Milestone2:

1.Preprocessing techniques:

1.1.we read csv and added to train

train=pd.read_csv("train.csv")

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y
FDA15	9.3	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	0
DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	2
FDN15	17.5	Low Fat	0.01676	Meat	141.618	OUT049	1999	Medium	Tier 1	0
FDX07	19.2	Regular	0	Fruits and	182.095	OUT010	1998		Tier 3	1
NCD19	8.93	Low Fat	0	Household	53.8614	OUT013	1987	High	Tier 3	0
FDP36	10.395	Regular	0	Baking Go	51.4008	OUT018	2009	Medium	Tier 3	2
FDO10	13.65	Regular	0.012741	Snack Foo	57.6588	OUT013	1987	High	Tier 3	0
FDP10		Low Fat	0.12747	Snack Foo	107.7622	OUT027	1985	Medium	Tier 3	3
FDH17	16.2	Regular	0.016687	Frozen Foc	96.9726	OUT045	2002		Tier 2	0
FDU28	19.2	Regular	0.09445	Frozen Foc	187.8214	OUT017	2007		Tier 2	0
FDY07	11.8	Low Fat	0	Fruits and	45.5402	OUT049	1999	Medium	Tier 1	0
FDA03	18.5	Regular	0.045464	Dairy	144.1102	OUT046	1997	Small	Tier 1	0
FDX32	15.1	Regular	0.100014	Fruits and	145.4786	OUT049	1999	Medium	Tier 1	0
FDS46	17.6	Regular	0.047257	Snack Foo	119.6782	OUT046	1997	Small	Tier 1	0
FDF32	16.35	Low Fat	0.068024	Fruits and	196.4426	OUT013	1987	High	Tier 3	0
FDP49	9	Regular	0.069089	Breakfast	56.3614	OUT046	1997	Small	Tier 1	0
NCB42	11.8	Low Fat	0.008596	Health anc	115.3492	OUT018	2009	Medium	Tier 3	2
FDP49	9	Regular	0.069196	Breakfast	54.3614	OUT049	1999	Medium	Tier 1	0
DRI11		Low Fat	0.034238	Hard Drink	113.2834	OUT027	1985	Medium	Tier 3	3

1.2. replace all empty cell with nan value to clean it next

```
train = train.replace(' ', np.nan)
```

1.3. we convert column label to string

```
train["Y"] = train["Y"].astype(str)
```

1.4.we find that column x2 and x9 have Nan value so we fill nan value by using :

```
train["X9"].fillna( method ='ffill', inplace = True)

train["X2"].fillna( method ='ffill', inplace = True)

train.isnull().sum()
```

1.5. in column 3 we find that its contain many word Have same mean for example :

```
Train["X3"].unique()
```

We find in dataset : "Low Fat" &"low fat"&"LF"→ " Low Fat"

Also: "Regular"&"reg"→"Regular"

So we use replace:

```
train["X3"]=train["X3"].replace(to_replace=["LF","low fat"],value="Low Fat")

train["X3"]=train["X3"].replace(to_replace=["reg"],value="Regular")
```

1.6. we handle zeros value that find in column X2&X4 Be get mean of column and use replace

```
m=train["X4"].mean()

train["X4"]=train["X4"].replace(to_replace=0,value=m)
```

1.7. we drop column X1: item id &X7: store id

```
train.drop("X1",inplace=True,axis=1)

train.drop("X7",inplace=True,axis=1)
```

train - DataFrame

Index	X2	X3	X4	X5	X6	X8	X9	X10	Y
0	9.3	0	0.0160473	Dairy	249.809	1999	1	0	0
1	5.92	1	0.0192782	Soft Drinks	48.2692	2009	1	2	2
2	17.5	0	0.0167601	Meat	141.618	1999	1	0	0
3	19.2	1	0.066132	Fruits and Vegetables	182.095	1998	1	2	1
4	8.93	0	0.066132	Household	53.8614	1987	0	2	0
5	10.395	1	0.066132	Baking Goods	51.4008	2009	1	2	2
6	13.65	1	0.0127411	Snack Foods	57.6588	1987	0	2	0
7	13.65	0	0.12747	Snack Foods	107.762	1985	1	2	3
8	16.2	1	0.0166871	Frozen Foods	96.9726	2002	1	1	0
9	19.2	1	0.0944496	Frozen Foods	187.821	2007	1	1	0
10	11.8	0	0.066132	Fruits and Vegetables	45.5402	1999	1	0	0
11	18.5	1	0.0454638	Dairy	144.11	1997	2	0	0
12	15.1	1	0.100014	Fruits and Vegetables	145.479	1999	1	0	0
13	17.6	1	0.0472573	Snack Foods	119.678	1997	2	0	0
14	16.35	0	0.0680243	Fruits and Vegetables	196.443	1987	0	2	0
15	9	1	0.069089	Breakfast	56.3614	1997	2	0	0
16	11.8	0	0.00859605	Health and Hygiene	115.349	2009	1	2	2
17	9	1	0.0691964	Breakfast	54.3614	1999	1	0	0
18	9	0	0.0342377	Hard Drinks	113.283	1985	1	2	3

2.Analysis on dataset:

2.1.we use label encoder foe column **(to avoid large number of feature)**

```
label=preprocessing.LabelEncoder()

train["X3"]=label.fit_transform(train["X3"])

train["X9"]=label.fit_transform(train["X9"])

train["X10"]=label.fit_transform(train["X10"])
```

2.2. we drop column X5: item category because we drop column item id also if we use label encoder it will make Priority

```
train.drop("X5",inplace=True,axis=1)
```

2.3. here we will change column name to make it easy to understand :

```
train.rename(columns={'X2': 'Weight of Item', 'X3': 'Amount of Fats in Item','X4': 'area allocated for item in store ','X6': 'Item Price','X8': 'Store Establishment Year','X9': 'Store Size','X10': 'Store Location Type','Y':'label'}, inplace=True)
```

2.4.apply correlation matrix to show correlation between feature and label

```
x=train.corr()
```

x - DataFrame

Index	Weight of Item	unt of Fats in	cated for item	Item Price	Establishment	Store Size	re Location Ty
Weight of Item	1	-0.0404948	-0.0221904	0.0256034	0.0226588	0.018763	-0.00420684
Amount of Fats in Item	-0.0404948	1	0.0460077	-0.019244	-0.00437766	-0.00493849	0.00582693
area allocated for item in store	-0.0221904	0.0460077	1	-0.0141835	-0.104073	0.0523311	-0.00674896
Item Price	0.0256034	-0.019244	-0.0141835	1	-0.00723278	-0.00545247	0.00175421
Store Establishment Year	0.0226588	-0.00437766	-0.104073	-0.00723278	1	0.218103	-0.0894962
Store Size	0.018763	-0.00493849	0.0523311	-0.00545247	0.218103	1	-0.517552
Store Location Type	-0.00420684	0.00582693	-0.00674896	0.00175421	-0.0894962	-0.517552	1

2.5. Second drop column that have small relation with my label

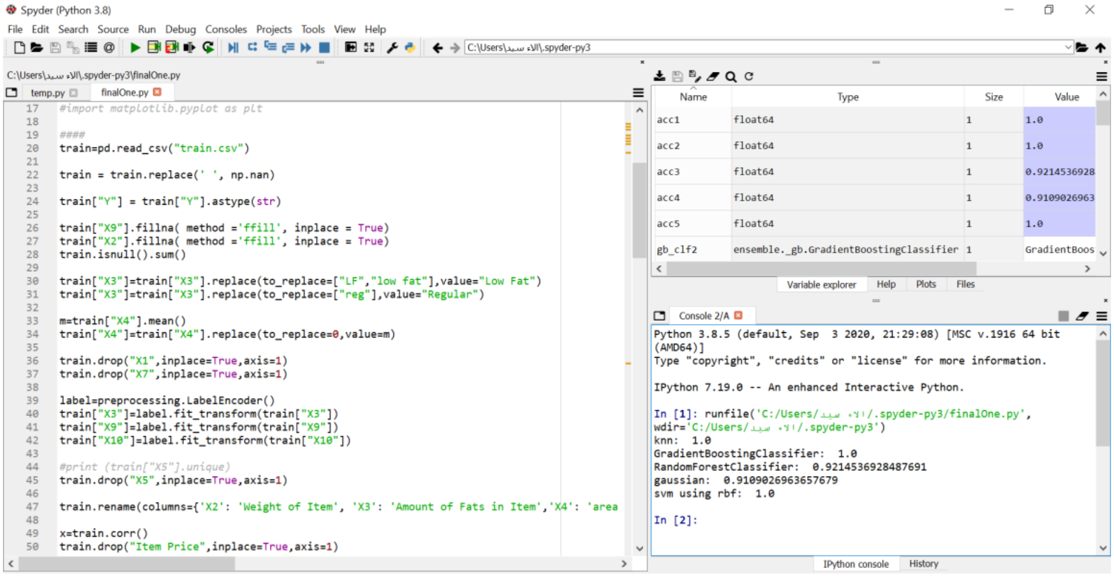
```
train.drop("Item Price",inplace=True,axis=1)
```

3.Split dataset into train and test:

```
x1=train[['Weight of Item','Amount of Fats in Item','area allocated for item in store ','Store Establishment Year','Store Size','Store Location Type']]
y=train['label']
x_train, x_test, y_train, y_test = train_test_split(x1, y,test_size = 0.1,random_state = 0)
```

4.classification techniques:

4.1.Accuracy score of our techniques in run time



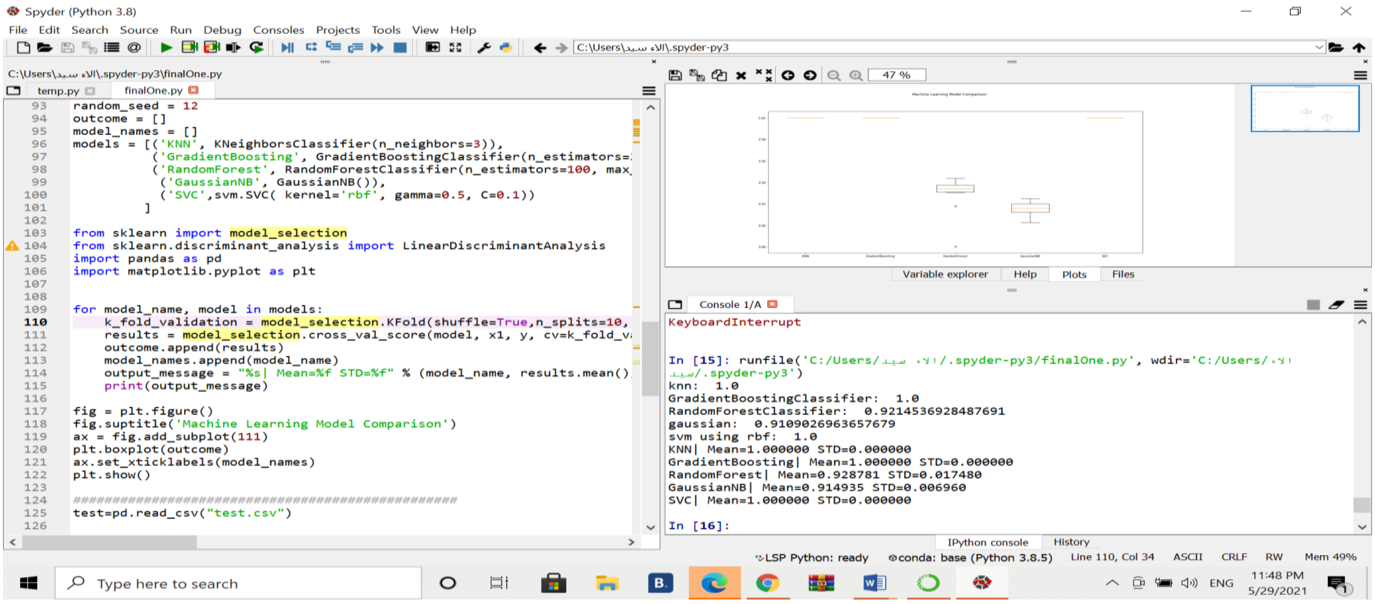
The output label of each technique:

F	E	D	C	B	A	
label GaussianNB	label RandomForestClassifie	label KNeighborsClassifier	label GradientBoostingClassifier	label SVM	row_id	1
0	0	0	0	0	0	2
0	0	0	0	0	1	3
0	0	1	1	1	2	4
0	0	0	0	0	3	5
3	3	3	3	3	4	6
0	0	0	0	0	5	7
2	2	2	2	2	6	8
3	3	3	3	3	7	9
0	0	0	0	0	8	10
0	0	0	0	0	9	11
0	0	0	0	0	10	12
0	0	0	0	0	11	13
1	1	1	1	1	12	14
0	0	0	0	0	13	15
0	0	0	0	0	14	16
0	0	0	0	0	15	17
0	0	0	0	0	16	18
2	2	2	2	2	17	19
3	3	3	3	3	18	20
1	0	1	1	1	19	21
0	0	0	0	0	20	22
0	0	0	0	0	21	23
0	0	0	0	0	22	24
0	1	1	1	1	23	25
0	0	0	0	0	24	26
0	0	1	1	1	25	27
0	0	0	0	0	26	28
0	0	0	0	0	27	29
0	0	0	0	0	28	30
0	0	0	0	0	29	31
0	1	1	1	1	30	32
2	2	2	2	2	31	33
2	2	2	2	2	32	34
3	3	3	3	3	33	35
0	0	0	0	0	34	36
3	3	3	3	3	35	37
3	3	3	3	3	36	38
0	0	1	1	1	37	39
0	0	0	0	0	38	40
3	3	3	3	3	39	41
2	2	2	2	2	40	42
0	0	1	1	1	41	43
0	0	0	0	0	42	44
2	2	2	2	2	43	45

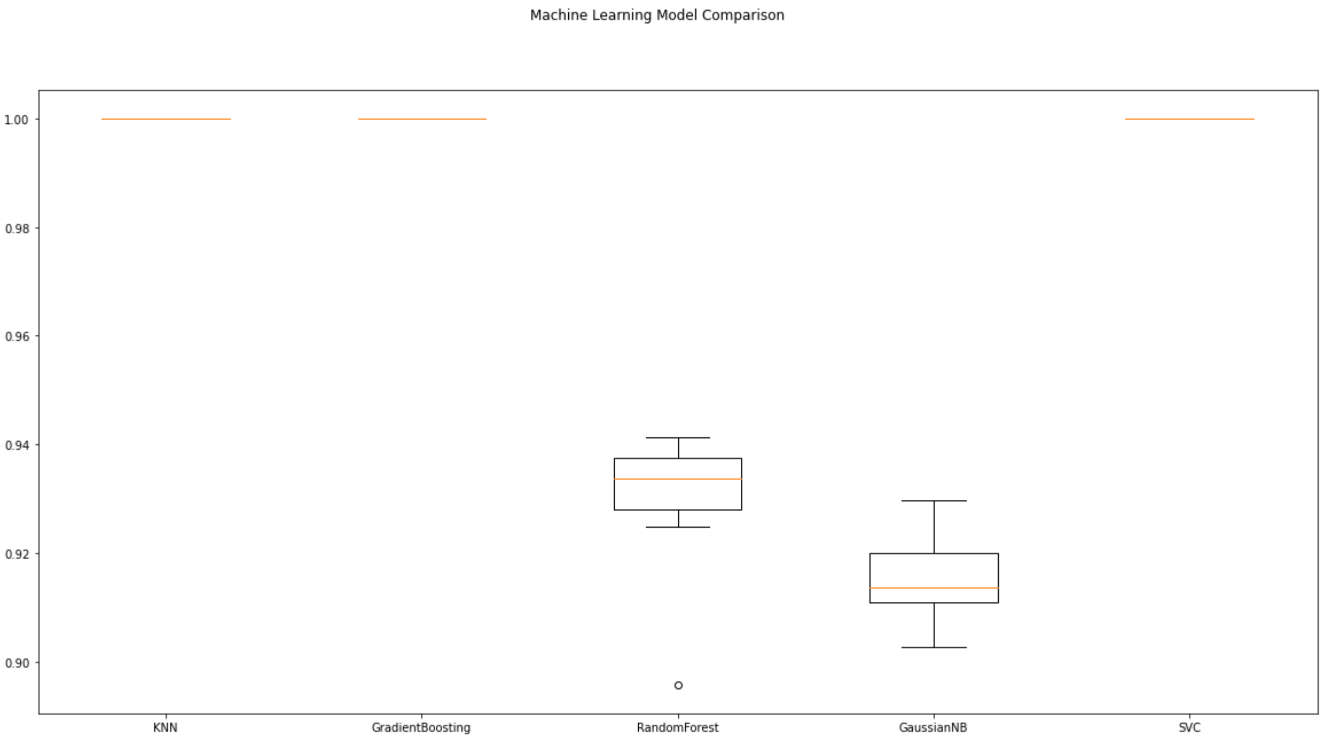
5.plot the accuracy of different models

5.1. code & screen:

```
random_seed = 12

outcome = []

model_names = []

models = [('KNN', KNeighborsClassifier(n_neighbors=3)),

    ('GradientBoosting', GradientBoostingClassifier(n_estimators=20,

            learning_rate=0.5,max_features=2, max_depth=2, random_state=0)),

    ('RandomForest', RandomForestClassifier(n_estimators=100, max_depth=2,

            random_state=0)),

    ('GaussianNB', GaussianNB()),

    ('SVC',svm.SVC(kernel='rbf', gamma=0.5, C=0.1))

    ]


from sklearn import model_selection

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

import pandas as pd

import matplotlib.pyplot as plt


for model_name, model in models:

  k_fold_validation = model_selection.KFold(shuffle=True ,n_splits=10, random_state=random_seed)

  results = model_selection.cross_val_score(model, x1, y, cv=k_fold_validation, scoring='accuracy')

  outcome.append(results)

  model_names.append(model_name)

  output_message = "%s| Mean=%f STD=%f" % (model_name, results.mean(), results.std())

  print(output_message)
```

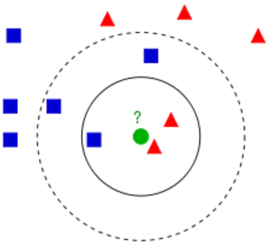



```
fig = plt.figure()

fig.suptitle('Machine Learning Model Comparison')

ax = fig.add_subplot(111)

plt.boxplot(outcome)

ax.set_xticklabels(model_names)

plt.show()
```



5.2. compared different models we used:

5.2.1. *K Neighbors Classifier*



KNeighborsClassifier implements classification based on voting by nearest k-neighbors of target point Example of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)

# " n_neighbors int, default=5 .Number of neighbors to use by default for kneighbors queries."

model.fit(x_train, y_train)

#" Fit the k-nearest neighbors classifier from the training dataset. "

KNN_pred =model.predict(x_test)

#" Predict the class labels for the provided data. "

acc1=accuracy_score(y_test, KNN_pred)

print("knn: ",acc1)
```

2. Gradient boosting classifiers

Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting. Gradient boosting models are becoming popular because of their effectiveness at classifying complex datasets, and have recently been used to win many Kaggle data science competitions.

```
from sklearn.ensemble import GradientBoostingClassifier

gb_clf2 = GradientBoostingClassifier(n_estimators=20, learning_rate=0.5, max_features=2, max_depth=2, random_state=0)

""""
```

" **n_estimators** : int (default=100) The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

learning_rate : float, optional (default=0.1) learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.

max_features : int, float, string or None, optional (default="auto")

max_depth : integer, optional (default=3) maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables. Ignored if max_samples_leaf is not None.

"

""""

gb_clf2.fit(x_train, y_train)

#" Fit the gradient boosting model. "

GBC_pred = gb_clf2.predict(x_test)

#" Predict class for X. "

acc2=accuracy_score(y_test, GBC_pred)

print("GradientBoostingClassifier: ",acc2)

3. A random forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0)

""""

" **n_estimators** ***int, default=100*** The number of trees in the forest

max_depth ***int, default=None*** The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

random_state ***int, RandomState instance or None, default=None*** Controls both the randomness of the bootstrapping of the samples used when building trees (if bootstrap=True) and the sampling of the features to consider when looking for the best split at each node (if max_features < n_features). See Glossary for details."

""""

RF.fit(x_train, y_train)

#" Build a forest of trees from the training set (X, y)."

```
RFC_pred= RF.predict(x_test)

#" Predict class for X."

acc3=accuracy_score(y_test, RFC_pred)

print("RandomForestClassifier: ",acc3)
```

4. Gaussian Naive Bayes

Naive Bayes are a group of supervised machine learning classification algorithms based on the **Bayes theorem**. It is a simple classification technique, but has high functionality. They find use when the dimensionality of the inputs is high. Complex classification problems can also be implemented by using Naive Bayes Classifier.

Bayes Theorem:

Bayes Theorem can be used to calculate conditional probability. Being a powerful tool in the study of probability, it is also applied in Machine Learning.

The Formula For Bayes' Theorem Is

$$P(A|B) = \frac{P(A \bigcap B)}{P(B)} = \frac{P(A) \cdot P(B|A)}{P(B)}$$

where:

$P(A) =$ The probability of A occurring

$P(B) =$ The probability of B occurring

$P(A|B) =$ The probability of A given B

$P(B|A) =$ The probability of B given A

$P\left(A \bigcap B\right)) =$ The probability of both A and B occurring

Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a gaussian distribution i.e, normal distribution.

```
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier

model = GaussianNB()

model.fit(x_train, y_train)
```

""

" Fit Gaussian Naive Bayes according to X, y

Xarray-like of shape (n_samples, n_features) Training vectors, where n_samples is the number of samples and n_features is the number of features.

yarray-like of shape (n_samples,) Target values."

```
"""
GNB_pred =model.predict(x_test)
#" perform classification on an array of test vectors X."
acc4=accuracy_score(y_test, GNB_pred)
print("gaussian: ",acc4)
```

5. SVM Classifier and RBF Kernel

Support Vector Machines (SVMs) are most frequently used for solving classification problems, which fall under the supervised machine learning category. In machine learning, the radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms. In particular, it is commonly used in support vector machine classification.[1]

The RBF kernel on two samples x and x', represented as feature vectors in some input space, is defined as[2]

```
from sklearn import svm
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(x_train, y_train)
"""
```

" **kernel*{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*** Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).

gamma*{'scale', 'auto'} or float, default='scale'* Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
coef0*float, default=0.0* Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

Fit the SVM model according to the given training data."

```
"""
rbf_pred = rbf.predict(x_test)
```

```
#" Perform classification on samples in X."

acc5=accuracy_score(y_test, rbf_pred)

print("svm using rbf: ",acc5)
```

Conclusion

In this phase of project we learn more about how to clean dataset and prepare it to run in model and this part need to try more than one scenario With taking careful to avoid overfiting or making large error try and then try to choose best model first step you need to do in machine learning project is see your dataset and learn more about it and what column may be much related to our label