Import Packages

In [3]:
```
# packages used in this tutorial
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

import numpy as np
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classificat
```

Load CSVs

In [4]:
```
# Load the CSV files into dataframes
dataframes = {}
keys = [str(i).zfill(2) for i in range(1, 13)] # strings '01' to '12'
for key in keys:
    df = pd.read_csv(f'CSVafterClean/{key}.csv')
    dataframes[key] = df
```

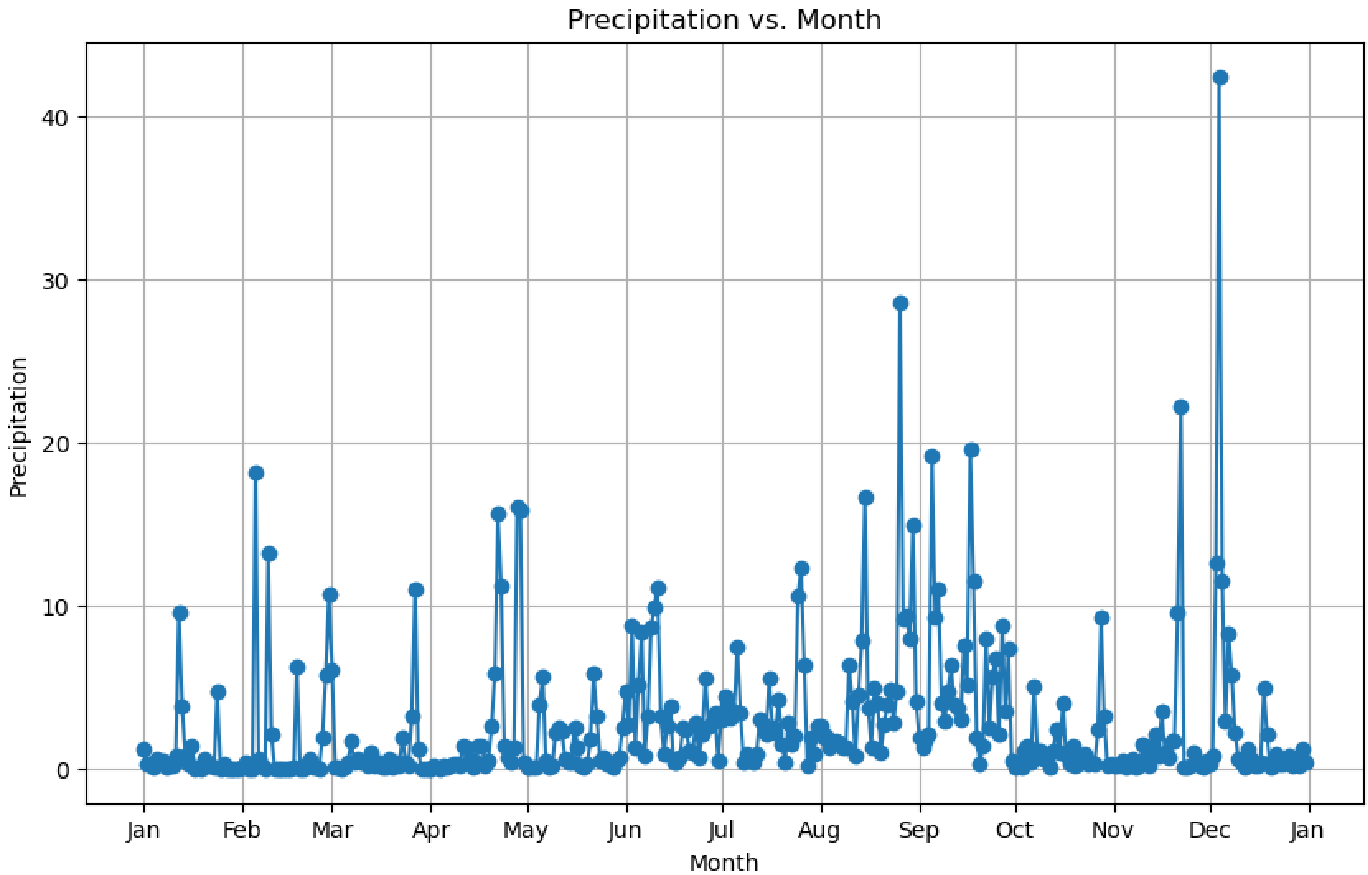
Data Exploration - Target Variable

In [5]:
```
# Create an empty list to store the concatenated data
concatenated_data = []
for i in dataframes:
    df = dataframes[i]
    # Add a 'Month' column to each dataframe
    snip = df.loc[:, df.columns.isin(['time', 'prcp_total'])]
    concatenated_data.append(snip)

# Concatenate dataframes vertically
combined_df = pd.concat(concatenated_data, ignore_index=True) #size is [101835 rows x 2

# Convert 'time' column to datetime
combined_df['time'] = pd.to_datetime(combined_df['time'])

# Group by date and calculate the average precipitation for each day
aggregated_df = combined_df.groupby(combined_df['time'].dt.date)['prcp_total'].mean().re

print(aggregated_df)
```

```
           time  prcp_total
0    2015-01-01    1.136654
1    2015-01-02    0.258093
2    2015-01-03    0.274102
3    2015-01-04    0.086851
4    2015-01-05    0.565326
..          ...         ...
360  2015-12-27    0.192383
361  2015-12-28    0.392772
362  2015-12-29    0.158494
363  2015-12-30    1.181893
```

```
364  2015-12-31    0.337404

[365 rows x 2 columns]
```

In [6]:
```
# Group by date and calculate the average precipitation for each day
aggregated_df = combined_df.groupby(combined_df['time'].dt.date)['prcp_total'].mean().re

# Create a line graph
plt.figure(figsize=(10, 6))
plt.plot(aggregated_df['time'], aggregated_df['prcp_total'], marker='o', linestyle='-')
plt.xlabel('Month')
plt.ylabel('Precipitation')
plt.title('Precipitation vs. Month')
plt.grid(True)

# Format the x-axis ticks to show one label per month
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))  # Set tick interval

plt.show()
```



Data Preprocessing

In [7]:
```
# Assuming 'dataframes' is your dictionary of dataframes
# Extract the 'prcp_total' column from each dataframe
X = []  # Input features
y = []  # Target variable

# List of columns to exclude
target = 'next_day_prcp_total'
exclude_columns = ['time', target]

for key, df in dataframes.items():
     # Select all columns except 'time', and 'prcp_total' temporal aspects
    features = df.loc[:, ~df.columns.isin(exclude_columns)].values
```

```
    X.append(features) #a list of arrays, where each array represents the features for o
    y.append(df[target].values) #a list of 1D NumPy arrays, where each array represents

# Combine data from all dataframes
X = np.vstack(X) #vertically stacks (concatenates) these arrays on top of each other, ef
#where each row represents a data point (sample), and each column represents a feature.
y = np.concatenate(y) # y becomes a 1D array of target data point values of the one targ
```

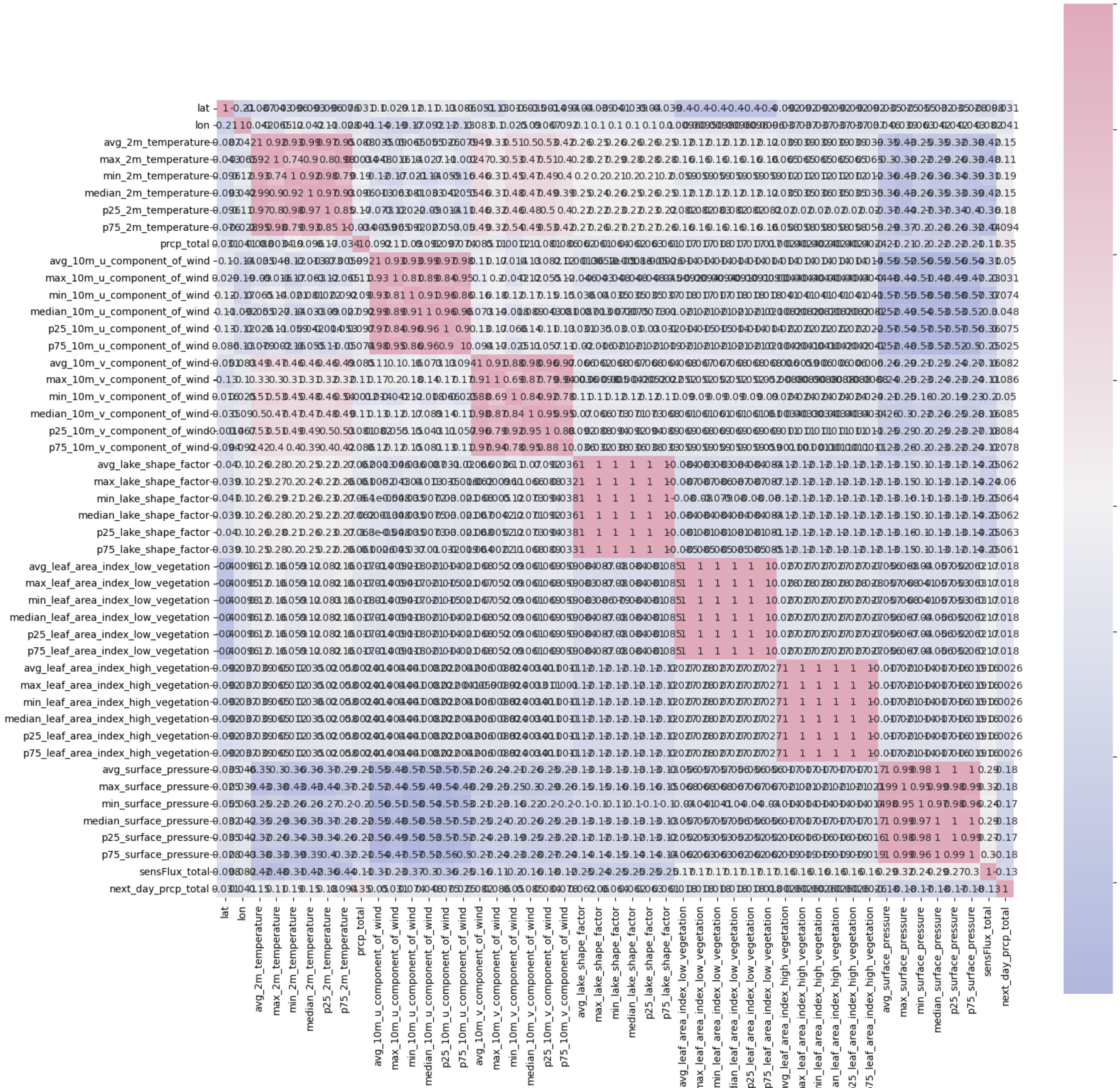
Data Visualization - Feature Importance

In [8]:

```
# Extract column names not listed in the exclusion list
col_names = [col for col in dataframes['01'].columns if col not in exclude_columns]
# Set up a standard scaler for the features
features = X
features = pd.DataFrame(features, columns=col_names)
#full data
features['next_day_prcp_total'] = y
data = features
# Correlation amongst numeric attributes
corrmat = data.corr()
cmap = sns.diverging_palette(260,-10,s=50, l=75, n=6, as_cmap=True)
plt.subplots(figsize=(18,18))
sns.heatmap(corrmat,cmap= cmap,annot=True, square=True)
```

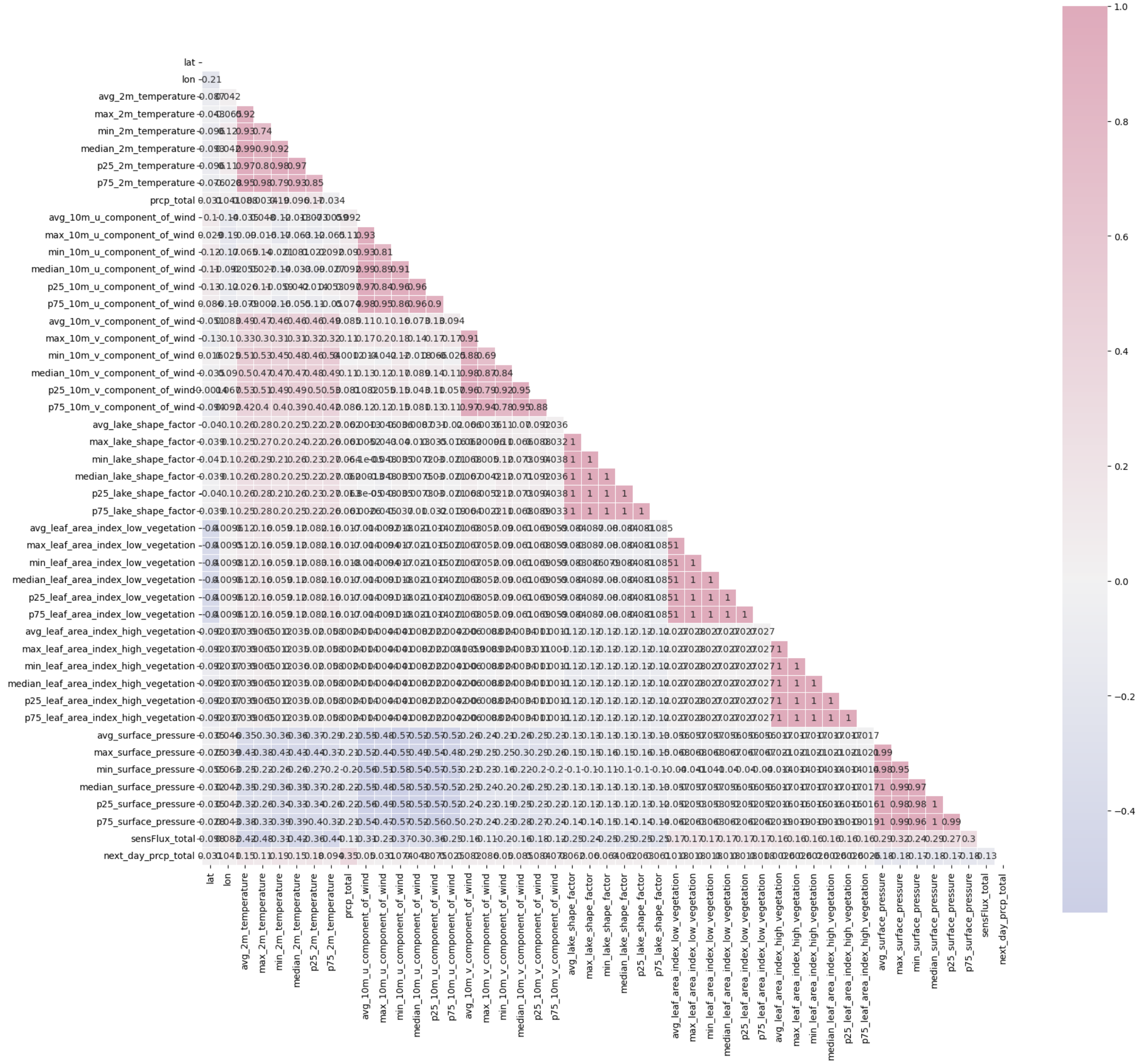
Out[8]:

```
<AxesSubplot:>
```



In [9]:
```
# Correlation Heatmap
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
corr = data.corr()
mask = np.triu(np.ones_like(corr)) #this and the next line make the correlation plot hal
f, ax = plt.subplots(figsize=(20, 20))
cmap = sns.diverging_palette(260,-10,s=50, l=75, n=6, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=None, center=0,square=True, annot=True, lin
```

Out[9]:
```
<AxesSubplot:>
```



Standardize Features

In [10]:

```
# Extract column names not listed in the exclusion list
col_names = [col for col in dataframes['01'].columns if col not in exclude_columns]

# Set up a standard scaler for the features
s_scaler = StandardScaler()
features = s_scaler.fit_transform(X)
features = pd.DataFrame(features, columns=col_names)

features.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%
lat	101835.0	-8.913200e-15	1.000005	-2.180517	-0.764305	0.044959	0.854223
lon	101835.0	2.761931e-15	1.000005	-2.114141	-0.716403	0.082304	0.881011
avg_2m_temperature	101835.0	-4.979067e-16	1.000005	-5.034658	-0.445812	0.234201	0.718118

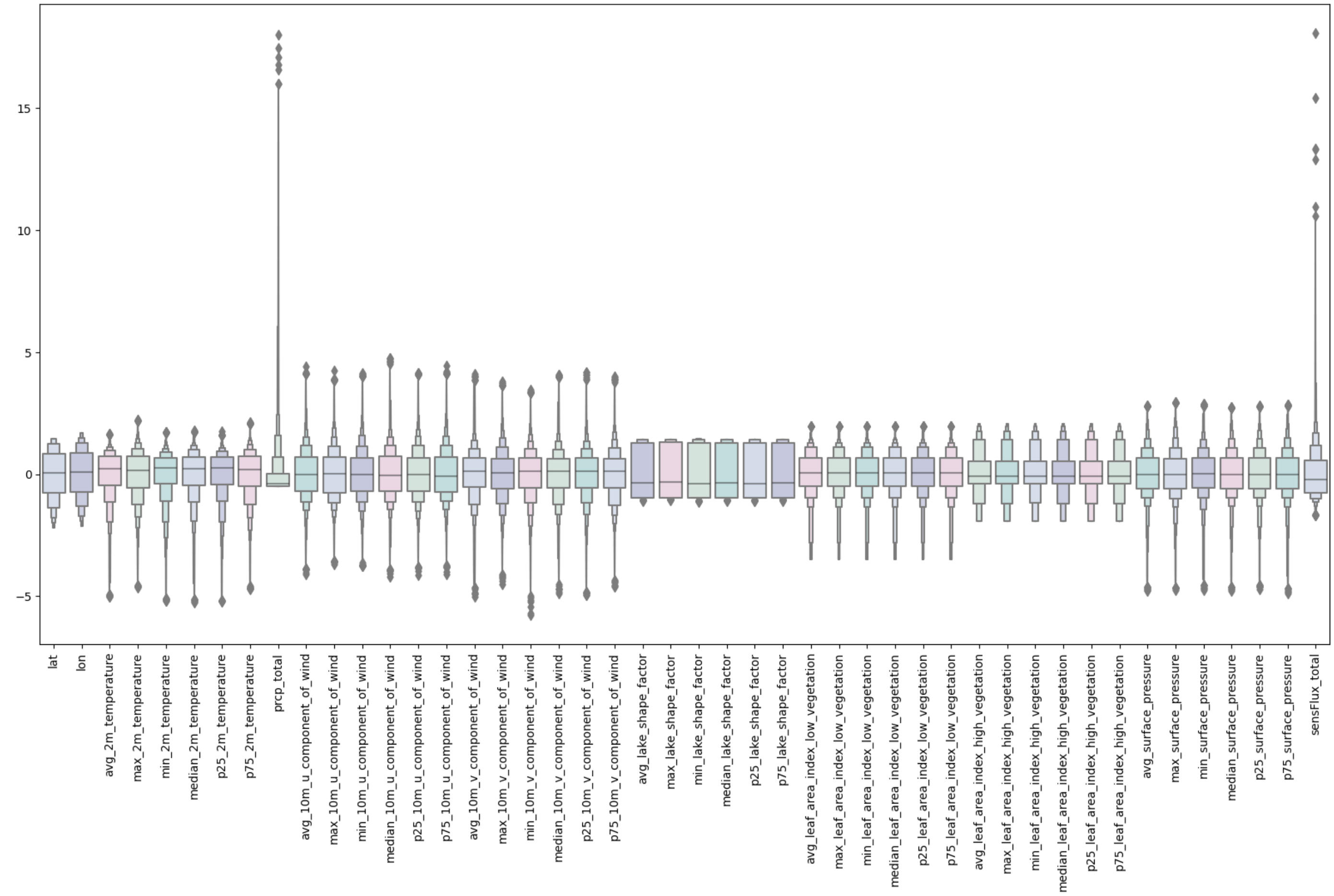
max_2m_temperature	101835.0	1.339659e-16	1.000005	-4.635613	-0.567524	0.167113	0.734910
min_2m_temperature	101835.0	4.242254e-17	1.000005	-5.197687	-0.393267	0.267346	0.683912
median_2m_temperature	101835.0	2.098800e-16	1.000005	-5.251599	-0.453864	0.218240	0.706703
p25_2m_temperature	101835.0	-5.738207e-16	1.000005	-5.209468	-0.406499	0.253726	0.695666
p75_2m_temperature	101835.0	-1.594195e-15	1.000005	-4.711178	-0.500744	0.195256	0.731812
prcp_total	101835.0	-9.879987e-17	1.000005	-0.499810	-0.481841	-0.389278	0.011919
avg_10m_u_component_of_wind	101835.0	-5.805190e-17	1.000005	-4.101912	-0.698792	-0.019214	0.711996
max_10m_u_component_of_wind	101835.0	-3.572425e-17	1.000005	-3.676495	-0.778239	0.021375	0.714730
min_10m_u_component_of_wind	101835.0	-3.572425e-17	1.000005	-3.774541	-0.679993	0.001949	0.659001
median_10m_u_component_of_wind	101835.0	-2.009489e-17	1.000005	-4.205522	-0.693084	-0.046455	0.718142
p25_10m_u_component_of_wind	101835.0	1.562936e-17	1.000005	-4.128567	-0.687848	-0.002232	0.676224
p75_10m_u_component_of_wind	101835.0	-2.902595e-17	1.000005	-4.116134	-0.723461	-0.068649	0.713732
avg_10m_v_component_of_wind	101835.0	5.302818e-18	1.000005	-5.021843	-0.541411	0.125994	0.657411
max_10m_v_component_of_wind	101835.0	-6.251743e-17	1.000005	-4.509447	-0.586238	0.059432	0.639499
min_10m_v_component_of_wind	101835.0	-1.032654e-17	1.000005	-5.763795	-0.574418	0.122938	0.667520
median_10m_v_component_of_wind	101835.0	-1.925760e-17	1.000005	-4.877706	-0.568708	0.129608	0.646900
p25_10m_v_component_of_wind	101835.0	4.632988e-17	1.000005	-4.963021	-0.575789	0.128065	0.662037
p75_10m_v_component_of_wind	101835.0	-1.786212e-17	1.000005	-4.617847	-0.574056	0.108271	0.633262
avg_lake_shape_factor	101835.0	1.431203e-15	1.000005	-1.115180	-0.960313	-0.364633	1.285405
max_lake_shape_factor	101835.0	2.012838e-15	1.000005	-1.085848	-0.968154	-0.325780	1.313830
min_lake_shape_factor	101835.0	9.065028e-16	1.000005	-1.129916	-0.951774	-0.393614	1.289870
median_lake_shape_factor	101835.0	-7.959809e-16	1.000005	-1.116869	-0.959505	-0.370417	1.283262
p25_lake_shape_factor	101835.0	-1.903433e-15	1.000005	-1.120033	-0.955777	-0.394687	1.289606
p75_lake_shape_factor	101835.0	-9.165502e-16	1.000005	-1.112380	-0.963315	-0.346859	1.288980

avg_leaf_area_index_low_vegetation	101835.0	2.188110e-16	1.000005	-3.499544	-0.488862	0.066289	0.678477
max_leaf_area_index_low_vegetation	101835.0	-4.420876e-16	1.000005	-3.501213	-0.488177	0.066125	0.678209
min_leaf_area_index_low_vegetation	101835.0	-1.964834e-16	1.000005	-3.497880	-0.489680	0.065948	0.678656
median_leaf_area_index_low_vegetation	101835.0	3.840357e-16	1.000005	-3.499544	-0.488928	0.066321	0.678407
p25_leaf_area_index_low_vegetation	101835.0	3.840357e-16	1.000005	-3.499544	-0.488928	0.066321	0.678407
p75_leaf_area_index_low_vegetation	101835.0	3.840357e-16	1.000005	-3.499544	-0.488928	0.066321	0.678407
avg_leaf_area_index_high_vegetation	101835.0	1.205693e-16	1.000005	-1.918176	-0.393488	-0.087491	0.521810
max_leaf_area_index_high_vegetation	101835.0	8.596147e-17	1.000005	-1.918438	-0.392695	-0.087692	0.522041
min_leaf_area_index_high_vegetation	101835.0	-3.248674e-16	1.000005	-1.917914	-0.394132	-0.087593	0.521220
median_leaf_area_index_high_vegetation	101835.0	8.931062e-18	1.000005	-1.918176	-0.393486	-0.087491	0.521827
p25_leaf_area_index_high_vegetation	101835.0	8.931062e-18	1.000005	-1.918176	-0.393486	-0.087491	0.521827
p75_leaf_area_index_high_vegetation	101835.0	8.931062e-18	1.000005	-1.918176	-0.393486	-0.087491	0.521827
avg_surface_pressure	101835.0	-9.415572e-15	1.000005	-4.798429	-0.596532	-0.002036	0.664485
max_surface_pressure	101835.0	5.422941e-14	1.000005	-4.756542	-0.601263	-0.016577	0.642322
min_surface_pressure	101835.0	-5.090817e-14	1.000005	-4.739806	-0.570598	0.006048	0.674365
median_surface_pressure	101835.0	3.916271e-15	1.000005	-4.772205	-0.594411	0.000420	0.659787
p25_surface_pressure	101835.0	-3.864917e-15	1.000005	-4.698882	-0.591755	-0.009206	0.664914
p75_surface_pressure	101835.0	-5.269996e-14	1.000005	-4.881858	-0.598269	-0.009879	0.655411
sensFlux_total	101835.0	2.232766e-17	1.000005	-1.691662	-0.760079	-0.212342	0.564928

Outlier Removal

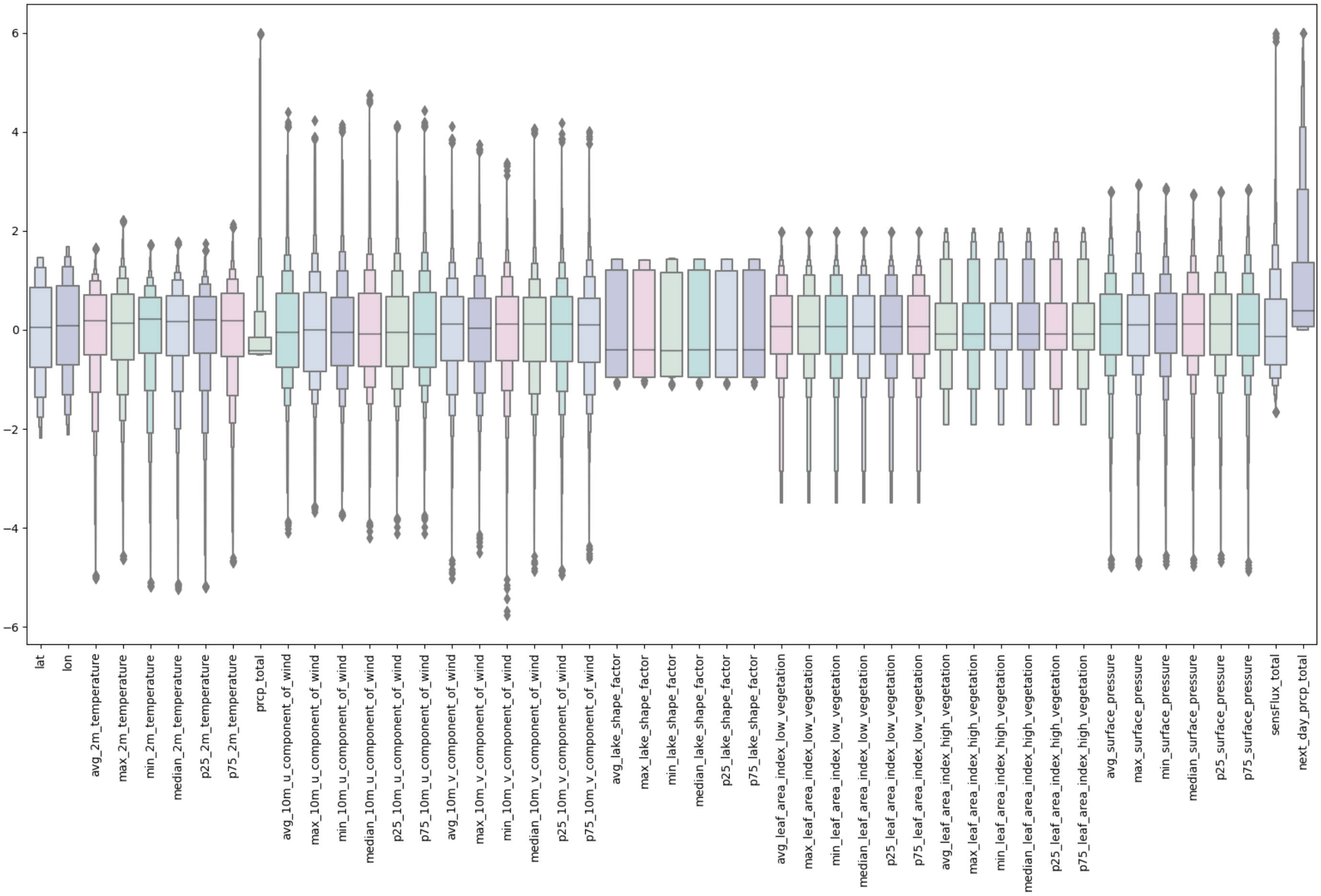
In [11]:

```
#Detecting outliers
#looking at the scaled features
colours = ["#D0DBEE", "#C2C4E2", "#EED4E5", "#D1E6DC", "#BDE2E2"]
plt.figure(figsize=(20,10))
sns.boxenplot(data = features,palette = colours)
plt.xticks(rotation=90)
plt.show()
```



In [12]:
```
#full data for
features['next_day_prcp_total'] = y

#Dropping outlier
features = features[(features.apply(lambda x: (x > -6) & (x < 6)).all(axis=1))]
```

In [13]:
```
#looking at the scaled features without outliers

plt.figure(figsize=(20,10))
sns.boxenplot(data = features,palette = colours)
plt.xticks(rotation=90)
plt.show()
```



Model Building

In [14]:
```
X = features.drop(['next_day_prcp_total'], axis=1)
y = features['next_day_prcp_total']

# Split data 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
```

Regression Analysis - Neural Network

In [15]:
```
# Build your neural network model
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)   # Output layer with a single neuron for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics = ['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test set
loss = model.evaluate(X_test, y_test)
print(f"Mean Squared Error on Test Set: {loss}")
```

```
Epoch 1/10
2199/2199 [==============================] - 3s 1ms/step - loss: 1.4084 - accuracy: 0.00
60 - val_loss: 1.2645 - val_accuracy: 0.0065
Epoch 2/10
2199/2199 [==============================] - 2s 1ms/step - loss: 1.2182 - accuracy: 0.00
```

```
64 - val_loss: 1.1532 - val_accuracy: 0.0069
Epoch 3/10
2199/2199 [==============================] - 2s 995us/step - loss: 1.1029 - accuracy: 0.
0066 - val_loss: 1.0591 - val_accuracy: 0.0070
Epoch 4/10
2199/2199 [==============================] - 2s 1ms/step - loss: 1.0215 - accuracy: 0.00
67 - val_loss: 0.9900 - val_accuracy: 0.0070
Epoch 5/10
2199/2199 [==============================] - 2s 996us/step - loss: 0.9613 - accuracy: 0.
0066 - val_loss: 0.9480 - val_accuracy: 0.0072
Epoch 6/10
2199/2199 [==============================] - 2s 992us/step - loss: 0.9129 - accuracy: 0.
0067 - val_loss: 0.9375 - val_accuracy: 0.0056
Epoch 7/10
2199/2199 [==============================] - 2s 974us/step - loss: 0.8744 - accuracy: 0.
0065 - val_loss: 0.8872 - val_accuracy: 0.0070
Epoch 8/10
2199/2199 [==============================] - 2s 978us/step - loss: 0.8370 - accuracy: 0.
0066 - val_loss: 0.8337 - val_accuracy: 0.0068
Epoch 9/10
2199/2199 [==============================] - 2s 987us/step - loss: 0.8111 - accuracy: 0.
0066 - val_loss: 0.8130 - val_accuracy: 0.0065
Epoch 10/10
2199/2199 [==============================] - 2s 954us/step - loss: 0.7850 - accuracy: 0.
0068 - val_loss: 0.7786 - val_accuracy: 0.0071
550/550 [==============================] - 0s 633us/step - loss: 0.7786 - accuracy: 0.00
71
Mean Squared Error on Test Set: [0.778622567653656, 0.007106714416295290]
```

In [16]:

```
#Shape Check
print(f"Shape of y: {y.shape}, shape of X: {X.shape}")
num_features = X.shape[1]
print(f"Number of features in X: {num_features}")
num_samples = X.shape[0]
print(f"Number of data points in X: {num_samples}")
```

```
Shape of y: (87941,), shape of X: (87941, 46)
Number of features in X: 46
Number of data points in X: 87941
```

In [17]:

```
X_train
```

Out[17]:

	lat	lon	avg_2m_temperature	max_2m_temperature	min_2m_temperature	median_2m_tempera
79818	-0.157357	0.281981	0.232389	0.307414	0.265452	0.06
84978	-1.978201	-0.516726	0.415915	-0.266578	0.869107	0.49
73508	0.651907	-0.117373	0.570201	0.394050	0.559886	0.37
74525	1.056539	-1.315433	0.759081	0.925019	0.375065	0.68
5722	0.651907	-1.315433	-2.555482	-1.915802	-2.992342	-2.45
...	...	...	...	...	...	
6476	0.854223	-0.516726	-2.975007	-2.004734	-3.115682	-3.18
61801	-0.561989	0.481658	1.007030	1.262306	0.971013	0.79
89295	0.247275	-0.516726	0.365967	0.628660	0.094047	0.50
861	-1.573569	1.080688	-2.635922	-2.559760	-2.783746	-2.34
17221	-1.573569	0.481658	0.130717	0.635691	-0.342037	0.26

70352 rows × 46 columns

In [18]:
```
y_train
```

Out[18]:
```
79818    0.025626
84978    1.936733
73508    4.237180
74525    3.612345
5722     0.724529
           ...
6476     0.000811
61801    1.091071
89295    0.342891
861      0.247908
17221    0.093117
Name: next_day_prcp_total, Length: 70352, dtype: float64
```

In [19]:
```
# show a summary of the data
model.summary()
```

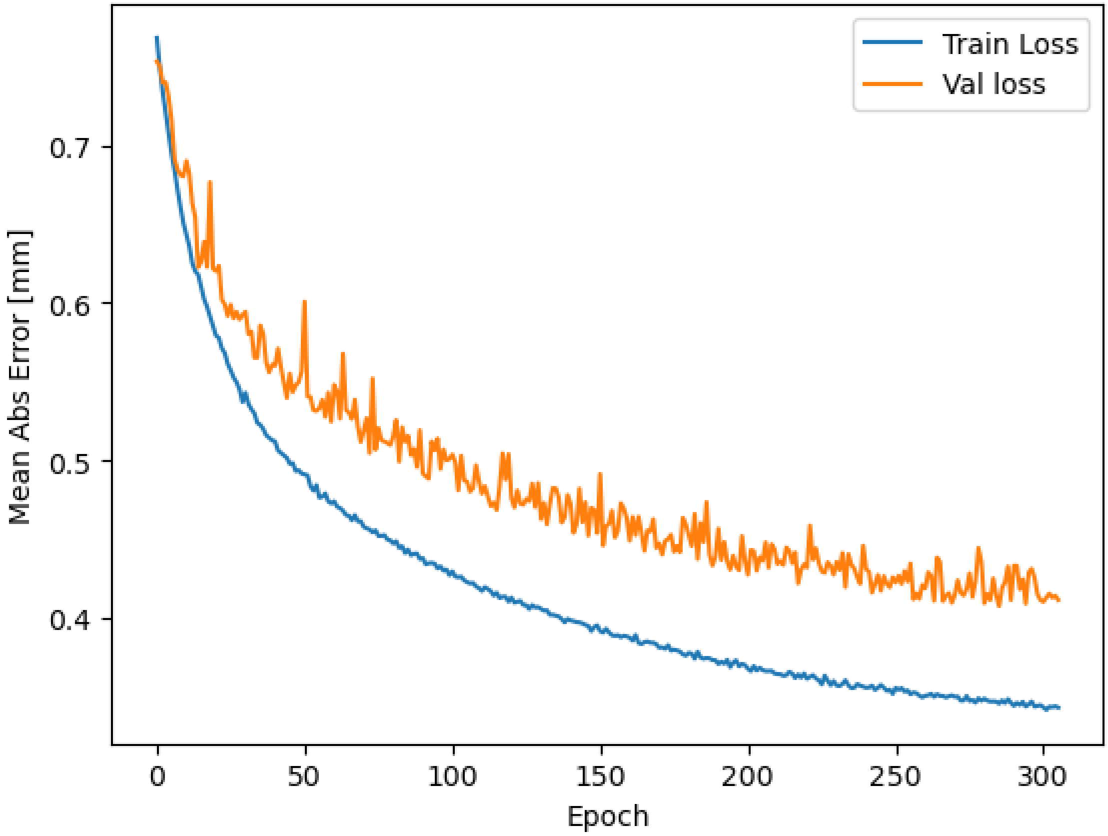
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                3008

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 5121 (20.00 KB)
Trainable params: 5121 (20.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [20]:
```
# Display training progress by printing a single dot for each completed epoch
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

# Function to plot how the model is doing during training
# Visualize the model's training progress using the stats stored in the history object.
# We want to use this data to determine how long to train before the model stops making
def plot_history(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error [mm]')
    plt.plot(history.epoch, np.array(history.history['loss']),
           label='Train Loss')
    plt.plot(history.epoch, np.array(history.history['val_loss']),
           label = 'Val loss')
    plt.legend()
    #plt.ylim([0, 5])
```

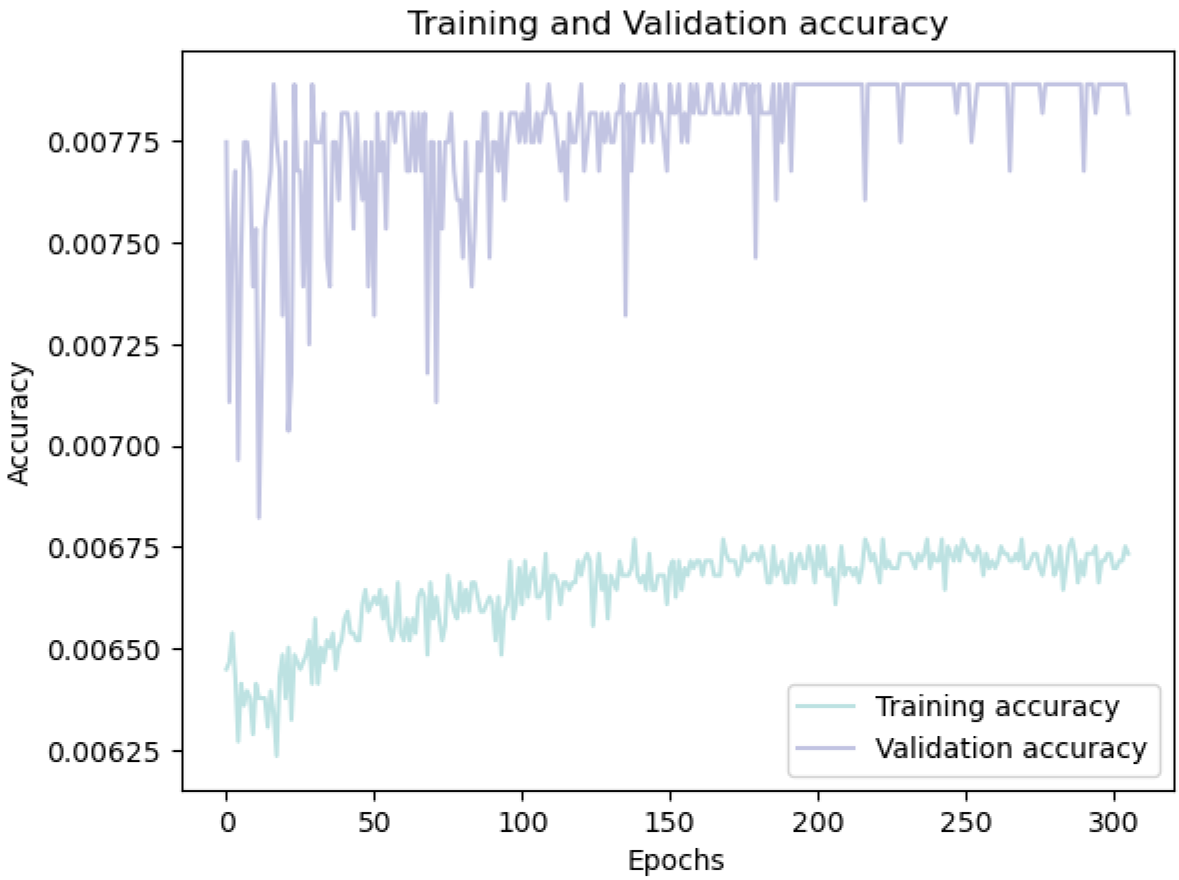
In [21]:
```
# If you train too long, you are prone to over-fitting
# this prevents the model from generalizing to data it has never seen before
# early stopping is one way to go about this
# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)

# Store training stats
history = model.fit(X_train, y_train, epochs=1000,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])
```

```
plot_history(history)
```

```
.............................................................................
.............................................................................
.............................................................................
......
```



In [24]:
```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['accuracy']], "#BDE2E2", label='Training accuracy')
plt.plot(history_df.loc[:, ['val_accuracy']], "#C2C4E2", label='Validation accuracy')

plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [25]:
```
print(history_df.columns)
```

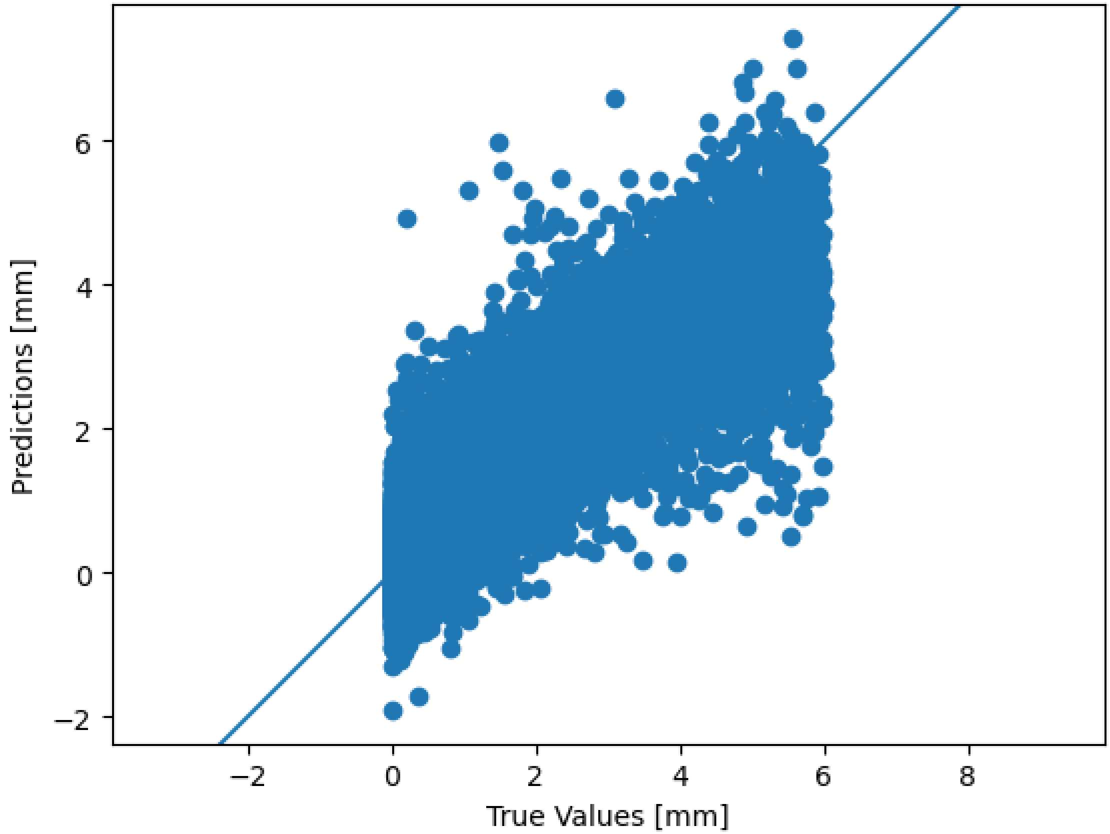
```
Index(['loss', 'accuracy', 'val_loss', 'val_accuracy'], dtype='object')
```

In [26]:
```
# Calculate MAE separately
from sklearn.metrics import mean_absolute_error
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error on Test Set: {mae} millimeters")
```

```
550/550 [==============================] - 0s 594us/step
Mean Absolute Error on Test Set: 0.4433258278176518 millimeters
```

In [27]:
```
test_predictions = model.predict(X_test).flatten()

plt.scatter(y_test, test_predictions)
plt.xlabel('True Values [mm]')
plt.ylabel('Predictions [mm]')
plt.axis('equal')
plt.xlim(plt.xlim())
plt.ylim(plt.ylim())
_ = plt.plot([-100, 100], [-100, 100])
```

```
550/550 [==============================] - 0s 587us/step
```



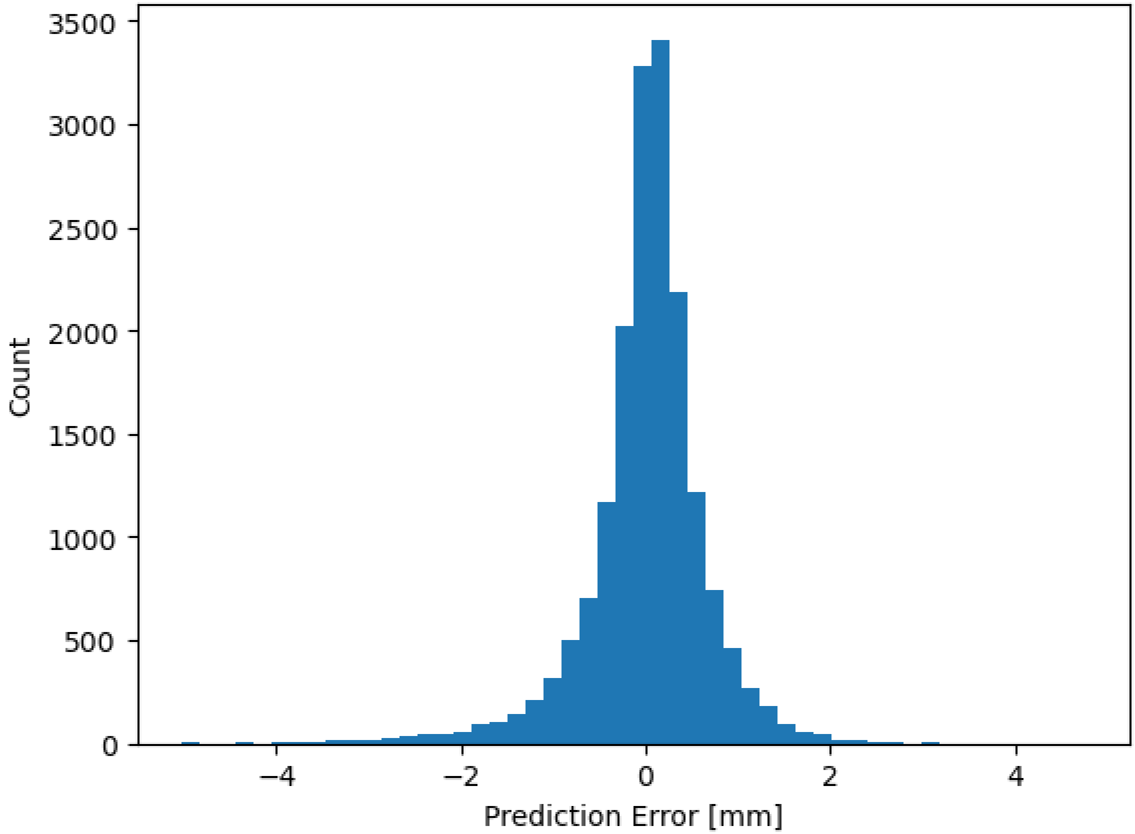
In [28]:
```
np.corrcoef(y_test,test_predictions)[0,1]
```

Out[28]:
```
0.8812278207459403
```

In [29]:
```
from sklearn.metrics import r2_score
r2_score(y_test,test_predictions)
```

Out[29]:
```
0.7759838324734877
```

In [30]:
```
error = test_predictions - y_test
plt.hist(error, bins = 50)
plt.xlabel("Prediction Error [mm]")
_ = plt.ylabel("Count")
```



Categorical Analysis - Data Preprocessing

In [32]:
```
import pandas as pd

# Calculate the 30th and 60th percentiles
percentile_30 = y_train.quantile(0.3)
percentile_60 = y_train.quantile(0.6)

# Create categorical labels based on percentiles
y_train_category = pd.cut(
    y_train,
    bins=[float('-inf'), percentile_30, percentile_60, float('inf')],
    labels=['0', '1', '2']
)

# Repeat the same process for y_test
y_test_category = pd.cut(
    y_test,
    bins=[float('-inf'), percentile_30, percentile_60, float('inf')],
    labels=['0', '1', '2']
)
```

In [33]:
```
print('Category 0 is from 0 to ' + str(round(percentile_30, 3)) + ' mm of rain')
print('Category 1 is from ' + str(round(percentile_30, 3)) + ' mm of rain to ' + str(rou
print('Category 2 is from ' + str(round(percentile_60, 3)) + ' mm of rain to the maximum
```

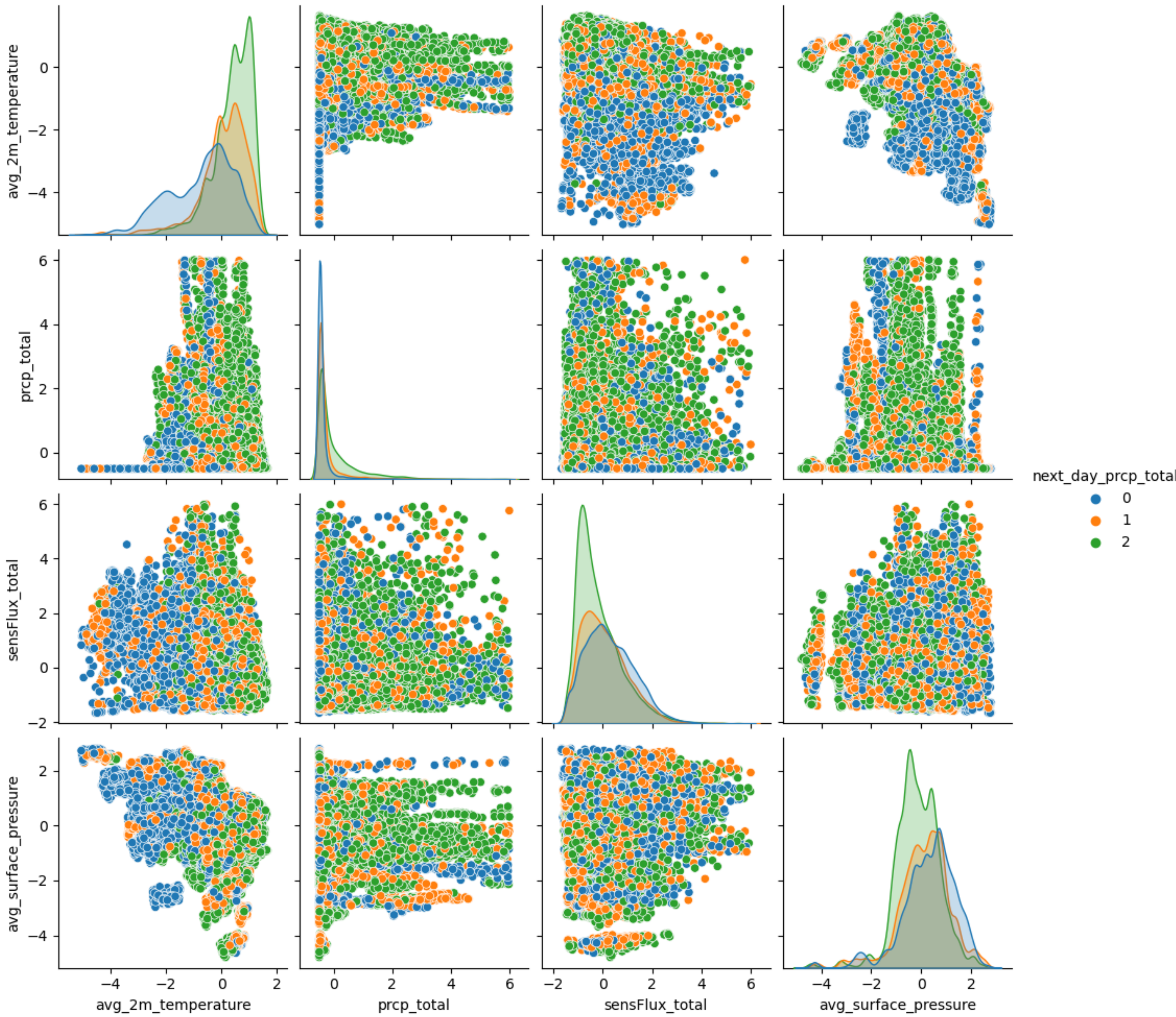
```
Category 0 is from 0 to 0.104 mm of rain
Category 1 is from 0.104 mm of rain to 0.649 mm of rain
Category 2 is from 0.649 mm of rain to the maximum
```

Categorical Analysis - Pairwise Correlation

In [34]:
```
X_cat = pd.concat([X_train, X_test,])
y_cat = pd.concat([y_train_category, y_test_category])
```

```python
X_cat['next_day_prcp_total'] = y_cat
data_cat = X_cat
sns.pairplot( data=data_cat, vars=('avg_2m_temperature','prcp_total','sensFlux_total','a
```

Out[34]: <seaborn.axisgrid.PairGrid at 0x1d8710e51f0>



Categorical Analysis - Model Building

In [35]:

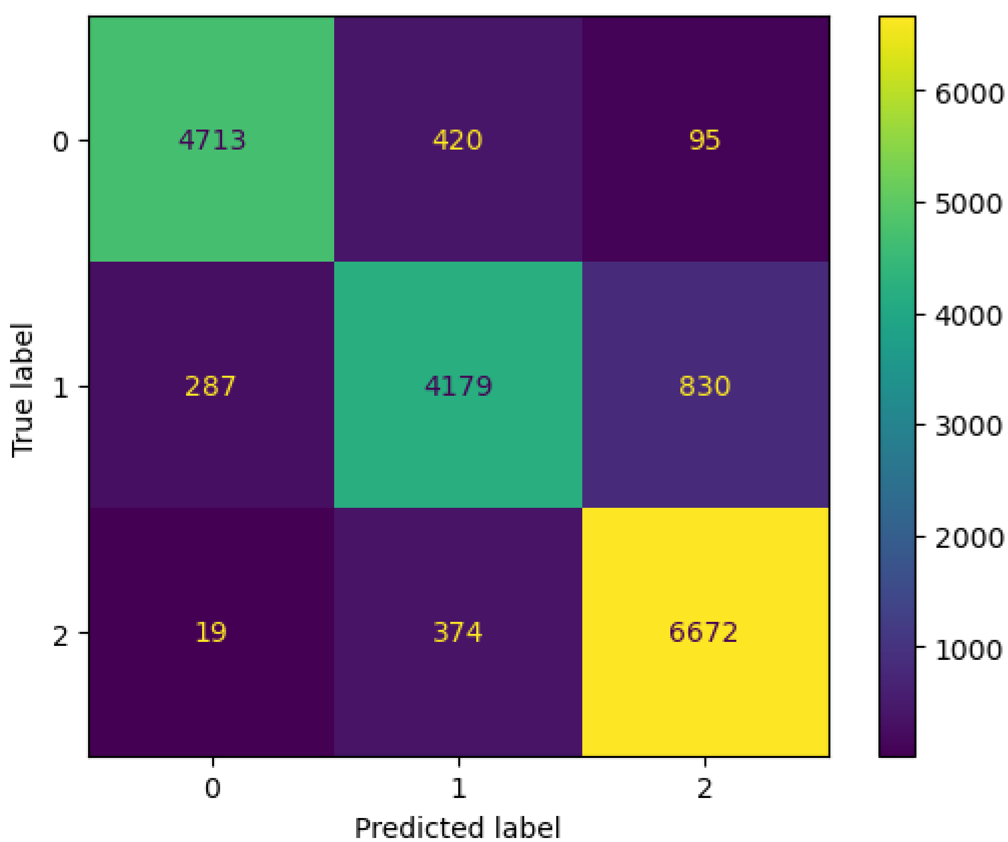
```python
import time
from sklearn.metrics import accuracy_score, cohen_kappa_score, classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
def run_model(model, X_train, y_train_category, X_test, y_test_category, verbose=True):
    t0=time.time()
    if verbose == False:
        model.fit(X_train,y_train_category, verbose=0)
    else:
        model.fit(X_train,y_train_category)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test_category, y_pred)
    coh_kap = cohen_kappa_score(y_test_category, y_pred)
    time_taken = time.time()-t0
    print("Accuracy = {}".format(accuracy))
    print("Cohen's Kappa = {}".format(coh_kap))
    print("Time taken = {}".format(time_taken))
    print(classification_report(y_test_category,y_pred,digits=5))
```

```
    cm = confusion_matrix(y_test_category, y_pred, labels=model.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=model.classes_)
    disp.plot()
    plt.show()

    return model, accuracy, coh_kap, time_taken
```

Categorical Analysis - Random Forest

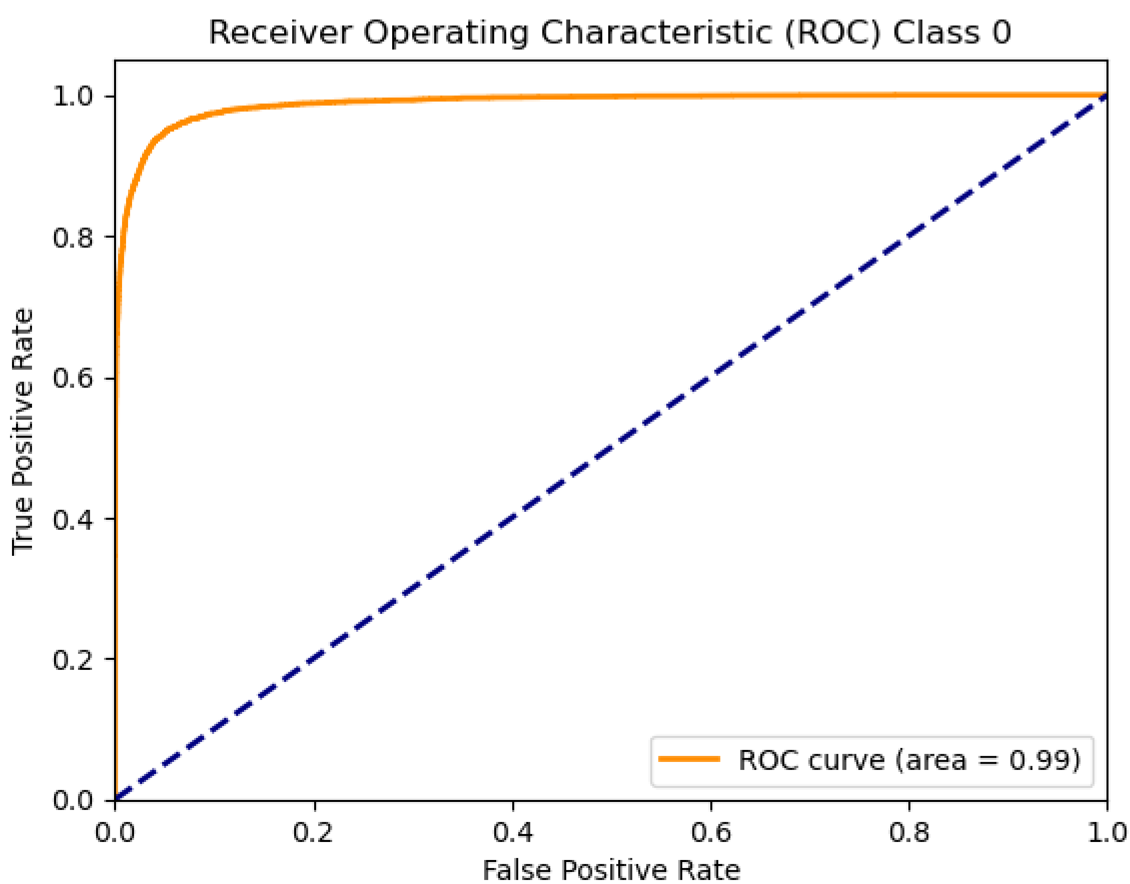
In [36]:
```
from sklearn.ensemble import RandomForestClassifier

params_rf = {'max_depth': 16,
             'min_samples_leaf': 1,
             'min_samples_split': 2,
             'n_estimators': 100,
             'random_state': 12345}

model_rf = RandomForestClassifier(**params_rf)
model_rf, accuracy_rf, coh_kap_rf, tt_rf = run_model(model_rf, X_train, y_train_category
```

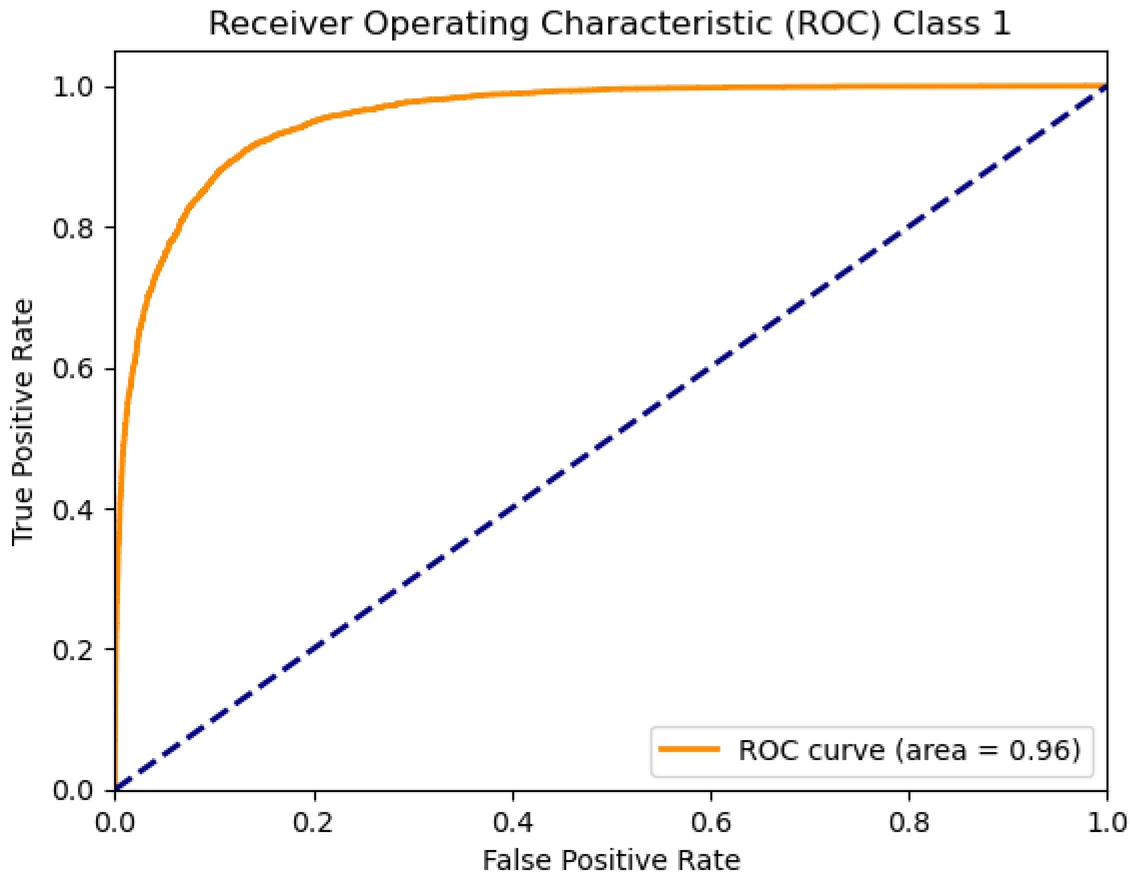
```
Accuracy = 0.884871226334641
Cohen's Kappa = 0.8246501433687409
Time taken = 80.55607771873474
              precision    recall  f1-score   support

           0    0.93903   0.90149   0.91988      5228
           1    0.84034   0.78909   0.81391      5296
           2    0.87824   0.94437   0.91011      7065

    accuracy                        0.88487     17589
   macro avg    0.88587   0.87832   0.88130     17589
weighted avg    0.88490   0.88487   0.88405     17589
```

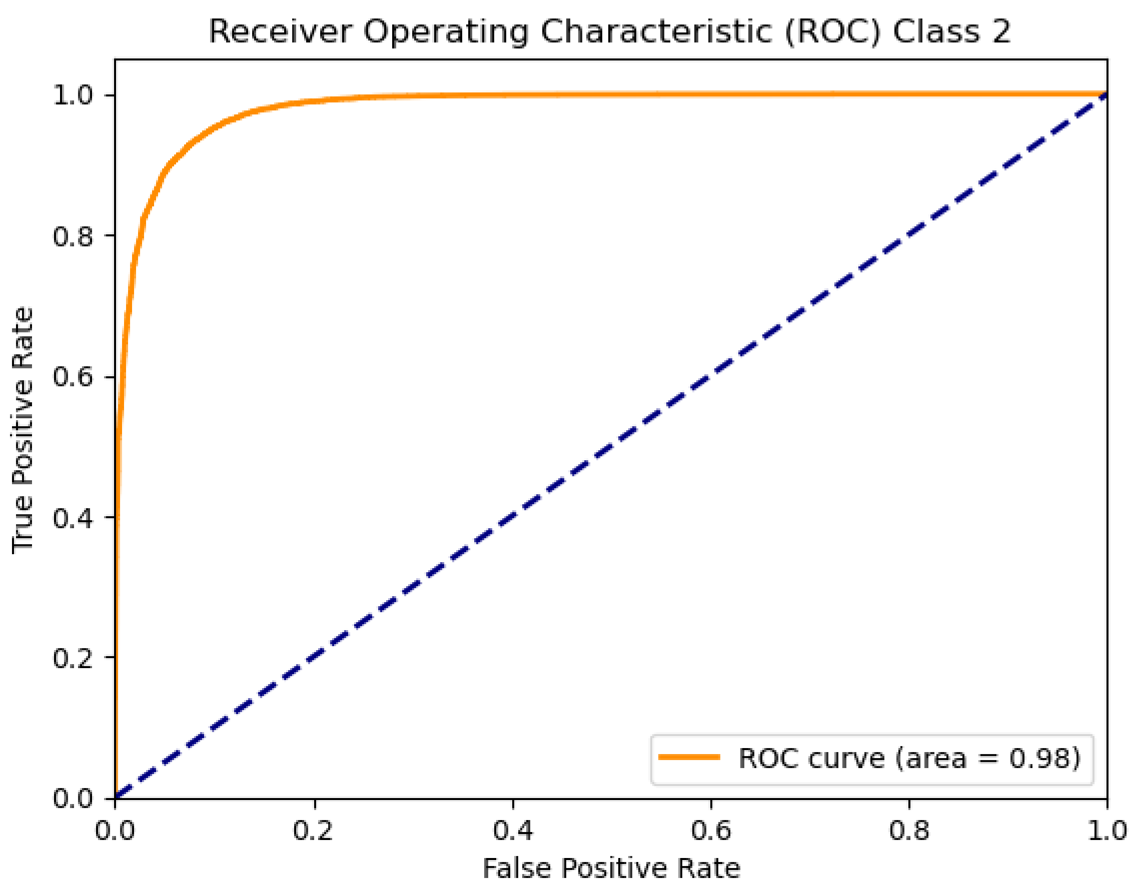


In [37]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
```

```python
import numpy as np

# Binarize the labels
y_test_binarized = label_binarize(y_test_category, classes=['0', '1', '2'])
n_classes = y_test_binarized.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

probs = model_rf.predict_proba(X_test)  # Calculate predicted probabilities


for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for each class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], color='darkorange', lw=2, label='ROC curve (area = {:.2f})'
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Class {}'.format(i))
    plt.legend(loc='lower right')
    plt.show()
```







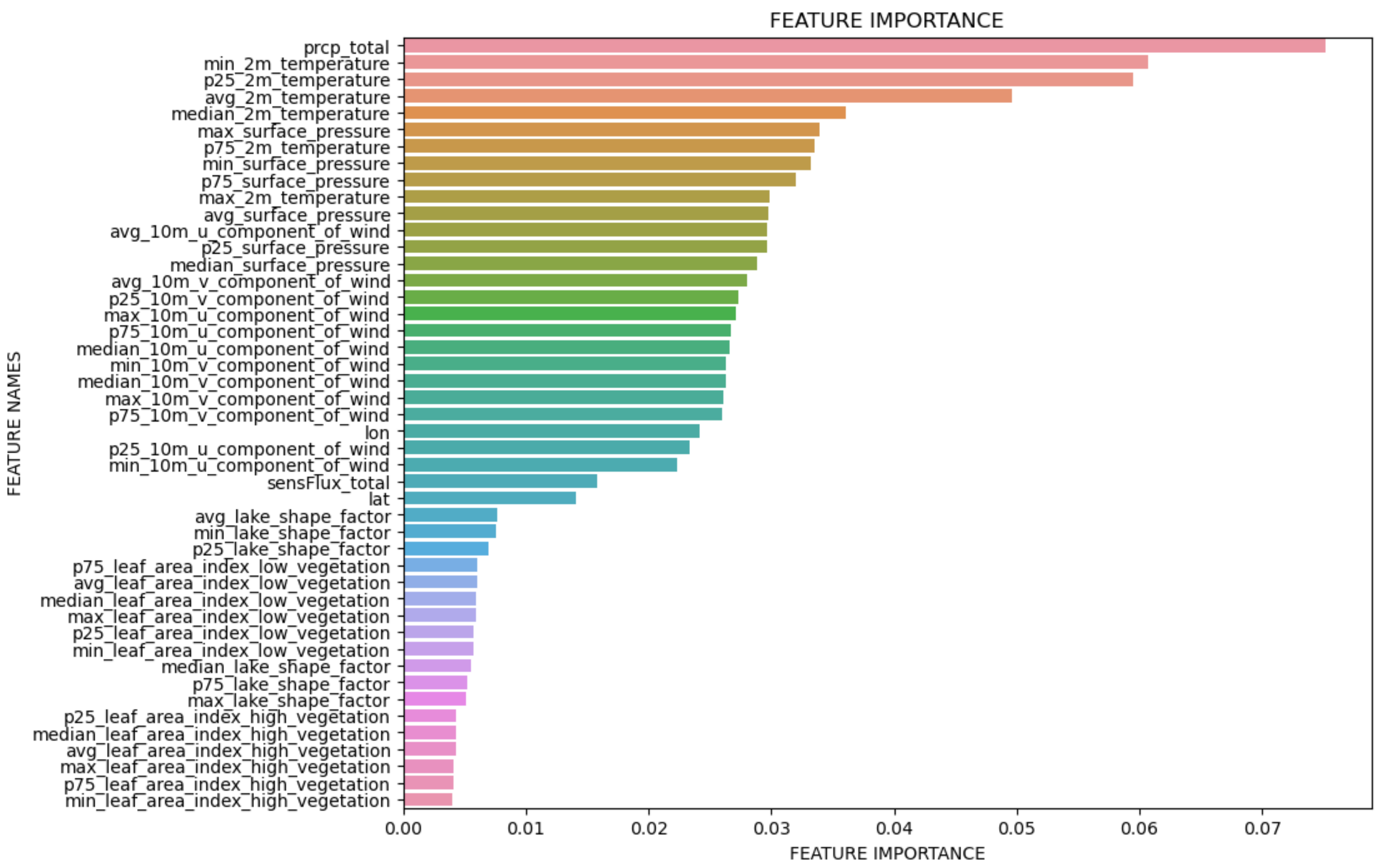
In [38]:

```
def plot_feature_importance(importance, names):
    #Create arrays from feature importance and feature names
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    #Create a DataFrame using a Dictionary
    data={'feature_names':feature_names,'feature_importance':feature_importance}
```

```
    fi_df = pd.DataFrame(data)

    #Sort the DataFrame in order decreasing feature importance
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    #Define size of bar plot
    plt.figure(figsize=(10,8))
    #Plot Searborn bar chart
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    #Add chart labels
    plt.title('FEATURE IMPORTANCE')
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
```

In [39]:
```
dfxTrain=pd.DataFrame(X_train, columns=col_names)
```

In [41]:
```
plot_feature_importance(model_rf.feature_importances_, dfxTrain.columns)
```

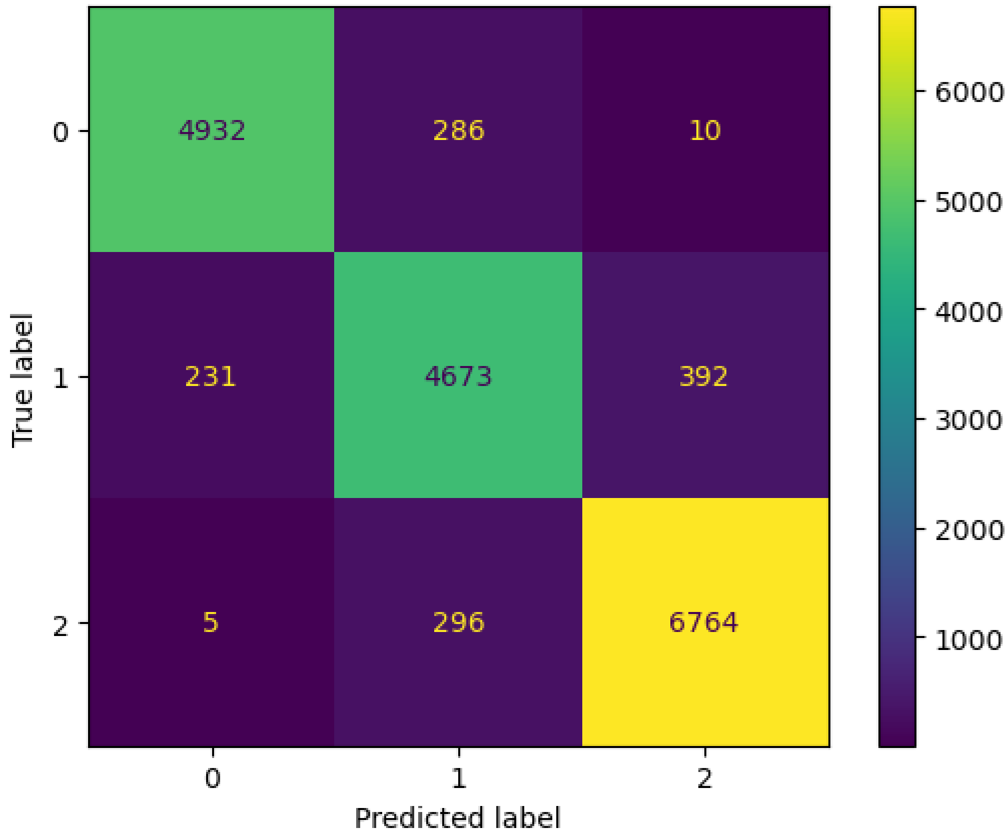


Categorical Analysis - XGBoost

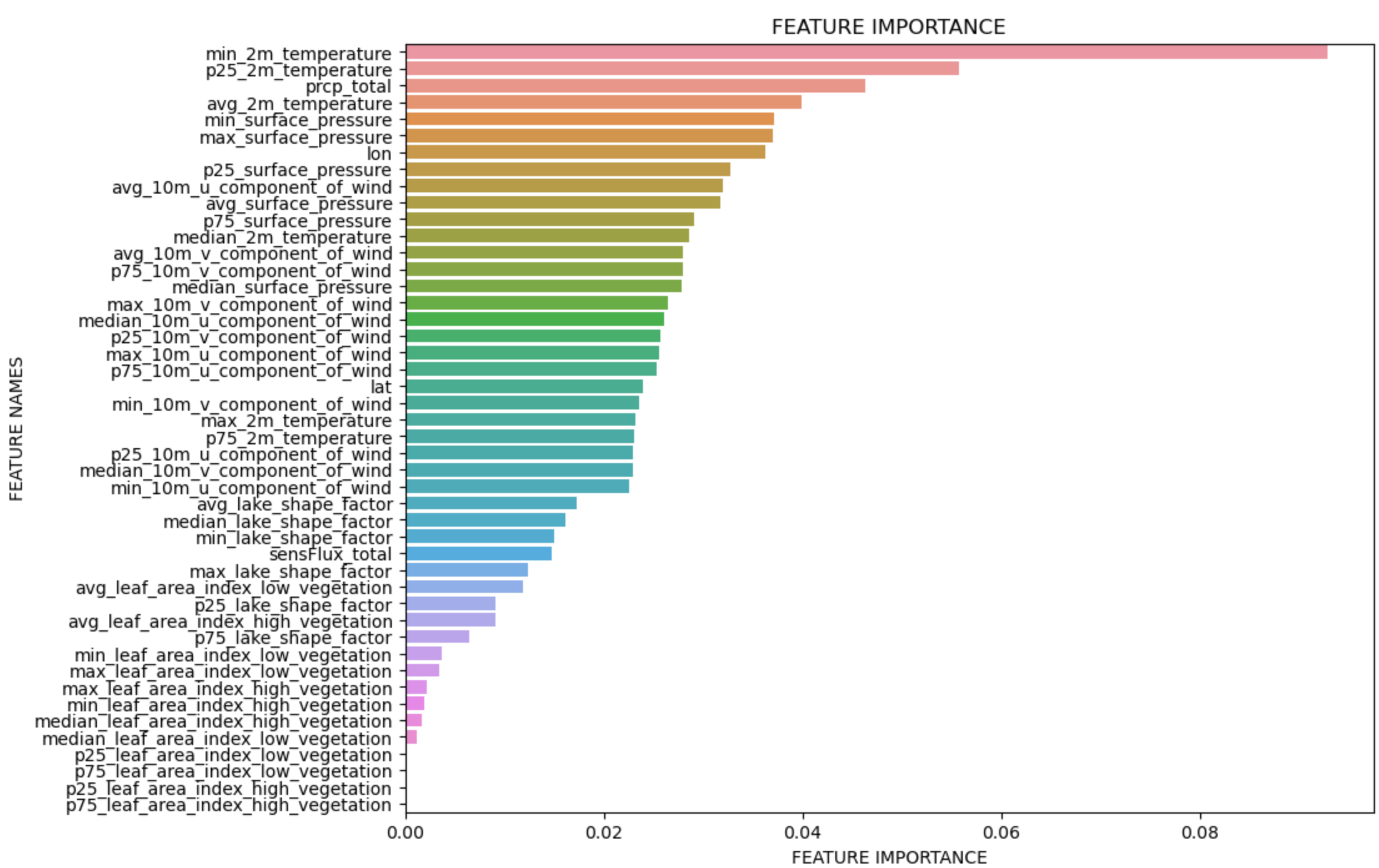
In [42]:
```
y_train_category = y_train_category.astype(int)
y_test_category = y_test_category.astype(int)

import xgboost as xgb
params_xgb ={'n_estimators': 500,
            'max_depth': 16}

model_xgb = xgb.XGBClassifier(**params_xgb)
model_xgb, accuracy_xgb, coh_kap_xgb, tt_xgb = run_model(model_xgb, X_train, y_train_cat
```

```
Accuracy = 0.930638467223833
Cohen's Kappa = 0.8947574082825525
Time taken = 31.319106578826904
              precision    recall  f1-score   support
```

```
           0    0.95433   0.94338   0.94883      5228
           1    0.88925   0.88236   0.88579      5296
           2    0.94390   0.95740   0.95060      7065

    accuracy                        0.93064     17589
   macro avg    0.92916   0.92771   0.92841     17589
weighted avg    0.93055   0.93064   0.93056     17589
```



In [43]:
```
plot_feature_importance(model_xgb.feature_importances_, dfxTrain.columns)
```



Saving the model to reuse it again

In [44]:
```
import joblib
joblib.dump(model_xgb, "xg_rain.pkl")
```

Out[44]:
```
['xg_rain.pkl']
```

Limitations and future work

Regressors and classifiers are usually not directly compared. It would require further processing to compare them. Adding more models, for example, a random forest regression, and a multi class prediction neural network could have yielded further comparison of the possible machine learning approaches. Even though latitude (lat) and longitude (lon) were added as features in one iteration, they were removed to produce more generalized models. A more generalized model facilitates applications to other datasets. Particularly since the model was only trained on a small subset of latitude and longitude, South Florida. Also, using a model without longitude and latitude dependency, creates a more physically based model (temperature, surface pressure, etc.) as opposed to a more location based model.

References

*Run model function/ roc and auc curve: https://github.com/azalahmadkhan/Precipitation-Prediction-using-ML/blob/main/Precipitation_Prediction.ipynb **

*Outlier removal, training and validation accuracy, standardization, confusion matrix:https://www.kaggle.com/code/karnikakapoor/rain-prediction-ann#MODEL-BUILDING **

*Pairwise correlation, heatmap, confusion matrix: https://www.kaggle.com/code/chandrimad31/rainfall-prediction-7-popular-models#Model-Comparison **

*Plot Feature Importance: https://www.analyseup.com/learn-python-for-data-science/python-random-forest-feature-importance-plot.html **

*Saving the model to reuse it again: https://github.com/Biswajit6844/rainfall-prediction/blob/master/Xgboost%20model.ipynb **