Weather Prediction using Machine Learning

JAMIA MILLIA ISLAMIA

By:
INSHA SALEEM

Course Instructor:
Dr. Nehal Ahmad

INDEX

Weather Prediction

Objective

In this project, we aim to predict the weather conditions in Seattle, a city in Washington state, in United States of America using Machine Learning Models. The weather data of Seattle city for the of years, 2012-2015 has been used for the study.

Machine Learning Models, namely, Logistic Regression, Support Vector Machine (SVM) and Naive Bayes have been used for weather prediction. These models belong to Classification technique under Supervised Machine Learning Algorithms.

Finally, Accuracy Score, Confusion Matrix and classification report has been used to evaluate the performance of our model.

Introduction

Weather prediction, often known as weather forecasting, is the act of predicting the state of the atmosphere at a specific time and location in the future. It is a critical field of research for comprehending and forecasting the behaviour of our planet's atmosphere, which is constantly changing owing to natural and man-made forces. Accurate weather prediction has multiple uses, like agriculture, transportation, aviation, and disaster management, to name a few.

People have attempted to predict the weather informally for millennia and formally since the 19th century. Weather forecasts are made by collecting quantitative data about the current state of the atmosphere, land, and ocean and using meteorology to project how the atmosphere will change at a given place.

Once calculated manually based mainly upon changes in barometric pressure, current weather conditions, and sky conditions or cloud cover, weather forecasting now relies on computer-based models that take many atmospheric factors into account. Human input is still required to pick the best possible model to base the forecast upon, which involves pattern recognition skills, teleconnections, knowledge of model performance, and knowledge of model biases.

The inaccuracy of forecasting is due to the chaotic nature of the atmosphere; the massive computational power required to solve the equations that describe the atmosphere, the land, and the ocean; the error involved in measuring the initial conditions; and an incomplete understanding of atmospheric and related processes. Hence, forecasts become less accurate as the difference between the current time and the time for which the forecast is being made (the *range* of the forecast) increases. The use of ensembles and model consensus helps narrow the error and provide confidence in the forecast.

Weather prediction stands at the forefront of critical decision-making in a multitude of industries, from agriculture and transportation to disaster management. Accurate

forecasts can mean the difference between a successful harvest and crop failure, the smooth operation of transportation networks, and the effective response to natural disasters. As we navigate a world where climate patterns are becoming increasingly unpredictable, the demand for precise weather forecasting is more pressing than ever.

In recent years, traditional methods of weather prediction have seen a transformative shift with the integration of machine learning. This amalgamation of meteorology and advanced data analytics holds the promise of significantly enhancing the accuracy and reliability of weather forecasts.

Because of their capacity to model complex connections and patterns in huge datasets, machine learning approaches have grown in popularity in weather prediction. With huge volumes of weather data available from multiple sources such as satellites, radars, and weather stations, machine learning systems may learn from this data to effectively anticipate weather conditions.

Methodology

In this project, a relatively small dataset has been considered. The dataset contains information about the weather conditions in the Seattle, a city in Washington state, in United States of America of years 2012-2015.

Machine Learning Models, namely, Logistic Regression, Support Vector Machine (SVM) and Naive Bayes have been used for weather prediction. These models belong to Classification algorithms under Supervised Machine Learning Algorithms.

Finally, Accuracy Score, Confusion Matrix and classification report has been used to evaluate the performance of our model.

Theoretical Background

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that allows computers to learn from data and perform tasks without explicit instructions. ML systems can analyse large amounts of data, identify patterns, and predict relationships between data.

Machine learning (ML) is a type of Artificial Intelligence (AI) that allows computers to learn and make decisions without being explicitly programmed. It involves feeding data into algorithms that can then identify patterns and make predictions on new data. Machine learning is used in a wide variety of applications, including image and speech recognition, natural language processing, and recommender systems.

Machine learning is able to learn, train from data and solve/predict complex solutions which cannot be done with traditional programming. It enables us with better decision making and solve complex business problems in optimized time. Machine learning has applications in various fields, like healthcare, finance, educations, sports and more.

Machine Learning can be used to, solve complex business problems, handle large volumes of data, automate repetitive tasks, personalize user experience etc. ML models are able to improve themselves based on more data, like user-behaviour and feedback.

Supervised Machine Learning

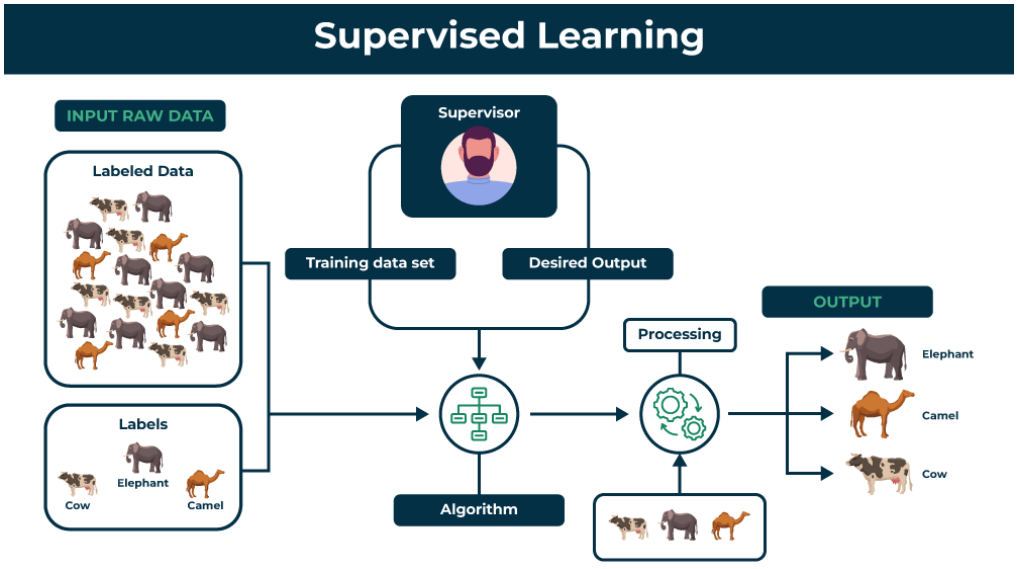
Supervised Machine Learning is a machine learning technique that uses labeled datasets to train AI models to recognize patterns and relationships between input and output. The data used in supervised learning is labeled which means that it contains both inputs (called features) and correct outputs (labels).

The goal is to create a model that can predict correct outputs for new data. Supervised learning is a type of machine learning algorithm that learns from labeled data.

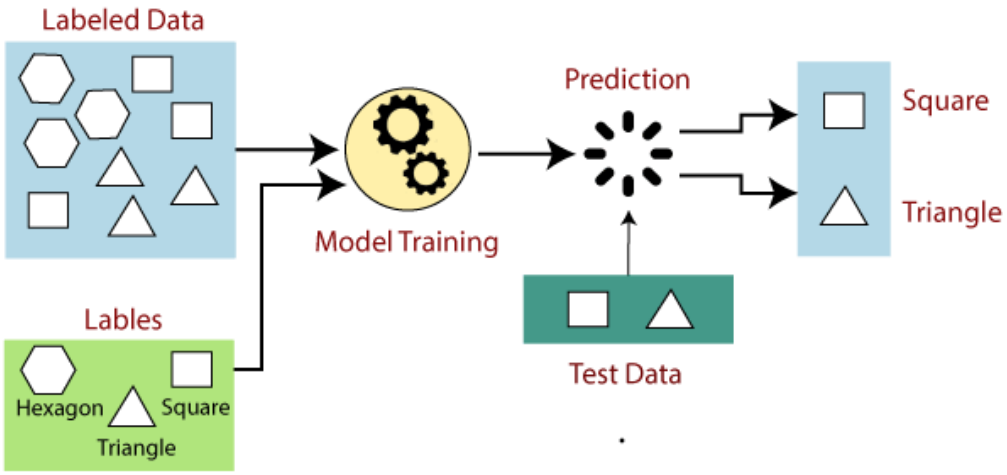
Labeled data, is a data that has been tagged with a correct answer or classification. It is a raw dataset that has been assigned labels to provide context and meaning. In a labeled dataset output of the corresponding input data is already known. These labels help machine learning models understand the data and interpret it effectively.

Supervised learning, as the name indicates, has the presence of a supervisor as a teacher. Supervised learning is when we teach or train the machine using data that is well-labelled. Which means some data is already tagged with the correct answer, which means each input comes with a corresponding correct output. Labeled data is carefully annotated with meaningful tags, or labels, that classify the data's elements or outcomes.

For example, in a dataset of emails, each email might be labeled as "spam" or "not spam." These labels then provide a clear guide for a machine learning algorithm to learn from. Similarly in a labeled dataset of images of Elephant, Camel and Cow would have each image tagged with either “Elephant”, “Camel” or “Cow.”



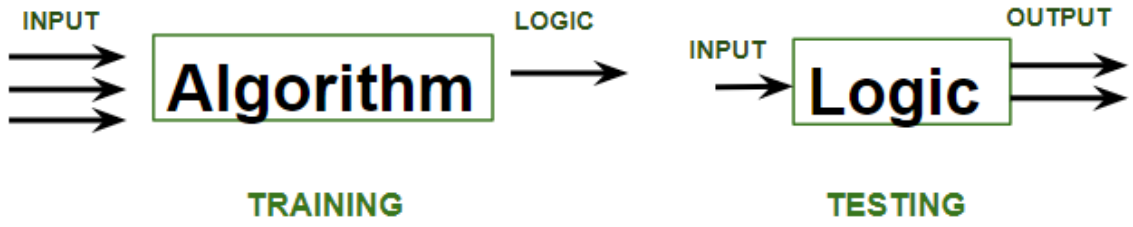
Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and hexagon.



While training the model learns about each type of data by comparing its predictions with the actual answers provided in the training data. The algorithm processes the training data, learning the relationships between the input features and the output labels. Over time, it adjusts itself to minimize errors and improve accuracy. The goal of supervised learning is to make accurate predictions when given new, unseen data. For example, if a model is trained to recognize handwritten digits, it will use what it learned to correctly identify new numbers it hasn't seen before.

After training, the machine learning model is provided with some new or unseen data, so that the supervised learning algorithm model can be tested on completely new or unseen data. The new or unseen data is same as the training dataset, but it is called new or unseen since the it was not present in the training dataset, so the machine learning model was not trained on it. This new/unseen data is known as testing data is mostly a subset of the training dataset.

The model based on this training produces an output that is, it gives a prediction for the new/unseen data analyses the by producing an output/prediction based on its labeled training data.



In the image above,

- **Training** phase involves feeding the algorithm labeled data, where each data point is paired with its correct output. The algorithm learns to identify patterns and relationships between the input and output data.

- **Testing** phase involves feeding the algorithm testing data this is, new/unseen data; and evaluating its ability to predict the correct output based on the learned patterns.

Types of supervised learning

Supervised learning in machine learning is generally divided into two categories:

- Classification
- Regression

Classification

Classification algorithms are used to group data by predicting a categorical label or output variable based on the input data. Classification is used when output variables are categorical, meaning there are two or more classes.

One of the most common examples of classification algorithms in use is the spam filter in your email inbox. Here, a supervised learning model is trained to predict whether an email is spam or not with a dataset that contains labeled examples of both spam and legitimate emails. The algorithm extracts information about each email, including the sender, the subject line, body copy, and more. It then uses these features and corresponding output labels to learn patterns and assign a score that indicates whether an email is real or spam.

Regression

Regression algorithms are used to predict a real or continuous value, where the algorithm detects a relationship between two or more variables.

A common example of a regression task might be predicting a salary based on work experience. For instance, a supervised learning algorithm would be fed inputs related to work experience (e.g., length of time, the industry or field, location, etc.) and the corresponding assigned salary amount. After the model is trained, it could be used to predict the average salary based on work experience.

Classification algorithms learn a function that maps from the input features to a probability distribution over the output classes.

Classification Algorithms

Classification algorithms are used for predicting discrete outcomes, if the outcome can take two possible values such as True or False, Default or No Default, Yes or No, it is known as Binary Classification. When the outcome contains more than two possible values, it is known as Multiclass Classification. There are many machine learning algorithms that can be used for classification tasks. Some of them are:

- Logistic Regression
- Support Vector Machines
- Naive Bayes
- Decision Tree Classifier
- K-Nearest Neighbor Classifier
- Random Forest Classifier
- Neural Networks

Logistic Regression

Logistic regression is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm which analyze the relationship between two data factors.

Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.

For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. It's referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

Key Points:

- Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Logistic Function – Sigmoid Function

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form.
- The S-form curve is called the Sigmoid function or the logistic function.

- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Types of Logistic Regression

On the basis of the categories, Logistic Regression can be classified into three types:

1. **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

2. **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as “cat”, “dogs”, or “sheep”

3. **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as “low”, “Medium”, or “High”.

Assumptions of Logistic Regression

We will explore the assumptions of logistic regression as understanding these assumptions is important to ensure that we are using appropriate application of the model. The assumption include:

1. Independent observations: Each observation is independent of the other. meaning there is no correlation between any input variables.

2. Binary dependent variables: It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories SoftMax functions are used.

3. Linearity relationship between independent variables and log odds: The relationship between the independent variables and the log odds of the dependent variable should be linear.

4. No outliers: There should be no outliers in the dataset.

5. Large sample size: The sample size is sufficiently large

Terminologies involved in Logistic Regression

Here are some common terms involved in logistic regression:

- **Independent variables:** The input characteristics or predictor factors applied to the dependent variable’s predictions.

- **Dependent variable:** The target variable in a logistic regression model, which we are trying to predict.

- **Logistic function:** The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.

- **Odds:** It is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur.

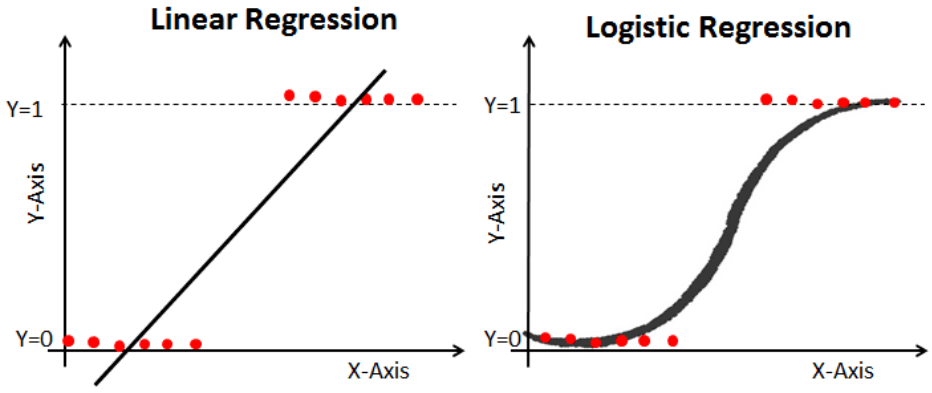
- **Log-odds:** The log-odds, also known as the logit function, is the natural logarithm of the odds. In logistic regression, the log odds of the dependent variable are modeled as a linear combination of the independent variables and the intercept.

- **Coefficient:** The logistic regression model's estimated parameters, show how the independent and dependent variables relate to one another.

- **Intercept:** A constant term in the logistic regression model, which represents the log odds when all independent variables are equal to zero.

- **Maximum likelihood estimation:** The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model.

Logistic Regression is a special case of Linear Regression where target variable (y) is discrete / categorical such as 1 or 0, True or False, Yes or No, Default or No Default. A log of the odds is used as the dependent variable. Using a logit function, logistic regression makes predictions about the probability that a binary event will occur.

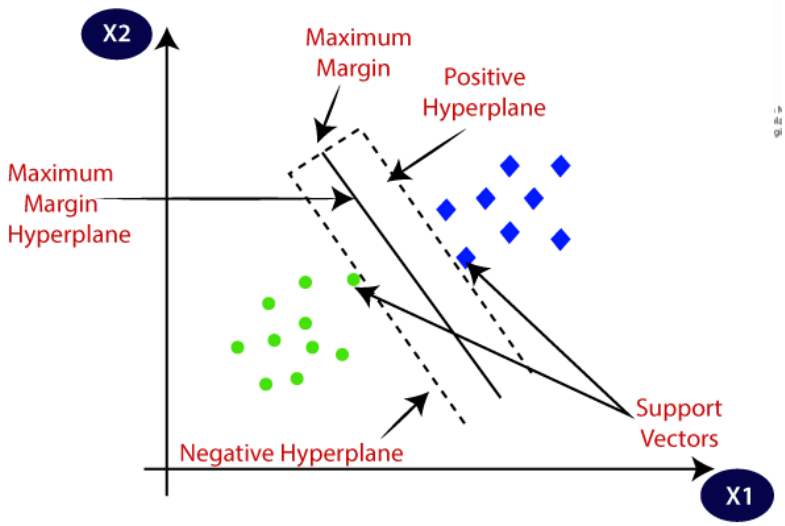


Support Vector Machine

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression tasks. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

The SVM algorithm is to identify the optimal hyperplane in an N-dimensional space that can effectively separate data points into different classes in the feature space. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. The algorithm ensures that the margin between the closest points of different classes, known as support vectors, is maximized.

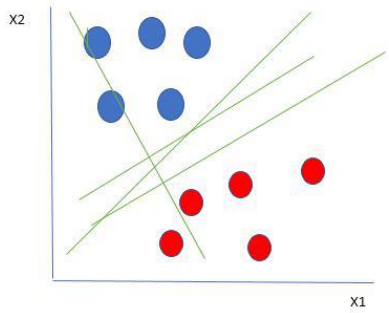
The dimension of the hyperplane depends on the number of features. For instance, if there are two input features, the hyperplane is simply a line, and if there are three input features, the hyperplane becomes a 2-D plane. As the number of features increases beyond three, the complexity of visualizing the hyperplane also increases.



Working of Support Vector Machine Algorithm

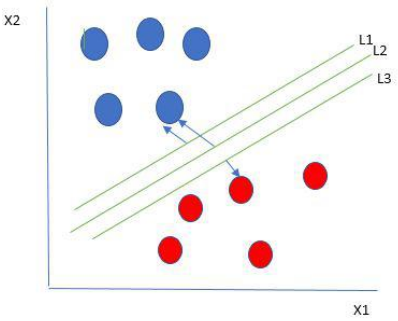
Consider two independent variables, x1 and x2, and one dependent variable represented as either a blue circle or a red circle.

- In this scenario, the hyperplane is a line because we are working with two features (x1 and x2).
- There are multiple lines (or hyperplanes) that can separate the data points.
- The challenge is to determine the best hyperplane that maximizes the separation margin between the red and blue circles.

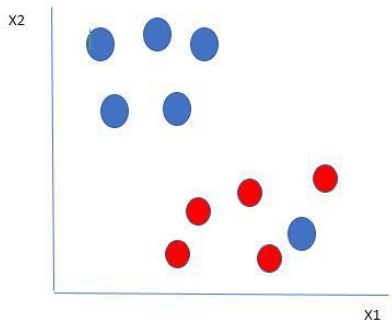


From the figure above it's very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregate our data points or do a classification between red and blue circles. So how do we choose the best line or in general the best hyperplane that segregates our data points?

One reasonable choice for the best hyperplane in a Support Vector Machine (SVM) is the one that maximizes the separation margin between the two classes. The maximum-margin hyperplane, also referred to as the hard margin, is selected based on maximizing the distance between the hyperplane and the nearest data point on each side.



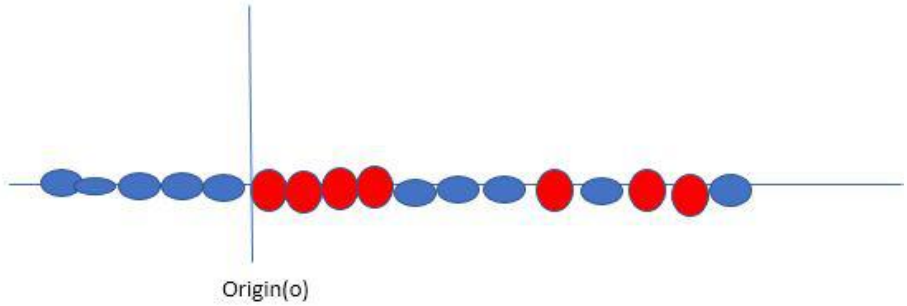
So, we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the maximum-margin hyperplane/hard margin. So, from the above figure, we choose L2. Let's consider a scenario like shown below.



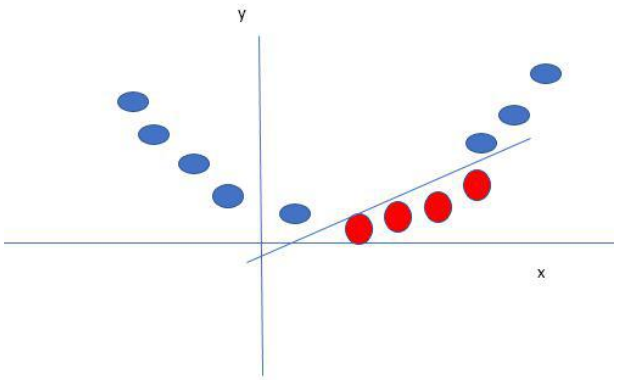
Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.

So, in this type of data point what SVM does is, finds the maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. So, the margins in these types of cases are called soft margins. When there is a soft margin to the data set, the SVM tries to minimize (1/margin+∧(∑penalty)). Hinge loss is a commonly used penalty. If no violations no hinge loss. If violations hinge loss proportional to the distance of violation.

Till now, we were talking about linearly separable data (the group of blue balls and red balls are separable by a straight line/linear line). What to do if data are not linearly separable?



Say, our data is shown in the figure above. SVM solves this by creating a new variable using a kernel. We call a point xi on the line and we create a new variable y as a function of distance from origin o.so if we plot this, we get something like as shown below.



In this case, the new variable y is created as a function of distance from the origin. A non-linear function that creates a new variable is referred to as a kernel.

Support Vector Machine Terminology

Hyperplane: The hyperplane is the decision boundary used to separate data points of different classes in a feature space. For linear classification, this is a linear equation represented as wx+b=0.

Support Vectors: Support vectors are the closest data points to the hyperplane. These points are critical in determining the hyperplane and the margin in Support Vector Machine (SVM).
Margin: The margin refers to the distance between the support vector and the hyperplane. The primary goal of the SVM algorithm is to maximize this margin, as a wider margin typically results in better classification performance.
Kernel: The kernel is a mathematical function used in SVM to map input data into a higher-dimensional feature space. This allows the SVM to find a hyperplane in cases where data points are not linearly separable in the original space. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.
Hard Margin: A hard margin refers to the maximum-margin hyperplane that perfectly separates the data points of different classes without any misclassifications.
Soft Margin: When data contains outliers or is not perfectly separable, SVM uses the soft margin technique. This method introduces a slack variable for each data point to allow some misclassifications while balancing between maximizing the margin and minimizing violations.
C: The C parameter in SVM is a regularization term that balances margin maximization and the penalty for misclassifications. A higher C value imposes a stricter penalty for margin violations, leading to a smaller margin but fewer misclassifications.
Hinge Loss: The hinge loss is a common loss function in SVMs. It penalizes misclassified points or margin violations and is often combined with a regularization term in the objective function.
Dual Problem: The dual problem in SVM involves solving for the Lagrange multipliers associated with the support vectors. This formulation allows for the use of the kernel trick and facilitates more efficient computation.

Types of Support Vector Machine

Based on the nature of the decision boundary, Support Vector Machines (SVM) can be divided into two main parts:

- **Linear SVM:** Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.

- **Non-Linear SVM:** Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a nonlinear decision boundary in this modified space.

Naive Bayes classifiers

Naive Bayes classifiers, are a family of algorithms based on Bayes' Theorem. Despite the "naive" assumption of feature independence, these classifiers are widely utilized for their simplicity and efficiency in machine learning.

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Naïve Bayes algorithm is used for classification problems. The advantage of using naïve Bayes is its speed. It is fast and making prediction is easy with high dimension of data. This model predicts the probability of an instance belongs to a class with a given set of feature value. It is a probabilistic classifier. It is because it assumes that one feature in the model is independent of existence of another feature. In other words, each feature contributes to the predictions with no relation between each other. In real world, this condition satisfies rarely. It uses Bayes theorem in the algorithm for training and prediction.

The "Naive" part of the name indicates the simplifying assumption made by the Naïve Bayes classifier. The classifier assumes that the features used to describe an observation are conditionally independent, given the class label. The "Bayes" part of the name refers to Reverend Thomas Bayes, an 18th-century statistician and theologian who formulated Bayes' theorem.
Assumption of Naive Bayes.

The fundamental Naive Bayes assumption is that each feature makes an:

- **Feature independence:** The features of the data are conditionally independent of each other, given the class label.
- **Continuous features are normally distributed:** If a feature is continuous, then it is assumed to be normally distributed within each class.
- **Discrete features have multinomial distributions:** If a feature is discrete, then it is assumed to have a multinomial distribution within each class.
- **Features are equally important:** All features are assumed to contribute equally to the prediction of the class label.
- **No missing data:** The data should not contain any missing values.

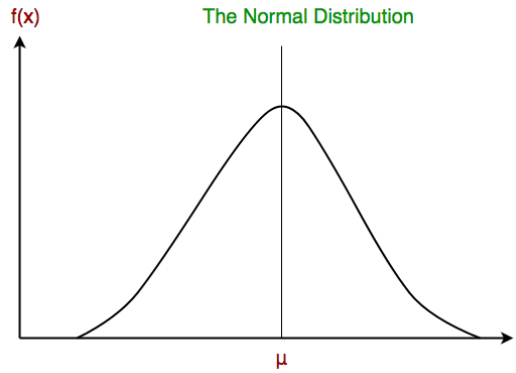
Types of Naive Bayes Model

There are three types of Naive Bayes Model:

- Gaussian
- Multinomial
- Bernoulli

Gaussian Naive Bayes classifier

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian/Normal distribution. When plotted, it gives a bell-shaped curve which is symmetric about the mean of the feature values as shown below:



Evaluating Supervised Learning Models

Evaluating supervised learning models is an important step in ensuring that the model is accurate and generalizable. There are a number of different metrics that can be used to evaluate supervised learning models, but some of the most common ones include:

For Classification

- **Accuracy:** Accuracy is the percentage of predictions that the model makes correctly. It is calculated by dividing the number of correct predictions by the total number of predictions.
- **Precision:** Precision is the percentage of positive predictions that the model makes that are actually correct. It is calculated by dividing the number of true positives by the total number of positive predictions.
- **Recall:** Recall is the percentage of all positive examples that the model correctly identifies. It is calculated by dividing the number of true positives by the total number of positive examples.
- **F1 score:** The F1 score is a weighted average of precision and recall. It is calculated by taking the harmonic mean of precision and recall.
- **Confusion matrix:** A confusion matrix is a table that shows the number of predictions for each class, along with the actual class labels. It can be used to visualize the performance of the model and identify areas where the model is struggling.

Dataset Description: Features/Columns

The dataset used for this project contains the weather conditions of Seattle. It is a frequently used dataset used in the process of Weather Prediction. The dataset contains 1461 rows and 6 attributes/features/columns. A brief description of the attributes:

- **Date**
- **Precipitation:** Indicates all forms in which the water falls on earth as rain, hail etc.
- **temp_max:** Indicates the maximum temperature
- **temp_min:** Indicates the minimum temperature
- **wind:** Indicates the wind speed
- **weather:** Indicates the type of the weather (drizzle, rain, sun, snow, fog)

Link to download the dataset: Seattle-Weather.csv

Tools and Libraries Used

The project makes use of the following Python libraries:

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Scikit-Learn
- Plotly

Steps for Weather Prediction using Machine Learning

1. In the first step, the necessary machine learning libraries will be imported.

[1404]:
```
#Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from imblearn.over_sampling import SMOTE
from collections import Counter

from sklearn.metrics import accuracy_score,  confusion_matrix, ConfusionMatrixDisplay, classification_report
```

2. In the next step, the dataset is read using the **read_csv()** function and the data set is read using read_csv() function.

```
[1405]: #Loading the dataset
        data = pd.read_csv("C:\\Users\\Insha Saleem\\Downloads\\weather-prediction\\Code and Data Files\\weather.csv")
```

3. Data Preprocessing

Data Preprocessing

Data preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

```
[1470]: data.head(5)
```

[1470]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

```
[1471]: data.tail()
```

[1471]:

	date	precipitation	temp_max	temp_min	wind	weather
1456	2015-12-27	8.6	4.4	1.7	2.9	rain
1457	2015-12-28	1.5	5.0	1.7	1.3	rain
1458	2015-12-29	0.0	7.2	0.6	2.6	fog
1459	2015-12-30	0.0	5.6	-1.0	3.4	sun
1460	2015-12-31	0.0	5.6	-2.1	3.5	sun

4. Now, basic Exploratory Data Analysis (EDA) is performed on the dataset.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a technique that helps understand data sets and their variables. Exploratory data analysis (EDA) is used to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods.

```
[1474]: data.shape
```

```
[1474]: (1461, 6)
```

```
[1475]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1461 entries, 0 to 1460
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           1461 non-null   datetime64[ns]
 1   precipitation  1461 non-null   float64
 2   temp_max       1461 non-null   float64
 3   temp_min       1461 non-null   float64
 4   wind           1461 non-null   float64
 5   weather        1461 non-null   object
dtypes: datetime64[ns](1), float64(4), object(1)
memory usage: 68.6+ KB
```

```
[1410…  #Checking for null values
        data.isnull().sum()
```

```
[1410]: date            0
        precipitation   0
        temp_max        0
        temp_min        0
        wind            0
        weather         0
        dtype: int64
```

```
1411… #Convert the data type into datetime
      data['date'] = pd.to_datetime(data['date'])
```

```
.412]: data.dtypes
```

```
.412]: date             datetime64[ns]
       precipitation           float64
       temp_max                float64
       temp_min                float64
       wind                    float64
       weather                  object
       dtype: object
```

```
.413]: data.duplicated().sum()
```

```
.413]: 0
```

```
.414]: data.date.dtype
```

```
.414]: dtype('<M8[ns]')
```

```
.415]: type(data.date)
```

```
.415]: pandas.core.series.Series
```

```
.416]: data.columns
```

```
.416]: Index(['date', 'precipitation', 'temp_max', 'temp_min', 'wind', 'weather'], dtype='object')
```

```
.417]: data.weather.unique()
```

```
.417]: array(['drizzle', 'rain', 'sun', 'snow', 'fog'], dtype=object)
```

```
[1418]: data.nunique()
```

```
[1418]: date             1461
        precipitation     111
        temp_max           67
        temp_min           55
        wind               79
        weather             5
        dtype: int64
```

```
[1419]: data['weather'].value_counts()
```

```
[1419]: weather
        rain       641
        sun        640
        fog        101
        drizzle     53
        snow        26
        Name: count, dtype: int64
```

```
[1420]: data.describe(include=[np.number])
```

[1420]:

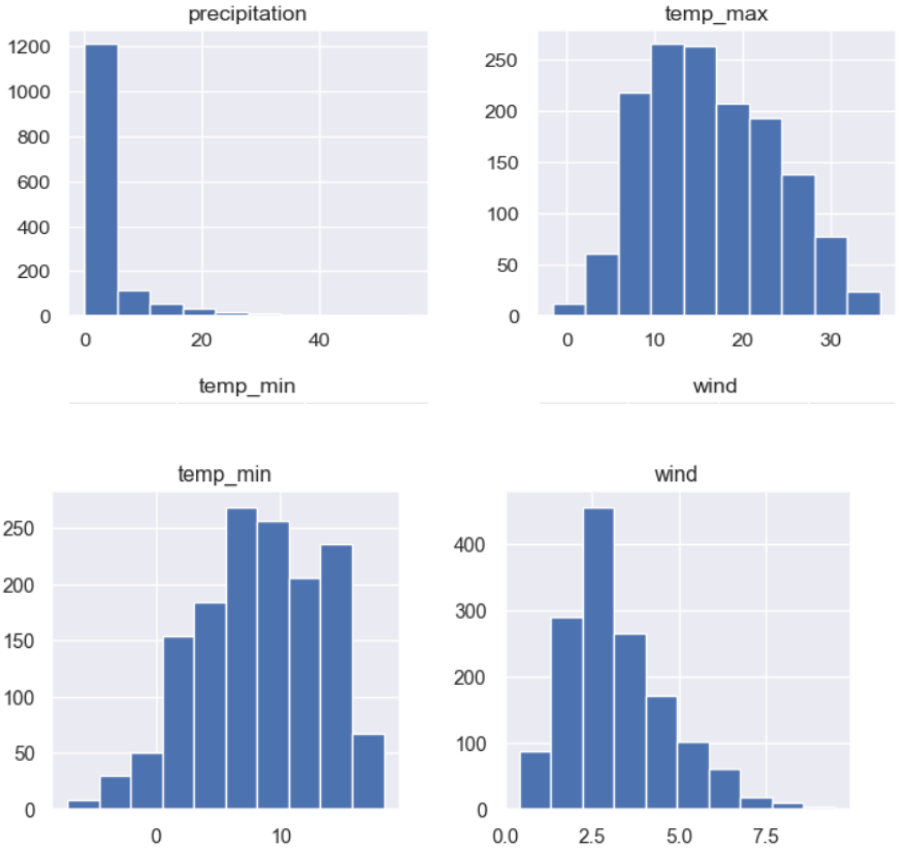
	precipitation	temp_max	temp_min	wind
count	1461.000000	1461.000000	1461.000000	1461.000000
mean	3.029432	16.439083	8.234771	3.241136
std	6.680194	7.349758	5.023004	1.437825
min	0.000000	-1.600000	-7.100000	0.400000
25%	0.000000	10.600000	4.400000	2.200000
50%	0.000000	15.600000	8.300000	3.000000
75%	2.800000	22.200000	12.200000	4.000000
max	55.900000	35.600000	18.300000	9.500000

5. Data Visualisation Techniques are used in order to understand the weather patterns better. We visualise our data using various plots.

Data Visualization

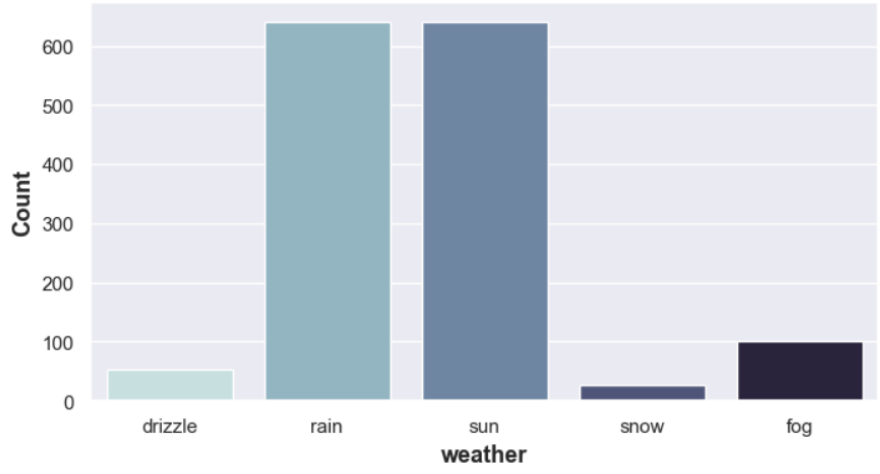
[1421]:
```
data.drop('date',axis=1).hist(bins=10, figsize=(8,7))
```

[1421]:
```
array([[<Axes: title={'center': 'precipitation'}>,
        <Axes: title={'center': 'temp_max'}>],
       [<Axes: title={'center': 'temp_min'}>,
        <Axes: title={'center': 'wind'}>]], dtype=object)
```



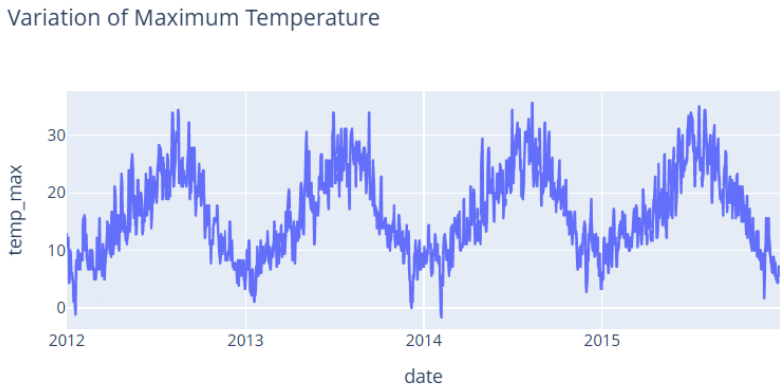
The

[1472]:
```
plt.figure(figsize=(8,4))
sns.set_theme()
sns.countplot(x = 'weather',data = data, hue='weather', palette="ch:start=.2,rot=-.3")
plt.xlabel("weather",fontweight='bold',size=13)
plt.ylabel("Count",fontweight='bold',size=13)
plt.show()
```

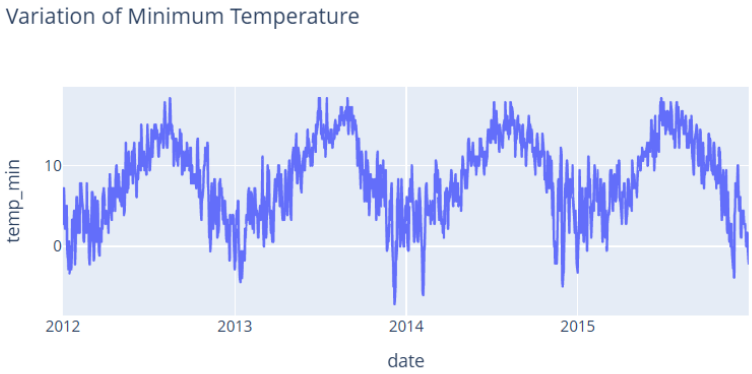


output displays the values present in the **weather** column using a countplot. It can be seen that the values **rain** and **sun** have the highest number of occurrences and the value **snow** has the least occurrence.

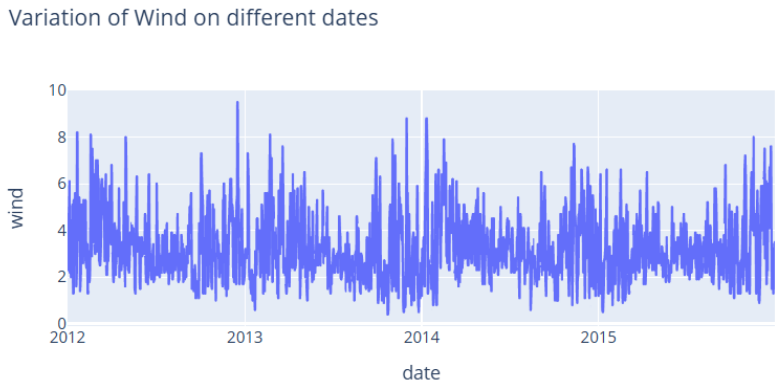
[1423]:
```
px.line(data_frame = data,
        x = 'date',
        y = 'temp_max',
        title = 'Variation of Maximum Temperature')
```



[1424]:
```
px.line(data_frame = data,
        x = 'date',
        y = 'temp_min',
        title = 'Variation of Minimum Temperature')
```



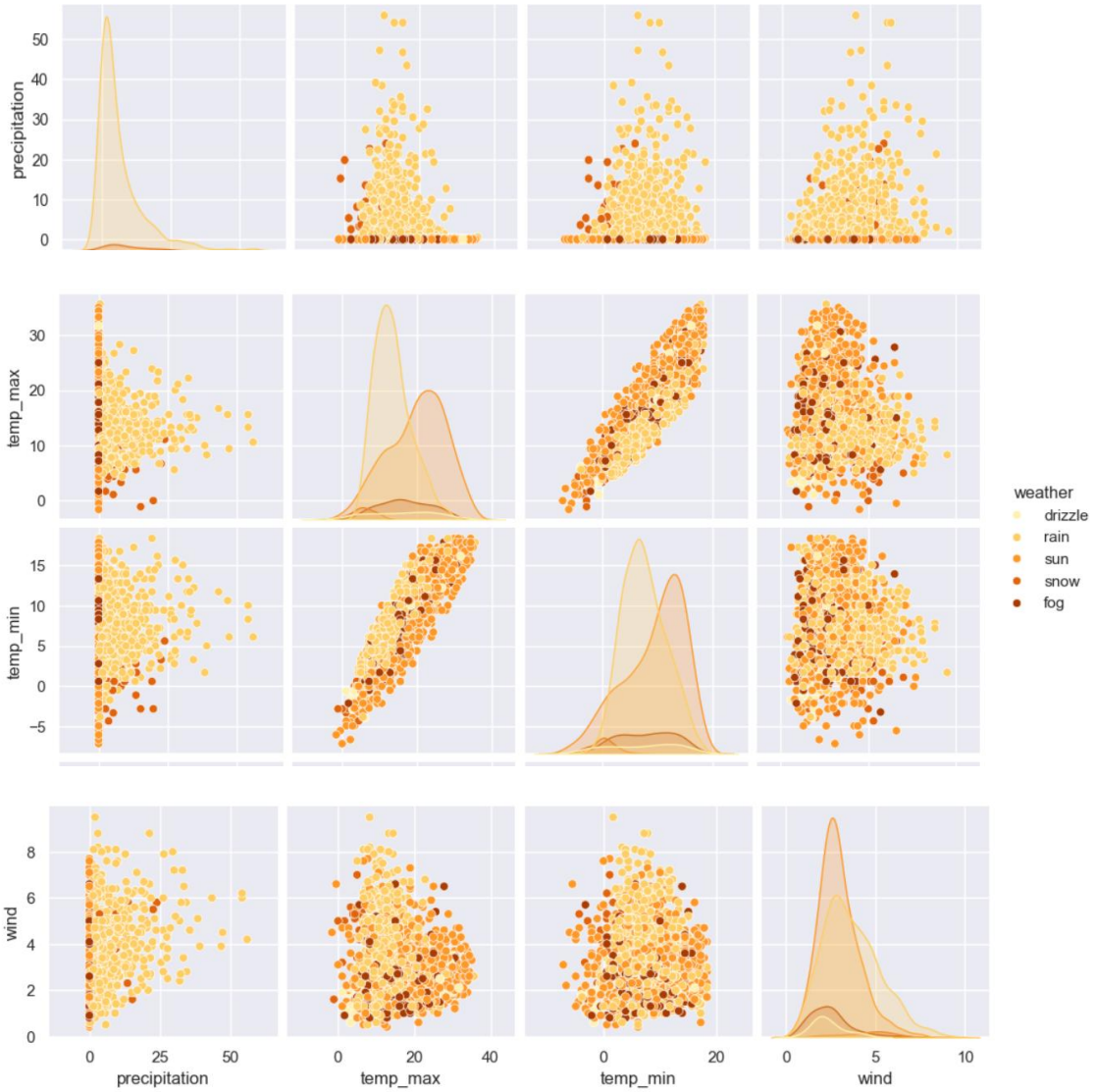
[1425]:
```
px.line(data_frame = data,
        x = 'date',
        y = 'wind',
        title = 'Variation of Wind on different dates')
```



Since the graph is made using **Plotly**, the graph can be zoomed in and zoomed out to find out more details or to gain more insights. The same line chart can be implemented for values present in other columns as well.

```
1473]: plt.figure(figsize=(10,4))
       sns.pairplot(data.drop('date',axis=1),hue='weather',palette="YlOrBr")
       plt.show()
```

```
<Figure size 1000x400 with 0 Axes>
```



Above a **pair plot** helps to understand the relationships between different variables in a dataset.

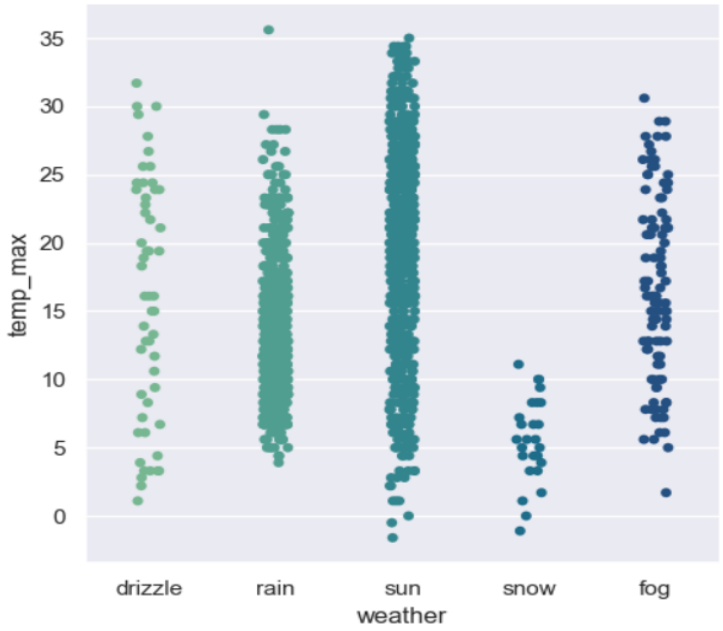
It is a type of data visualization that helps you understand the relationships between different variables in a dataset. It creates a grid of scatter plots, where each scatter plot shows the relationship between two specific variables. The diagonal of the grid typically displays histograms or kernel density estimations for each individual variable.

It creates a grid of plots to show the relationships between all pairs of remaining numerical columns in your data.

- **Scatter Plots:** For each pair of numerical variables, pairplot creates a scatter plot. This helps you see how these variables are related (e.g., is there a positive correlation, negative correlation, or no clear relationship).
- **Diagonal Plots:** Along the diagonal of the grid, pairplot creates histograms (or kernel density plots) to show the distribution of each individual numerical variable.

[1427]:
```
plt.figure(figsize=(10,5))
sns.catplot(x='weather',y ='temp_max',data=data, hue='weather', palette="crest")
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```

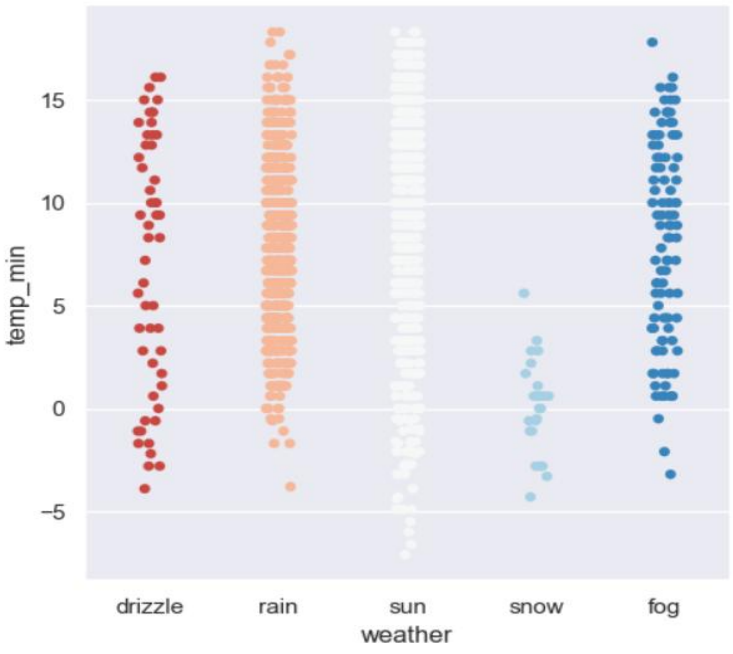


Catplot is a shorthand for categorical plot, which can be implemented to analyse the relationship between variables. The catplot provides the ability to create different categorical plots.

The output contains the catplot for the **weather** column and the **temp_max** column. It displays the relationship between the categorical variable(**weather**) and the numeric variable (**maximum temperature**). Each data point is represented by a scatter point, and the categorical factors decide how the scatter points are coloured and organised.

[1428]:
```
plt.figure(figsize=(10,5))
sns.catplot(x='weather', y ='temp_min', data=data, hue='weather', palette = "RdBu")
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```



The output contains the catplot for the **weather** column and the **temp_min** column. It displays the relationship between the categorical variable(**weather**) and the numeric variable (**minimum temperature**). Each data point is represented by a scatter point, and the categorical factors decide how the scatter points are coloured and organised.

6. Encoding is performed. But only on the target variable, since rest of the data is numerical.

Encoding

Encoding is a technique of converting categorical variables into numerical values so that it could be easily fitted to a machine learning model.

[1429]:
```
#Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models
#which only take numerical data. It is an important pre-processing step in a machine-learning project.

data1=data.copy()
label_encoder = LabelEncoder()
data1['weather'] = label_encoder.fit_transform(data1['weather'])
```

[1430]:
```
data1.head(10)
```

[1430]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	0
1	2012-01-02	10.9	10.6	2.8	4.5	2
2	2012-01-03	0.8	11.7	7.2	2.3	2
3	2012-01-04	20.3	12.2	5.6	4.7	2
4	2012-01-05	1.3	8.9	2.8	6.1	2
5	2012-01-06	2.5	4.4	2.2	2.2	2
6	2012-01-07	0.0	7.2	2.8	2.3	2
7	2012-01-08	0.0	10.0	2.8	2.0	4
8	2012-01-09	4.3	9.4	5.0	3.4	2
9	2012-01-10	1.0	6.1	0.6	3.4	2

[1431]:
```
data1.weather.unique()
```

[1431]:
```
array([0, 2, 4, 3, 1])
```

[1432]:
```
data1.head(10)
```

[1432]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	0
1	2012-01-02	10.9	10.6	2.8	4.5	2
2	2012-01-03	0.8	11.7	7.2	2.3	2
3	2012-01-04	20.3	12.2	5.6	4.7	2
4	2012-01-05	1.3	8.9	2.8	6.1	2
5	2012-01-06	2.5	4.4	2.2	2.2	2
6	2012-01-07	0.0	7.2	2.8	2.3	2
7	2012-01-08	0.0	10.0	2.8	2.0	4
8	2012-01-09	4.3	9.4	5.0	3.4	2
9	2012-01-10	1.0	6.1	0.6	3.4	2

7. Scaling is performed on the data set.

Scaling

```
Scaling in machine learning, also known as feature scaling or data normalization, is a preprocessing step that involves transforming
numerical features in a dataset to a common scale
```

439]:
```
#StandardScaler operates on the principle of normalization, where it transforms the distribution of each feature to have a mean of zero
#and a standard deviation of one. This process ensures that all features are on the same scale, preventing any single feature from dominating
#the learning process due to its larger magnitude.

sc = StandardScaler()
xx = sc.fit_transform(x)
```

8. Splitting the data in Training Data and Testing Data.

- Our dataset was imbalanced. So, before splitting over-sampling technique was used to address the problem of imbalanced dataset.
- Imbalanced datasets impact the performance of the machine learning models and the Synthetic Minority Over-sampling Technique (SMOTE) addresses the class imbalance problem by generating synthetic samples for the minority class.
- SMOTE-ENN combines the SMOTE method with the Edited Nearest Neighbors (ENN) rule. ENN is used to clean the data by removing any samples that are misclassified by their nearest neighbors. This combination helps in cleaning up the synthetic samples, improving the overall quality of the dataset. The objective of ENN is to remove noisy or ambiguous samples, which may include both minority and majority class instances.

Working Procedure of SMOTE-ENN (Edited Nearest Neighbors)

- **SMOTE Application**: First, apply SMOTE to generate synthetic samples.
- **ENN Application**: Then, use ENN to remove synthetic or original samples that have a majority of their nearest neighbors belonging to the opposite class.
- **Cleaning Data**: This step helps in removing noisy instances and those that are likely to be misclassified.

Train Test Split

[511]:
```
print("Original class distribution:", Counter(y))
```
```
Original class distribution: Counter({2: 641, 4: 640, 1: 101, 0: 53, 3: 26})
```
```
Imbalanced datasets impact the performance of the machine learning models and the Synthetic Minority Over-sampling Technique (SMOTE)
addresses the class imbalance problem by generating synthetic samples for the minority class.
```

[521]:
```
smote=SMOTE(sampling_strategy='minority')
xx,y=smote.fit_resample(xx, y)
y.value_counts()
```

[521]:
```
weather
0    641
2    641
4    641
3    641
1    641
Name: count, dtype: int64
```

[526]:
```
len(xx)
```

[526]:
```
3205
```

[527]:
```
#Splitting the training and testing data; 25% => 802 rows are used for testing and 75% => 2404 rows are used for training.
x_train, x_test, y_train, y_test = train_test_split(xx, y, test_size = 0.25, random_state = 0)
```

9. The final step, involves applying the Machine Learning Models on our dataset.

- **Logistic Regression**

Logistic Regression

```
[1516]: from sklearn.linear_model import LogisticRegression
        model1 = LogisticRegression()
        model1.fit(x_train, y_train)
```

```
[1516]: LogisticRegression()
```

```
[1517]: y_pred = model1.predict(x_test)
```

```
[1518]: train_acc=accuracy_score(y_train , model1.predict(x_train))
        print(f"Training Accuracy: {train_acc}")
        test_acc1=accuracy_score(y_test, y_pred)
        print(f"Testing Accuracy: {test_acc1}")
```

```
Training Accuracy: 0.7926940639269406
Testing Accuracy: 0.7622950819672131
```

```
[1519]: acc_score1=test_acc1*100
        print(f"Accuracy Score: {acc_score1:.2f}%")
```

```
Accuracy Score: 76.23%
```

An Accuracy of 76.23% was obtained from this model.

```
1448]: for i in range(len(y_pred)):
         y_pred[i]=round(y_pred[i],2)
       pd.DataFrame({'Actual':y_test,'Prediction':y_pred,'diff':(y_test-y_pred)})
```

1448]:

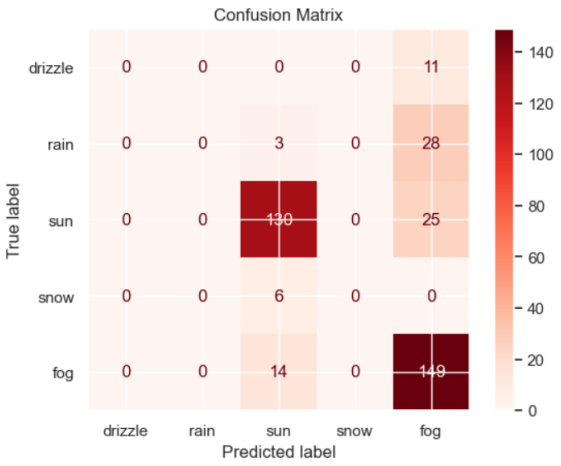
	Actual	Prediction	diff
530	4	4	0
657	4	2	2
459	2	2	0
279	4	4	0
656	4	4	0
...	...	...	...
781	2	2	0
1391	1	4	-3
1168	2	2	0
847	2	2	0
1425	4	4	0

366 rows × 3 columns

```
[1449]: cm = confusion_matrix(y_test, y_pred)
        print(cm)
```

```
[[  0   0   0   0  11]
 [  0   0   3   0  28]
 [  0   0 130   0  25]
 [  0   0   6   0   0]
 [  0   0  14   0 149]]
```

```
[1450]: disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data.weather.unique())
        disp.plot(cmap=plt.cm.Reds)
        plt.title('Confusion Matrix')
        plt.show()
```




```
[1451]: classification_rep = classification_report(y_test, y_pred, zero_division=1)
        print("Classification Report:")
        print(classification_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        11
           1       1.00      0.00      0.00        31
           2       0.85      0.84      0.84       155
           3       1.00      0.00      0.00         6
           4       0.70      0.91      0.79       163

    accuracy                           0.76       366
   macro avg       0.91      0.35      0.33       366
weighted avg       0.80      0.76      0.71       366
```

- **Support Vector Machine (SVC-Linear)**

SVM

```
[1452]: from sklearn.svm import SVC
        model2 = SVC(kernel = 'rbf', random_state = 0)
        model2.fit(x_train, y_train)
```

```
[1452]: SVC(random_state=0)
```

```
[1453]: y_pred = model2.predict(x_test)
```

```
[1454]: train_acc=accuracy_score(y_train , model2.predict(x_train))
        print(f"Training Accuracy: {train_acc}")
        test_acc2=accuracy_score(y_test, y_pred)
        print(f"Testing Accuracy: {test_acc2}")
```

```
Training Accuracy: 0.7835616438356164
Testing Accuracy: 0.7650273224043715
```

```
[1455]: acc_score2=test_acc2*100
        print(f"Accuracy Score: {acc_score2:.2f}%")
```

```
Accuracy Score: 76.50%
```

An Accuracy of 76.50% was obtained from this model.

[1456]:
```
for i in range(len(y_pred)):
  y_pred[i]=round(y_pred[i],2)
pd.DataFrame({'Actual':y_test, 'Prediction':y_pred,'diff':(y_test-y_pred)})
```

[1456]:

	Actual	Prediction	diff
530	4	4	0
657	4	2	2
459	2	2	0
279	4	4	0
656	4	2	2
...	...	...	...
781	2	2	0
1391	1	2	-1
1168	2	2	0
847	2	2	0
1425	4	4	0

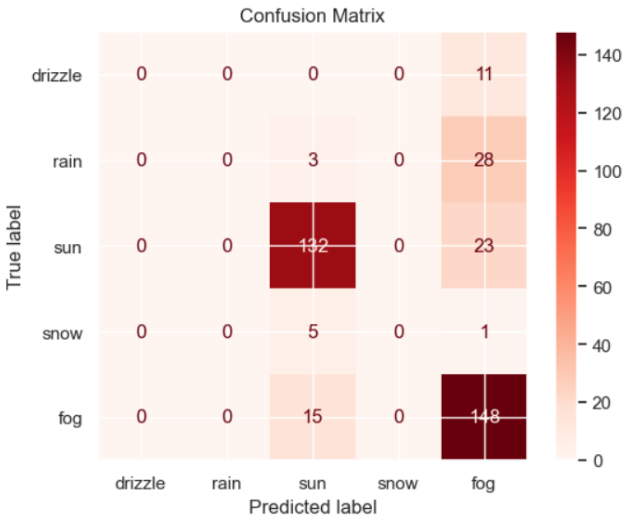
366 rows × 3 columns

[1457]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[  0   0   0   0  11]
 [  0   0   3   0  28]
 [  0   0 132   0  23]
 [  0   0   5   0   1]
 [  0   0  15   0 148]]
```

[1458]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data.weather.unique())
disp.plot(cmap=plt.cm.Reds)
plt.title('Confusion Matrix')
plt.show()
```

[1458]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data.weather.unique())
disp.plot(cmap=plt.cm.Reds)
plt.title('Confusion Matrix')
plt.show()
```



[1459]:
```
classification_rep = classification_report(y_test, y_pred, zero_division=1)
print("Classification Report:")
print(classification_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        11
           1       1.00      0.00      0.00        31
           2       0.85      0.85      0.85       155
           3       1.00      0.00      0.00         6
           4       0.70      0.91      0.79       163

    accuracy                           0.77       366
   macro avg       0.91      0.35      0.33       366
weighted avg       0.80      0.77      0.71       366
```

- # Naive Bayes

Naive Bayes

[1460]:
```
from sklearn.naive_bayes import GaussianNB
model3 = GaussianNB()
model3.fit(x_train, y_train)
```

[1460]:
```
GaussianNB()
```

[1461]:
```
y_pred = model3.predict(x_test)
```

[1462]:
```
train_acc=accuracy_score(y_train , model3.predict(x_train))
print(f"Training Accuracy: {train_acc}")
test_acc3=accuracy_score(y_test, y_pred)
print(f"Testing Accuracy: {test_acc3}")
```

```
Training Accuracy: 0.8493150684931506
Testing Accuracy: 0.8415300546448088
```

[1463]:
```
acc_score3=test_acc3*100
print(f"Accuracy Score: {acc_score3:2f}%")
```

```
Accuracy Score: 84.153005%
```

[1464]:
```
for i in range(len(y_pred)):
  y_pred[i]=round(y_pred[i],2)
pd.DataFrame({'Actual':y_test, 'Prediction':y_pred,'diff':(y_test-y_pred)})
```

An Accuracy of 84.15% was obtained from this model.

[1464]:
```
for i in range(len(y_pred)):
  y_pred[i]=round(y_pred[i],2)
pd.DataFrame({'Actual':y_test, 'Prediction':y_pred,'diff':(y_test-y_pred)})
```

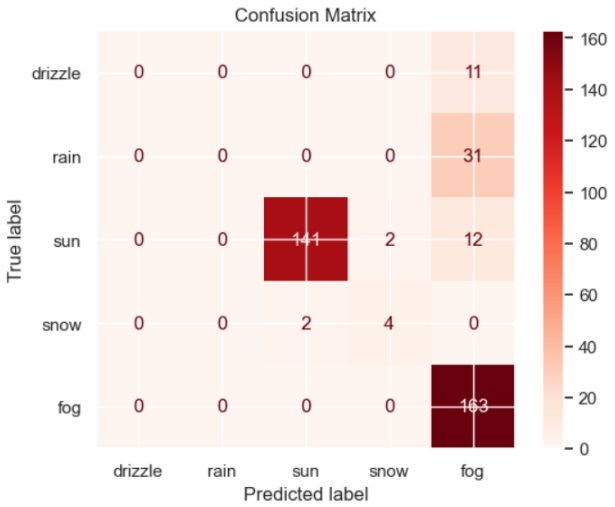
	Actual	Prediction	diff
530	4	4	0
657	4	4	0
459	2	2	0
279	4	4	0
656	4	4	0
...	...	...	...
781	2	2	0
1391	1	4	-3
1168	2	2	0
847	2	2	0
1425	4	4	0

366 rows × 3 columns

[1465]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[  0   0   0   0  11]
 [  0   0   0   0  31]
 [  0   0 141   2  12]
 [  0   0   2   4   0]
 [  0   0   0   0 163]]
```

[1466]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data.weather.unique())
disp.plot(cmap=plt.cm.Reds)
plt.title('Confusion Matrix')
plt.show()
```



[1467]:
```
classification_rep = classification_report(y_test, y_pred, zero_division=1)
print("Classification Report:")
print(classification_rep)
```

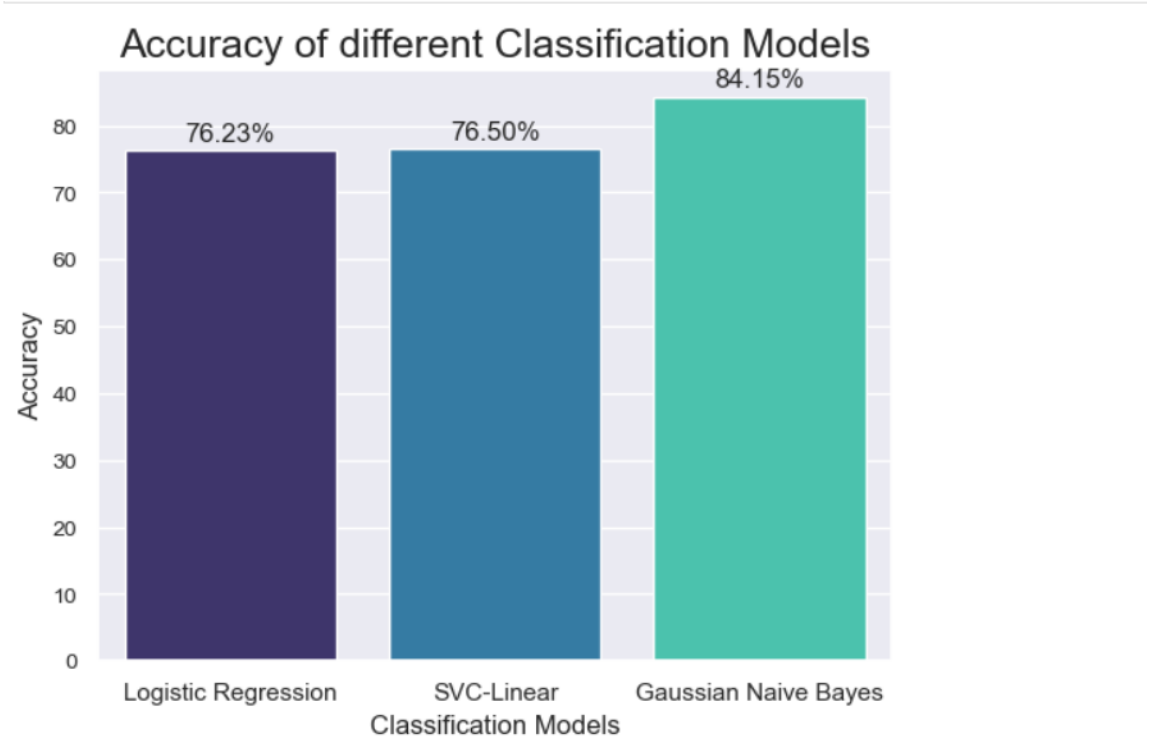
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        11
           1       1.00      0.00      0.00        31
           2       0.99      0.91      0.95       155
           3       0.67      0.67      0.67         6
           4       0.75      1.00      0.86       163

    accuracy                           0.84       366
   macro avg       0.88      0.52      0.49       366
weighted avg       0.88      0.84      0.79       366
```

The performance of all the models has been evaluated using metrics like accuracy score confusion matrix and classification report.

Lastly, a simple bar graph is plotted to view the accuracy of all models simultaneously.

[1468]:
```
mylist=[acc_score1, acc_score2, acc_score3]
mylist2=['Logistic Regression', 'SVC-Linear', 'Gaussian Naive Bayes']
```

[1469]:
```
ax = sns.barplot(x=mylist2, y=mylist, hue=mylist2, palette = "mako", saturation =1.5)
plt.xlabel("Classification Models", fontsize = 12 )
plt.ylabel("Accuracy", fontsize = 12)
plt.title("Accuracy of different Classification Models", fontsize = 18)
plt.xticks(fontsize = 11, horizontalalignment = 'center')
plt.yticks(fontsize = 10)
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.2f}%', (x + width/2, y + height*1.02), ha='center')
plt.show()
```



Conclusion

We observe that all three models are giving satisfactory predictions of our weather column. Among them Gaussian Naïve Bias Model is giving the best prediction, with an Accuracy Score of 84.5%.

We conclude that because of their capacity to model complex connections and patterns in huge datasets, Machine Learning Models can be used in predicting and/or in improving weather forecasting, aiding in decision-making processes. With huge volumes of weather data available from multiple sources such as satellites, radars, and weather stations, machine learning systems may learn from this data to effectively anticipate weather conditions.