Overview:

This assignment involves picking a Kaggle dataset, researching the existing work that has been carried out and develop solutions to work on the gaps that were identified.

1. Dataset description:

The dataset chosen was **Rain in Australia**, to predict whether it will rain tomorrow in Australia. It has 24 attributes with the target attribute 'raintomorrow' and has 142,194 instances totally.

https://www.kaggle.com/jsphyg/weather-dataset-rattle-package

2. Existing work:

- Initial data exploration, data cleaning steps were carried out.
- Removing null values, dropping columns that don't add much value to the target variable were performed.
- Outliers were detected and removed if they crossed the +/- 3.29 limit.
- Data were normalized by performing minmaxscaler() function on the dataset.
- The categorical attributes were converted into numerical data. Few contributors had split the categorical attributes as separate dummy variable columns using get_dummies() function.
- Two contributors had interestingly performed an univariate analysis for feature selection, to identify the best features that contribute to the target variable significantly, using SelectKBest method, and dropped the variables that didn't contribute significantly.
- Principal Component Analysis was done on the dataset and few attributes were removed by performing dimensionality reduction techniques.
- Exploratory Data Analysis was performed. Data was visualized using various plots, to find the underlying relationships between each of the dependent and independent variables.
- Data was split as training and testing sets to train the model and predict the target variable. A general split of 80/20 or 79/30 was followed.
- Machine learning models such as Logistic Regression, Decision Tree classifier, Neural Networks have been used.
- Ensemble model such as Random Forest classifier was used and parameters were tuned again to see if prediction got better.
- A general performance accuracy of 0.7 to 0.85% has been achieved by the models used by the contributors.
- The models were evaluated by using confusion matrix.

3. Gaps identified:

- During data preparation phase, many contributors had simply dropped all rows containing NA values, without attempting to impute the values of those attributes, either by replacing with mean or median or mode values.

  Instead of dropping the NA values, imputing their values with mean will allow us to make use of the other attributes in that instance. Also, replacing with the mean value gives a good guessing power of what this instance could be able to predict, instead of just removing the instance entirely because of the presence of NA.

- Correlation matrix was not used to check for collinearity.

  Multicollinearity is a phenomenon in which one or more predictor variables are very similar to each other, that it essentially means we are using the same variable in finding a relationship. Any two variables having a collinearity of more than 0.8 are said to be highly collinear.

- Cross validation was not performed on the training data set.
  Cross Validation is used to assess the predictive performance of the models and and to judge that the model doesn't take too much of noise from the data, resulting in overfitting of model.

- The model's accuracy was not evaluated based on confusion matrix, precision, recall measures and f1-score. Only accuracy_score was used.

4. Solution:

Data pre-processing:

The target variable is the raintomorrow variable. We are going to classify it is going to rain tomorrow or not, based on the independent variables.

Dropping variables:

```
rain = rain.drop(columns=['sunshine', 'cloud9am', 'cloud3pm', 'date','location','evaporation','windgustdir',
                          'winddir3pm', 'winddir9am'])
```

The above columns were dropped initially.

The columns cloud9am, cloud3pm, sunshine and evaporation were dropped on the basis that there were extremely large number of NA values.

The columns date, location, windgustdir, winddir3pm, winddir9am were dropped mainly because they were categorical string type columns, which doesn't necessarily add value to the independent variable.

Recoding:

```
rain['raintoday'].replace({'No': 0, 'Yes': 1},inplace = True)
rain['raintomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)
```

The variables raintoday and raintomorrow were recoded as No = 0 and Yes = 1, indicating it didn't rain and it rained, respectively.

Imputing variables:

```
rain.groupby('raintoday', as_index=False)['windspeed9am'].mean()
```

	raintoday	windspeed9am
0	0.0	13.503050
1	1.0	15.684721

```
rain['windspeed9am'][rain.raintoday == 1 ] = rain['windspeed9am'].fillna(15.7)

rain['windspeed9am'][rain.raintoday == 0 ] = rain['windspeed9am'].fillna(13.5)
```

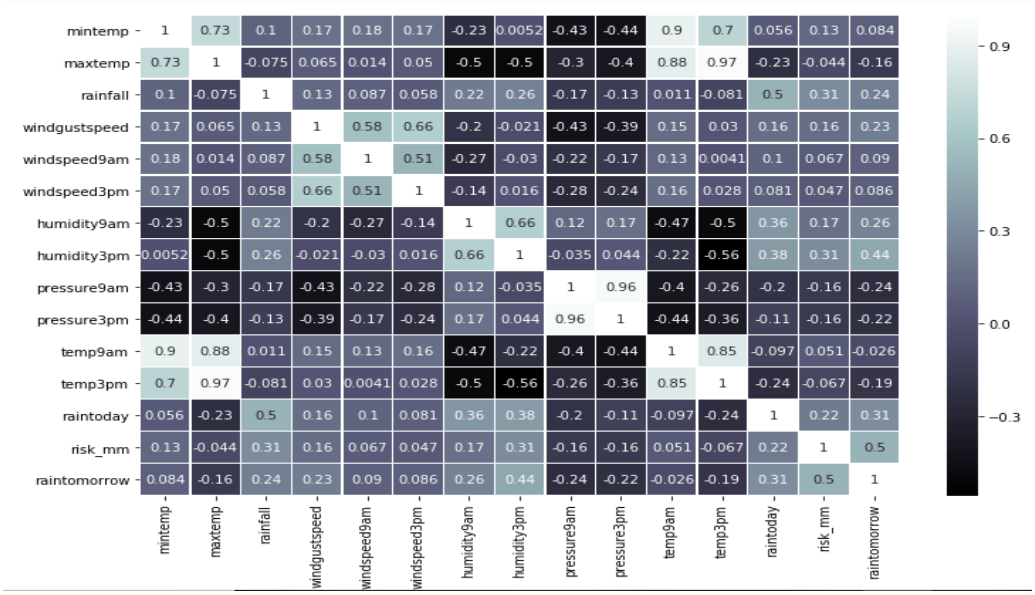
To impute the NA values of variables, I performed a groupby on the raintoday variable, to find the different means of windspeed9am variable based on the raintoday variable.

Then I filled the instances having NA with the mean found in groupby function for raintoday. For example, missing windspeed9am instances that have raintoday = 0, will be replaced with the mean value of raintoday = 0, as found in the previous step.

This step was repeated for all of the variables that had missing NA values, to find the separate means based on the value of raintoday variable.

A final round of dropping NA values was performed, for not being able to impute with values beyond this.

Correlation Matrix:



High correlation can be found between the variables mintemp and temp9am.

High correlation can be found between the variables maxtemp and temp9am & temp3pm.

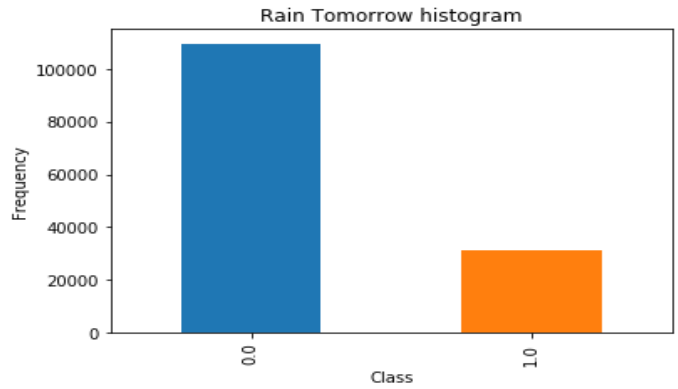
Thus, I have removed the variables temp9am and temp3pm, so that we don't have redundant information in our dataset.

Scaling:

```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
scaler.fit(rain)
rain = pd.DataFrame(scaler.transform(rain), index=rain.index, columns=rain.columns)
```

Transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one. This transformation is often used as an alternative to zero mean, unit variance scaling.

Histogram:



Although it isn't an even distribution, the class 'raintomorrow' instance is definitely at a good range for our models to learn the distinction between the two instances.

Splitting the dataset:

```
X_train, X_test, y_train, y_test = train_test_split(rain.drop('raintomorrow',axis=1), rain['raintomorrow'],test_size=0.2)
```

The data is split into training and testing sets in a 80:20 ratio

5. Local Evaluation and Data Modelling:

Used the following four models to compare between Random Forest Classifier, Decision Tree, KNearest Neighbor and Naïve Bayes.

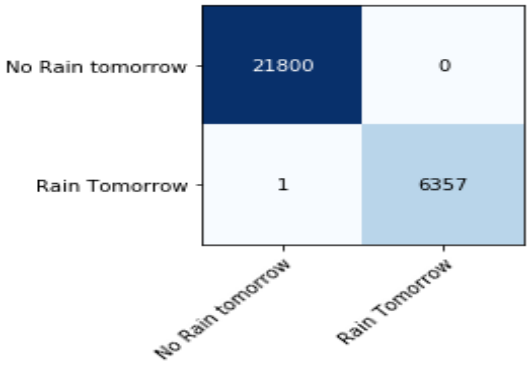
Results for Random Forest:

Random Forest is a supervised learning algorithm. It creates a forest, which is actually an ensemble of decision trees and makes it somehow random, which is most of the time trained with the "bagging" method. The general idea is that the bagging method uses a combination of models to learn and it increases the overall result. It only uses a random subset of the features to be considered for the algorithm while splitting the nodes.

```
Random Forest : 0.9999644861140706
K fold result:  [1. 1. 1. 1. 1.] Mean Score:  0.9999556064601339
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     21800
         1.0       1.00      1.00      1.00      6358

avg / total       1.00      1.00      1.00     28158
```



As we can see, Random Forest performs really well for this dataset. An accuracy of 0.99 has been achieved. Only one instance, which is actually raintomorrow has been wrongly classified as noraintomorrow. This shows us that, the instance that we wish to identify mainly, which is 'raintomorrow' is almost 1/3rd of the 'noraintomorrow' instance and it is not really an imbalanced dataset, which could be one of the main reasons this model has performed so well. Even during cross validation, every single k-instance has achieved the same result.

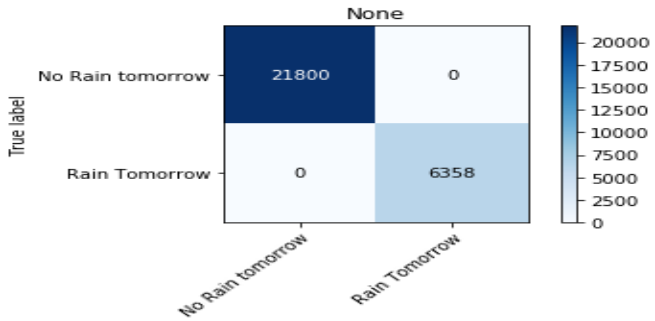
Results for Decision Tree:

A decision tree as the name suggests can be thought of as having an upside down version of a tree, with the root node at the top and has a deciding factor (condition/internal node) to split data based on an attribute and splits the data into branches or edges. The end of the branch that doesn't split anymore is the decision/leaf. Using a tree involves in deciding on which attributes to choose and what conditions to use for splitting the data and also to know when to stop.

```
Decision Tree: 1.0
K fold result:  [1. 1. 1. 1. 1.] Mean Score:  1.0
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     21800
         1.0       1.00      1.00      1.00      6358

avg / total       1.00      1.00      1.00     28158
```



As we can see, Decision Tree exceeds expectations by giving us an 100% accuracy and for all of the k-cross validation instances as well! It could be the same reason as to how Random Forest performed very well, as the instances are not really imbalanced.

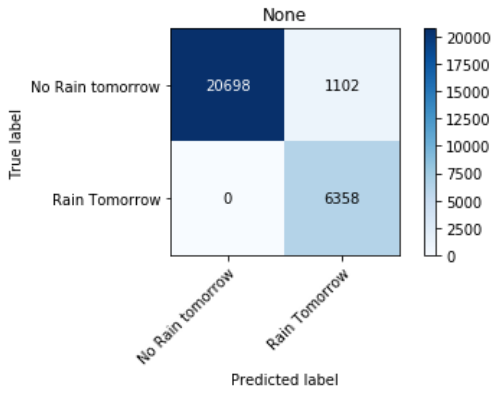
Results for Naïve Bayes:

Naive Bayes classifiers assume strong, or naive, independence between attributes of data points (i.e) The algorithm assumes that the predictor features are not related to each other, the presence of a particular feature doesn't affect the other. The main goal of the NB algorithm, is to find the conditional probability of an attribute based on prior probability.

```
Naive Bayes : 0.9608636977058029
K fold result:  [0.96 0.96 0.96 0.96 0.96] Mean Score:  0.9590602705193151
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      0.95      0.97     21800
         1.0       0.85      1.00      0.92      6358

avg / total       0.97      0.96      0.96     28158
```



Naïve-Bayes is also doing well with an accuracy of 0.96. Only the 'noraintomorrow' instance has been classified wrongly.

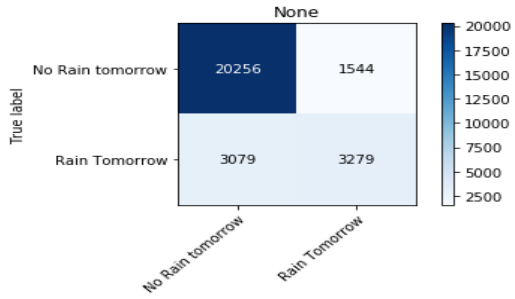
Results for KNN:

KNN usually estimates how likely an instance is likely to be a member of one group or another group, depending on what group the instances nearest to it belong to. It is an example of a "lazy learner" algorithm (i.e) it does not build a model, until a general query of the data set is performed. The only calculations it makes are when it is asked to poll the data point's neighbors.

```
KNN : 0.8358193053483912
K fold result:  [0.84 0.84 0.83 0.84 0.84] Mean Score:  0.8368981372911982
Classification report:
              precision    recall  f1-score   support

        0.0       0.87      0.93      0.90     21800
        1.0       0.68      0.52      0.59      6358

avg / total       0.83      0.84      0.83     28158
```



KNN is not performing well, compared to the accuracy we saw with Random Forest, Decision Tree and Naïve Bayes. Here both the instances are classified wrongly.

6. Reflection on results:

Based on the results, we can see that the Decision Tree classifier outperforms everything else with an accuracy of 100%. Achieving 100% accuracy is a miracle as mostly what we expect is a general accuracy around 80-85%. The other contributors' models in Kaggle, were able to predict correctly only with an accuracy around 80-90%.

Based on the gaps that were found, the steps that were taken to work on those gaps, has worked out well for three of the classifiers that were used. Having used cross-validation, we can be certain that the models does really perform well and doesn't have a problem of overfitting the data.

We can see that, if our data is pre-processed in the right way, by means of imputing the variables, checking for outliers, scaling the variables, performing a collinearity check and removing variables that do not contribute significantly to our independent variable, our model will be able to achieve excellent results.

As something that we always see in most of the machine learning guides and tutorials, we have to devote almost 80% of our time to perform the data cleaning and data pre-processing part. Once we have the clean data, it is a cakewalk from there on, to build and train models and evaluating them.

Having a dataset that has a good number of instances of the class we really intend to find out, also helps in the model to learn to predict the positive instances correctly, as was the case with this dataset.