Part I: Random Forest Optimization

All Stations, 1960 - 1969

ROUND 1

Grid Search

```
grid_space = {'max_depth': [50, 75, 80, None],
        'n_estimators': [100, 200, 300],
        'max_features':[ 1,3, 5,7]}
```

Random Search

```
rs_space = {'max_depth': list(np.arange(10, 200, step=20)) + [None],
         'n_estimators':np.arange(10, 500, step=50),
        'max_features': randint(1,7),
        'criterion': ['gini','entropy'],
        'min_samples_leaf': randint(1,4),
        'min_samples_split':np.arange(2, 10, step=2)}
```

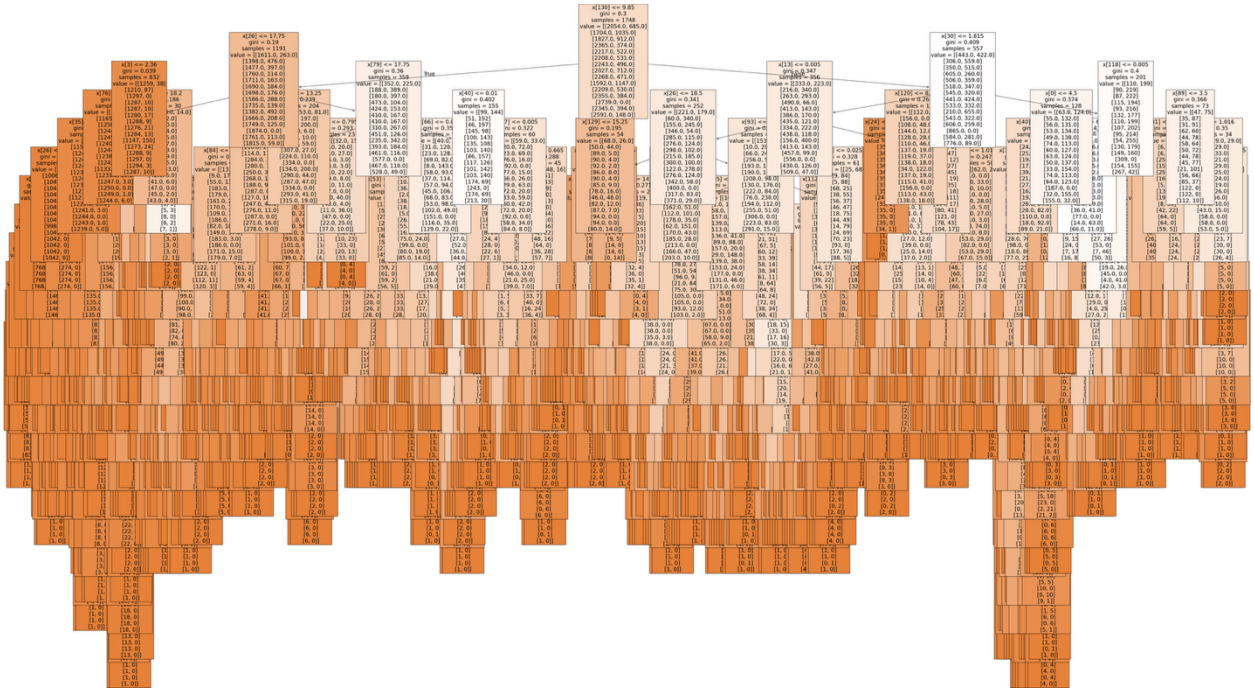
Results

Hyperparameters	**GRID search**	**RANDOM search**
max_depth	50	13
max_features	7	6
n_estimators	300	310
criterion	--	gini
min_samples_leaf	--	1
min_samples_split	--	2
best score	**0.541**	**0.535**

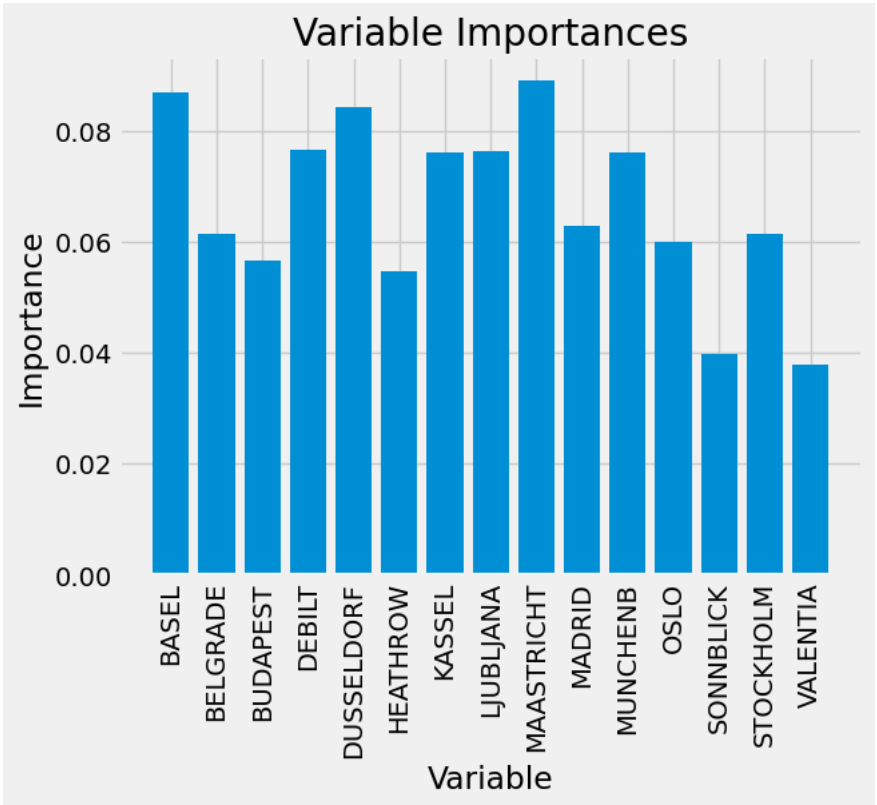
Model run on:
(n_estimators = 300, max_depth=50, max_features = 7)

Model Accuracy: 0.572

Tree



Importances



ROUND 2

Grid Search

```
grid_space = {'max_depth': [20, 50, 100, None],
              'n_estimators': [200, 300, 400],
              'max_features': [5, 7, 9]}
```

Random Search

```
rs_space = {'max_depth': list(np.arange(10, 200, step=20)) + [None],
            'n_estimators': np.arange(100, 500, step=50),
            'max_features': randint(1,7),
            'criterion': ['gini','entropy'],
            'min_samples_leaf': randint(1,4),
            'min_samples_split': np.arange(2, 10, step=2)}
```

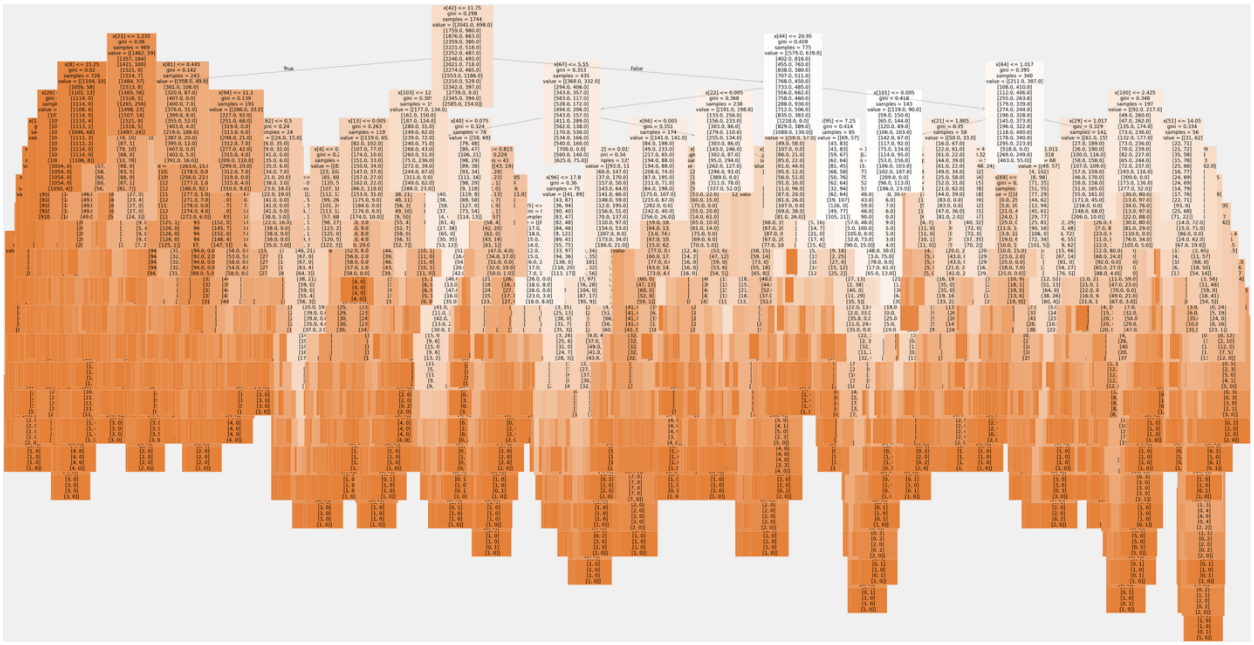
Results

Hyperparameters	**GRID search**	**RANDOM search**
max_depth	20	11
max_features	9	6
n_estimators	300	200
criterion	--	gini
min_samples_leaf	--	1
min_samples_split	--	6
best score	**0.554**	**0.529**

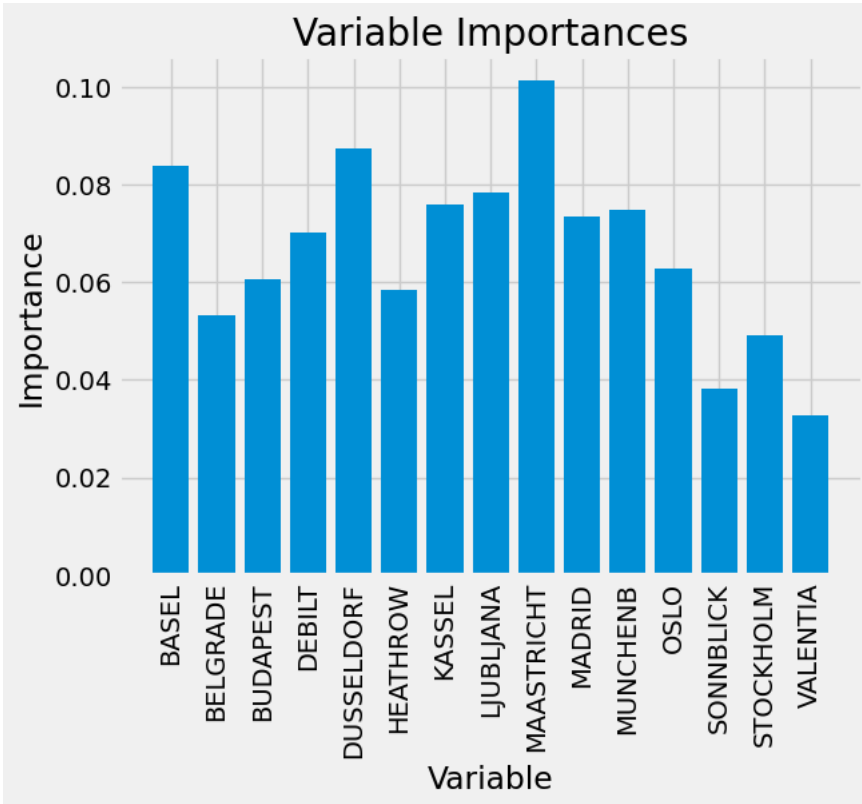
Model run on:
(n_estimators = 300, max_depth=20, max_features = 9)

Model Accuracy: 0.589

Tree



Importances



Compare with before-optimization

	Before Optimization	Round 1 Optimization	Round 2 Optimization
n_estimators	100	300	300
max_depth	--	50	20
max_features	--	7	9
Model Accuracy	0.606	0.572	0.589
Importances	1. Basel 2. Dusseldorf 3. Maastricht	1. Maastricht 2. Basel 3. Dusseldorf	1. Maastricht 2. Dusseldorf 3. Basel

Observations:

For the Random Forest model, optimization does not seem to improve the results. The initial model run on the simplest hyperparameter of n_estimators = 100 has provided the highest accuracy.

BASEL Station, 1960 – 2022

Grid Search

```
grid_space = {'max_depth': [50, 75, 80, None],
        'n_estimators': [100, 200, 300],
        'max_features': [1,3, 5,7]}
```

Random Search

```
rs_space = {'max_depth':list(np.arange(10, 200, step=20)) + [None],
        'n_estimators':np.arange(10, 500, step=50),
        'max_features': randint(1,7),
         'criterion': ['gini','entropy'],
         'min_samples_leaf': randint(1,4),
        'min_samples_split':np.arange(2, 10, step=2)}
```

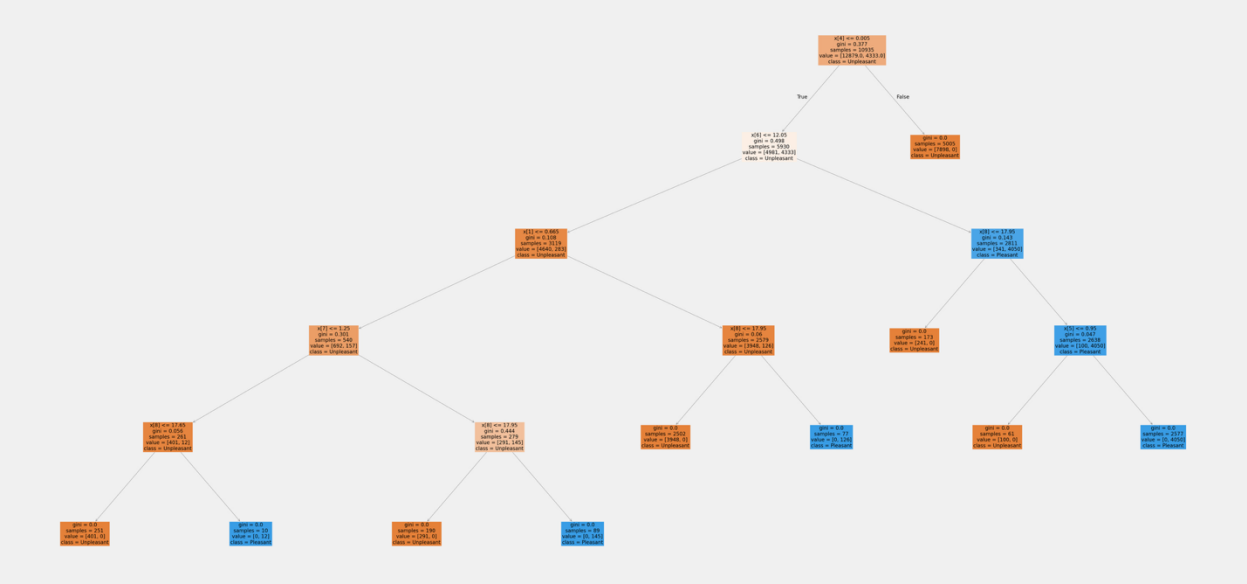
Results

Hyperparameters	**GRID search**	**RANDOM search**
max_depth	50	10
max_features	3	6
n_estimators	200	310
criterion	--	entropy
min_samples_leaf	--	3
min_samples_split	--	2
best score	**1.0**	**1.0**

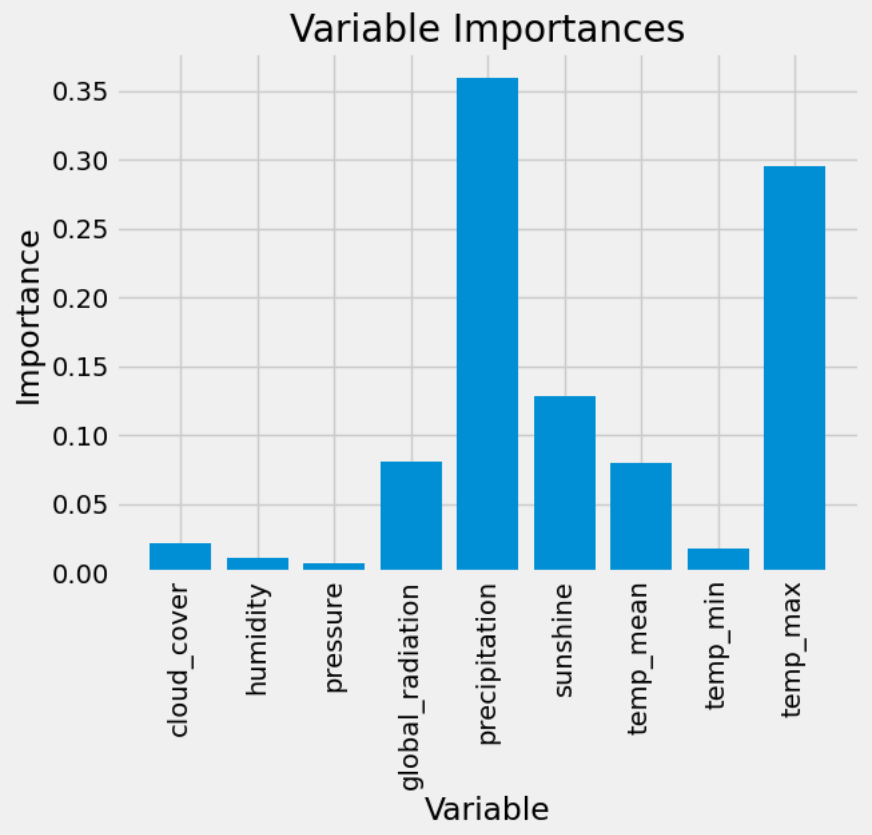
Model run on:
(n_estimators = 200, max_depth=50, max_features=3)

Model Accuracy: 1.0

Tree



Importances



Compare with before-optimization

	Before Optimization	After Optimization
n_estimators	100	200
max_depth	5	50
max_features	--	3
Model Accuracy	1.0	1.0
Importances	1. precipitation 2. temp_max 3. sunshine	1. precipitation 2. temp_max 3. sunshine

Observations:

As the model initially already ran at 100% accuracy, there was no expectation that an optimization could improve on the model.

Part II: Deep Learning

Bayesian Search

Optimum Results

Hyperparameters	Values
neurons (n_hidden)	85
kernel	np.float64(1.649)
activation	softsign
optimizer	SGD
learning_rate	np.float64(0.331)
batch_size	620
epochs	54
layers1	2
layers2	2
normalization	np.float64(0.346)
dropout	np.float64(0.516)
dropout_rate	np.float64(0.130)

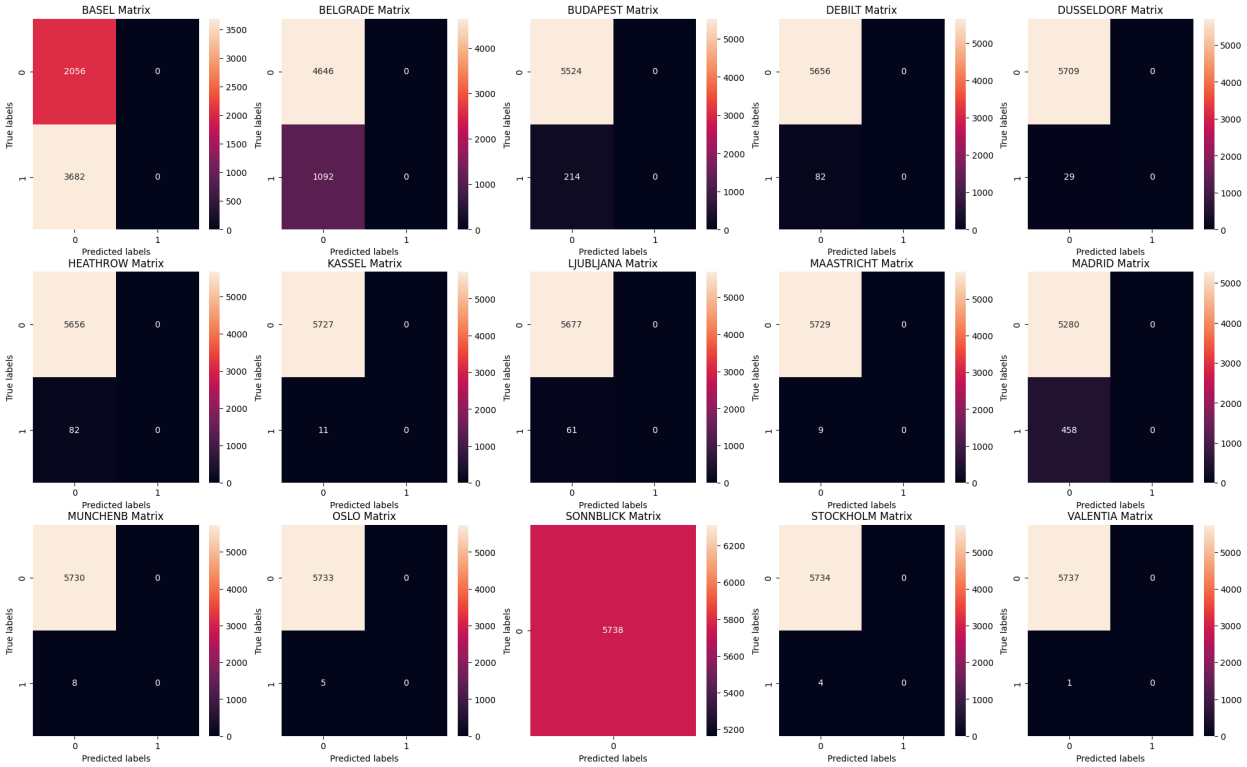
Model Accuracy: 0.7013

Final Confusion Matrix

(with pd.crosstab):

Pred	BASEL	BELGRADE	MADRID
True			
BASEL	3456	226	0
BELGRADE	568	524	0
BUDAPEST	124	89	1
DEBILT	38	44	0
DUSSELDORF	22	7	0
HEATHROW	60	20	2
KASSEL	8	3	0
LJUBLJANA	53	8	0
MAASTRICHT	8	1	0
MADRID	394	62	2
MUNCHENB	8	0	0
OSLO	3	1	1
STOCKHOLM	1	3	0
VALENTIA	1	0	0

(with Confusion Matrix display):



Compare with before-optimization

	Before Optimization	**After Optimization**
epochs	20	54
batch_size	32	620
n_hidden	32	85
activation 1	relu	softsign
activation 2	relu	softsign
activation 3 (final layer)	relu	softmax
loss	categorical_crossentropy	sparse_categorical_crossentropy
optimizer	adam	SGD
Model Accuracy	0.644	0.7013

Observations:

The Bayesian search optimized the model accuracy by only about 5%, which is not very impressive. In the end, the CNN model could only produce 70% accuracy. An RNN model might have performed better, since the data has a time element.

The confusion matrices are showing some concerning issues in the prediction of the model. The first matrix, produced by the using pd.crosstab, is showing 15 true stations and only 3 predicted stations. This confusion matrix, although produced by following the script, is confusing me, because I thought the objective was for the model to predict whether a particular day was pleasant or not (the y data set comes from the pleasant weather answers). This confusion matrix is showing how the model predicts if the data belongs to a certain station.

I then produced a confusion matrix that shows the results of predicting pleasant weather, but the matrix also shows that the model is performing poorly; in fact, it predicts no pleasant weather at all.

Part III: Iteration

How might you break the data down into smaller components to test and iterate upon.

I would break the data down by time, first iterating on just a decade of data. I could test on two different decades and see if I can optimize on both.

Another option is to choose a select number of stations, say 3 stations from different geographical areas, and iterate on those first. We have seen that with stations too similar in geography, the model will overfit. So it would be better to select a station each from northern Europe, central Europe, and southern Europe.

Which model would you use for each iteration? Expand on your observations from the random forest and deep learning models.

For the iterations on these smaller datasets, I would use the deep learning models, as the random forest has proven to be less accurate and difficult to interpret. I would like to try optimizing the RNN model, which may provide better results for this data set with a time element.

What variables would you recommend that Air Ambulance pay the most attention to while deciding whether it's safe to fly?

I would recommend that Air Ambulance pay attention to precipitation and maximum temperature, as those are the strongest indicators of whether the weather is pleasant or not. I assume wind speed is also important for flying safely, but I would inform Air Ambulance that we did not have enough data on wind speed to include it in training the models.