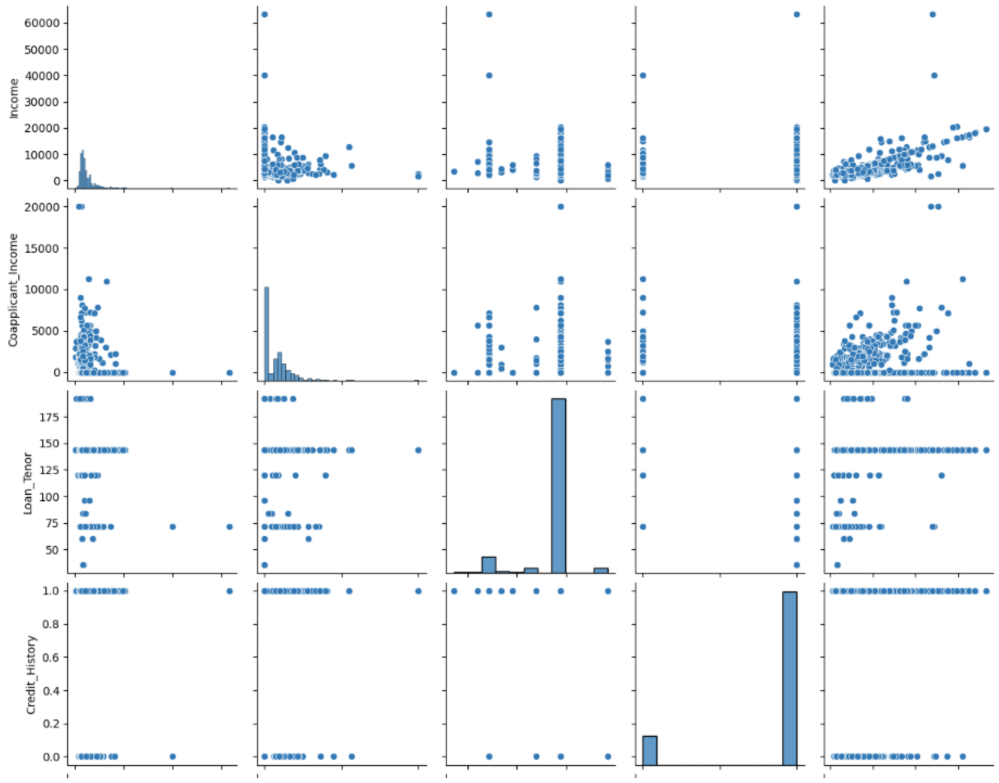
```python
[6]: for column in data.select_dtypes(include=['float64', 'int64']).columns:
         mean = data[column].mean()
         std = data[column].std()
         print(f"Column: {column}")
         print(f"Mean: {mean}")
         print(f"Standard Deviation: {std}")
         print("------------")
```

Some of the numerical features have high standard deviation thus not all at same scale.

4- Pairplot:

```
sns.pairplot(nf)
plt.show()
```

```
c:\Users\Essam\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Data Processing:

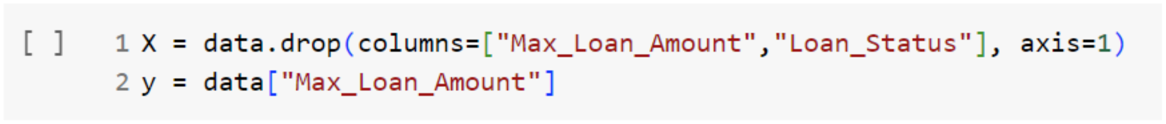
1- Removing records containing missing values:

```
[ ]   data = data.dropna()
```

```
data.isnull().sum()
```

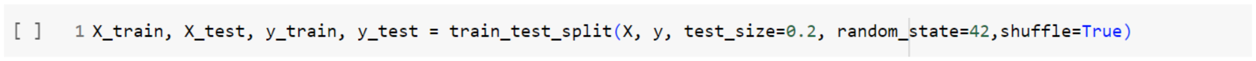
```
Gender                0
Married               0
Dependents            0
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor            0
Credit_History        0
Property_Area         0
Max_Loan_Amount       0
Loan_Status           0
dtype: int64
```

2- Separating features and targets:



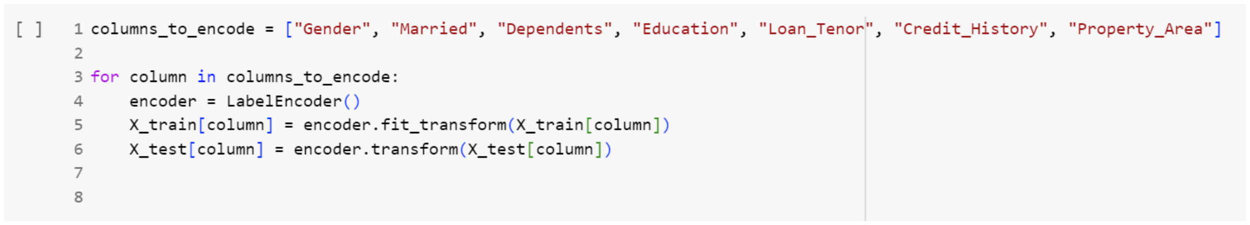
```
X = data.drop(columns=["Max_Loan_Amount","Loan_Status"], axis=1)
y = data["Max_Loan_Amount"]
```

3- Data is shuffled and split into training and testing sets:

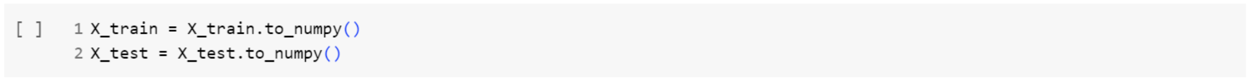


```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,shuffle=True)
```

4- Categorical attributes are encoded:

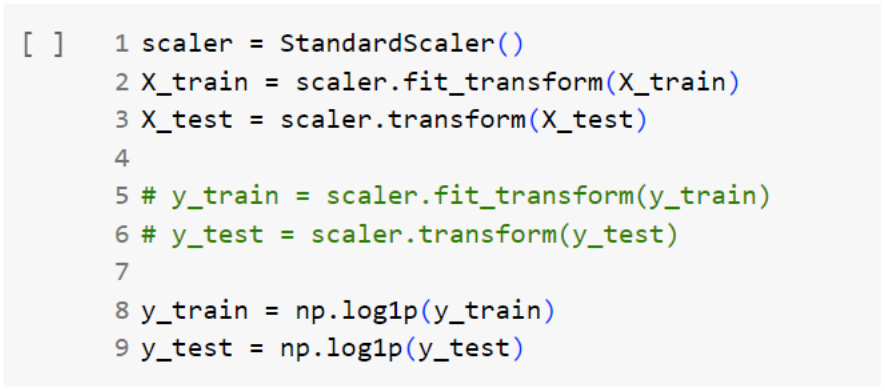


```
columns_to_encode = ["Gender", "Married", "Dependents", "Education", "Loan_Tenor", "Credit_History", "Property_Area"]

for column in columns_to_encode:
    encoder = LabelEncoder()
    X_train[column] = encoder.fit_transform(X_train[column])
    X_test[column] = encoder.transform(X_test[column])


```



```
X_train = X_train.to_numpy()
X_test = X_test.to_numpy()
```

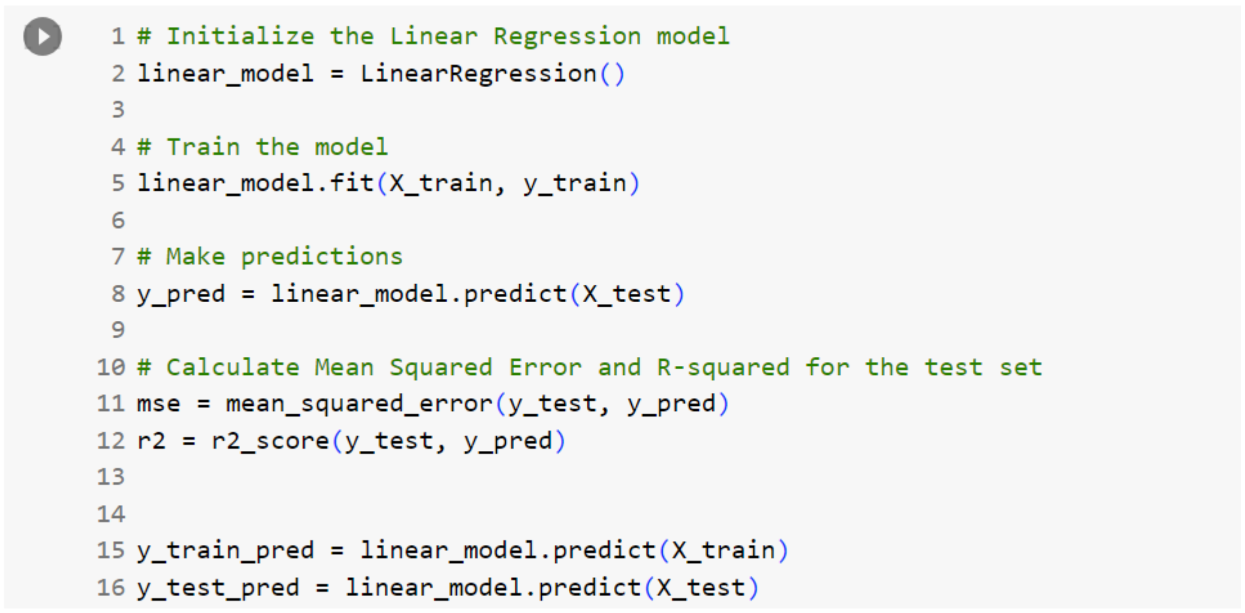
5- Numerical features are standardized:



```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# y_train = scaler.fit_transform(y_train)
# y_test = scaler.transform(y_test)

y_train = np.log1p(y_train)
y_test = np.log1p(y_test)
```

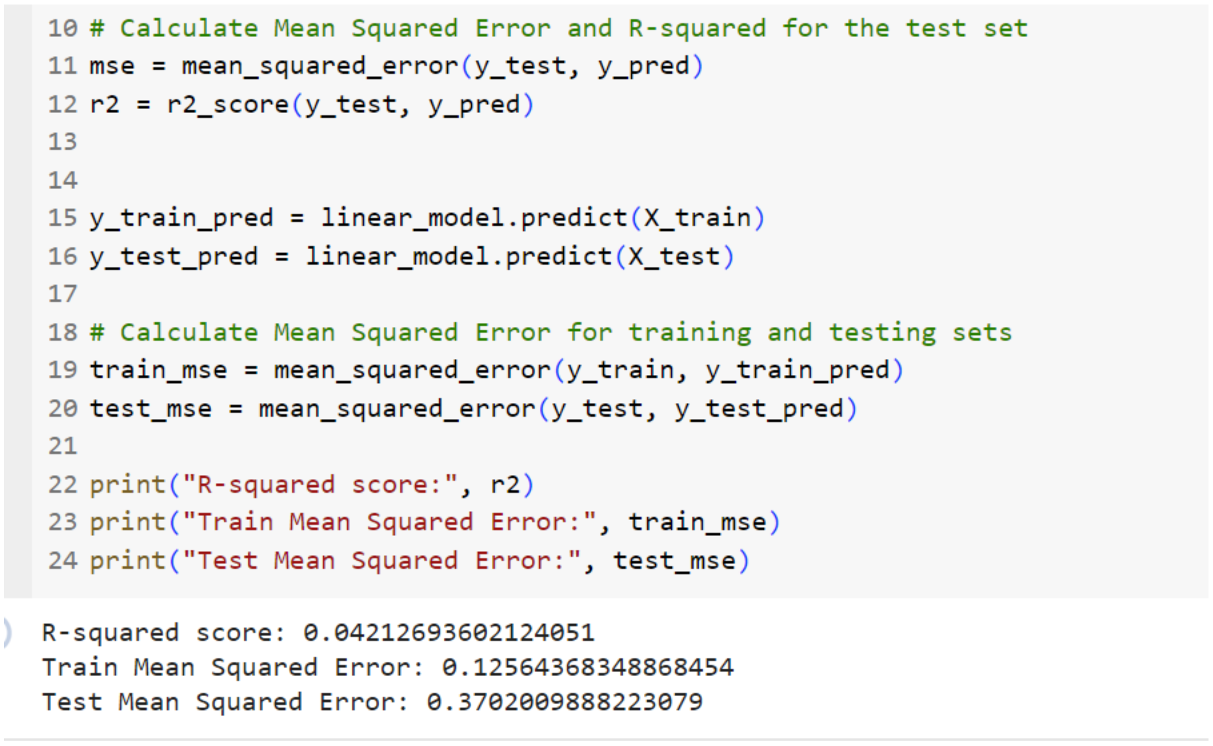
Linear Regression Model:

1- Predicting loan account:



```python
# Initialize the Linear Regression model
linear_model = LinearRegression()

# Train the model
linear_model.fit(X_train, y_train)

# Make predictions
y_pred = linear_model.predict(X_test)

# Calculate Mean Squared Error and R-squared for the test set
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)


y_train_pred = linear_model.predict(X_train)
y_test_pred = linear_model.predict(X_test)
```

2- Calculate Mean Squared Error and R-Squared Score:

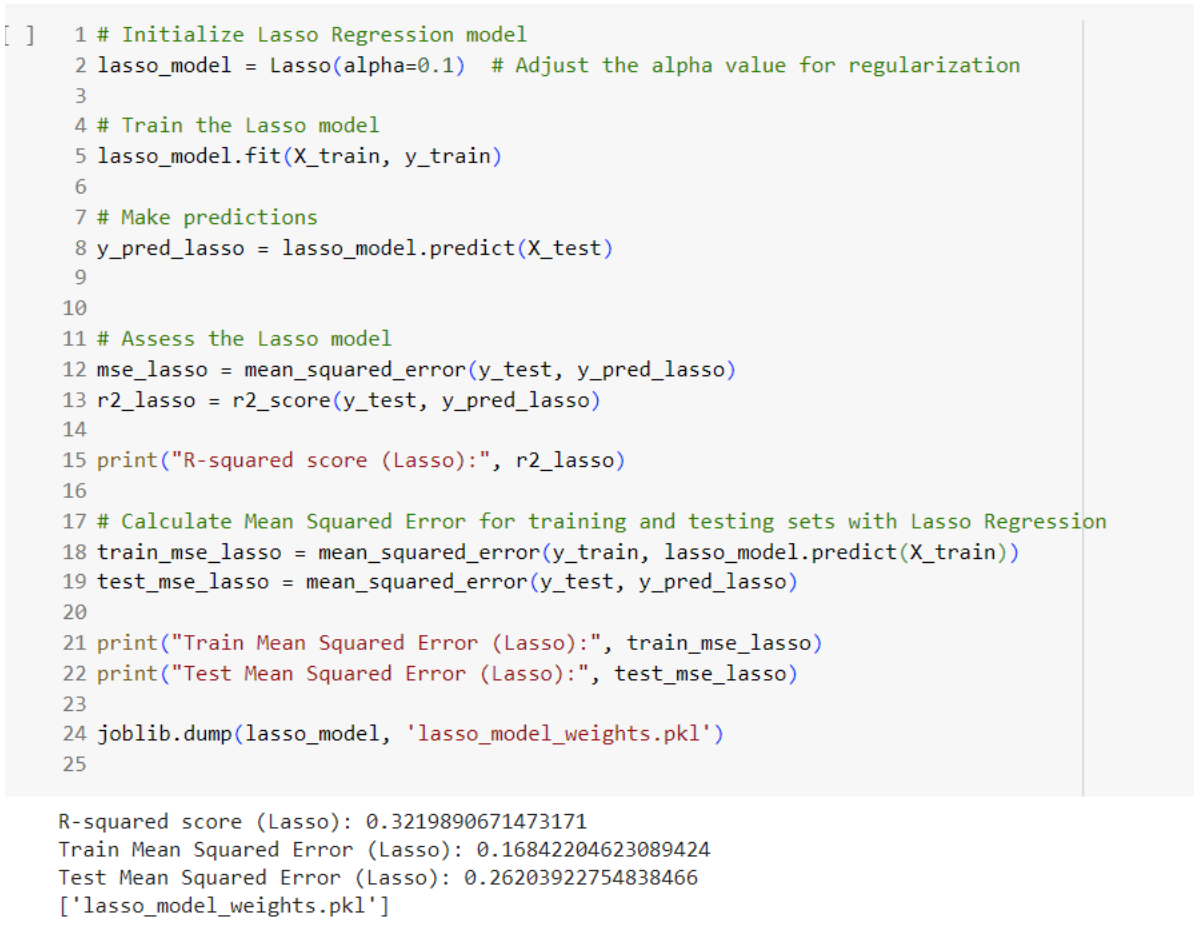


```python
# Calculate Mean Squared Error and R-squared for the test set
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)


y_train_pred = linear_model.predict(X_train)
y_test_pred = linear_model.predict(X_test)

# Calculate Mean Squared Error for training and testing sets
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)

print("R-squared score:", r2)
print("Train Mean Squared Error:", train_mse)
print("Test Mean Squared Error:", test_mse)
```

```
R-squared score: 0.04212693602124051
Train Mean Squared Error: 0.12564368348868454
Test Mean Squared Error: 0.3702009888223079
```

MSE on the training set is significantly lower than the MSE on the test set, it could be an indication that the model is overfitting.

3- Try lasso model to reduce the overfitting:



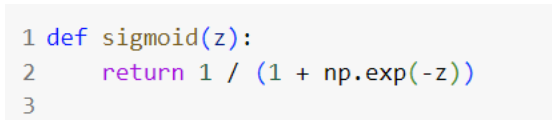
```
[ ]
# Initialize Lasso Regression model
lasso_model = Lasso(alpha=0.1)  # Adjust the alpha value for regularization

# Train the Lasso model
lasso_model.fit(X_train, y_train)

# Make predictions
y_pred_lasso = lasso_model.predict(X_test)


# Assess the Lasso model
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

print("R-squared score (Lasso):", r2_lasso)

# Calculate Mean Squared Error for training and testing sets with Lasso Regression
train_mse_lasso = mean_squared_error(y_train, lasso_model.predict(X_train))
test_mse_lasso = mean_squared_error(y_test, y_pred_lasso)

print("Train Mean Squared Error (Lasso):", train_mse_lasso)
print("Test Mean Squared Error (Lasso):", test_mse_lasso)

joblib.dump(lasso_model, 'lasso_model_weights.pkl')
```

```
R-squared score (Lasso): 0.3219890671473171
Train Mean Squared Error (Lasso): 0.16842204623089424
Test Mean Squared Error (Lasso): 0.26203922754838466
['lasso_model_weights.pkl']
```

Better results!!

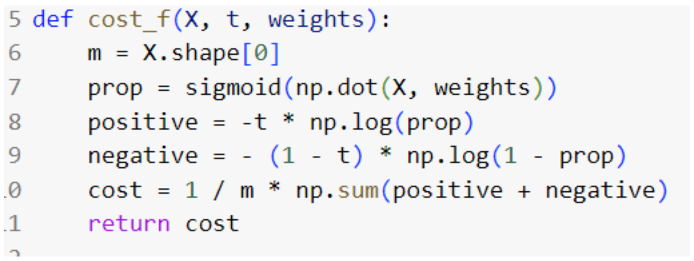
Logistic Regression Model:

1- Sigmoid Function:



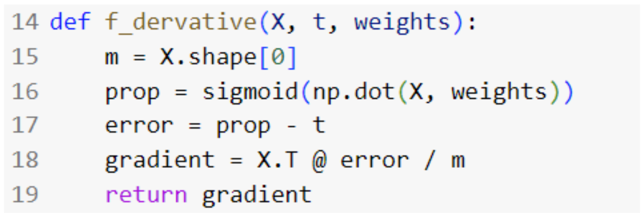
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

2- Cost Function:



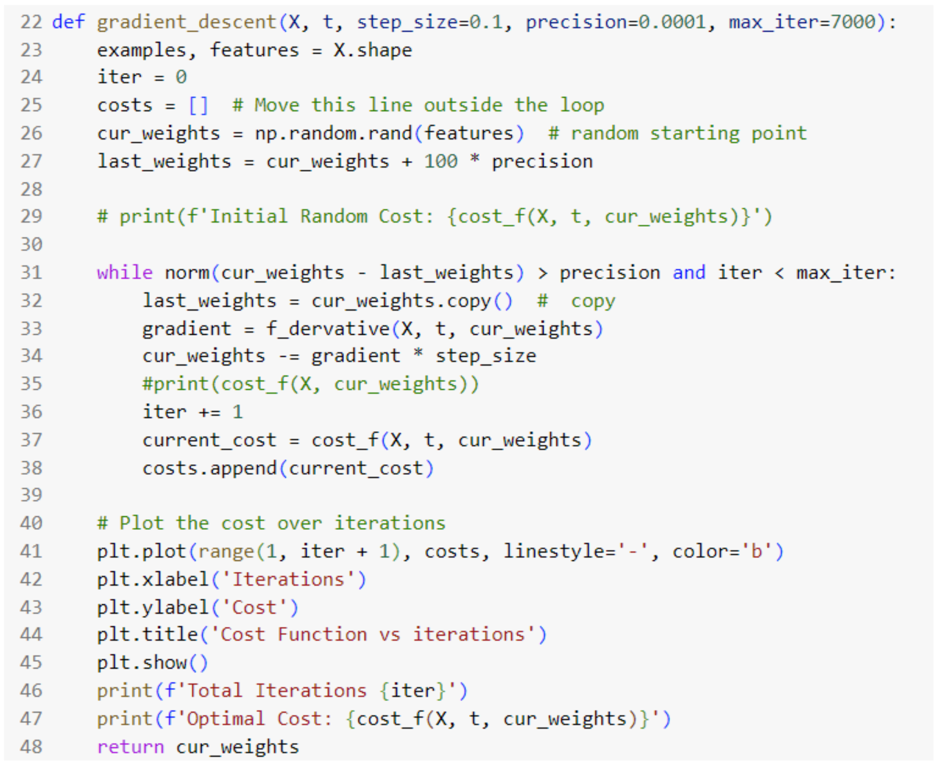
```python
def cost_f(X, t, weights):
    m = X.shape[0]
    prop = sigmoid(np.dot(X, weights))
    positive = -t * np.log(prop)
    negative = - (1 - t) * np.log(1 - prop)
    cost = 1 / m * np.sum(positive + negative)
    return cost
```

3- Derivative Function:



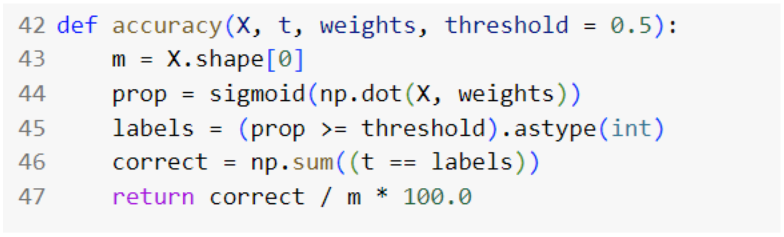
```python
def f_dervative(X, t, weights):
    m = X.shape[0]
    prop = sigmoid(np.dot(X, weights))
    error = prop - t
    gradient = X.T @ error / m
    return gradient
```

4- Gradient Descent Function:



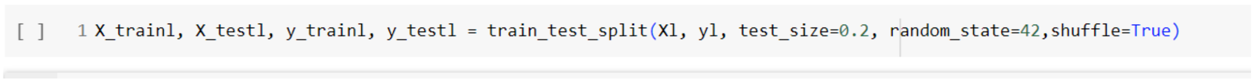
```python
def gradient_descent(X, t, step_size=0.1, precision=0.0001, max_iter=7000):
    examples, features = X.shape
    iter = 0
    costs = []  # Move this line outside the loop
    cur_weights = np.random.rand(features)  # random starting point
    last_weights = cur_weights + 100 * precision

    # print(f'Initial Random Cost: {cost_f(X, t, cur_weights)}')

    while norm(cur_weights - last_weights) > precision and iter < max_iter:
        last_weights = cur_weights.copy()  #  copy
        gradient = f_dervative(X, t, cur_weights)
        cur_weights -= gradient * step_size
        #print(cost_f(X, cur_weights))
        iter += 1
        current_cost = cost_f(X, t, cur_weights)
        costs.append(current_cost)

    # Plot the cost over iterations
    plt.plot(range(1, iter + 1), costs, linestyle='-', color='b')
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.title('Cost Function vs iterations')
    plt.show()
    print(f'Total Iterations {iter}')
    print(f'Optimal Cost: {cost_f(X, t, cur_weights)}')
    return cur_weights
```

6- Accuracy Function:



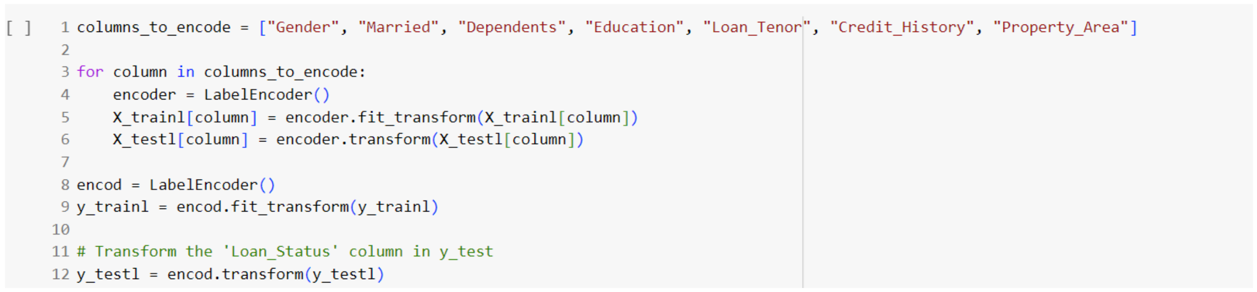
```python
def accuracy(X, t, weights, threshold = 0.5):
    m = X.shape[0]
    prop = sigmoid(np.dot(X, weights))
    labels = (prop >= threshold).astype(int)
    correct = np.sum((t == labels))
    return correct / m * 100.0
```

7- Separating features and targets:



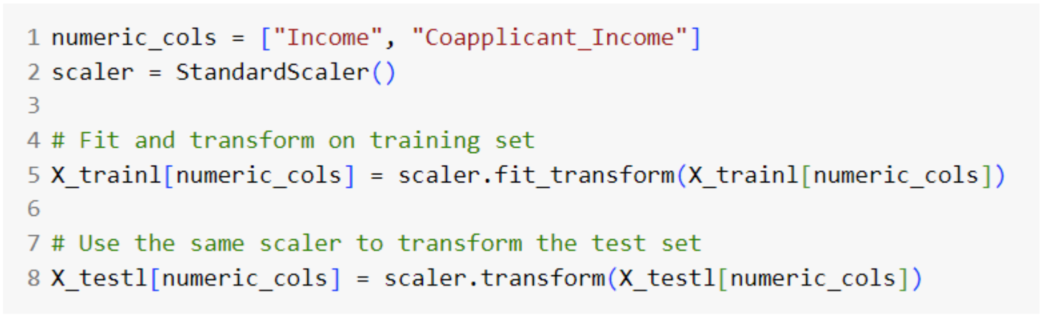
```python
[ ]  X_trainl, X_testl, y_trainl, y_testl = train_test_split(Xl, yl, test_size=0.2, random_state=42,shuffle=True)
```

8- Categorical attributes are encoded:



```python
[ ]  columns_to_encode = ["Gender", "Married", "Dependents", "Education", "Loan_Tenor", "Credit_History", "Property_Area"]

     for column in columns_to_encode:
         encoder = LabelEncoder()
         X_trainl[column] = encoder.fit_transform(X_trainl[column])
         X_testl[column] = encoder.transform(X_testl[column])

     encod = LabelEncoder()
     y_trainl = encod.fit_transform(y_trainl)

     # Transform the 'Loan_Status' column in y_test
     y_testl = encod.transform(y_testl)
```

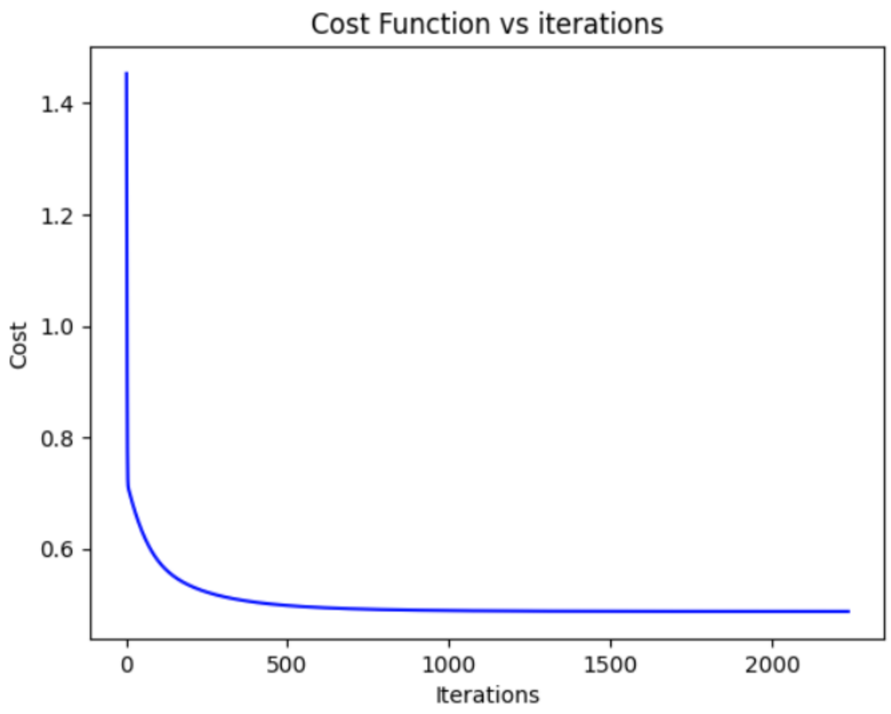
9- Numerical features are standardized:



```python
numeric_cols = ["Income", "Coapplicant_Income"]
scaler = StandardScaler()

# Fit and transform on training set
X_trainl[numeric_cols] = scaler.fit_transform(X_trainl[numeric_cols])

# Use the same scaler to transform the test set
X_testl[numeric_cols] = scaler.transform(X_testl[numeric_cols])
```

10- Training the model using gradient descent and evaluating its accuracy:

```
weights = gradient_descent(X_trainl,y_trainl)
print(f'Accuracy: {accuracy(X_testl,y_testl,weights)}')
np.save('Logistic_model_weights.npy', weights)
```



```
Total Iterations 2236
Optimal Cost: 0.48674880863286263
Accuracy: 80.58252427184466
```

Loading the Testing Set:

```python
data_new = pd.read_csv(r"C:\Users\Essam\Desktop\Assignment 1 ML\loan_new.csv")
# Removing spaces in column name
data_new = data_new.dropna()
# Write the cleaned data to a new CSV file
data_new.to_csv(r"C:\Users\Essam\Desktop\Assignment 1 ML\cleaned_data.csv", index=False)
```

```python
data_new = data_new.drop(["Loan_ID"],axis=1)
```

```python
for column in columns_to_encode:
    encoder = LabelEncoder()
    data_new[column] = encoder.fit_transform(data_new[column])
```

```python

numeric_cols = data_new.select_dtypes(include=['int64', 'float64']).columns

# Subsetting the DataFrame to only include these columns
numeric_data = data_new[numeric_cols]

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform only the selected columns
scaled_data = scaler.fit_transform(numeric_data)

# Replace the original values with the scaled values
data_new[numeric_cols] = scaled_data
data_new
```

	Gender	Married	Dependents	Education	Income	Coapplicant_Income	Loan_Tenor	Credit_History	Property_Area
0	1	1	0	0	0.208582	-0.656381	0.314250	0.46082	2
1	1	1	1	0	-0.349612	-0.013323	0.314250	0.46082	2
2	1	1	2	0	0.056577	0.115289	0.314250	0.46082	2
4	1	0	0	1	-0.307389	-0.656381	0.314250	0.46082	2
5	1	1	0	1	-0.541940	0.810649	0.314250	0.46082	2
...	...	...	...	...	...	...	...	...	...
361	1	1	1	0	-0.519984	0.272624	0.314250	0.46082	1
362	1	1	3	1	-0.152640	0.105429	0.314250	0.46082	2
363	1	1	0	0	-0.121183	-0.352429	0.314250	0.46082	2
365	1	1	0	0	0.056577	0.369511	0.314250	0.46082	0
366	1	0	0	0	0.943270	-0.656381	-3.088317	0.46082	0

314 rows × 9 columns

```python
loaded_weights = np.load('Logistic_model_weights.npy')
def predict_loan_status(X, weights):
    probabilities = sigmoid(np.dot(X, weights))
    return (probabilities >= 0.5).astype(int)

# Predict loan status for data_new
predicted_loan_status = predict_loan_status(data_new, loaded_weights)
predicted_loan_status
```

```
array([1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1])
```

```python
cleaned_data = pd.read_csv(r"C:\Users\Essam\Desktop\Assignment 1 ML\cleaned_data.csv")

# Adding the predicted values as a new column 'Loan_Status'
cleaned_data['Loan_Status'] = predicted_loan_status
cleaned_data['Loan_Status'] = cleaned_data['Loan_Status'].apply(lambda x: 'Y' if x == 1 else 'N')

# Save the updated data to a new CSV file
cleaned_data.to_csv(r"C:\Users\Essam\Desktop\Assignment 1 ML\cleaned_data.csv", index=False)
```

```python
loaded_model = joblib.load('lasso_model_weights.pkl')
predictions_lasso = loaded_model.predict(data_new)
predictions_lasso
```

```
c:\Users\Essam\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:457: UserWarning: X has feature names, but Lasso was fitted without feature names
  warnings.warn(
array([ 5.2356461 ,  5.13765852,  5.31898798,  5.03543999,  5.21500048,
        4.94942673,  5.08500001,  5.45879464,  5.15344357,  5.02028527,
        5.06593521,  5.76425902,  5.14930517,  5.23130448,  5.37306832,
        5.10354751,  6.14875079,  4.43949758,  5.04982882,  4.22927292,
        5.0754976 ,  5.55575039,  6.66471071,  5.67796407,  4.48797995,
        4.97006991,  5.34389808,  5.1678561 ,  5.20917087,  5.11337606,
        5.0251184 ,  4.97929681,  5.17354321,  5.24490276,  5.22071417,
        5.29286924,  5.06558559,  5.07046769,  5.50511847,  5.06738777,
        5.11046333,  5.44218475,  5.10020587,  5.29708174,  4.47487898,
        5.20066737,  4.99235144,  5.12767897,  4.6344282 ,  5.28443221,
        4.44528908,  5.10098515,  5.36489889,  5.17404422,  5.05778906,
        5.13488771,  5.28933231,  5.15209036,  5.04928403,  5.39505734,
        5.37207373,  5.13267594,  4.39176549,  5.25212484,  5.43948688,
        5.33218722,  5.21546714,  5.28774556,  5.36712529,  5.41832188,
        5.16795232, 10.70846511,  5.24757687,  5.45919866,  4.42366289,
        5.07087014,  5.21117899,  5.01823734,  5.19140341,  5.18141678,
        5.87443792,  5.32223612,  5.28764284,  5.32910153,  5.27988149,
        5.3721949 ,  5.3107605 ,  5.56323375,  5.13177484,  5.09406368,
        5.08663836,  4.27875103,  5.14875087,  5.13677181,  5.13570688,
        5.24433278,  5.15553089,  5.04699034,  5.27495618,  5.48431768,
        4.9062383 ,  5.12221236,  5.1039155 ,  5.02307047,  5.25888751,
        5.16484796,  5.47901427,  5.79104601,  5.14938708,  5.29954166,
        5.38145905,  4.31037114,  5.14052565,  5.04486049,  5.16087462,
        5.06361957,  4.52799649,  5.11588287,  5.13570688,  5.16333415,
        5.15430213,  4.83218261,  5.36537328,  4.52565562,  5.54379573,
        5.03543999,  5.48000568,  5.21222265,  5.25039123,  5.35935096,
        5.11017773,  5.17666558,  5.15544164,  5.16478755,  4.81030292,
        5.45062261,  5.16148355,  5.44404657,  5.41381642,  5.44354922,
        5.00988177,  5.22442341,  4.98891091,  4.95286726,  5.02002652,
        5.26667969,  5.14249864,  5.0590322 ,  5.09130765,  5.13431428,
```