

21AIE205
Python for Machine Learning
Credit card Fraud Detection
Project Abstract

Group Members: -

Abhishek Ashokkumar	-	AMENU4AIE21002
Eric Oommen Mathew	-	AMENU4AIE21027
Manav Maniprasad	-	AMENU4AIE21043
Vishnujith Manu	-	AMENU4AIE21082
Nikhil Kumar Singh	-	AMENU4AIE21086

Abstract :

Due to the rise and rapid growth of E-Commerce, use of credit cards for online purchases has dramatically increased and it caused an explosion in credit card fraud. In an era of digitalization, credit card fraud detection is of great importance to financial institutions.

The obtained results from databases of credit card transactions show the power of these techniques in the fight against banking fraud, comparing them to others in the same field.

Introduction :

'Fraud' in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account. Necessary prevention measures can be taken to stop this abuse and the behavior of such fraudulent practices can be studied to minimize it and protect against similar occurrences in the future. In other words, Credit Card Fraud can be defined as a case where a person uses someone else's credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used.

Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behavior, which consist of fraud, intrusion, and defaulting.

This is a very relevant problem that demands the attention of communities such as machine learning and data science where the solution to this problem can be automated.

Problem definition :

- Fraud detection is a set of activities undertaken to prevent money or property from being obtained through false pretenses.
- Fraud detection is applied to many industries such as banking or insurance. In banking, fraud may include forging checks or using stolen credit cards.
- The challenge is to recognize fraudulent credit card transaction so that the customer of credit card companies are not charged for items that they did not purchase
- Using the linear regression in this project we will be able to detect the anomaly and classify it as fraudulent transaction as quickly as possible

Methodology :

The approach that this project proposes, uses the latest machine learning algorithms to detect anomalous activities, called outliers.

1. First of all, we obtained our dataset from Kaggle, a data analysis website which provides datasets.
2. We will split our data into the training and testing data set.
3. We will use that data set to train our model using different Machine learning algorithms.
4. Then we will test our model using the testing dataset
5. Then we will compare Original output with the predicted output to compute the Mean squared error. If the mean squared error is less then our model is good to use , else we need to change the input features and may increase or decrease the number of data in the dataset as per the requirement.

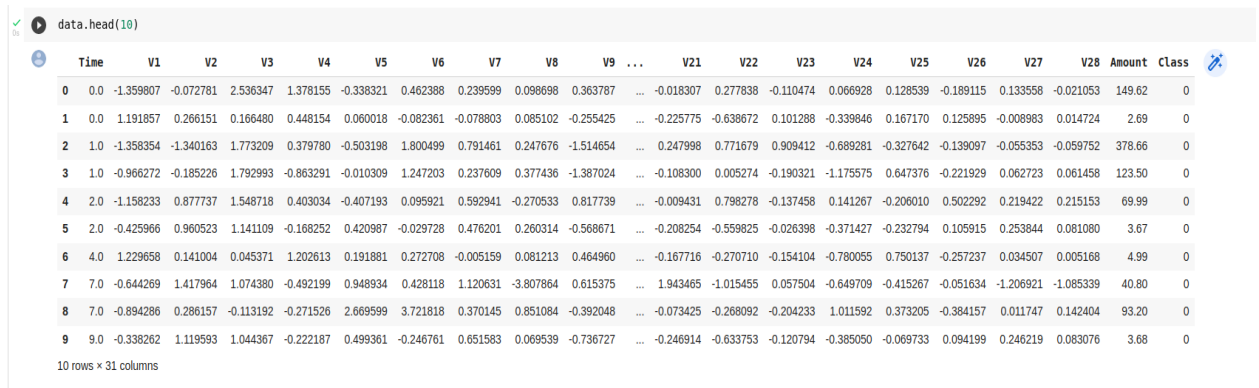
Dataset definition :

source : <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

Dataset is a collection of various types of data stored in a digital format. Data is the key component of any Machine Learning project.

For our project we have collected the data from the Kaggle dataset ,having 284807 rows and 31 columns. This dataset describe the frequency of the card used in the different times along with the amount used and the class. Observing these features we'll predict the credit card Fraud detection.

A sample of the dataset is shown below:



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.280314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67	0
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99	0
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20	0
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68	0

The values of columns V1-V28 in the dataset may be result of a PCA(Principal Component Analysis) Dimensionality reduction to protect user identities and sensitive information, whereas the Amount column is the amount of transaction done with the credit cards.The class column has only 0's and 1's , where 1's are the fraud and 0's are the legits in the dataset.

Data pre -processing:

Data preprocessing, a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure.

Standardization:

Here in the dataset all the columns from v1-v28 are already consistent. So we're going to standardize the amount column of the dataset, and transform it as normalized amount.

```

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

#We are standardizing the amount columns and replace it with normalisedamount
data['normalizedAmount'] = sc.fit_transform(data['Amount'].values.reshape(-1,1))
data = data.drop(['Amount'], axis = 1)

```

And the Amount in the dataset is transformed to normalisedAmount :

V28	Amount	Class
-0.021053	149.62	0
0.014724	2.69	0
-0.059752	378.66	0
0.061458	123.50	0
0.215153	69.99	0
0.081080	3.67	0
0.005168	4.99	0
-1.085339	40.80	0
0.142404	93.20	0
0.083076	3.68	0

Now Drop the Time from the data set , since we are not going to use the time factor for our model:



```
data = data.drop(['Time'], axis = 1)
data.head()
```

Handling Missing Inputs:

We are dropping the rows with the missing inputs:



```
data=data.dropna( )
```

Data Summarization:

Number of rows and columns in the dataset.

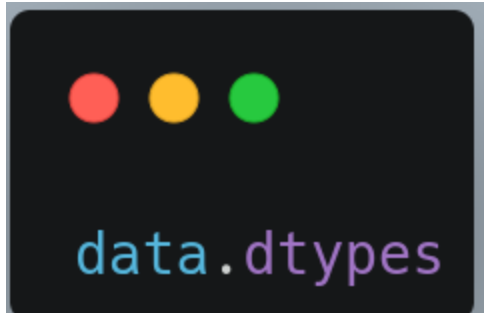


```
data.shape
```

(284807, 30)

The dataset contains 284807 rows and 30 columns.

Data types of all the columns in the dataset:



From v1-v28 and normalized amount all the datatypes are float type but class is of type integer as it contains only 0's and 1's.

Describe():

The `describe()` method returns a description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains this information for each column: count - The number of not-empty values. mean - The average (mean) value.



```

<bound method NDFrame.describe of
0      -1.359807  -0.072781  2.536347  1.378155  -0.338321  0.462388
1      1.191857   0.266151  0.166480  0.448154  0.060018  -0.082361
2     -1.358354  -1.340163  1.773209  0.379780  -0.503198  1.800499
3     -0.966272  -0.185226  1.792993  -0.863291  -0.010309  1.247203
4     -1.158233   0.877737  1.548718  0.403034  -0.407193  0.095921
...
284802 -11.881118  10.071785  -9.834783  -2.066656  -5.364473  -2.606837
284803  -0.732789  -0.055080  2.035030  -0.738589  0.868229  1.058415
284804  1.919565  -0.301254  -3.249640  -0.557828  2.630515  3.031260
284805  -0.240440   0.530483  0.702510  0.689799  -0.377961  0.623708
284806  -0.533413  -0.189733  0.703337  -0.506271  -0.012546  -0.649617

      V7      V8      V9      V10  ...      V21      V22  \
0      0.239599  0.098698  0.363787  0.090794  ...  -0.018307  0.277838
1     -0.078803  0.085102  -0.255425  -0.166974  ...  -0.225775  -0.638672
2      0.791461  0.247676  -1.514654  0.207643  ...   0.247998  0.771679
3      0.237099  0.377436  -1.387024  -0.054952  ...  -0.108300  0.005274
4      0.592941  -0.270533  0.817739  0.753074  ...  -0.009431  0.798278
...
284802 -4.918215  7.305334  1.914428  4.356170  ...   0.213454  0.111864
284803  0.024330  0.294869  0.584800  -0.975926  ...   0.214205  0.924384
284804  -0.296827  0.708417  0.432454  -0.484782  ...   0.232045  0.578229
284805  -0.686180  0.679145  0.392087  -0.399126  ...   0.265245  0.800049
284806  1.577006  -0.414650  0.486180  -0.915427  ...   0.261057  0.643078

      V23      V24      V25      V26      V27      V28  Class  \
0     -0.110474  0.066928  0.128539  -0.189115  0.133558  -0.021053      0
1      0.101288  -0.339846  0.167170  0.125895  -0.008983  0.014724      0
2      0.909412  -0.689281  -0.327642  -0.139097  -0.055353  -0.059752      0
3     -0.190321  -1.175575  0.647376  -0.221929  0.062723  0.061458      0
4     -0.137458  0.141267  -0.206010  0.502292  0.219422  0.215153      0
...
284802  1.014480  -0.509348  1.436807  0.250034  0.943651  0.823731      0
284803  0.012463  -1.016226  -0.606624  -0.395255  0.068472  -0.053527      0
284804  -0.037501  0.640134  0.265745  -0.087371  0.004455  -0.026561      0
284805  -0.163298  0.123205  -0.569159  0.546668  0.108821  0.104533      0
284806  0.376777  0.008797  -0.473649  -0.818267  -0.002415  0.013649      0

```

```

normalizedAmount
0      0.244964
1     -0.342475
2      1.160686
3      0.140534
4     -0.073403
...
284802  -0.350151
284803  -0.254117
284804  -0.081839
284805  -0.313249
284806   0.514355

```

```
[284807 rows x 30 columns]>
```

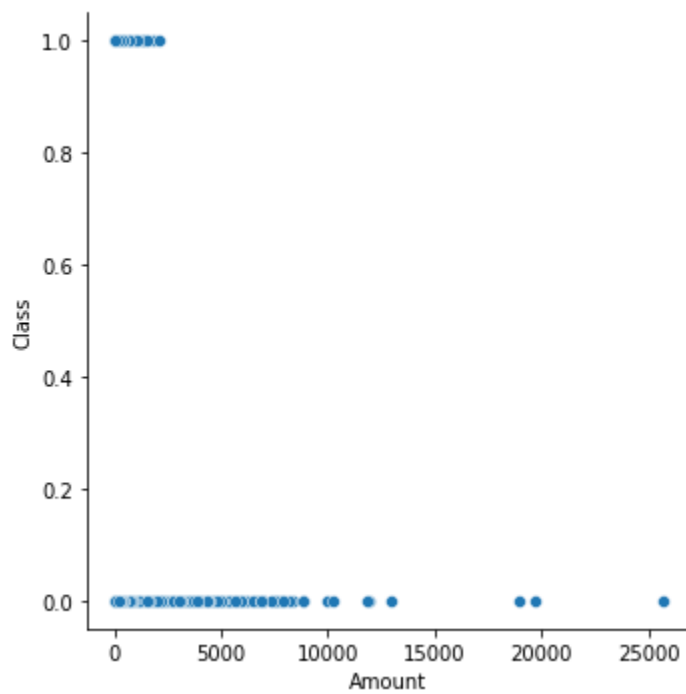
Data Visualization:

Plot : Amount vs class



```
import seaborn as sns
sns.relplot(x= 'Amount', y='Class',data=data)
```

<seaborn.axisgrid.FacetGrid at 0x7f9717a57c70>

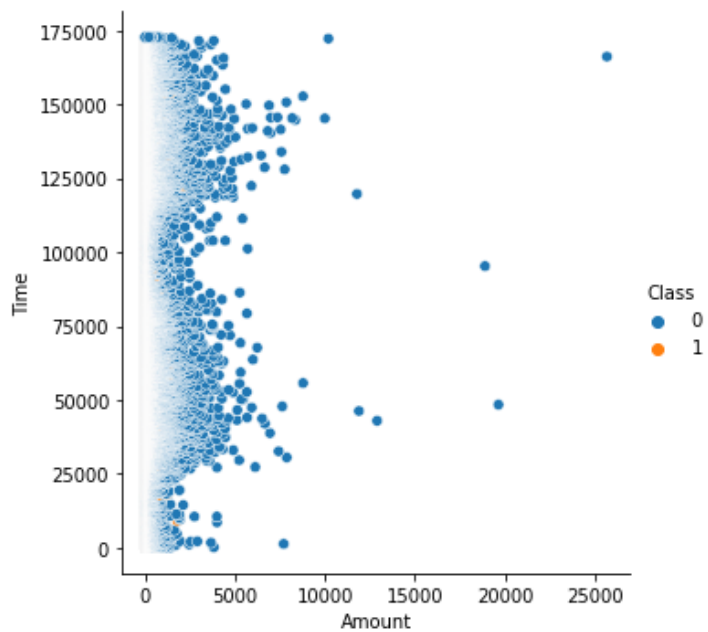


Plot: Time vs Amount



```
sns.relplot(x= 'Amount', y='Time', hue='Class',data=data)
```

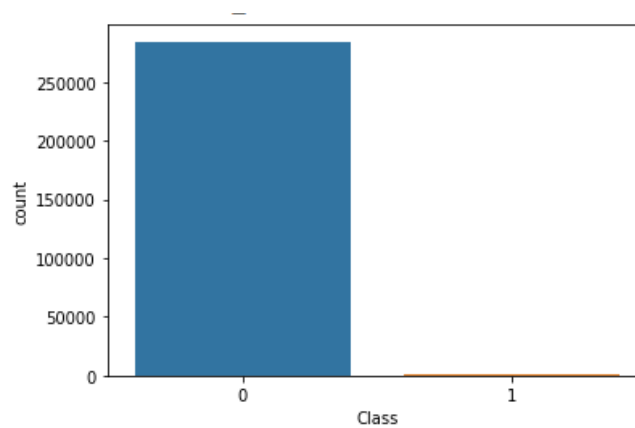
<seaborn.axisgrid.FacetGrid at 0x7f9717a0baf0>



countplot : fraud vs legit



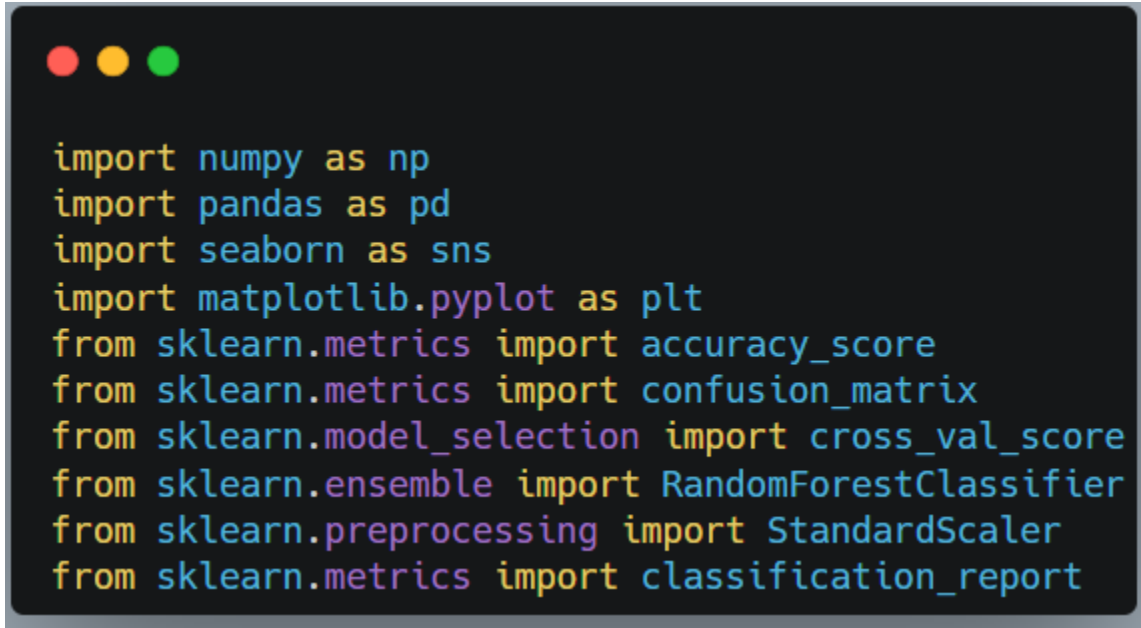
```
sns.countplot(data['Class'])
```



Python packages:

The python packages need to be used for implementation of Machine Learning algorithms pertaining to our projects are:

- **scikit-learn** - a library for machine learning and data mining
- **pandas** - a library for data manipulation and analysis
- **NumPy** - a library for scientific computing with Python
- **Matplotlib** - a plotting library for Python
- **Seaborn** - a visualization library based on Matplotlib



```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
```

This is how we have imported these packages and used in our Project

Learning Algorithms:

We have used two Machine Learning algorithm in our project,

- Random Forest classifier
- SVM(support Vector Machine)

Random Forest classifier:

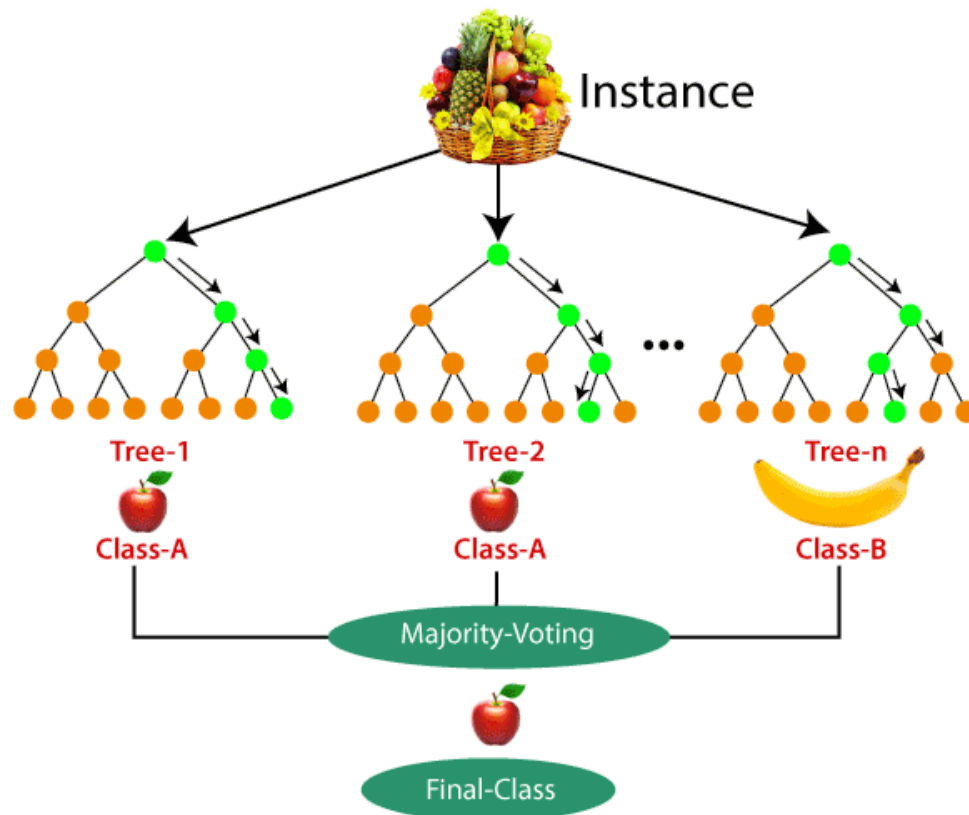
A random forest classifier is an ensemble method that is built using multiple decision trees. The basic idea behind the method is to construct multiple decision trees, where each tree is built using a different random subset of the training data, and then combine the predictions of all the trees to make a final prediction.

The decision trees are constructed using a technique called bagging, which stands for bootstrap aggregating. Bagging involves randomly selecting a subset of the training data (with replacement) to build each decision tree. This helps to reduce the variance in the predictions of the final model, as each tree is built using a different subset of the data.

Once the decision trees are built, they are combined to make a prediction for a new data point. This is done by taking a majority vote of the predictions made by all the trees in the forest (for classification problems) or averaging the predictions (for regression problems).

Random Forest classifiers also provide the feature importance measure, which is a measure of the relative importance of each feature in the dataset for making accurate predictions. The feature importance measure can be used to select a subset of the most important features for building a more compact and accurate model.

Random Forest is also less prone to overfitting. In Random Forest, multiple decision trees are created, so the overfitting problem is reduced.



The random forest ensemble technique has been selected to predict fruit types that are expected to contain a high number of positive samples.

SVM(support Vector Machine)

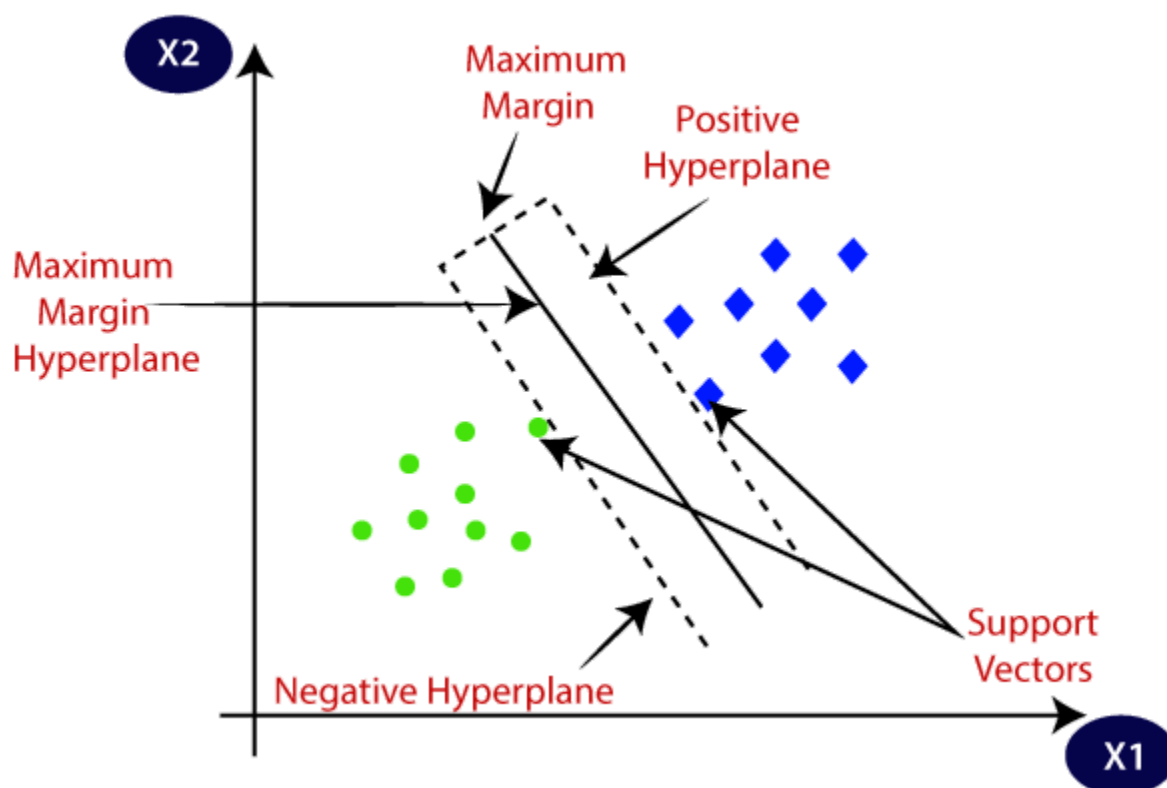
Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. The key idea behind SVMs is to find a boundary (or hyperplane) that separates the different classes in the training data, in a way that maximizes the margin, which is the distance between the boundary and the closest data points from each class.

SVMs are based on the idea of finding a hyperplane that best separates the data into different classes. A hyperplane is a line, plane or n-dimensional space that separates the data into different classes. The best hyperplane is one that has the maximum margin, which is the distance between the hyperplane and the closest data points from each class.

SVMs are particularly useful when the data is not linearly separable, which means that a straight line or plane cannot be used to separate the different classes. In such cases, SVMs use a technique called the kernel trick to transform the data into a higher-dimensional space, where it becomes linearly separable. Common kernels are linear, polynomial and radial basis function (RBF) kernels.

SVMs also have a regularization parameter, C which trades off correct classification of training examples against maximization of the decision function's margin.

SVMs are widely used in classification of images, bioinformatics and text classification.



The graph you can see describes the working of the support vector machine, the line is drawn to classify the blue and green samples as you can see in the graph.

Split your data into training, validation, and testing:

Splitting data into training, validation, and testing sets is a common practice in machine learning to ensure that the model is able to generalize well to new, unseen data. The data is typically split into three sets:

- **Training set:** This is the largest set of data and is used to train the model. The model learns the patterns and relationships in the training data, and the parameters of the model are adjusted to minimize the error.
- **Validation set:** The validation set is a smaller set of data that is used to evaluate the model during training. It is used to tune the hyperparameters of the model, such as the learning rate, the number of hidden units, or the regularization parameter. The goal is to find the best set of hyperparameters that result in the best performance on the validation set.
- **Testing set:** The testing set is a separate set of data that is used to evaluate the final performance of the model. It is used to provide an unbiased estimate of the model's performance on unseen data. The testing set should be kept separate from the training and validation sets to prevent overfitting and over-optimistic performance estimates.

Validation set is used to evaluate the performance of different models or different configurations of the same model. The goal of using a validation set is to avoid overfitting by keeping a portion of data for the model to test its ability to generalize to new data. When the model performs well on the validation set, it is likely to perform well on unseen data. The validation set is used to tune the hyperparameters of the model, such as the learning rate, the number of hidden units, or the regularization parameter.

```
from sklearn.model_selection import train_test_split

# Assuming that you have the data in 'X' and 'y' variables
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=5)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.23, random_state=5)
```

Use k-fold cross-validation to evaluate your ML algorithm code;

K-fold cross-validation is a method for evaluating the performance of a machine learning algorithm. It involves dividing the dataset into k subsets, or "folds," and training the algorithm on k-1 of the folds while evaluating it on the remaining one. This process is then repeated k times, with each fold serving as the evaluation set once. The

performance measure is then averaged across all k iterations to give an overall estimate of the algorithm's performance. To implement k-fold cross-validation in your code, you can use the KFold class from the **sklearn.model_selection** module.

RandomForest Algorithm:

```
from sklearn.model_selection import KFold, cross_val_score

k_folds = KFold(n_splits = 5)
scores_randomforest = cross_val_score(RandomClassifier, x, y, cv= k_folds)
print("Cross Validation Scores: ", scores_randomforest)
print("Average CV Score: ", scores_randomforest.mean())
print("Number of CV Scores used in Average: ", len(scores_randomforest))
```

```
Cross Validation Scores: [0.99827955 0.998578  0.9986482  0.99903443 0.99889398]
Average CV Score:  0.9986868318408941
Number of CV Scores used in Average:  5
```

SVM(support vector Machine):

```
from sklearn.model_selection import KFold, cross_val_score
k_folds = KFold(n_splits = 5)
scores_svm = cross_val_score(classifier, x, y, cv= k_folds)
print("Cross Validation Scores: ", scores_svm)
print("Average CV Score: ", scores_svm.mean())
print("Number of CV Scores used in Average: ", len(scores_svm))
```

```
, Cross Validation Scores: [0.99880622 0.99917489 0.99920999 0.99945577 0.99920999]
Average CV Score:  0.9991713699509834
Number of CV Scores used in Average:  5
```

Create models and estimate their accuracy on unseen data using the specified ML algorithms:

Random Forest Classifier:

```
from sklearn.ensemble import RandomForestClassifier
#from sklearn.datasets import make_classification
Randomclassifier = RandomForestClassifier(max_depth=2,n_estimators=100, random_state=5)
Randomclassifier.fit(x_train,y_train)
predicted_values = Randomclassifier.predict(x_test) |
```

SVM(Support Vector Machine):

```
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
predicted_values = classifier.predict(x_test) |
```

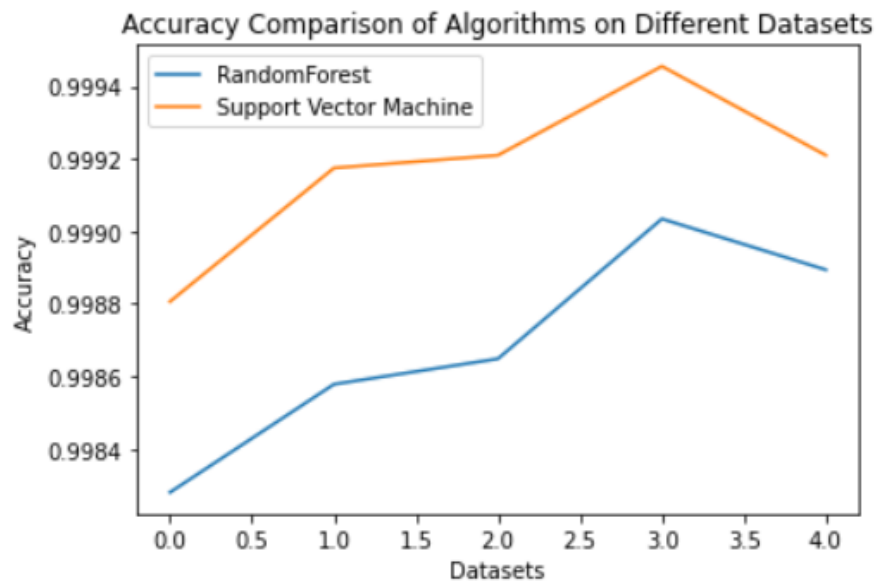
Plot a comparison graph showing the accuracy comparison of various algorithms on each of your datasets.

```
import matplotlib.pyplot as plt

# Create the line graph
plt.plot(scores_randomforest, label="RandomForest")
plt.plot(scores_svm, label="Support Vector Machine")

# Add axis labels, a title, and a legend
plt.xlabel("Datasets")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison of Algorithms on Different Datasets")
plt.legend()

# Display the graph
plt.show()
```



Make Predictions on the validation dataset. Plot accuracy and time for varying parameters:

Random Forest Classifier:

```

import time
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

# Initialize the lists to store the results
accuracy_randomforest = []
time_elapsed_randomforest = []

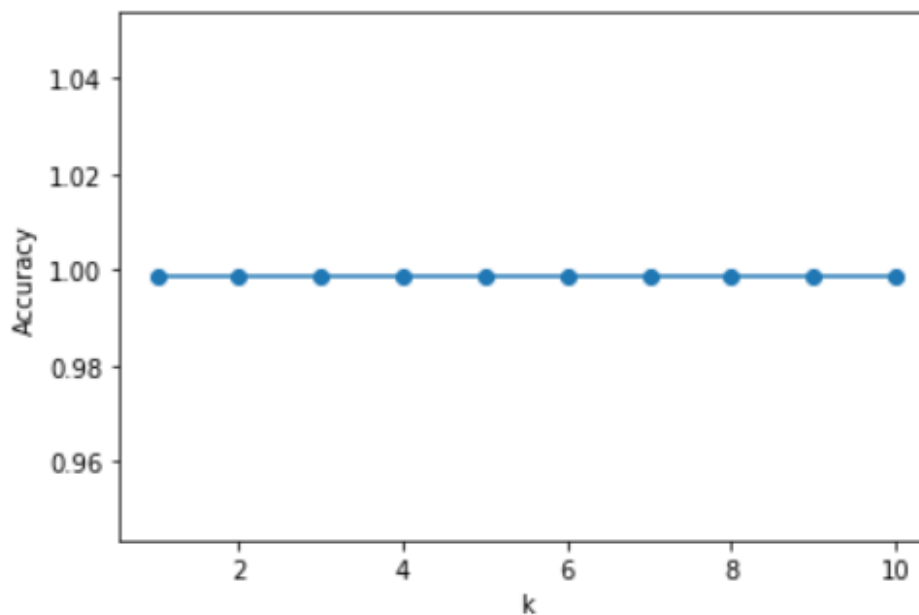
# Try different values of k
for k in range(1, 11):
    # Initialize the model
    Randomclassifier = RandomForestClassifier(max_depth=2, n_estimators=100, random_state=5)
    # Start the timer
    start_time = time.time()
    Randomclassifier.fit(x_train, y_train)
    val_pred = Randomclassifier.predict(x_val)
    accuracy_randomforest.append(accuracy_score(y_val, val_pred))
    time_elapsed_randomforest.append(time.time() - start_time)

# Plot the accuracy for different values of k
plt.plot(range(1, 11), accuracy_randomforest, marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.show()

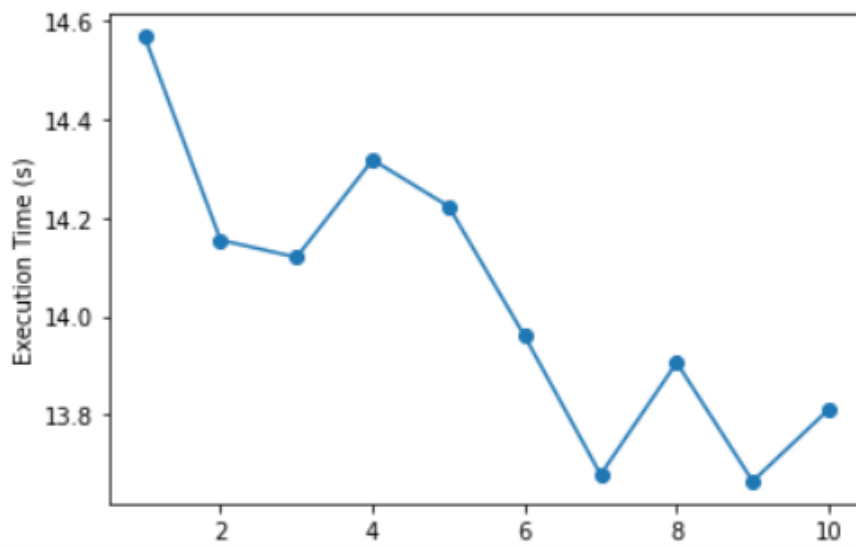
plt.plot(range(1, 11), time_elapsed_randomforest, marker='o')
plt.xlabel('k')
plt.ylabel('Execution Time (s)')
plt.show()

```


Plot accuracy for different values of K:



plot execution time for different values of K:



Support Vector Machine(SVM):



```
import time
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

# Initialize the lists to store the results
accuracy_svm = []
time_elapsed_svm = []

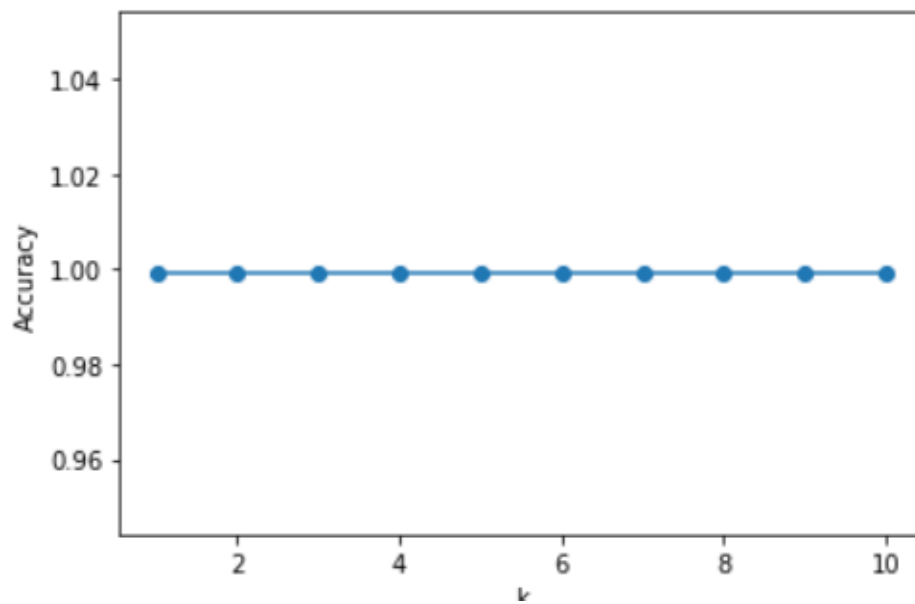
# Try different values of k
for k in range(1, 11):
    # Initialize the model
    classifier = SVC(kernel='linear', random_state=0)
    classifier.fit(x_train, y_train)

    # Start the timer
    start_time = time.time()
    classifier.fit(x_train, y_train)
    val_pred = classifier.predict(x_val)
    accuracy_svm.append(accuracy_score(y_val, val_pred))
    time_elapsed_svm.append(time.time() - start_time)

# Plot the accuracy for different values of k
plt.plot(range(1, 11), accuracy_svm, marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.show()

plt.plot(range(1, 11), time_elapsed_svm, marker='o')
plt.xlabel('k')
plt.ylabel('Execution Time (s)')
plt.show()
```

Plot accuracy for different values of K:



plot execution time for different values of K:

