1 Question 1

Create a linear regression model to predict the temperature of the day. You can choose either maximum, minimum or average temperature. Use one part of the full data set as your training data to find the parameters of the model, and use the other part as test data to find the accuracy of the model. Use at least two different type of methods to find the parameters of your model.

1.1 Linear Regression

For this question, we had to build a linear model. One of the methods chosen to determine the parameters of the linear model was using Linear Regression from Scikit learn library.

Prior to creating the model, the data had to be pre-processed. The weather data set has 10 attributes, and every column was checked for null values. To ensure the model is accurate, the null values were replaced with average values for that column.

```
#Preporocess the data

#Check which columns have missing data
print(weather.mean_temp.isna().sum())
print(weather.date.isna().sum())
print(weather.cloud_cover.isna().sum())
print(weather.sunshine.isna().sum())
print(weather.global_radiation.isna().sum())
print(weather.precipitation.isna().sum())
print(weather.pressure.isna().sum())
print(weather.snow_depth.isna().sum())
```

```
#Replaces all the missing data with the mean of that column.
columns = weather.columns
print(columns)

for col in columns:
    mean_val = weather[col].mean()
    print(mean_val)
    weather[col].fillna(mean_val, inplace = True)
    print(weather[col].isna().sum())
```

The target variable was chosen as the meantemp. The other two columns for max and min temperature were dropped. All the other columns were chosen as the predictor variables.

The data set was then split into training set and a testing set. These sets were obtained using the test train split package in Python. The training set was used to train the model. A test size of 0.2 was used which meant 80% of the total data was used for training the model and remaining 20% was used for testing purposes.

```
#Define x and y

#Predictor variables
x = weather.drop(['max_temp', 'mean_temp', 'min_temp'], axis = 1)
print(x)
print(len(x))

#Target Variable
y = weather['mean_temp']
print(y)
print(len(y))
```

```
#Splitting dataset into 80% trainign data set and 20% testing data set

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state =0)
```

The r2 score is a metric for model accuracy and should have a value between 0 and 1. The model has a r2 score of 0.50 for the test data and 0.49 for the training data. A value of 0.50 indicates that the model is not very accurate and can be improved. Different ratios for training and testing data were used. Using a test size of 0.2 produced a better model with higher r2 score. Initially, in the pre-processing stage, the null values were replaced with 0 instead of the mean values, which also produced a lower r2 score (about 48%).

The mean squared error was calculated for the model and a mse of 16.46 was obtained for training data and mse of 16.27 was obtained for the testing data.

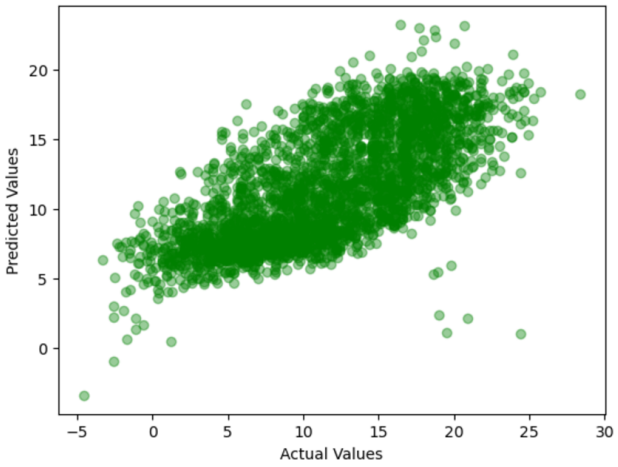
```
#Evaluating the model
from sklearn.metrics import r2_score, mean_squared_error

y_pred1 = lr.predict(X_train)
mse1 = mean_squared_error(y_train, y_pred1)

mse2 = mean_squared_error(y_test, y_predict)

print("The mean squared error for training data is " + str(mse1))
print("The mean squared error for testing data is " + str(mse2))

r2_test = r2_score(y_test, y_predict)
r2_train = r2_score(y_train, y_pred1)

print(r2_test)
print(r2_train)
```

```
The mean squared error for training data is 16.45684525828288
The mean squared error for testing data is 16.270843106473983
0.5018007248502101
0.49784080447277457
```

The figure below shows the scatter plot for actual and predicted values.

```
#Plot actual vs predicted values
plt.scatter(y_test, y_predict, color = 'green', marker = 'o', alpha = 0.4)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

plt.show
```

87]: <function matplotlib.pyplot.show(close=None, block=None)>



1.2 Gradient Descent

- Used independent variables: Sunshine, cloud cover and precipitation and dependent variable: max temperature.
- Split the data into train and test sets (80%/20% ratio)
- Reshaped data for gradient descent model
- Set learning rate and number of iterations and initialize w with 0.
- Perform gradient descent loop
- Review the slope and intercept parameters

```
In [38]: X2 = new_weather[['sunshine', 'cloud_cover', 'precipitation']]
         y2 = new_weather['max_temp']
```

```
In [18]: #split data
         X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
```

```
In [19]: X2_train = np.c_[np.ones((X2_train.shape[0], 1)), X2_train]
         X2_test = np.c_[np.ones((X2_test.shape[0], 1)), X2_test]
```

```
In [44]: eta = 0.01
         num_iterations = 1000
         w = np.zeros(X2_train.shape[1])

         #loop to perform gradient descent
         for i in range(num_iterations):
             # Calculate the predicted values
             y2_pred = np.dot(X2_train, w)

             # Calculate cost function
             gradient = (1/len(y2_train)) * np.dot(X2_train.T, (y2_pred - y2_train))

             # Update the parameters using the gradient
             w = w - eta * gradient
```

```
In [45]: #Parameter of model
         print("Intercept ", w[0], "Slope: ", w[1])

Intercept  1.4300508289691358 Slope:  1.3332806053573605
```

Evaluation of Gradient Model

Predict with test data to perform evaluation of MSE and R2 which produced results of MSE = 30.6699 and R2 = 0.2868.

MSE indicates that the squared difference between predicted and actual max temperature values is approximately 30.6699.

R2 with 28.7% is relatively low, indicating that there is a significant amount of variance unexplained. Meaning that more independent variables may be required in the model.

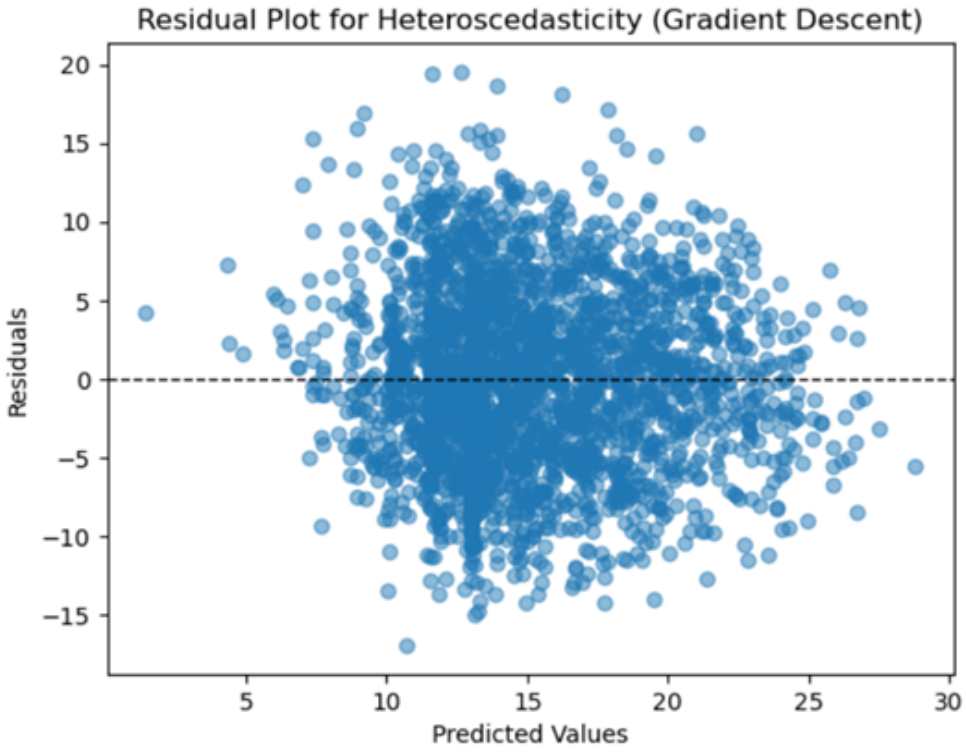
```
In [46]: # pred with test data
         y2_pred = np.dot(X2_test, w)
         y2_pred

Out[46]: array([13.23491388,  9.50897976, 19.81030031, ..., 10.3649923 ,
                17.69607421, 13.01073717])
```

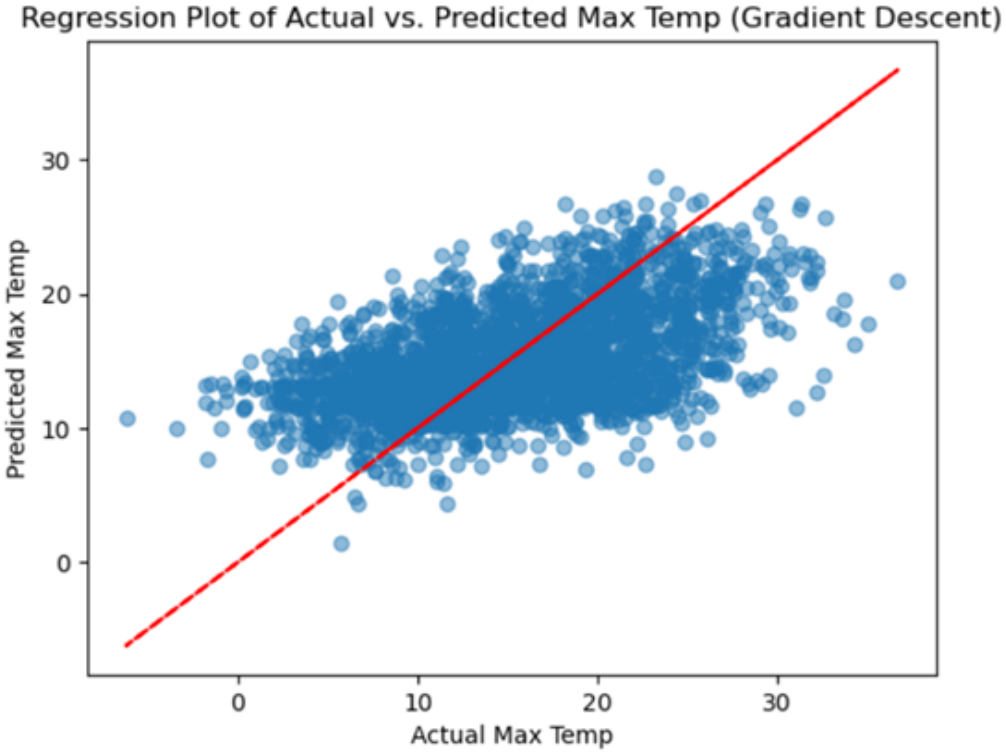
```
In [47]: #evaluate performance
         mse = mean_squared_error(y2_test, y2_pred)
         r2_gd = r2_score(y2_test, y2_pred)

         print("Mean Squared Error:", mse)
         print("R-squared:", r2_gd)

Mean Squared Error: 30.669866752882513
R-squared: 0.28680277045902014
```

Check for Heteroscedasticity

The data points appear to be scattered evenly above and below the line indicating a linear relationship, however there appears to be several outlier data points which indicate heteroscedasticity.

Residual plot of actual vs predicted max temp





Comparison of both models

The results obtained with both models are listed below:

Linear regression model: MSE = 16.27 and R2 = 50.45

Gradient descent: MSE = 30.6699 and R2 = 28.68

The difference in the two models may have been because the gradient descent model uses fewer predictor variables compared to the other model. Therefore, this may not be an appropriate comparison.

2 Question 2

Create a logistic regression model to predict the temperature of each day. Again you can pick one of the three temperatures provided as the target variable. As with the previous question, split your data into training and test sets. You should create models to at least perform one binary and one multi-class classification task on the data set. Hint: How would you transform the values in temperature into something you can use for classification?

2.1 Binary Classification

For the preprocessing of the dataset, we have first looked at if there are any null values. Since there are a few, and we have 15341 rows of data, we have decided to drop the rows with the null values using `weather.dropna()`. After this we were left with 13843 rows which is enough to train and test our model.

	date	cloud_cover	sunshine	global_radiation	max_temp	mean_temp	min_temp	precipitation	pressure	snow_depth
count	1.384300e+04	13843.00000	13843.000000	13843.000000	13843.000000	13843.000000	13843.000000	13843.000000	13843.000000	13843.000000
mean	1.998330e+07	5.32818	4.262609	114.529148	14.951911	11.085408	7.212302	1.667493	101538.493101	0.037853
std	1.195655e+05	2.03417	3.987488	87.758136	6.510220	5.700936	5.319409	3.733947	1066.084413	0.545712
min	1.979010e+07	0.00000	0.000000	12.000000	-6.200000	-7.600000	-11.800000	0.000000	95960.000000	0.000000
25%	1.988071e+07	4.00000	0.400000	39.000000	10.200000	6.800000	3.200000	0.000000	100900.000000	0.000000
50%	1.998011e+07	6.00000	3.400000	89.000000	14.400000	10.800000	7.300000	0.000000	101630.000000	0.000000
75%	2.009010e+07	7.00000	7.100000	180.000000	19.700000	15.550000	11.400000	1.600000	102260.000000	0.000000
max	2.019123e+07	9.00000	15.700000	352.000000	37.900000	29.000000	22.300000	61.800000	104430.000000	22.000000

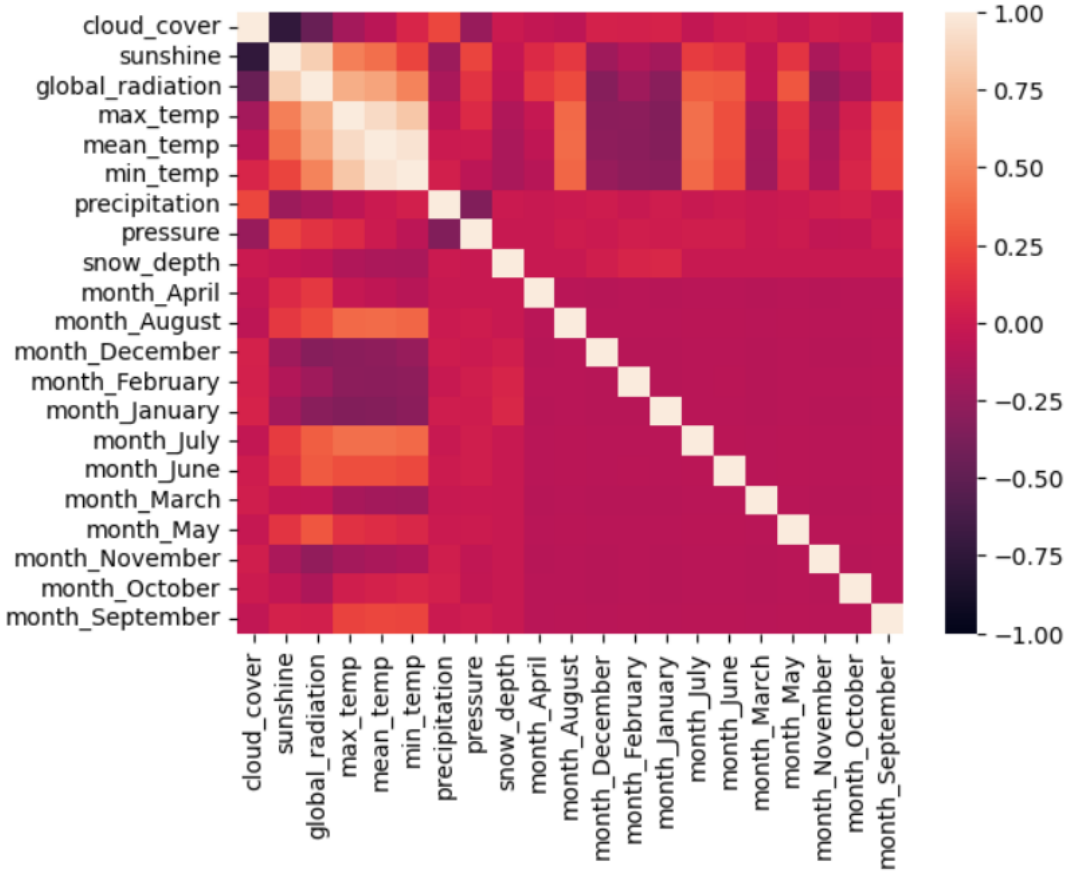
Afterwards we have looked at the summary of the dataset to understand how to classify the target variable. In this case we have used **meantemp** as our target variable. Since 50% of the mean temperature shows 10.8, thus we have used it for a classification of "High" and "Low" denoted as 0 and 1.

Since the dates are stored as integers in the dataset, we have converted the dates into dates and separated the months in a different column to use it as a predictor.

```
new_weather['date'] = pd.to_datetime(new_weather['date'], format = '%Y%m%d')
new_weather['month'] = new_weather['date'].dt.strftime('%B')
new_weather = pd.get_dummies(new_weather)
new_weather.head()
```

Now to understand the correlation between the variables, we have illustrated a `sns.heatmap` for correlation as shown below.

We can observe that **sunshine** and **cloudcover** are dependent on each other. So we can use one of them in our model.

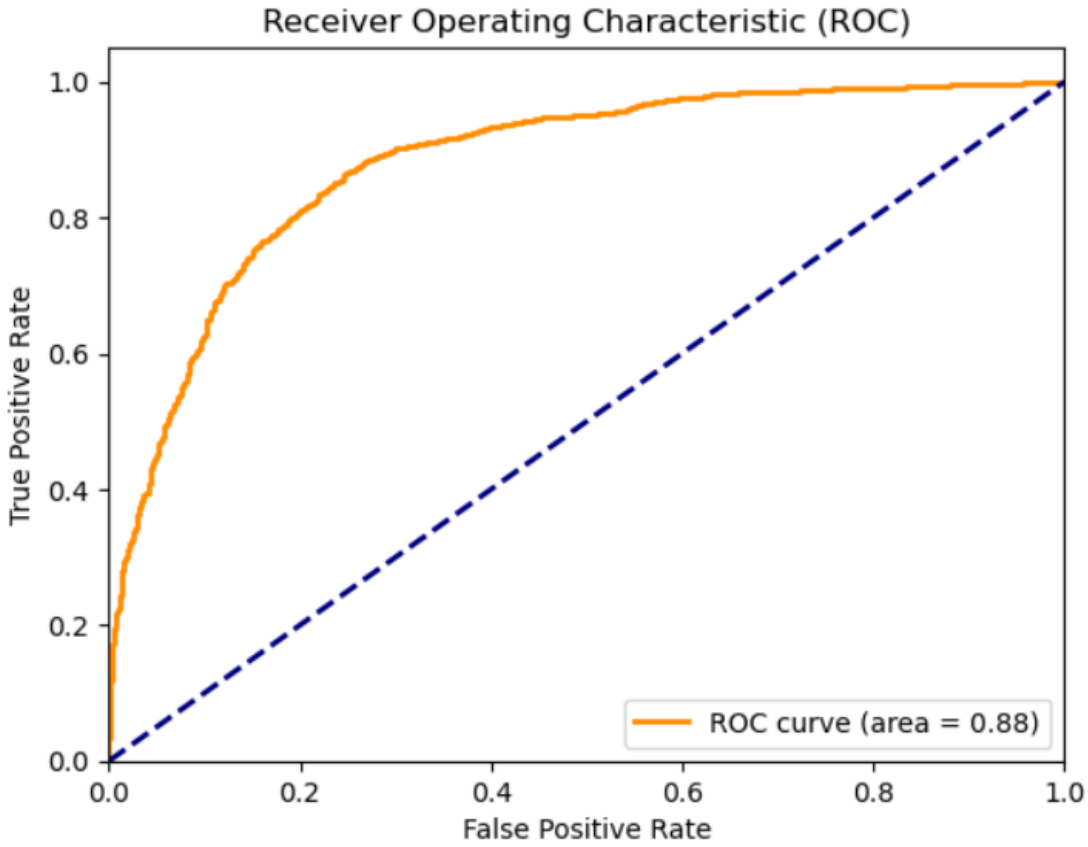



```
lr = LogisticRegression(solver = 'liblinear')
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
```

```
0.8012381883349625
```

```
binary_model = LogisticRegression(max_iter=1000)
binary_model.fit(X_train, y_train)
binary_predictions = binary_model.predict(X_test)
binary_accuracy = accuracy_score(y_test, binary_predictions)
binary_classification_report = classification_report(y_test, binary_predictions)
print("Binary Classification Report:\n", binary_classification_report)
print("Binary Accuracy:", binary_accuracy)
```

```
Binary Classification Report:
              precision    recall  f1-score   support

         0.0       0.72      0.79      0.76      1454
         1.0       0.80      0.73      0.76      1615

    accuracy                           0.76      3069
   macro avg       0.76      0.76      0.76      3069
weighted avg       0.76      0.76      0.76      3069

Binary Accuracy: 0.7588791137178234
```



At first we have applied logistic regression model with liblinear solver, where we achieved an accuracy of 80.12%.

After fitting the binary model we got a precision ratio of 0.72 for 0 and 0.80 for 1. We also got an overall accuracy of 75.89% which represents the proportion of correctly classified samples in the test set.

Finally AUC is calculated (0.88) which indicates that the model is strong to distinguish between the classes.

2.2 Multiclass Classification

For the Multiclass classification, we used pandas, numpy, seaborn, and matplotlib. We have also used sklearn and imported train test split, Logistic Regression, metrics, accuracy score, classification report, confusion matrix, precision score, recall score, and f1 score. We dropped the missing values and converted the data type of date from integer to date. The image below shows that we chose the mean temperature as our target variable just as we did in question 1 for Linear Regression and question 2 for Binary Logistic Regression. The maximum temperature for the mean temperature is 29 and the minimum temperature is negative 7.6. From there, we divided the mean temperature into three parts and they are low, moderate, and high as seen below.

```
new_weather.loc[new_weather['mean_temp'] >= 16.8, 'mean_temp_cat_multi'] = 'High'
new_weather.loc[(new_weather['mean_temp'] >= 4.6) & (new_weather['mean_temp'] < 16.8),
                'mean_temp_cat_multi'] = 'Moderate'
new_weather.loc[new_weather['mean_temp'] < 4.6, 'mean_temp_cat_multi'] = 'Low'
```

```
new_weather['mean_temp_cat_multi'].unique()
```

```
array(['Low', 'Moderate', 'High'], dtype=object)
```

```
new_weather.head().T
```

	0	1	2	3	4
date	1979-01-01 00:00:00	1979-01-02 00:00:00	1979-01-03 00:00:00	1979-01-04 00:00:00	1979-01-05 00:00:00
cloud_cover	2.0	6.0	5.0	8.0	6.0
sunshine	7.0	1.7	0.0	0.0	2.0
global_radiation	52.0	27.0	13.0	13.0	29.0
max_temp	2.3	1.6	1.3	-0.3	5.6
mean_temp	-4.1	-2.6	-2.8	-2.6	-0.8
min_temp	-7.5	-7.5	-7.2	-6.5	-1.4
precipitation	0.4	0.0	0.0	0.0	0.0
pressure	101900.0	102530.0	102050.0	100840.0	102250.0
snow_depth	9.0	8.0	4.0	2.0	1.0
month_April	0	0	0	0	0
month_August	0	0	0	0	0
month_December	0	0	0	0	0
month_February	0	0	0	0	0
month_January	1	1	1	1	1
month_July	0	0	0	0	0
month_June	0	0	0	0	0
month_March	0	0	0	0	0
month_May	0	0	0	0	0
month_November	0	0	0	0	0
month_October	0	0	0	0	0
month_September	0	0	0	0	0
mean_temp_cat_multi	Low	Low	Low	Low	Low

We used cloud cover, sunshine, global radiation, snow depth, and all the months for the X variable. We used the stepwise backward selection in choosing the best features to use that will improve our model accuracy. We started with all the features and removed one by one those features that were not useful or important.

```
#generate 2 classes
X = new_weather[['cloud_cover','sunshine','global_radiation', 'snow_depth','month_January',
                 'month_February', 'month_March', 'month_April', 'month_May', 'month_June',
                 'month_July', 'month_August', 'month_September', 'month_October', 'month_November',
                 'month_December']]
y = new_weather['mean_temp_cat_multi']
```

The figure below shows that eighty percent of the data is used for training and 20 percent for testing the model. We used the lbfgs for the solver to get the local minimum and multinomial for the multiclass as we have three classes. Sample predictions can also be seen below.

```
#test and train for data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#fit the model
lr=LogisticRegression(solver="lbfgs", multi_class='multinomial', max_iter=10000).fit(X_train, y_train)
```

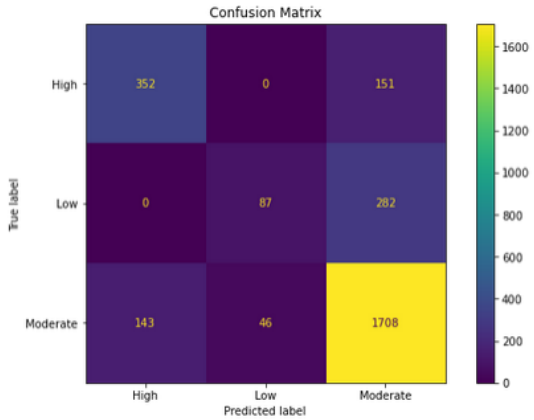
```
#prediction
predictions = lr.predict(X_test)
print('Predicted labels: ', predictions[:15])
print('Actual labels: ', y_test[:15])
```

```
Predicted labels:  ['High' 'Moderate' 'Moderate' 'Moderate' 'Moderate' 'Moderate' 'High'
 'High' 'Moderate' 'Moderate' 'Moderate' 'High' 'Moderate' 'Moderate'
 'Moderate']
Actual labels:  14093       High
12710    Moderate
4143         High
4801     Moderate
1611         High
4019     Moderate
1696         High
4989     Moderate
4154     Moderate
8037     Moderate
3197     Moderate
7516         High
1229     Moderate
5560     Moderate
13409    Moderate
Name: mean_temp_cat_multi, dtype: object
```

The confusion matrix below shows that our model is doing good. The diagonal elements showing the values 352, 87, and 1708 mean that our model correctly labeled the predicted class. The off-diagonal elements display the wrong predictions that our model classified. In the first row, our model correctly classified 352 mean temperatures as high out of all the 503 high mean temperatures but misclassified 151 as moderate. In the second row, our model correctly classified 87 mean temperatures as low out of all the 369 low mean temperatures but labeled 282 as moderate. In the third row, our model correctly classified 1708 mean temperatures as moderate out of all the 1897 moderate mean temperatures but misclassified 143 as high and 46 as low. All in all, the total number of correct predictions is 2147 and the total number of incorrect predictions is 622.

```
#confusion matrix
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title('Confusion Matrix')

display = metrics.plot_confusion_matrix(lr, X_test, y_test, ax = ax)
display.confusion_matrix
```

```
array([[ 352,    0,  151],
       [   0,   87,  282],
       [ 143,   46, 1708]], dtype=int64)
```




```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

        High       0.71      0.70      0.71       503
         Low       0.65      0.24      0.35       369
    Moderate       0.80      0.90      0.85      1897

    accuracy                           0.78      2769
   macro avg       0.72      0.61      0.63      2769
weighted avg       0.76      0.78      0.75      2769
```

Our model is 77.54% accurate in performing correct predictions across the entire dataset. The overall precision rate is 72.10%, overall recall rate is 61.20%, and overall F1 score is 63.27% which gives us a lower value. It means that our model is performing well in predicting the mean temperatures of each day but may have issues in determining classes correctly.

```
#data scoring
print("Overall Accuracy:",accuracy_score(y_test, predictions))
print("Overall Precision:",precision_score(y_test, predictions, average='macro'))
print("Overall Recall:",recall_score(y_test, predictions, average='macro'))
print("Overall F1 Score",f1_score(y_test, predictions, average="macro"))
```

```
Overall Accuracy: 0.775370169736367
Overall Precision: 0.7210015021465647
Overall Recall: 0.6119808514188522
Overall F1 Score 0.6326625718840647
```

Out of these two models, the binary classification model has a higher accuracy score of 80.12% so it is better to use this model.

3 Question 3

Create a Ridge regression model to predict the temperature (as in Question 1).

```
y = weather['mean_temp']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, rando

#Creating a ridge model
from sklearn.metrics import r2_score, mean_squared_error

ridge = Ridge(alpha=1)

ridge.fit(X_scaled, y_train)
new_y = ridge.predict(X_test_scaled)
y_pred = ridge.predict(X_scaled)

r2_test = r2_score(y_test, new_y)
r2_train= r2_score(y_train, y_pred)

mse1 = mean_squared_error(y_test, new_y)
mse2 = mean_squared_error(y_train, y_pred)
print("The mean squared error for the ridge model using test data is " + str(mse
print("The mean squared error for the ridge model using train data is " + str(ms

print(r2_test)
print(r2_train)

# Do the ridge regression over a range of lambda values
ridge_reg = Ridge(alpha=1, random_state=47)
ridge_reg.fit(X_scaled, y_train)
ridge_coef = []
```

```
The mean squared error for the ridge model using test data is 16.23961115045296
The mean squared error for the ridge model using train data is 16.4568470226989
7
0.5027570205842029
0.497840750633916
```

Ridge Regression is used to introduce a small amount of bias to our model to ensure the variance of the model is lowered. The data was preprocessed prior to creating the Ridge Model. The Ridge model was created as an instance of the Ridge object.

As seen in the image below, we imported r2 score and mean squared error from sklearn. We also used the StandardScaler to standardize the features for more stable performance because it will be less influenced by the range of variables by transforming it so the distribution will have a mean value of zero and standard deviation of one. We used the training and test data that we used in question 1.

The Ridge model is slightly better in comparison to the model in question 1. The mean squared error for test data is slightly lower compared to the mean squared error

for training data. This model is better as it has a slightly higher r2score of 0.5028 and lower MSE.

The different values of alpha were also used for determining the most appropriate ridge model, a loop was used to iterate through the different values of alpha to check which value determine the lowest MSE.