OPIM 5512 - Data Science using Python

Rain in Australia

Group 8

Deep Prakash Bhanushali, Genghui Cui, Yuxuan Qian, Jiannan Zhu

Table of Contents

Executive Summary

As machine learning and analytics continue to become more and more relevant in today's world, weather departments are constantly searching for ways to utilize data to get ahead, and using it for forecasting rainfall is no exception. Machine learning algorithms are being used not only to forecast rainfall, but also determine the magnitude with which features like Temperature, Evaporation, Sunshine and wind speed influence the chances of rainfall. With advancements in computing power, such as the utilization of super computers, weather forecasts have become more accurate than they were before.

In our project, we have made an effort to use the predictive power of machine learning algorithms to predict whether it will rain tomorrow or not in a particular Australian city. The data set that we used is the Rain in Australia dataset which contains about 10 years of daily weather observations from numerous Australian weather stations. Target variable is 'Rainfall_Status' which was obtained by recoding the 'Risk_MM' column to zeros and ones. We dropped the predictor variables like RainToday, RainTomorrow and RISK_MM which were acting as direct predictors and would have tampered the performance of our models.

The most appropriate technique to use for the purpose of this prediction was spot checking technique since our plan was to fit nine models and using any other technique would have made our programming work quite cumbersome. Based on the results obtained using spot checking, Random Forest Classifier turned out to be our best model on the basis of its accuracy. With the help of pipelining and hyperparameter tuning, we were able to optimize the performance of our best fitting model further and made pretty accurate predictions.

Introduction

The recent bushfire in Australia made the entire world realize how extreme weather can be and its capacity to bring an entire country to its toes. Looking at such situations makes us realise the importance of using machine learning to make accurate weather forecasts. This will help government agencies tremendously in making suitable arrangements and also make efforts to limit the damage that can be possibly caused.

Taking inspiration from the Bushfire event and the significance of forecasting in the weather department, we decided to utilize the predictive power of machine learning algorithms to forecast whether it will rain or not the day after in a particular Australian city. We believe that the model that we will build would not only provide us academic benefits but it will also provide valuable insights about how various predictors influence the weather of our surrounding areas. These things are pretty important if someone wishes to build his career in the field of data science and wants to work in the weather domain.

While working on this extensive complex dataset, the first step that we did was to clean the data and get rid of insignificant columns just by eyeballing the dataset. Then with the help of some graphs and plots (Figure 1), we got valuable insights about the data we are dealing with and future course of actions that will be needed from our side to work with the dataset. Polynomial features were added as a part of the feature engineering process and data was standardized with the help of z-score transformation to ensure that all the predictors are on the same scale and the dataset is ready to fit a model on.

Data Description

This dataset contained about 10 years of daily weather observations from numerous Australian weather stations. The target of the project was to predict whether it will rain tomorrow. The dataset has 142193 rows and 24 columns. We created a new column called Rainfall_Status as our target.

The data of Rainfall_Status from the column Risk_MM, if RISK_MM > 0 mm, then 1 (It will rain), if RISK_MM = 0 mm, then 0 (No rain). Then we deleted the columns RainToday, RainTomorrow, and RISK_MM. We excluded the variable RISK_MM because it was acting as a direct predictor.. If we did not exclude it, we would have leaked the answers to our model and reduced its predictability.

We also built a correlation matrix to get some information about our data variables. (Figure2). Since there was no hint of the presence of multicollinearity, we picked up all of them.

Then we checked the outliers of the numerical variables. We used the four columns as an example. (Figure 3) In the case of Australia, Rainfall is one of those features that tend to fall heavily when it rains. So We thought these outliers exist and decided not to drop these columns.

We found this new dataset about the latitude and longitude of Australia cities and combined it with our dataset for further analysis. We wanted to use the column called Location about the cities of Australia to build our model, so we chose the cities which were around the sea and not inland to reduce the other factors that may affect the accuracy of our model. (Figure 4)

Before adding polynomial features, we first made subsets for numeric and categorical variables because categorical variables could not be dealt with directly in the polynomial features. For the categorical variables, we used the one-hot encoding, which is a representation of categorical variables as binary vectors. For the numerical variables, first, we used the KNNImputer to deal with missing values. Each sample's missing values were imputed using the mean value from n_neighbors nearest neighbors found in the training set. Then we used the PolynomialFeatures to transform the existing features into higher degree features. Later, we used the Z-score Standardization to avoid features having a large numerical range that would play a major role when calculating metrics to improve model accuracy.

The last thing we did before building the model is to balance our data for modeling. We chose 25000 of each class to balance our data.

<u>Model Description and Results</u>

1.Spot Checking

Once the data was cleansed and all the preprocessing operations were performed, we splitted the data into 80% for training and 20% for validation. We then assessed the performance of 8 machine learning algorithms on the partitioned data by using the spot checking technique (Figure 5). The accuracy of GBC is 0.77, the accuracy of CART is 0.69, the accuracy of RFC is 0.77, the accuracy of LDA is 0.77, the accuracy of KNN is 0.74, the accuracy of NB is 0.73, the accuracy of ETC is 0.77 and the accuracy of Bagc is 0.75.

Based on the results obtained, we decided to use the RandomForestClassifier (RFC) algorithm to fit the model and make our predictions. We managed to get an accuracy score of 76.67% in the initial run which was quite good considering the size and complexity of our dataset (Figure 6). Once we selected our model, we decided to perform hyperparameter tuning operations on our model to fine tune it and increase its accuracy further to obtain better results.

2.Hyperparameter Tuning of RFC

Firstly, let us check the Parameters currently existing in RFC by default. Once we get to know the default parameters, we could then make the necessary changes in them to get optimum results. These are the parameters existing in RFC:

```
 'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 123,
 'verbose': 0,
 'warm_start': False
```

After taking a thorough look at all of them and understanding the significance of every parameter, we decided to perform tuning operation on picked six of them which we thought were the most important ones through experimental approach to achieve our goal of increasing the model efficiency namely:

```
'bootstrap': [True, False],
'max_depth': [30, 40, 50, 60, None],
```

'max_features': ['auto', 'sqrt'],

'min_samples_leaf': [1, 2, 3, 4],

'min_samples_split': [5, 6, 7, 8, 9],

'n_estimators': [820, 880, 940]

After utilising every combination of parameters possible using the grid search technique, we obtained valuable information about the parameters which had the ability to provide us the best results as well as overpowering the others. They were:

n_estimators=940

bootstrap=False

max_depth=60

max_features='auto'

min_samples_split=8

min_samples_leaf=2

Using these parameters, we fitted our RFC model and were able to increase the score by 0.18%. Now we can see that the accuracy of the new tuned RFC model is 76.85% (Figure 7).

3.Feature Importance:

After we got the new model, we were still curious about how the original variables would influence the target variable. So we decided to add Feature Importance analysis. We refrained from converting the categorical variables and also did not add polynomial features and Z-score transformation for carrying out feature importance analysis. This is because we wanted to see which original variables influenced our target the most without any feature engineering.

After that (figure 8), in terms of feature importance of the variables regarding the rainfall status of the next day, we found that the humidity of 3pm plays the most important role in the model fit, which was followed by sunshine and the rainfall status of the current day.

4.Something behind the modeling:

Actually, during our test, we figured out that if we do not use polynomial features and Z-score, the model performance will be better. And if we just dropped NAN values rather than KNNimputation, we will have a slightly better accuracy although the sample will shrink to 20K records from 50K records. This may imply that complication is not always better. Sometimes, a simpler operation may yield the best results. But in another way, we cannot be sure how the model will do as mentioned before without performing such complex operations and analysis.

(Figure 9-12)

Discussion

After using Spot Checking algorithms as well as the hyperparameter tuning method to compare and retrieve the best model and its parameters, we did not stop at this stage, but rather took a step backward and contemplated on our model performance. This is because we rebalanced and sampled our data before modeling to reduce the model training time and for parsimony, and we need to resample our dataset several times and observe if an overfitting problem still exists.

We ended up with around 50,000 rows in our current approach in the modeling part after data preprocessing, and that number may look intimidating and has contained lots of features and traits for prediction already, but that’s nothing as compared to the original 142,000 rows of raw data.

Therefore, we resampled and ran our modeling for another couple of times, in which we tried to add and drop certain data preprocessing steps to improve our results. Our methodology turned out to be an interesting experiment which once again relied on our judgements and domain knowledge rather than pure data science and machine learning techniques. We not only tested on 20,000 rows, 30,000 rows and 50,000 rows of data, but also have changed measures taken in dealing with missing values from KNN Imputer to simply exclude them. It is through this process we raised concerns when struggling with the order of data preprocessing and have abandoned our previous approach, in which we first converted the categorical variables to dummy variables and then ran feature engineering, resulting in over 1000 abundant features that made little sense. (Figure 9) in the Appendix showed our raw data as the baseline, our current approach and its performance as well as this other attempt where we sampled 20,000 rows, chose drop missing values instead of KNN Imputers, and got rid of the polynomials and standardization process. Surprisingly, with fewer data preprocessing techniques, our RandomForestClassifier model performed better than before, with an 80% accuracy as compared to 77%.

How come those effective and presumably textbook example-like data preprocessing steps actually made our model perform a little worse than without them? One explanation would be that our sample still had minor overfitting issues and happened to capture most of the patterns for prediction. Another argument would be that our original dataset was already large enough, so sampling and training were necessary steps, where the way we handled the missing values was identified as the main cause of the gap. Since the dataset was already big enough, KNN Imputer did not improve our data significantly whereas dropping those NAs has proven sufficient. Overall, our interpretation might be diversified from different perspectives, but the key lesson learned is that simplicity is always the best.

<u>Conclusions and Next Steps</u>

Our analysis of the Rain in Australia dataset started from data exploration and visualization on both numeric and categorical columns, where we found that not only numeric data such as wind speed and humidity were excellent predictor variables, categorical data like location and wind directions also had shown an impact on our target variable. Because of this, we decided to keep the categorical data and convert them into dummy variables, and continued our preprocessing steps to deal with missing values, performed feature engineering and standardization. Then we moved on to modeling and hyperparameter tuning, where we found our best model is RandomForestClassifier along with its best parameters like minimum sample splits should be 5. We resampled our dataset to test the possibility of overfitting, and we got similar accuracy scores, and for our approach discussed, we had a model performance of 77% in terms of forecasting. Through many rounds of trying different samples, we also discovered how data preprocessing has affected the results, and we believe the insights gained from this dataset could be used to better off future prediction.

Moreover, our findings brought us to the next steps, in which we identified the feature importance from the dataset for data deployment, and since weather forecasting in real life is often challenged by natural events occur on an hourly basis, an additional variable named HOUR is highly encouraged to be introduced in future predictions, where lag variables could also play a huge part in forecasting the rainfall status.

Last but not least, we should refocus our attention on the SEMMA approach, where we can adjust and refine the modification, modeling and assessment process iteratively and expect back and forths for changes to improve our model and predictions continuously.

References

Das, S. (2020). *Google Using Machine Learning For Weather Forecasting*. [online] Analytics India Magazine. Available at: https://analyticsindiamag.com/from-weather-forecasting-to-nowcasting-how-google-is-using-ml-to-predict-precipitation/ [Accessed 29 Apr. 2020].

Ganshin, A. (2019). *Advancing weather forecasting with machine learning*. [online] ITProPortal. Available at: https://www.itproportal.com/features/advancing-weather-forecasting-with-machine-learning/.

hackernoon.com. (n.d.). *What is One Hot Encoding? Why And When do you have to use it? | Hacker Noon*. [online] Available at: https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f [Accessed 29 Apr. 2020].

simplemaps.com. (n.d.). *Australia Cities Database | Simplemaps.com*. [online] Available at: https://simplemaps.com/data/au-cities [Accessed 29 Apr. 2020].

Koehrsen, W. (2018). *Hyperparameter Tuning the Random Forest in Python*. [online] Medium. Available at: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74.

Reilly Meinert (2019). *Optimizing Hyperparameters in Random Forest Classification*. [online] Medium. Available at: https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6.

Brownlee, J. (2019). *Tune Hyperparameters for Classification Machine Learning Algorithms*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/ [Accessed 29 Apr. 2020].

Appendix

Figure 1

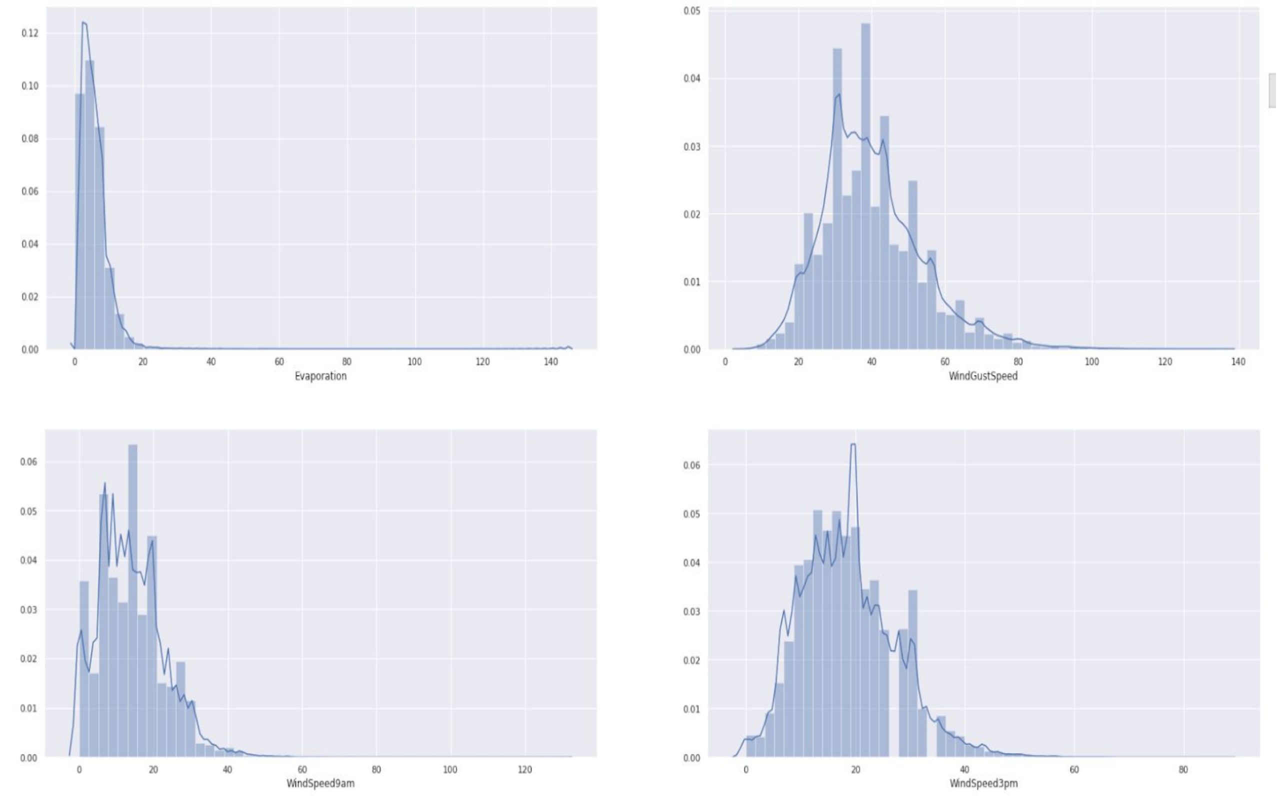


Figure 2

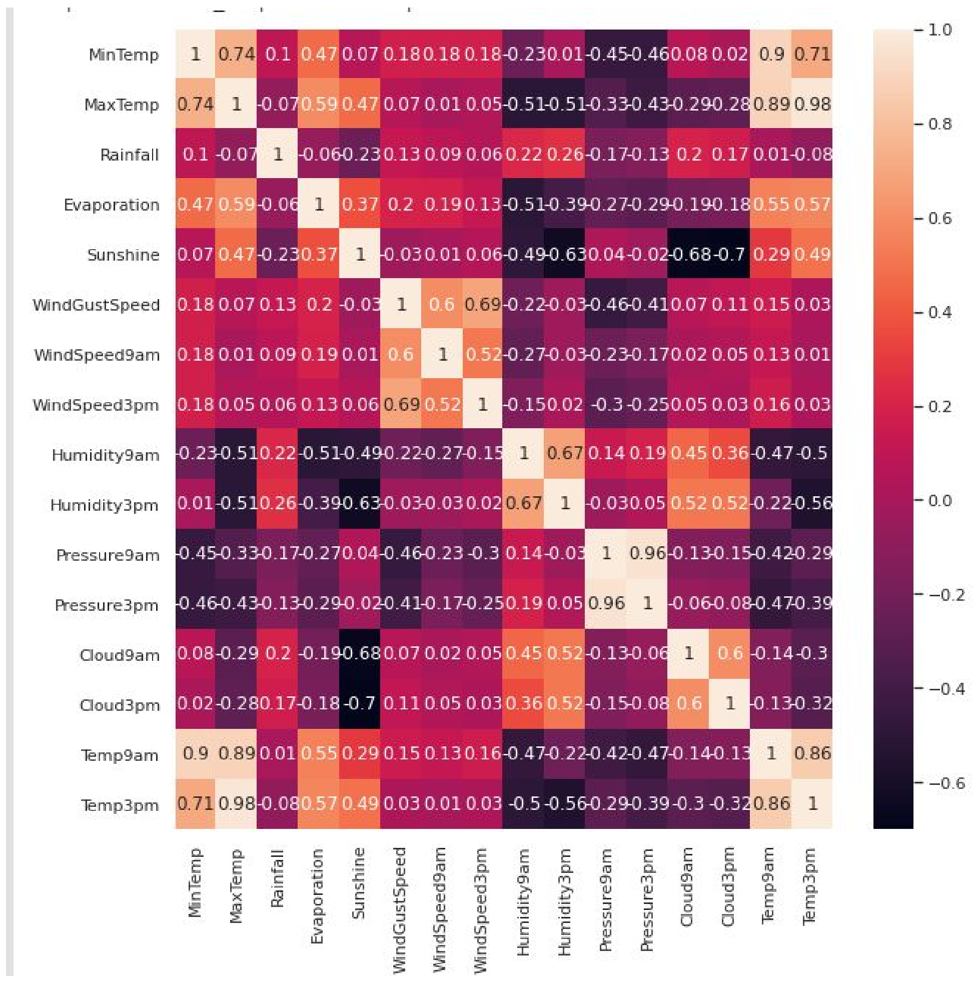
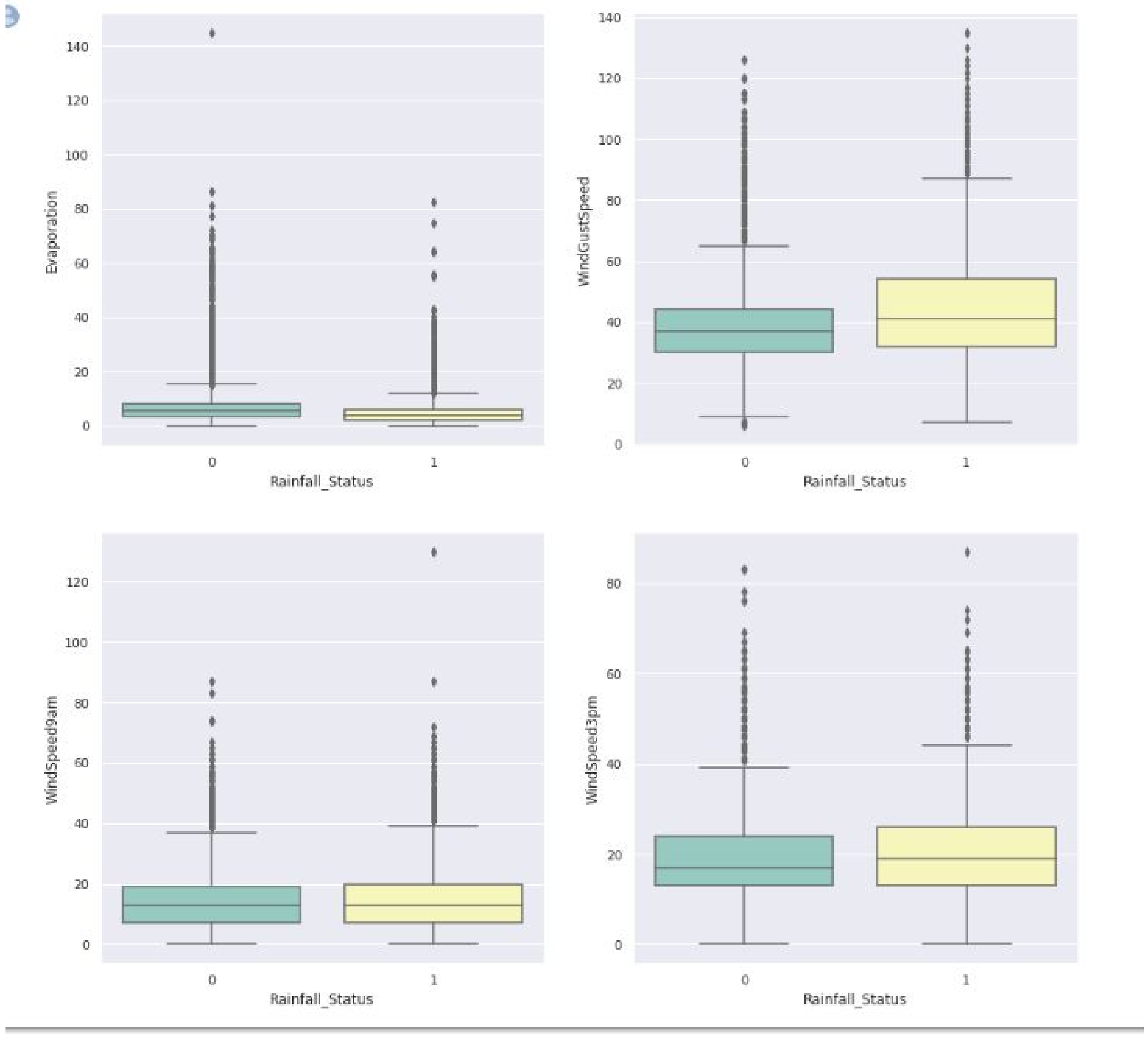

MinTemp
MaxTemp
Rainfall
Evaporation
Sunshine
WindGustSpeed
WindSpeed9am
WindSpeed3pm
Humidity9am
Humidity3pm
Pressure9am
Pressure3pm
Cloud9am
Cloud3pm
Temp9am
Temp3pm
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Figure 3



Evaporation
WindGustSpeed
WindSpeed9am
WindSpeed3pm
Rainfall_Status
0
1

Figure 4

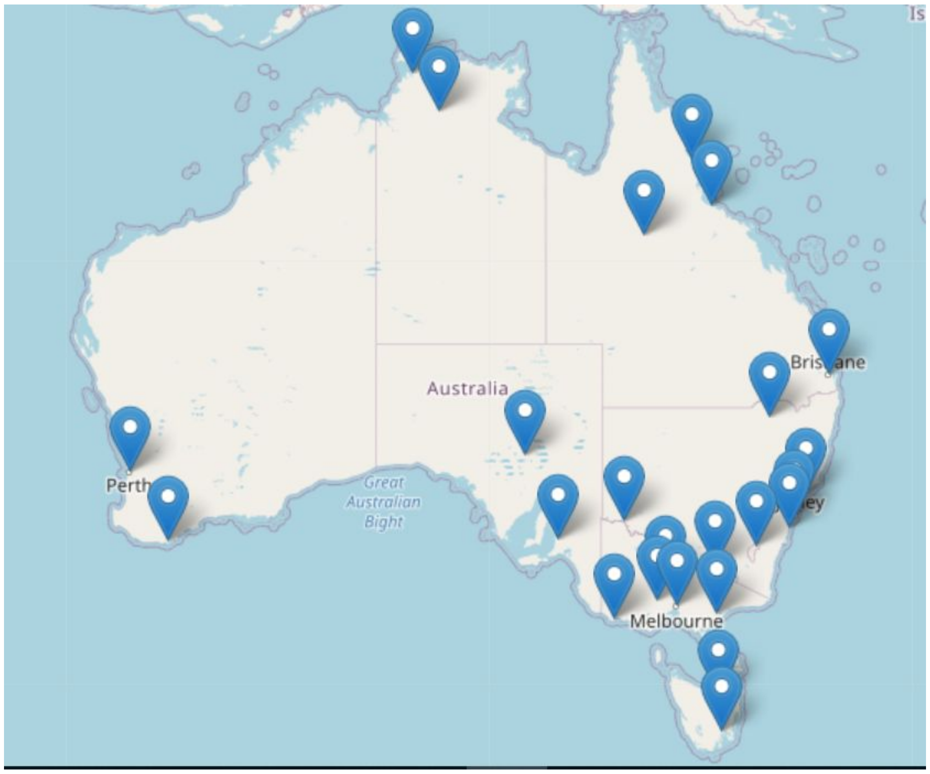
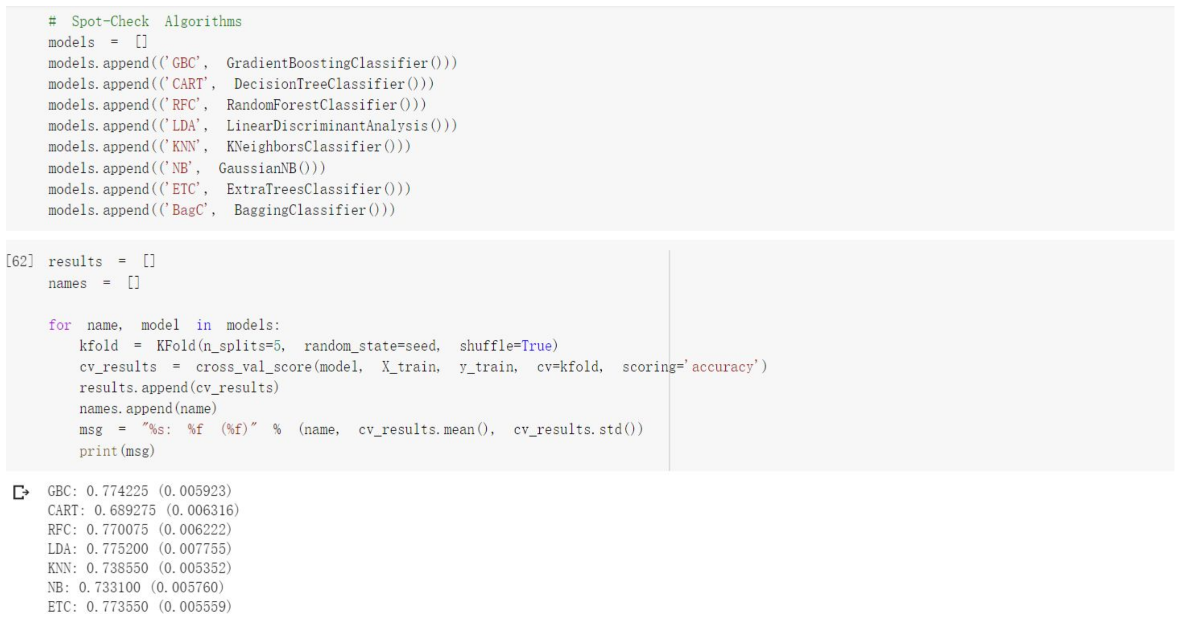


Figure 5

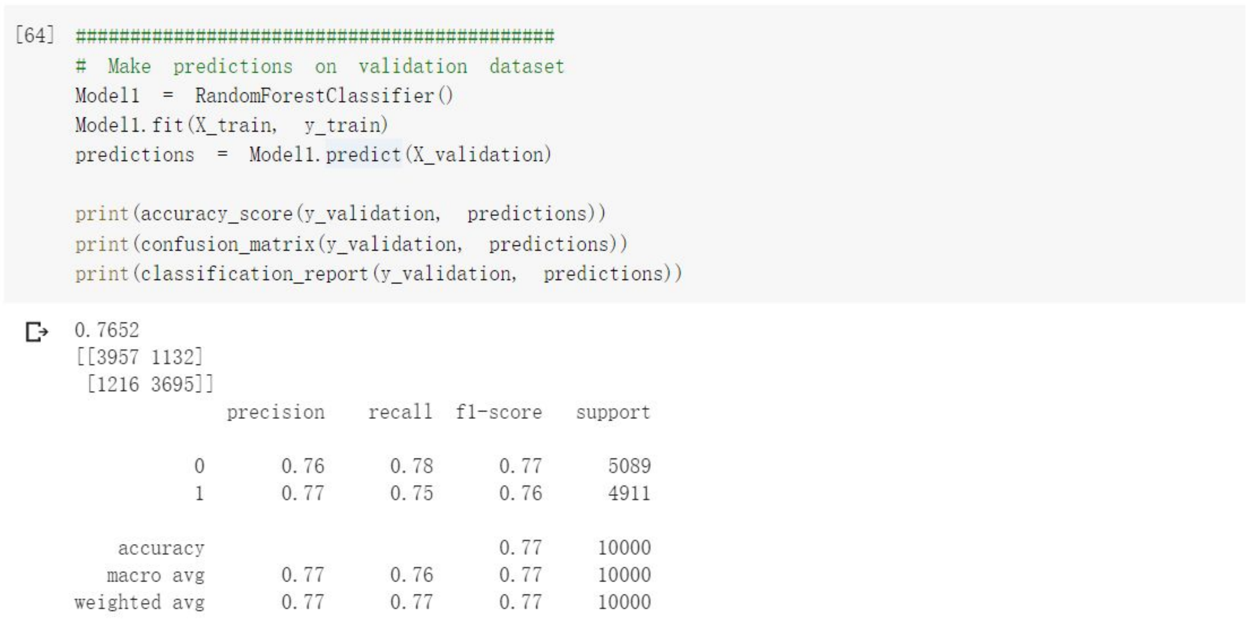


```
# Spot-Check Algorithms
models = []
models.append(('GBC', GradientBoostingClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('RFC', RandomForestClassifier()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
models.append(('ETC', ExtraTreesClassifier()))
models.append(('BagC', BaggingClassifier()))
```

```
[62] results = []
     names = []

     for name, model in models:
         kfold = KFold(n_splits=5, random_state=seed, shuffle=True)
         cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
         results.append(cv_results)
         names.append(name)
         msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
         print(msg)
```

```
GBC: 0.774225 (0.005923)
CART: 0.689275 (0.006316)
RFC: 0.770075 (0.006222)
LDA: 0.775200 (0.007755)
KNN: 0.738550 (0.005352)
NB: 0.733100 (0.005760)
ETC: 0.773550 (0.005559)
```

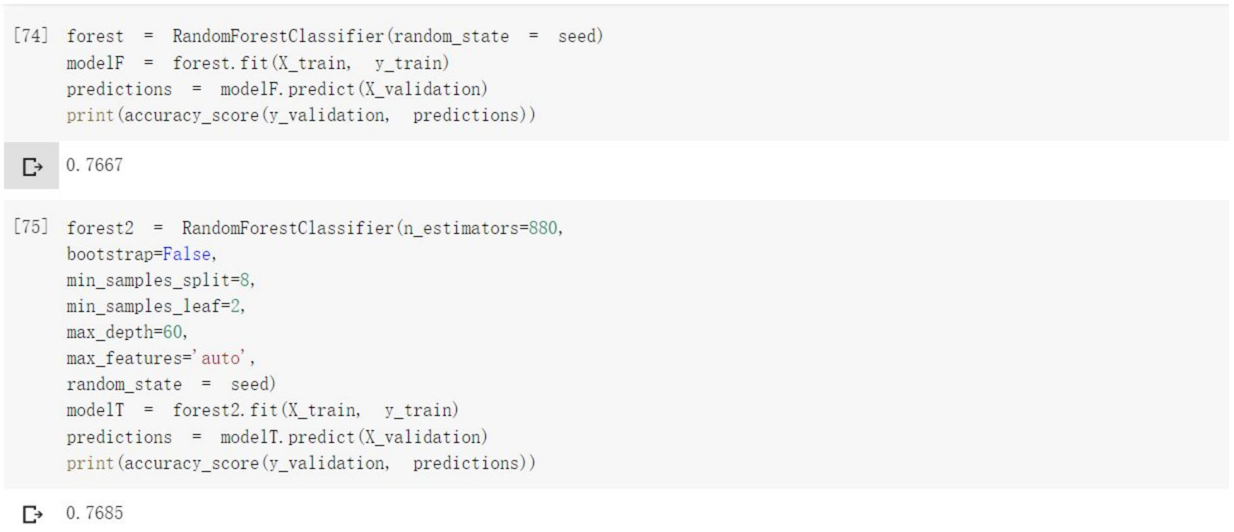
Figure 6



```
[64] ##############################################
     # Make predictions on validation dataset
     Model1 = RandomForestClassifier()
     Model1.fit(X_train, y_train)
     predictions = Model1.predict(X_validation)

     print(accuracy_score(y_validation, predictions))
     print(confusion_matrix(y_validation, predictions))
     print(classification_report(y_validation, predictions))
```

```
0.7652
[[3957 1132]
 [1216 3695]]
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      5089
           1       0.77      0.75      0.76      4911

    accuracy                           0.77     10000
   macro avg       0.77      0.76      0.77     10000
weighted avg       0.77      0.77      0.77     10000
```

Figure 7



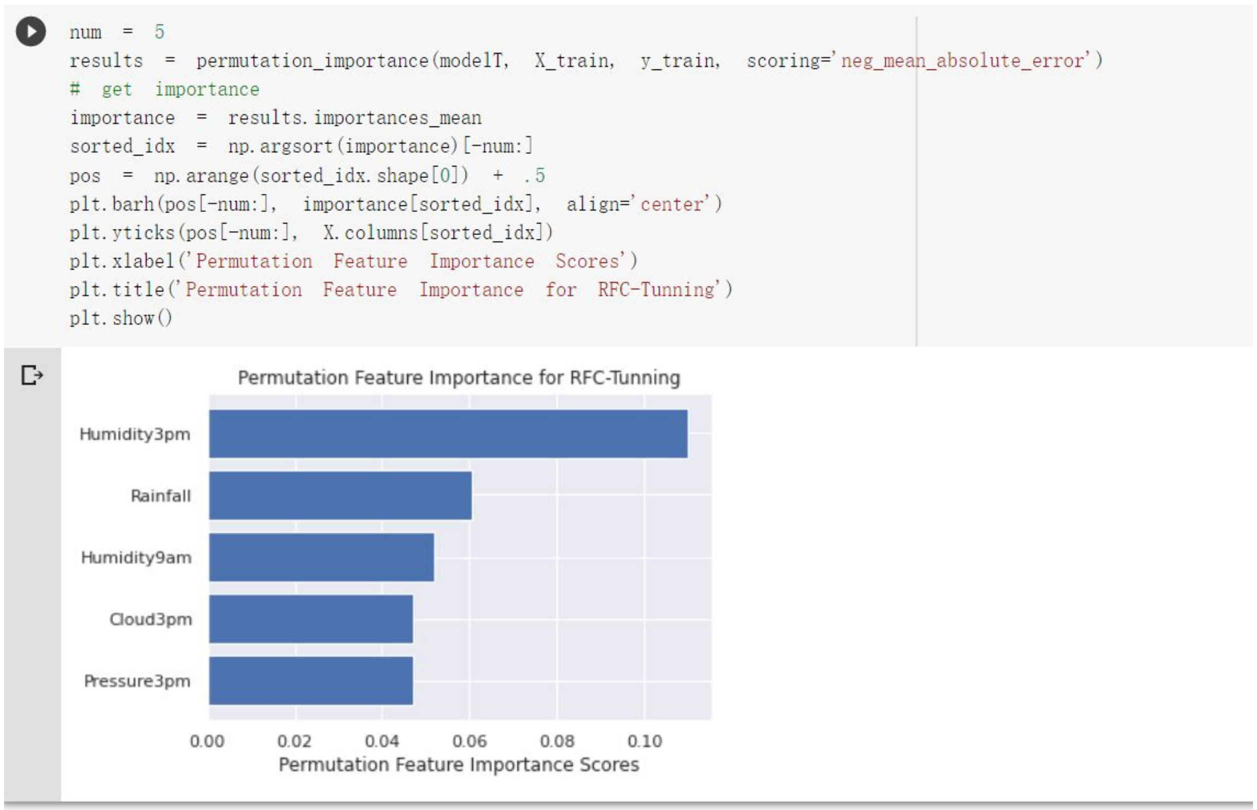
```
[74] forest = RandomForestClassifier(random_state = seed)
     modelF = forest.fit(X_train, y_train)
     predictions = modelF.predict(X_validation)
     print(accuracy_score(y_validation, predictions))
```

```
0.7667
```

```
[75] forest2 = RandomForestClassifier(n_estimators=880,
     bootstrap=False,
     min_samples_split=8,
     min_samples_leaf=2,
     max_depth=60,
     max_features='auto',
     random_state = seed)
     modelT = forest2.fit(X_train, y_train)
     predictions = modelT.predict(X_validation)
     print(accuracy_score(y_validation, predictions))
```

```
0.7685
```

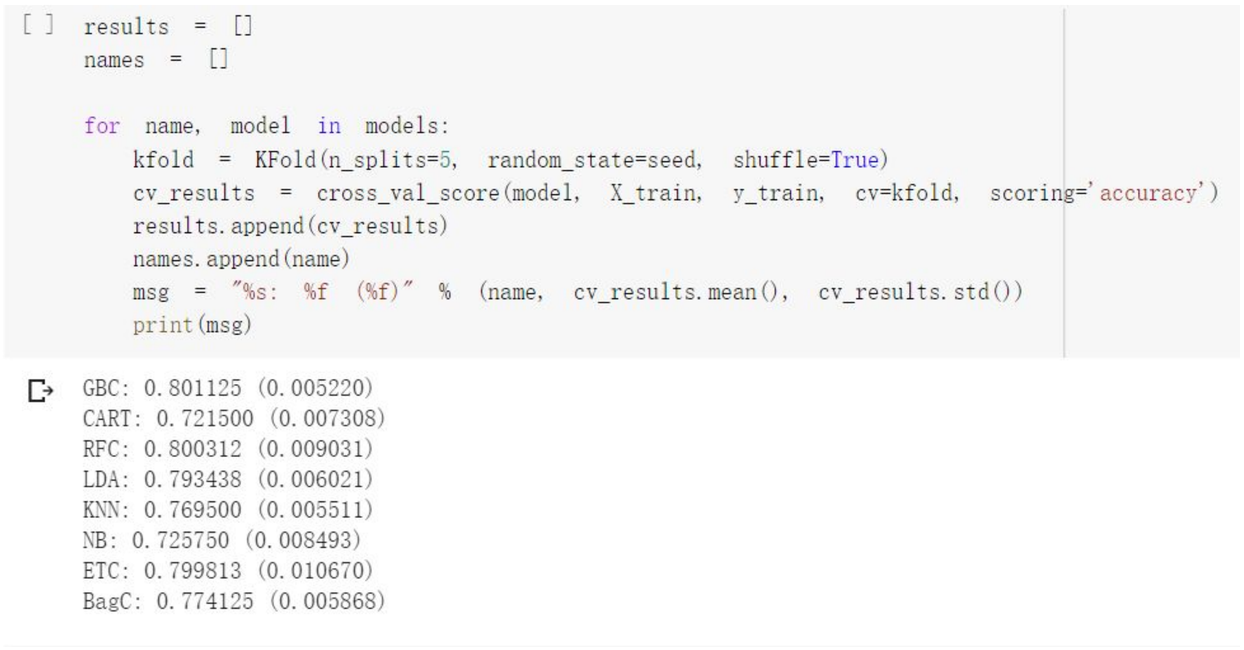
Figure 8



```
num = 5
results = permutation_importance(modelT, X_train, y_train, scoring='neg_mean_absolute_error')
# get importance
importance = results.importances_mean
sorted_idx = np.argsort(importance)[-num:]
pos = np.arange(sorted_idx.shape[0]) + .5
plt.barh(pos[-num:], importance[sorted_idx], align='center')
plt.yticks(pos[-num:], X.columns[sorted_idx])
plt.xlabel('Permutation Feature Importance Scores')
plt.title('Permutation Feature Importance for RFC-Tunning')
plt.show()
```

Permutation Feature Importance for RFC-Tunning
Humidity3pm
Rainfall
Humidity9am
Cloud3pm
Pressure3pm
0.00
0.02
0.04
0.06
0.08
0.10
Permutation Feature Importance Scores

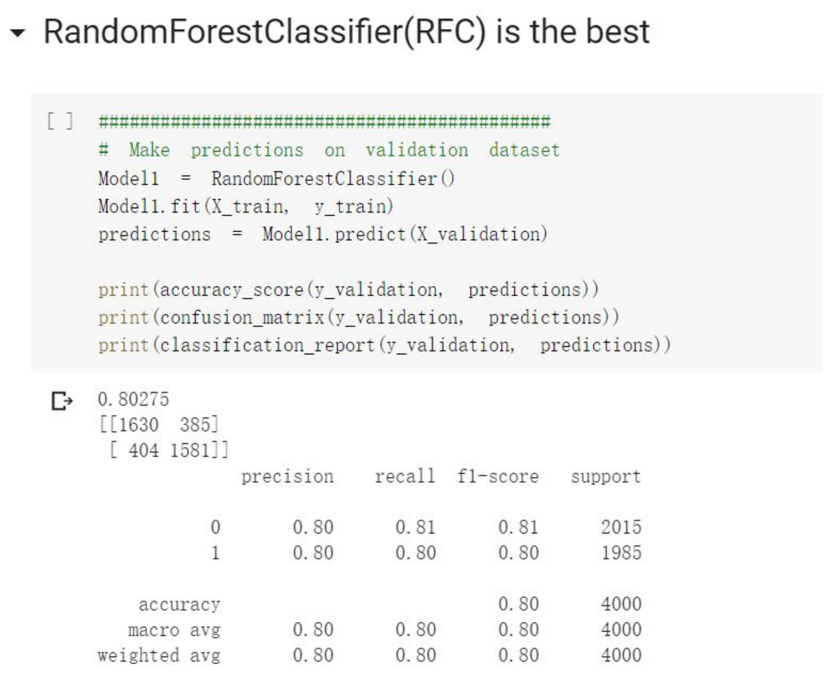
Figure 9



```
[ ] results = []
    names = []

    for name, model in models:
        kfold = KFold(n_splits=5, random_state=seed, shuffle=True)
        cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
```

```
GBC: 0.801125 (0.005220)
CART: 0.721500 (0.007308)
RFC: 0.800312 (0.009031)
LDA: 0.793438 (0.006021)
KNN: 0.769500 (0.005511)
NB: 0.725750 (0.008493)
ETC: 0.799813 (0.010670)
BagC: 0.774125 (0.005868)
```

Figure 10



RandomForestClassifier(RFC) is the best

```
[ ] ##############################################
    # Make predictions on validation dataset
    Model1 = RandomForestClassifier()
    Model1.fit(X_train, y_train)
    predictions = Model1.predict(X_validation)

    print(accuracy_score(y_validation, predictions))
    print(confusion_matrix(y_validation, predictions))
    print(classification_report(y_validation, predictions))
```

```
0.80275
[[1630  385]
 [ 404 1581]]
              precision    recall  f1-score   support

           0       0.80      0.81      0.81      2015
           1       0.80      0.80      0.80      1985

    accuracy                           0.80      4000
   macro avg       0.80      0.80      0.80      4000
weighted avg       0.80      0.80      0.80      4000
```

Figure 11

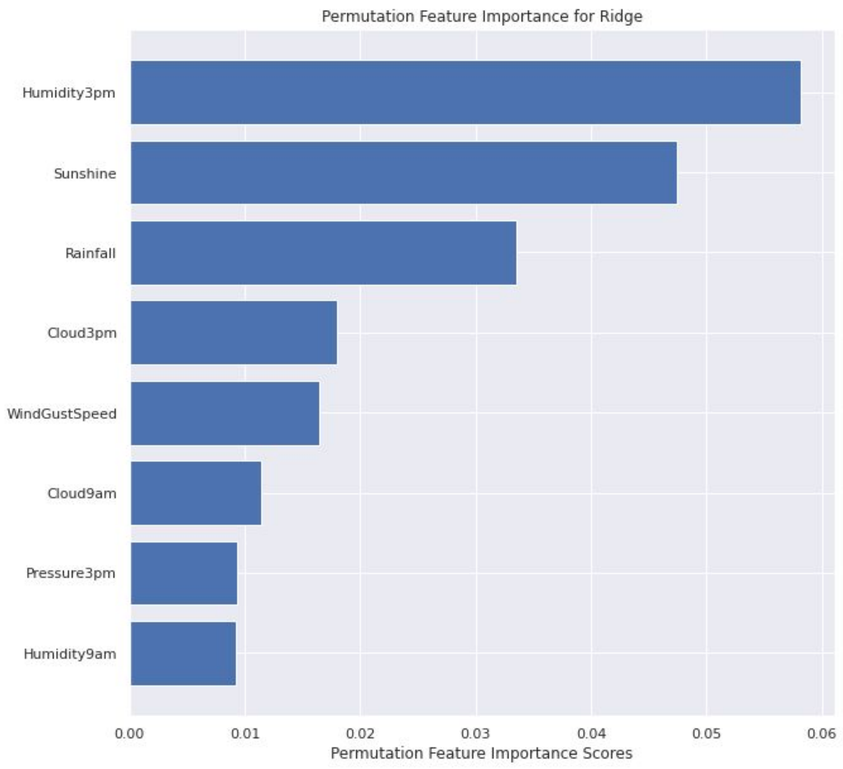


Figure 12

	Raw Data	**Our Approach**	**Best Performance**
Data Preprocessing			
Missing Values	✗	KNN Imputer	DropNA
Sampling	142k rows	50k rows	20k rows
Polynomials	✗	✓	✗
Z Score Std.	✗	✓	✗
Model Performance			
Model Name	NA	Random Forest Classifier	Random Forest Classifier
Accuracy Score	NA	77%	80%