International Institute of Information Technology Bangalore

Machine Learning

AI-511

Assignment - 1 (Weather Classification)

Team Members:

Achintya Harsha (IMT2021525)

Adithya Nagaraja Somasalle (IMT2021054)

Devendra Rishi Nelapati (IMT2021076)

November 16, 2023

Contents

1 Exploratory Data Analysis (EDA)

1.1 Dealing with Null Values

First we look for null values in the dataframe. Using the library missingno, we can visualise the percentage of missing values in each column.

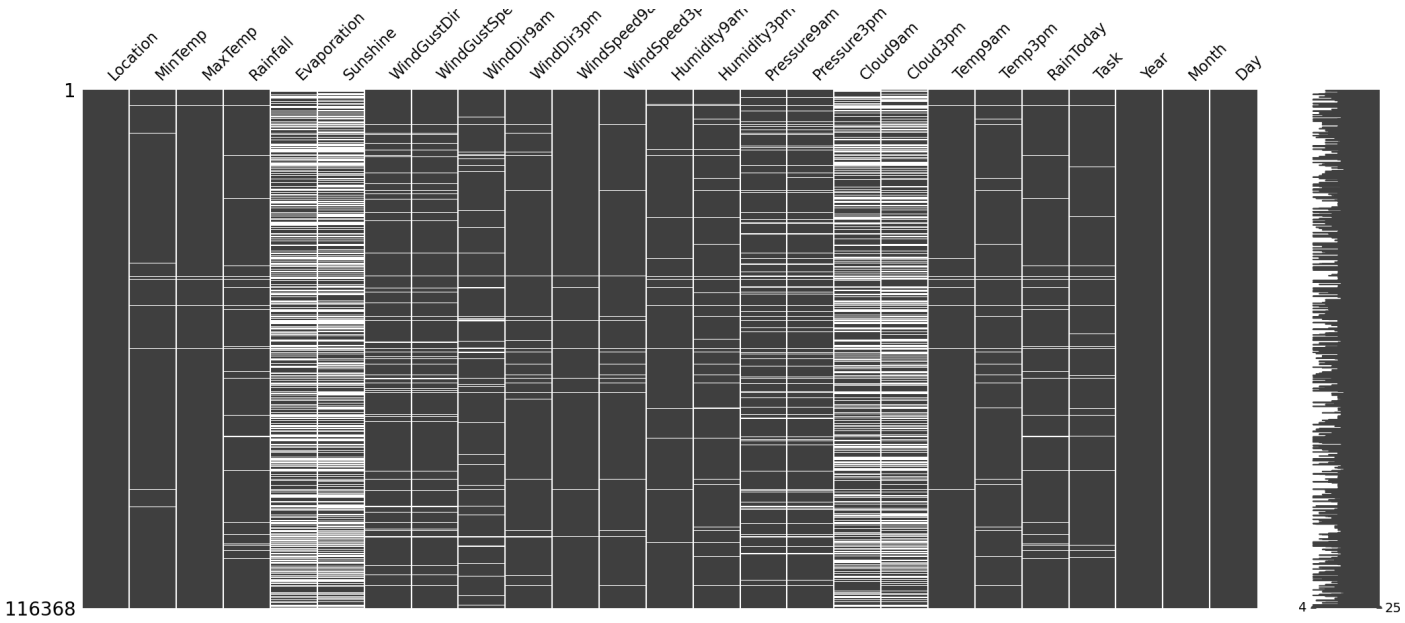

Figure 1: *Missing values in Dataframe*

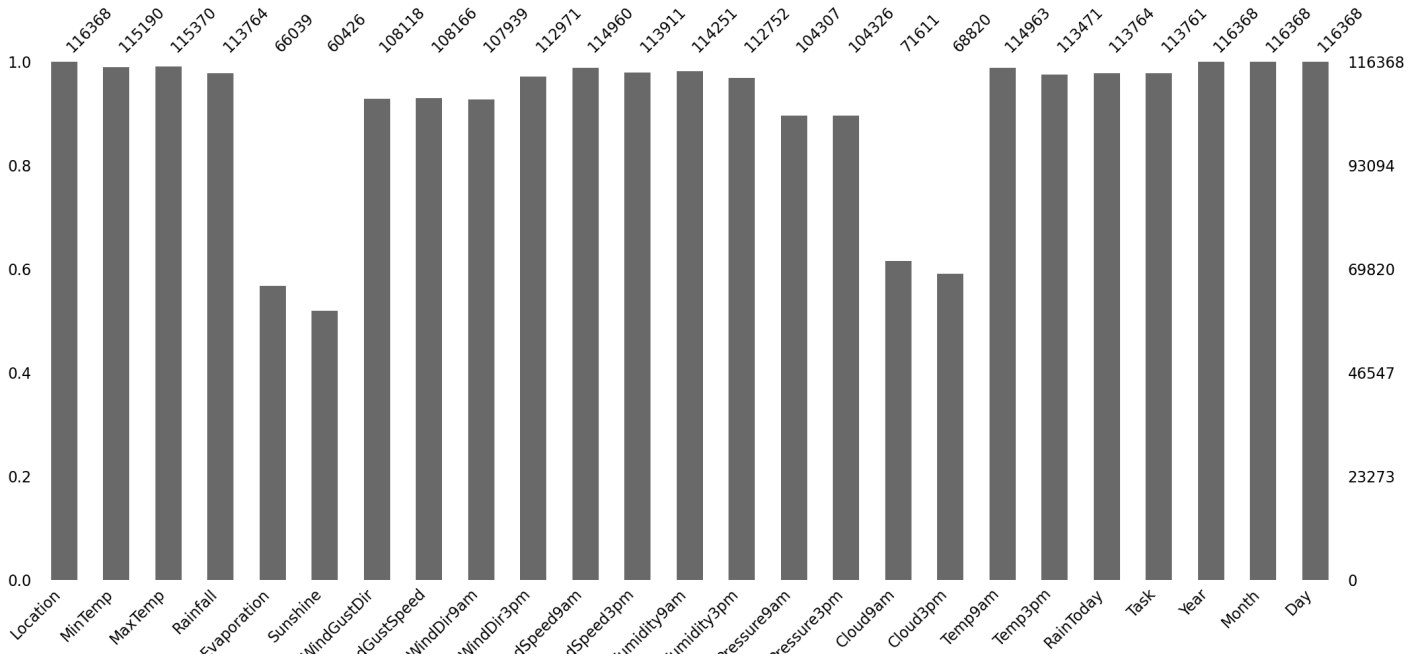

Figure 2: *Percentage of Missing values in Dataframe*

From the graphs we observe that columns Sunshine, Evaporation, Cloud9am, Cloud3pm has large percent of null values. Among these, Sunshine and Evaporation have null percentage nearly equal to

50%.

So we drop the respective columns from the data-set such a high percent of null values can cause inaccuracy in the prediction model.

1.2 Distribution of Features

Here we will probe into the distribution of the values of different features. We will look at the mean and variance of a particular feature and the probability of the occurrence of a particular value.

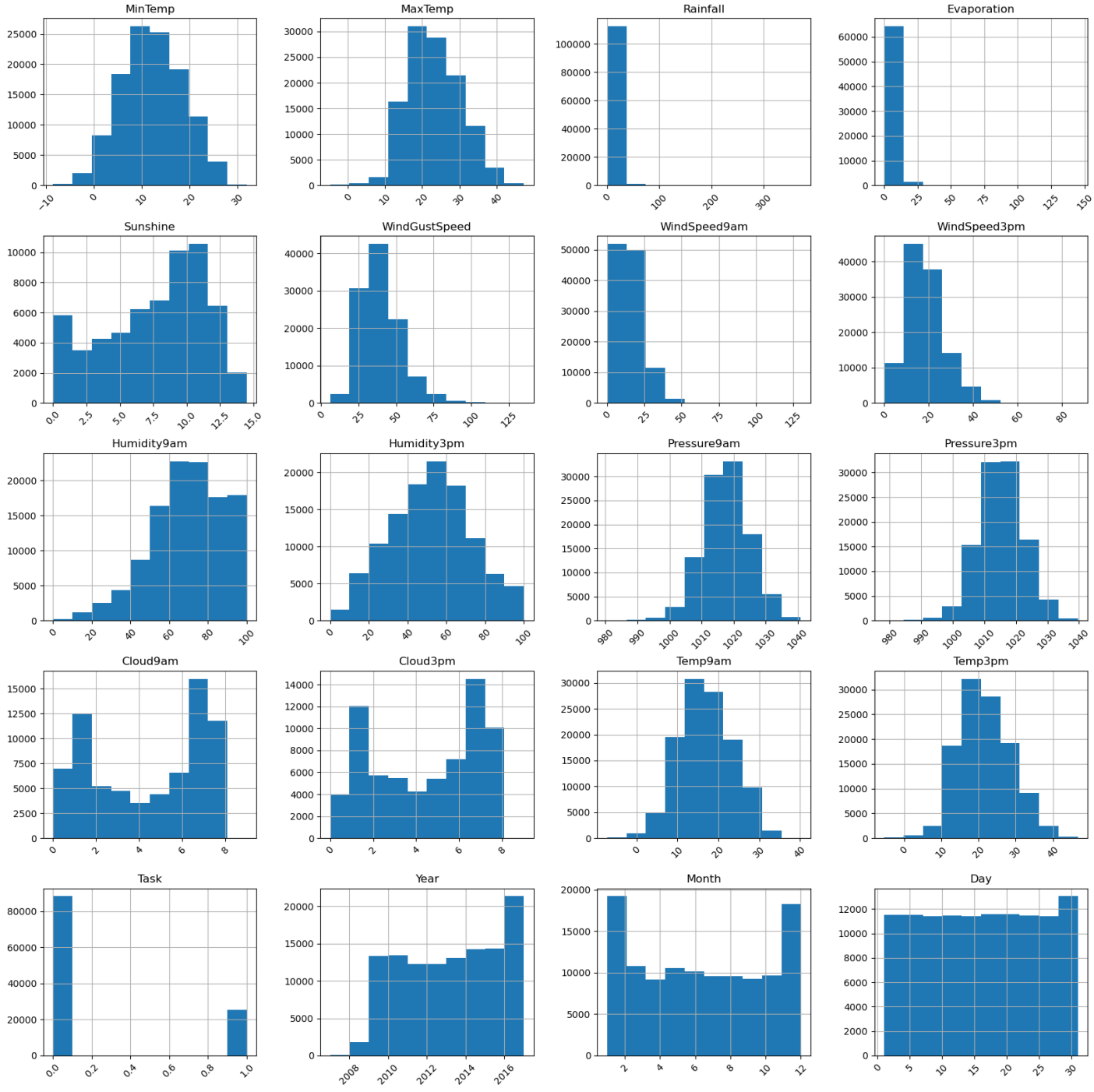

Figure 3: *Histogram showing the mean and spread of numerical features*

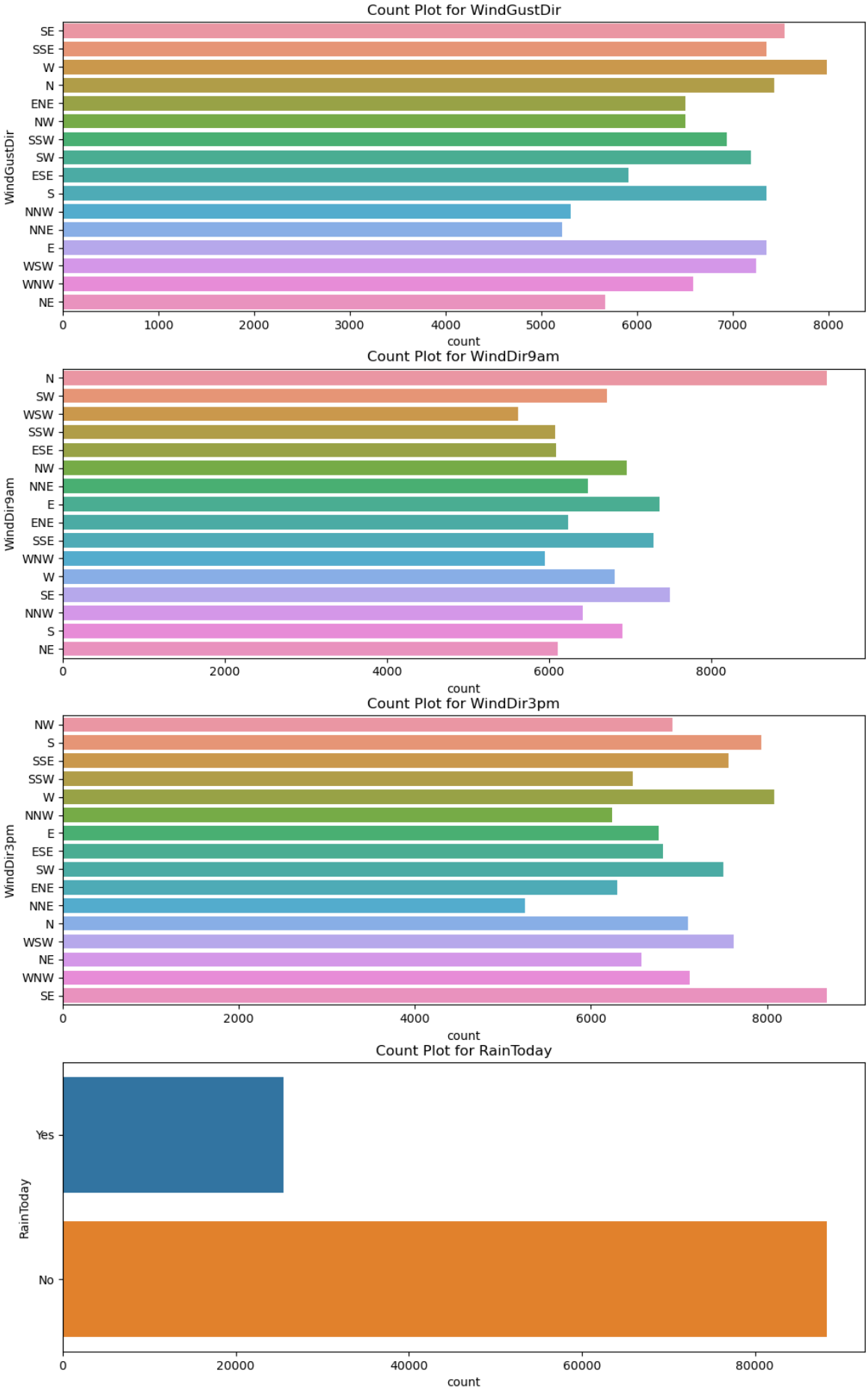

Figure 4: *Count plot showing the mean and spread of categorical features*

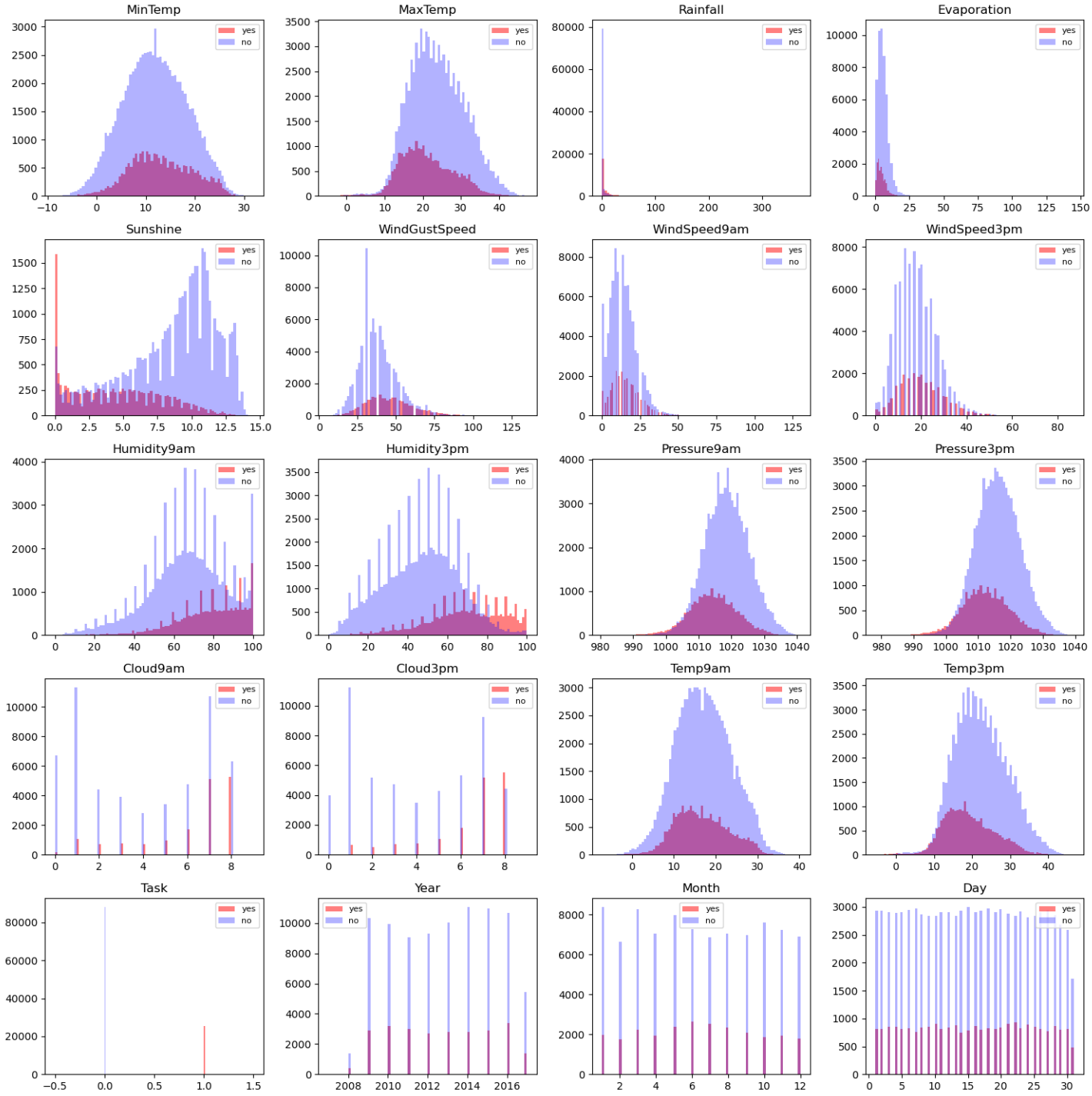

Figure 5: *Comparison of Characteristics for Task Values 0 and 1*

From the above graphs we can remove the features that have low variance. Also if we look that Figure5, some features like 'MinTemp', 'MaxTemp', 'Day' have similar mean and variance for cases where Task is 0 and 1. So they are not really useful for prediction.

1.3 Correlation

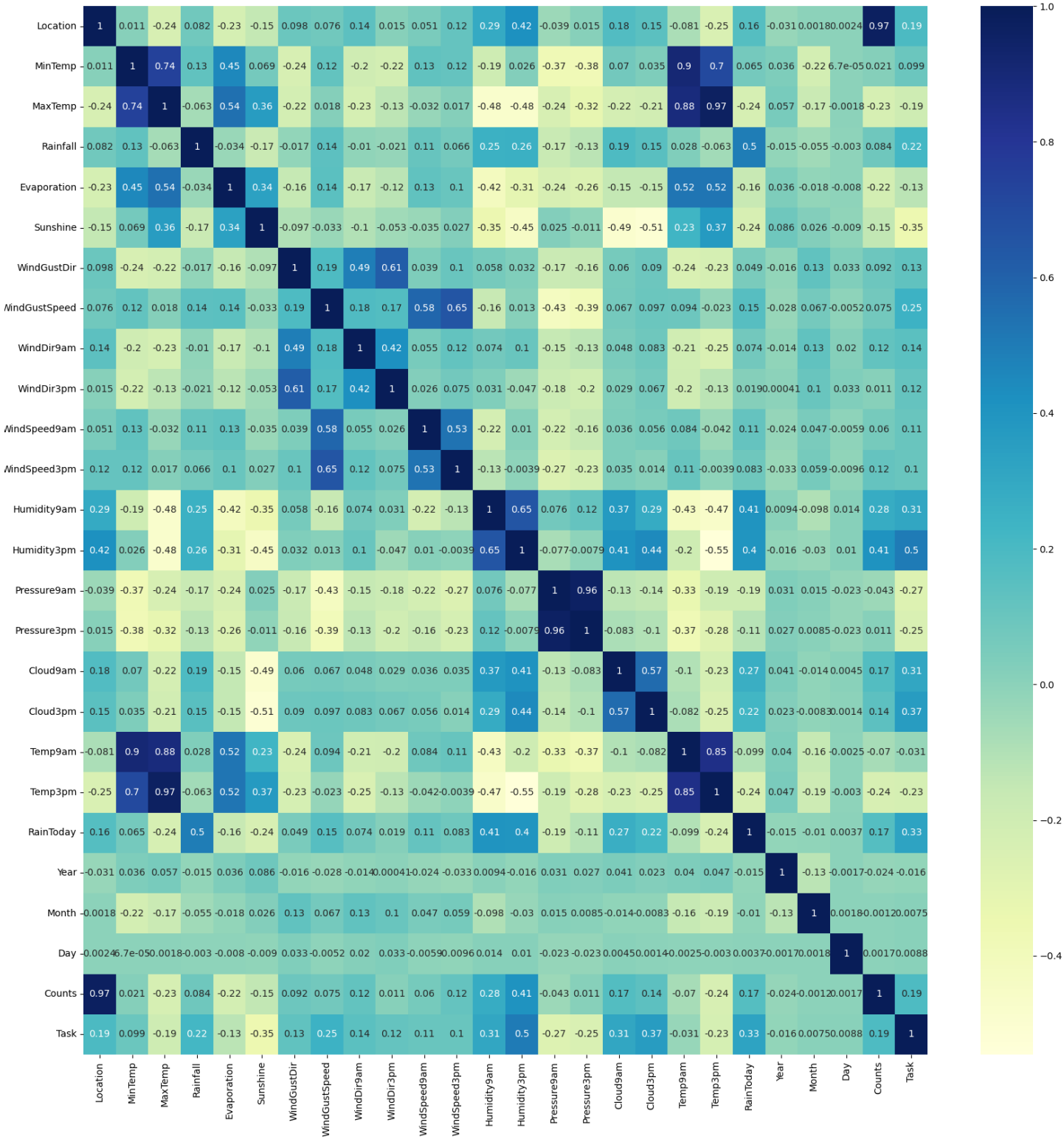

Figure 6: *Correlation heat map of every feature w.r.t. each other*

Features that are poorly correlated with Task(correlation close to 0) are removed as they do not contribute to improving accuracy. Example: 'Day' feature is removed.

1.4 Outlier Detection and removal

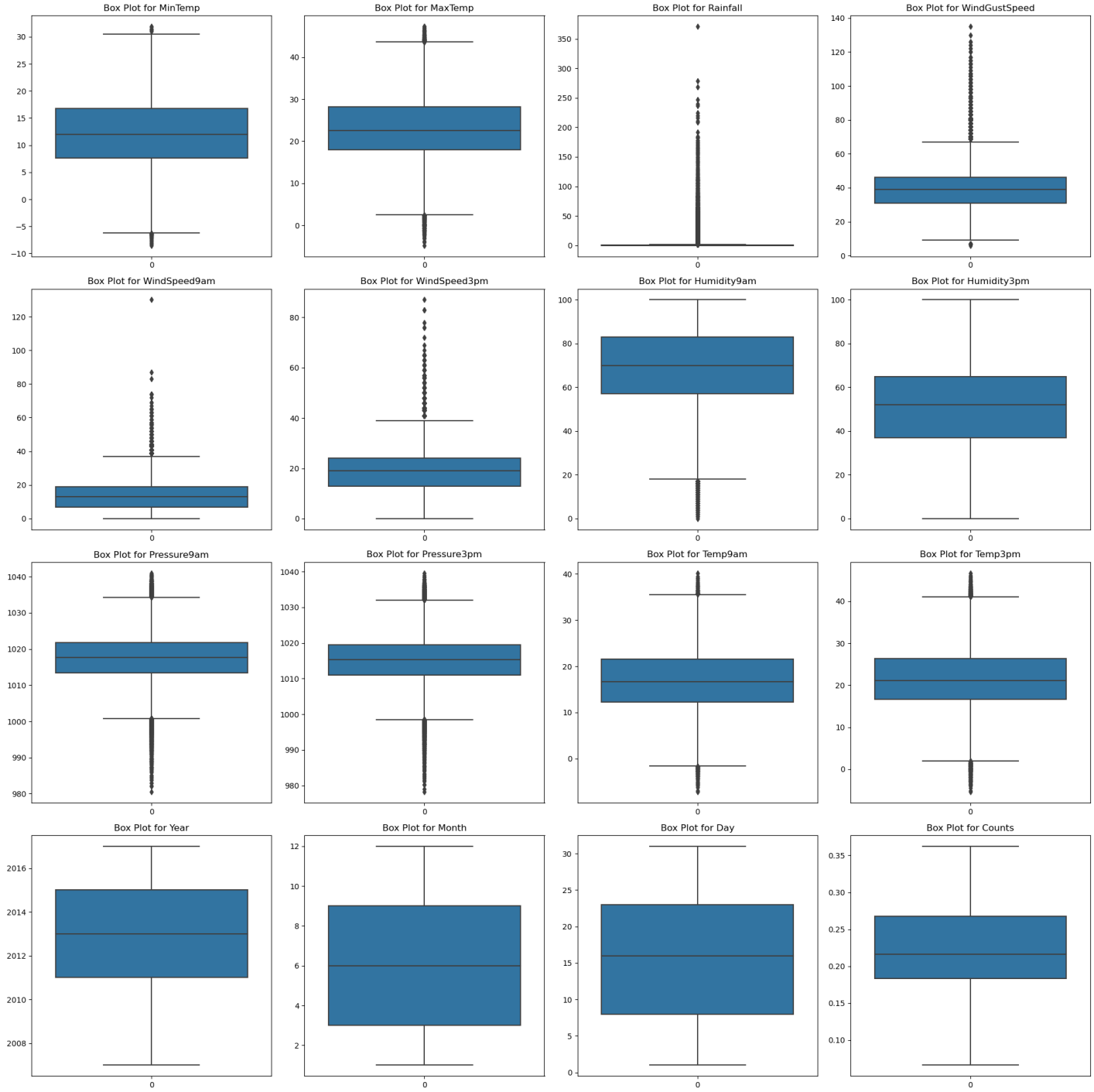

Figure 7: *Outliers plotted using boxplot*

In our analysis, we identified and addressed outliers in the dataset. We either removed them to improve data quality and, potentially, enhance model performance.

2 Pre-processing of Data

2.1 General Data Preprocessing

We began by examining the provided dataset, consisting of test and train CSV files. One of our initial steps involved breaking down the date column into more categorical columns, such as date, month, and year. This action was taken to manage the potential issue of the date column having a large number of unique values. Consequently, we simplified it into 31 days, 12 months, and several years.

```
len(pd.unique(train_df['Date']))

3414
```

Figure 8: *Number of unique date values without splitting is very large*

We also removed rows with null values in columns where the null value percentage was less than 5%. These rows were considered inconsequential as the columns contained sufficient data for training our model.

2.2 Location-Based Priority Probability Calculation

Subsequently, we conducted priori probability calculations based on location. This involved dividing the count of each location value by its location coin value when the Task value of that particular row equaled 1. This calculation provided us with the probability that the task value would be 1 at a given location. Also total number of occurrences of a particular location was calculated. Using this we calculated the probability of target being 1 given the place.The resulting probability values were then added as a new column(Counts) to the data set.

$$trainDF[Counts] = P(Task = 1|Location)$$

2.3 Seasonal Data Categorization

Given that the task in question is 'Humidity,' it's a logical assumption that humidity is influenced by the season. Thus, we organized the data frame into four sub-dataframes, categorized by seasons: Spring_df (March, April, May), Summer_df (June, July, August), Fall_df (September, October, November), and Winter_df (December, January, February).

For data imputation, we addressed the missing values within the respective sub-dataframes. For numerical data, we replaced missing values with the mean of the column values for that particular sub-data frame, while for categorical data, we used the mode of the sub-data frame.

2.4 Upsampling and Downsampling

Another noteworthy observation was the substantial disparity in the number of rows with task value 0 compared to those with task value 1. To address this, we employed a combination of up-sampling and

down-sampling. We divided the dataframe into two sub-dataframes based on the task value, then found the midpoint of the total rows. We up-sampled the task=1 rows to match this value and down-sampled the task=0 rows accordingly. Subsequently, we merged these sub-dataframes into a single one, ensuring the rows were shuffled to maintain randomness.

2.5 Categorical Data Encoding

Further, we had to deal with categorical data that required encoding. Initially, we attempted one-hot encoding, which resulted in an accuracy of approximately 20%. However, after implementing label encoding, the accuracy score improved to 64%. Further enhancements were achieved by employing target encoding, where each unique value in a categorical column was assigned a unique float value. This approach elevated the accuracy to 68

It's crucial to note that all these preprocessing steps were also applied to the test data. The primary difference was that for the test data, we performed null value imputation directly on the dataset to prevent any shuffling of rows.

3 Model training and testing

To approach to the best model with the best hyperparameters, we tried out a lot of different models and tuned its hyper parameters. Some algorithms we used were

- K-Nearest Neighbours Classifier
- Logistic Regression
- Light Gradient Boosting Classifier
- Random Forest Classifier
- Adaptive Boost Classifier
- Extreme Gradient Boosting Classifier

3.1 K-Nearest Neighbours Classifier

KNN implementation code

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5, weights='uniform', leaf_size=30, n_jobs=1)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Best Accuracy : 52.055%

3.2 Logistic Regression

Logistic Regression implementation code

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(
    penalty='l2',
    C=1.0,
    solver='liblinear',
    max_iter=100,
    class_weight='balanced',
    random_state=42
)
model.fit(X_train,Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Best Accuracy : 54.769%

3.3 LGBM Classifier

LGBM Classifier implementation code

```
import lightgbm as lgb
model = lgb.LGBMClassifier(
    learning_rate=0.05,
    n_estimators=100,
    max_depth=6,
)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Best Accuracy : 63.195%

3.4 Random Forest Classifier

Random Forest Classifier implementation code

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(
    n_estimators=100,
    max_depth=5,
    random_state=42
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Best Accuracy : 65.197%

3.5 AdaBoost Classifier

AdaBoost Classifier implementation code

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

base_estimator = DecisionTreeClassifier(max_depth=1)
model = AdaBoostClassifier(
    base_estimator=base_estimator,
    n_estimators=50,
    random_state=42
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Best Accuracy : 59.664%

3.6 XGBoost Classifier (Best Model)

XGBoost Classifier implementation code

```
from sklearn.model_selection import train_test_split
import xgboost as xgb

X_train, X_valid, y_train, y_valid = train_test_split(X, Y, test_size=0.05, random_state=42)

dtrain = xgb.DMatrix(X_train, label=y_train)
dvalid = xgb.DMatrix(X_valid, label=y_valid)

params = {
    'objective': 'binary:logistic',
    'max_depth': 8,
    'learning_rate': 0.05,
    'eval_metric': 'logloss',
    'n_estimators':750,
    'min_child_weight':1,
    'subsample':0.6,
    'colsample_bytree': 1.0,
    'gamma':0.5,
    'reg_alpha':0.1,
    'reg_lambda':2
}

num_boost_round = 3000

evals = [(dvalid, 'eval')]
early_stopping_rounds = 10

model = xgb.train(params, dtrain, num_boost_round=num_boost_round, evals=evals,
                  early_stopping_rounds=early_stopping_rounds)
```

Best Accuracy : 68.132%

4 XGBoost accuracy Timeline

XGBoost turned out to be the best model for the given dataset.

- **64.404% :** Normal XGBoost with minimal preprocessing(null value imputation) and no hyper parameter tuning.
- **65.524% :** Dropped columns with correlation and low variance
- **66.821% :** We noticed that there was an imbalance in the the number of Task = 0 and 1. So we implemented up and down sampling
- **67.175% :** Encoding algorithm was changed from Label encoding to Categorical encoding.
- **67.973% :** Seasonal Data Categorization was implemented
- **68.132% :** XGBoost early stopping was implemented to prevent overfitting.

For our best model, we utilized the XGBoost classifier for model training. Subsequently, we proceeded to train our model on the preprocessed training data, creating a validation split of 95% for training and 5% for validation.

Additionally, we implemented the concept of early stopping, set to 10 iterations, a parameter in XGBoost. The training process continued until the validation loss consistently decreased, and it halted as soon as the validation loss began to increase to prevent overfitting.