Naive Bayesian Classifier

Exp : 1
Date : August 18 2020

Aim :

Write a program to implement the naive bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. Feature set <Domestic/International, Climate, Bet value, Win/Loss>.

Procedure :

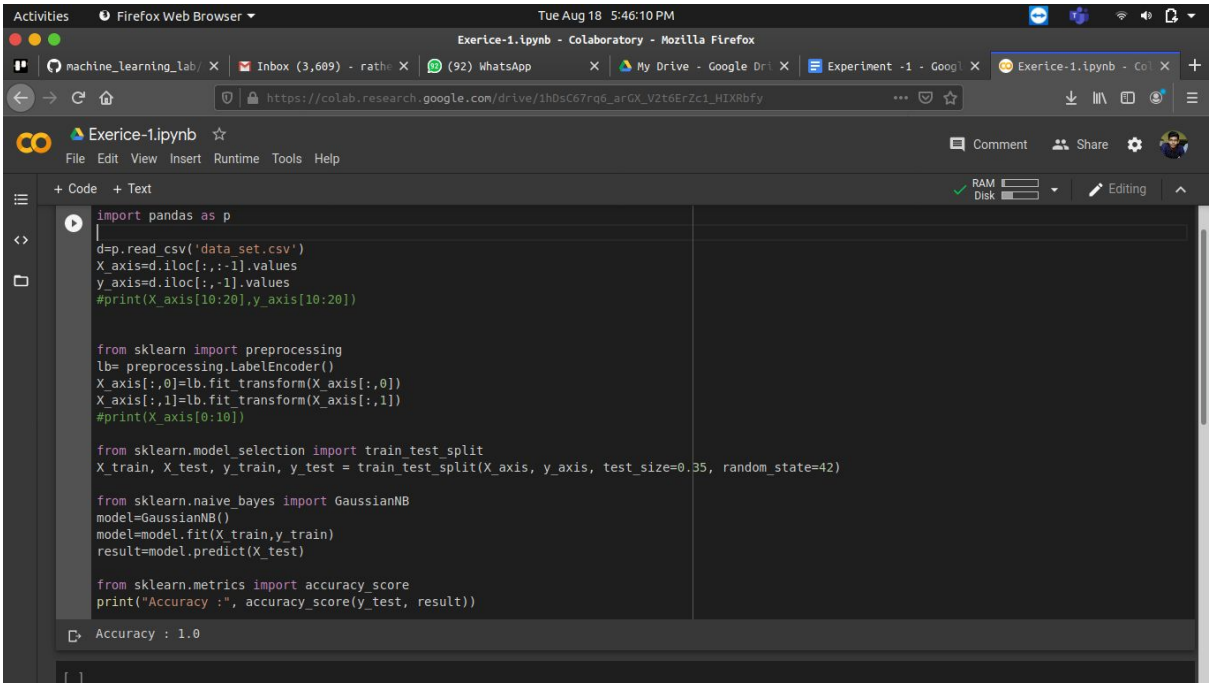
- Open Google Colab, and create a notebook file.
- Importing the necessary packages includes pandas and numpy, naive bayes from sklearn in the GaussianNB package.
- Import the dataset from data_set.csv file.
- Convert all the data into data frames using numpy arrays.
- Then store the independent values in X_data and y_data respectively.
- Convert text values into numerical values using labelencoder.
- Use the test_train module to split the dataset into test and train_data.
- Train the model using fit and predict the model using predict functionality and store the predicted results in a variable.
- Calculate the accuracy and print the result.

Code :

```
import numpy as np
import pandas as p

d=p.read_csv('data_set.csv')
X_axis=d.iloc[:,:-1].values
y_axis=d.iloc[:,-1].values

from sklearn import preprocessing
lb= preprocessing.LabelEncoder()
X_axis[:,0]=lb.fit_transform(X_axis[:,0])
X_axis[:,1]=lb.fit_transform(X_axis[:,1])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_axis, y_axis,
test_size=0.35, random_state=42)

from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model=model.fit(X_train,y_train)
result=model.predict(X_test)
from sklearn.metrics import accuracy_score
print("Accuracy :", accuracy_score(y_test, result))
```

Naive Bayesian Classifier

Output :



	Marks	Signature
Procedure		
Observation		
Viva		
Result		
Record		

Result : Thus the model is trained and accuracy is attained using Naive Bayesian classifier.

K-Nearest Neighbors

Exp : 2
Date : August 24 2020

Aim :

Write a program to classify data using the *K-Nearest Neighbors* algorithm in Python. Use **breast cancer dataset** from the **sklearn.datasets module.**

Procedure :

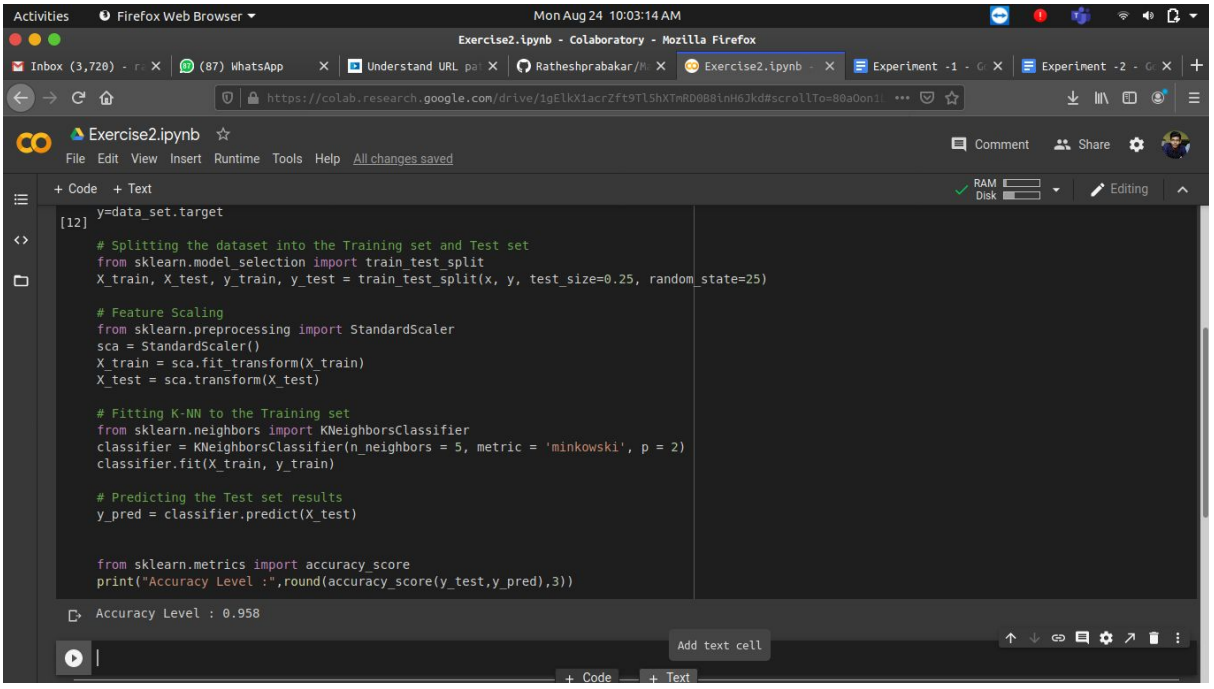
- Open Google Colab, and create a notebook file.
- Importing the necessary packages includes numpy and sklearn.
- Import the dataset from sklearn.datasets().
- Convert the dataset into the training and test set.
- Then, Implement standard scaling techniques to transform the training and test set.
- Now implement the KNeighborsClassifier to our training set.
- Use the accuracy score function to predict the accuracy of the dataset.
- Rounded upto 3 decimals and print the accuracy value.

Code :

```
# Importing the libraries
import numpy as np
from sklearn.datasets import load_breast_cancer
data_set = load_breast_cancer()
x=data_set.data
y=data_set.target
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.25, random_state=25)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sca = StandardScaler()
X_train = sca.fit_transform(X_train)
X_test = sca.transform(X_test)
# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric =
'minkowski', p = 2)
classifier.fit(X_train, y_train)
# Predicting the Test set results
y_pred = classifier.predict(X_test)
from sklearn.metrics import accuracy_score
print("Accuracy Level :",round(accuracy_score(y_test,y_pred),3))
```

K-Nearest Neighbors

Output :



	Marks	Signature
Procedure		
Observation		
Viva		
Result		
Record		

Result : Thus the model is trained and accuracy is attained using K- Nearest Neighbors algorithm.

Decision Tree Classifier

Exp : 3
Date : August 25 2020

Aim :

Write a python program to implement **Decision Tree Classifier**, visualize and evaluate the tree for the diabetes dataset using Scikit-Learn Package.

Procedure :

- Open Google Colab, and create a notebook file.
- Importing the necessary packages includes numpy and sklearn.
- Import the dataset using pandas.
- Convert the dataset into the training and test set.
- Then, Implement standard scaling techniques to transform the training and test set.
- Now implement the Decision Tree Classifier to our training set.
- Use the accuracy score function to predict the accuracy of the dataset.
- Plot the values into the decision tree.
- Rounded upto 3 decimals and print the accuracy value.

Code :

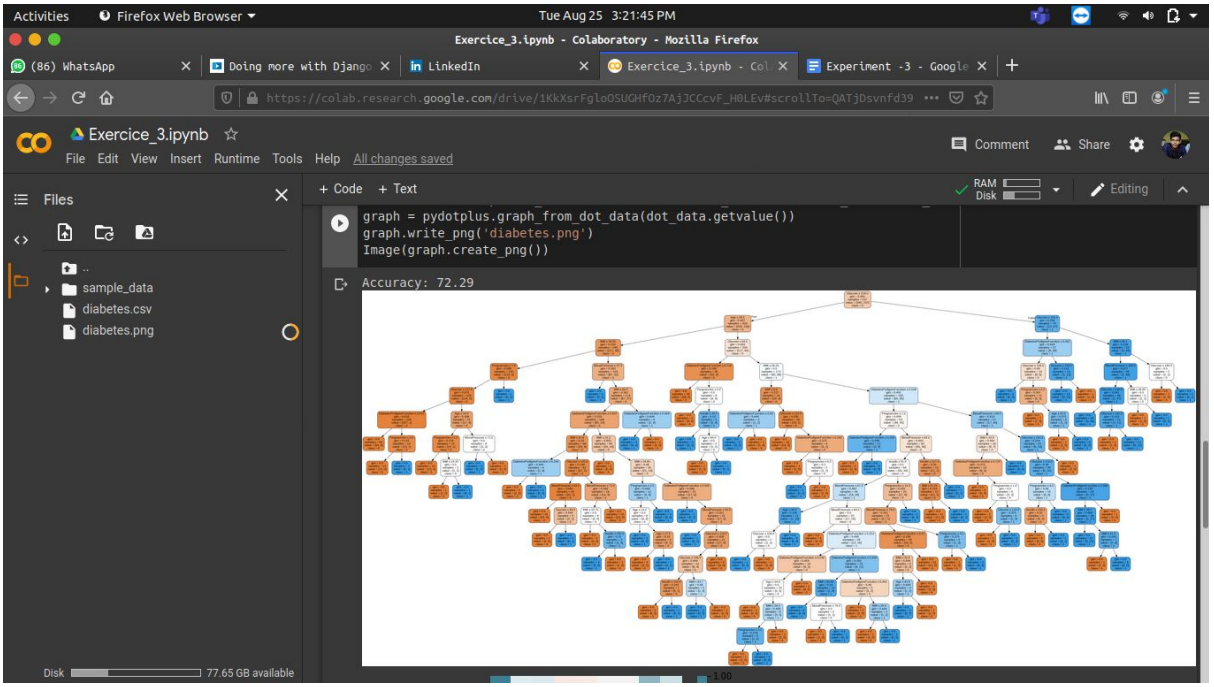
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

p =  pd.read_csv("diabetes.csv")
p.head()

import seaborn as sns
corr = p.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);

# feature selection
```

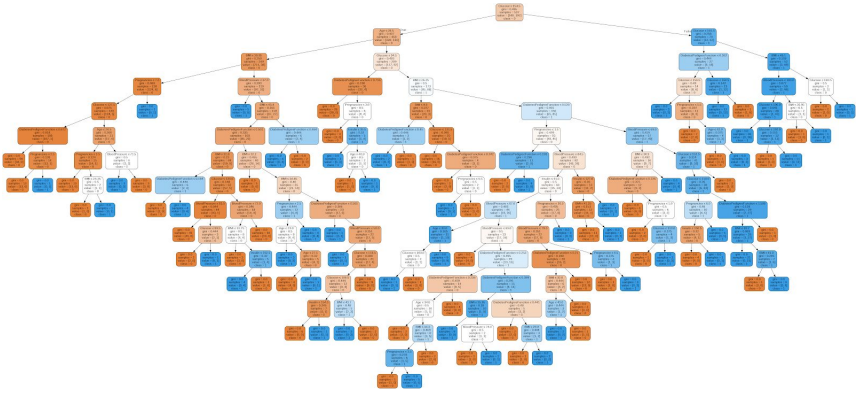
Decision Tree Classifier

```
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose',
'BloodPressure', 'DiabetesPedigreeFunction']
x = p[feature_cols]
y = p.Outcome

# split data
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size =
0.3, random_state=5)

# build model
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train, Y_train)

# predict
y_pred = classifier.predict(X_test)
#print(y_pred)

# confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, y_pred)
#print(confusion_matrix(Y_test, y_pred))

# accuracy
print("Accuracy:", round(metrics.accuracy_score(Y_test,y_pred)*100,2))

from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names =
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Decision Tree Classifier

Output :





	Marks	Signature
Procedure		
Observation		
Viva		
Result		
Record		

Result : Thus the model is trained, visualized and the accuracy is attained using Decision Tree Classifier algorithm.

Simple Linear Regression

Exp No : 4
Date : August 29 2020
Register No : 1718131

Aim :

Write a python program to implement,visualize and evaluate the performance metrics of **simple linear regression**. *Note* : Predict the maximum temperature taking input feature as the minimum temperature using Weather Conditions in **World War Two dataset** from **Kaggle.**

Procedure :

- Open Google Colab, and create a notebook file.
- Importing the necessary packages includes numpy,pandas, matplotlib, seaborn, sklearn.
- Import the LinearRegression model from sklearn package.
- Upload the World War II dataset in CSV format.
- Now plot the Minimum and Maximum temperature values which are obtained from the dataset using matplotlib.
- Now we are training our model by splitting the dataset into the training and test set.
- Predict the values using our trained model.
- Visualize the results using matplotlib.
- Calculate the Mean absolute value, Mean squared value, Root mean squared value.
- Rounded upto 3 decimals and print the values.

Code :

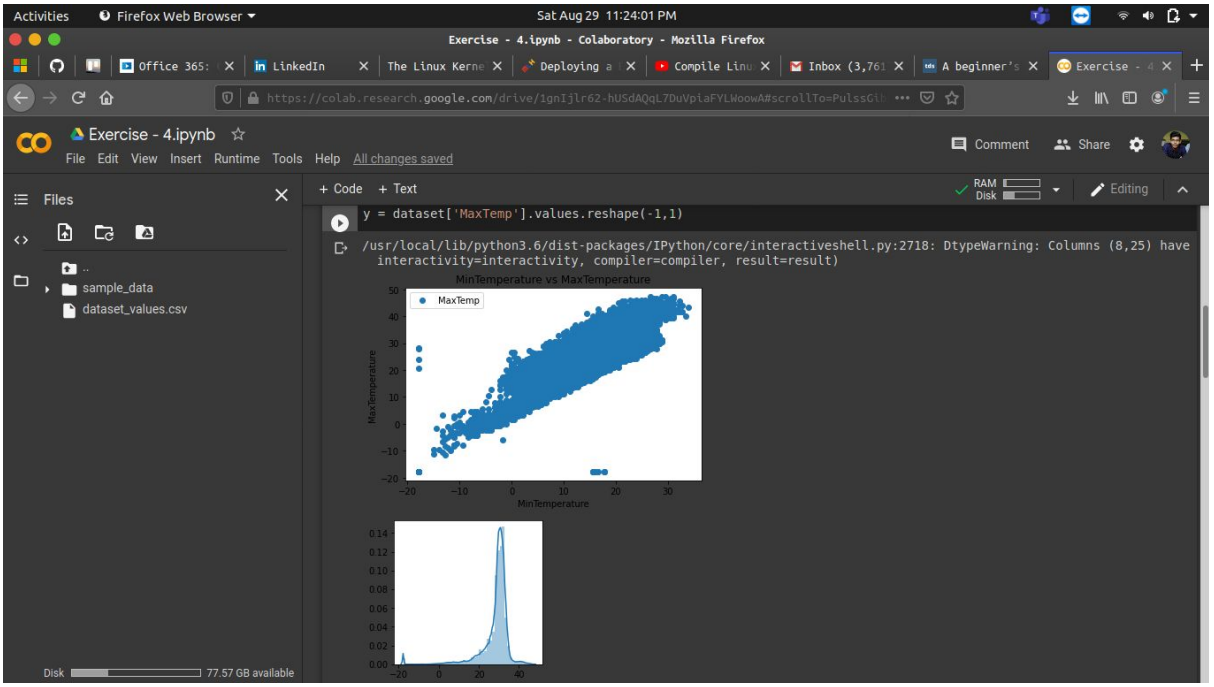
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as pl
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline

dataset = pd.read_csv('dataset_values.csv')
dataset.plot(x='MinTemp', y='MaxTemp', style='o')
pl.title('MinTemperature vs MaxTemperature')
pl.xlabel('MinTemperature')
pl.ylabel('MaxTemperature')
pl.show()
pl.figure(figsize=(3,3))
pl.tight_layout()
seabornInstance.distplot(dataset['MaxTemp'])
```

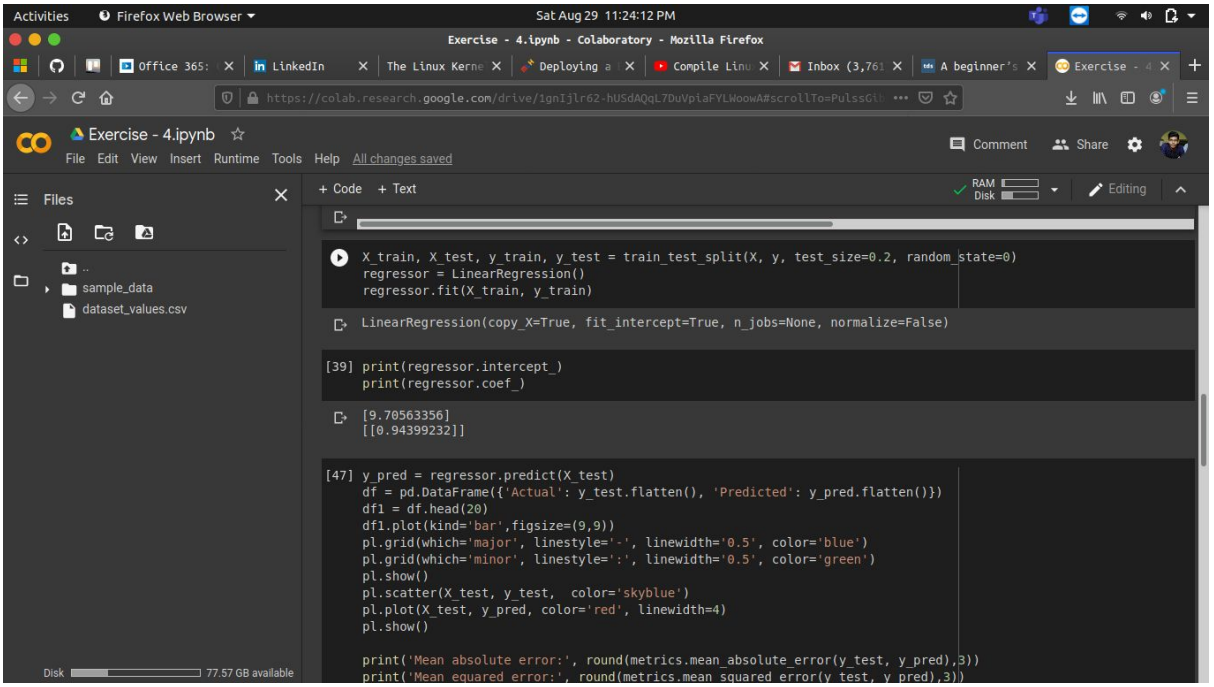
Simple Linear Regression

```
X = dataset['MinTemp'].values.reshape(-1,1)
y = dataset['MaxTemp'].values.reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)

print(regressor.intercept_)
print(regressor.coef_)

y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted':
y_pred.flatten()})

df1 = df.head(20)
df1.plot(kind='bar',figsize=(4,4))
pl.grid(which='major', linestyle='-', linewidth='0.5', color='blue')
pl.grid(which='minor', linestyle=':', linewidth='0.5', color='green')
pl.show()

pl.scatter(X_test, y_test,  color='skyblue')
pl.plot(X_test, y_pred, color='red', linewidth=3)
pl.show()

print('Mean absolute error:', round(metrics.mean_absolute_error(y_test,
y_pred),3))
print('Mean squared error:', round(metrics.mean_squared_error(y_test,
y_pred),3))
print('Root mean squared error:',
round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)),4))
```

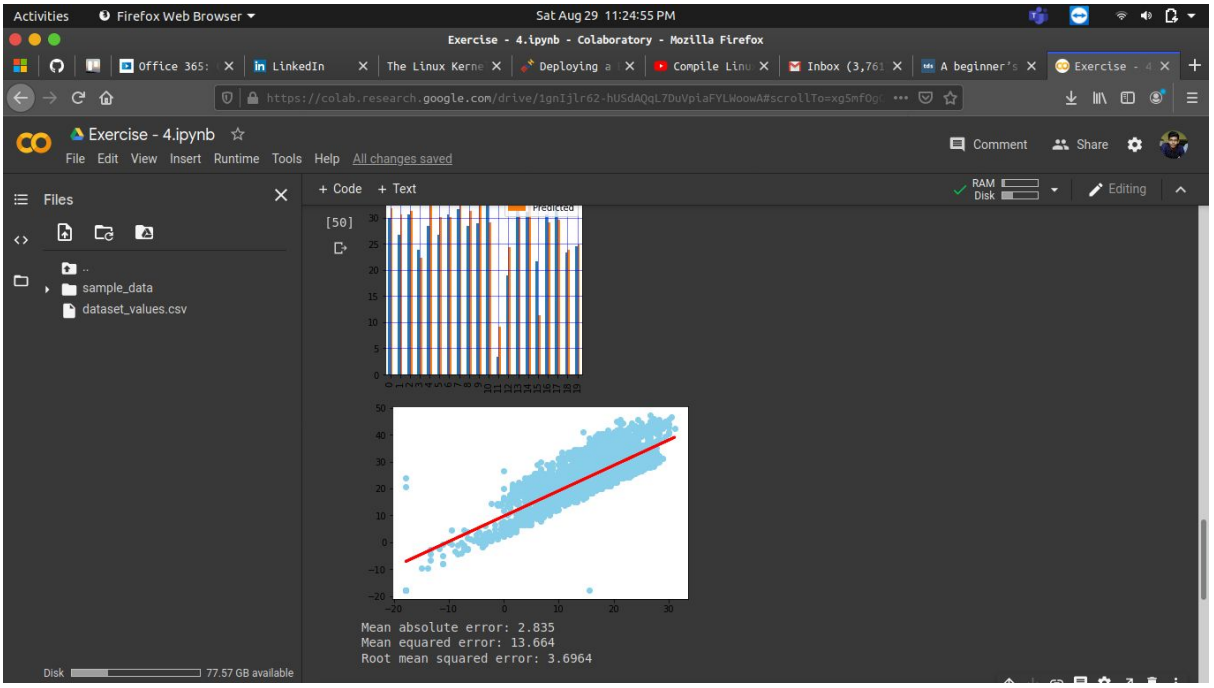
Simple Linear Regression

Output :





Simple Linear Regression



	Marks	Signature
Procedure		
Observation		
Viva		
Result		
Record		

Result :

Thus the **simple linear regression** is implemented, visualized and the performance is evaluated successfully.

Multiple Linear Regression

Exp No : 5
Date : September 1 2020
Register No : 1718131

Aim :

Write a python program to implement **multiple linear regression** for **Red Wine Quality dataset** from kaggle. Consider input features like fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density pH, sulphates, alcohol. and predict the quality of the wine. Also evaluate the performance of your algorithm.

Procedure :

- Open Google Colab, and create a notebook file.
- Importing the necessary packages includes numpy,pandas, matplotlib, seaborn, sklearn.
- Import the LinearRegression model from sklearn package.
- Upload the Red Wine Quality dataset in CSV format.
- Now plot the values which are obtained from the dataset using matplotlib.
- Now we are training our model by splitting the dataset into the training and test set.
- Predict the values using our trained model.
- Visualize the results using matplotlib.
- Calculate the Mean squared value, Root mean squared value and evaluate the values.
- Rounded upto 3 decimals and print the values.

Code :

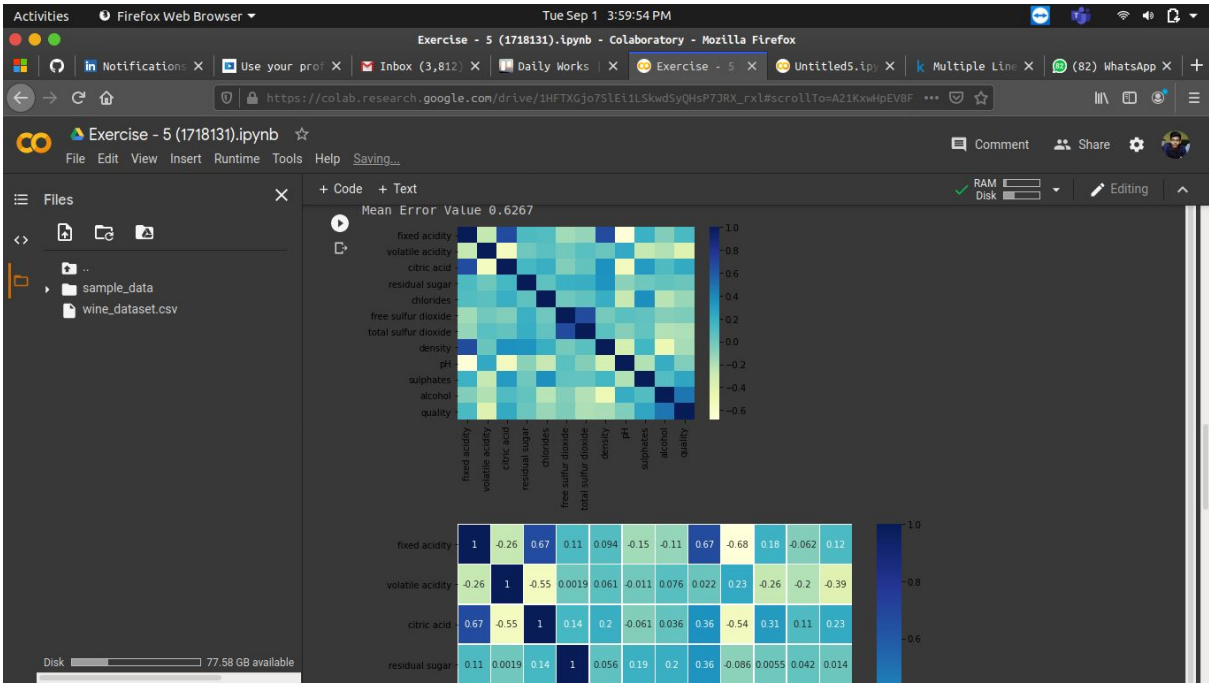
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
import seaborn as sns
from sklearn.model_selection import train_test_split
import statsmodels.formula.api as sm
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('wine_dataset.csv')

corr_matrix = dataset.corr()
sns.heatmap(corr_matrix,cmap="YlGnBu")

fig, ax = pl.subplots(figsize=(10,10))
sns.heatmap(corr_matrix,cmap="YlGnBu", annot=True, linewidths=.5, ax =
ax)
```

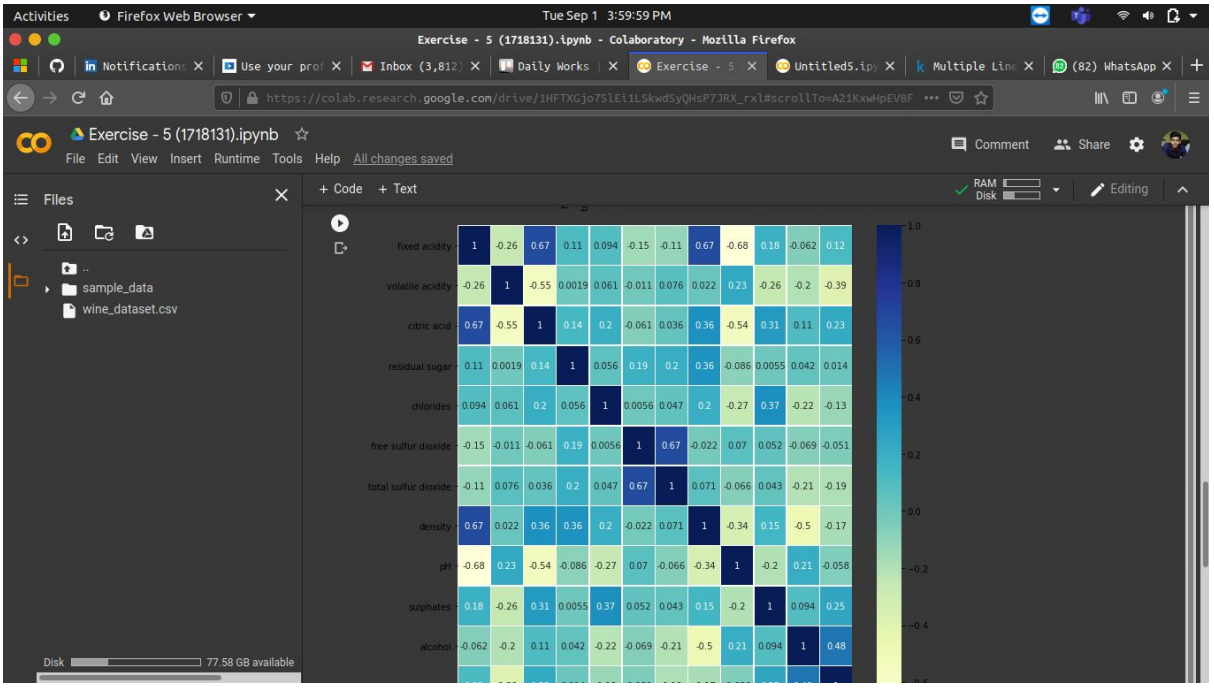
Multiple Linear Regression

```
linear_dataset = dataset.drop(['fixed acidity','density','citric
acid','free sulfur dioxide','total sulfur dioxide'],axis =
1,inplace=False)

X = linear_dataset.loc[:,'volatile acidity' : 'alcohol'].values
y = linear_dataset.loc[:, 'quality'].values

import statsmodels.api as sm
X_opt = X[:,0:6]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:,[0,2,3,4,5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_train,X_test,y_train,y_test = train_test_split(X_opt,y,test_size =
0.2, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

regressor = LinearRegression()
regressor.fit(X_train,y_train)

y_pred = regressor.predict(X_test)
from sklearn.metrics import mean_squared_error
from math import sqrt

rmse = sqrt(mean_squared_error(y_test, y_pred))
print("Mean Error Value" ,round(rmse,4))
```

Multiple Linear Regression

Output :





Multiple Linear Regression

	Marks	Signature
Procedure		
Observation		
Viva		
Result		
Record		

Result :

Thus the **Multiple linear regression** is implemented, visualized and the performance is evaluated successfully.

Naive Bayes Classification(Gender)

Exp No : 6
Date : September 5 2020
Register No : 1718131

Aim :

To implement Naive Bayes algorithm to predict whether the person is a male or female using height (in feet), weight (in lbs) and foot size (in inches).

Procedure :

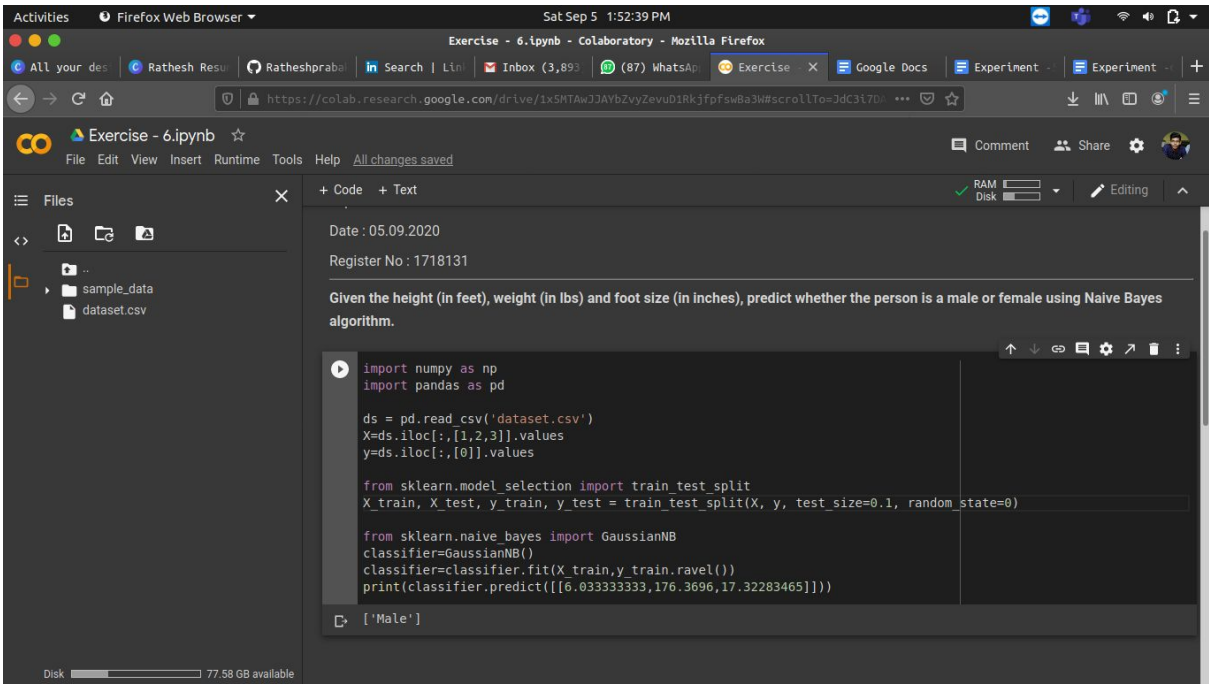
- Open Google Colab, and create a notebook file.
- Importing the necessary packages includes numpy,pandas, sklearn.
- Upload the Height, weight, foot size dataset in CSV format.
- Now we are training our model by splitting the dataset into the training and test set.
- Import the GaussianNB model from sklearn package.
- Fit the GaussianNB classifier to the training set.
- Predict the values using our trained model.

Code :

```
import numpy as np
import pandas as pd

ds = pd.read_csv('dataset.csv')
X=ds.iloc[:,[1,2,3]].values
y=ds.iloc[:,[0]].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=0)

from sklearn.naive_bayes import GaussianNB
classifier=GaussianNB()
classifier=classifier.fit(X_train,y_train.ravel())
print(classifier.predict([[6.033333333,176.3696,17.32283465]]))
```

Naive Bayes Classification(Gender)

Output :



	Marks	Signature
Procedure		
Observation		
Viva		
Result		
Record		

Result :

Thus a person's gender has been predicted successfully using **Naive Bayes algorithm**.

Genetic Algorithm

Exp No : 7
Date : September 7 2020
Register No : 1718131

Aim :

For the given equation : Y = w1x1 + w2x2 + w3x3 + w4x4 + w5x5 + w6x6, find the maximize such equation using Genetic Algorithm, where (x1,x2,x3,x4,x5,x6) = (4,-2,7,5,11,1)

Procedure :

- Open Google Colab, and create a notebook file.
- Importing the necessary packages includes numpy,pygad,matplotlib.
- Give the function inputs and desired output.
- Now we are predicting our model by generation, fitness and correctness value.
- Print the parameters of the best solution, fitness value of the best solution.
- Plot the values using the plot module.

Code :

```
import pygad
import numpy

function_inputs = [4,-2,7,5,11,1] # Function inputs.
desired_output = 44 # Function output.

def fitness_func(solution, solution_idx):
   output = numpy.sum(solution*function_inputs)
   fitness = 1.0 / numpy.abs(output - desired_output)
   return fitness

fitness_function = fitness_func

num_generations = 100 # Number of generations.
num_parents_mating = 7 # Number of solutions to be selected as parents
in the mating pool.

sol_per_pop = 50 # Number of solutions in the population.
num_genes = len(function_inputs)

init_range_low = -2
init_range_high = 5

parent_selection_type = "sss" # Type of parent selection.
```

Genetic Algorithm

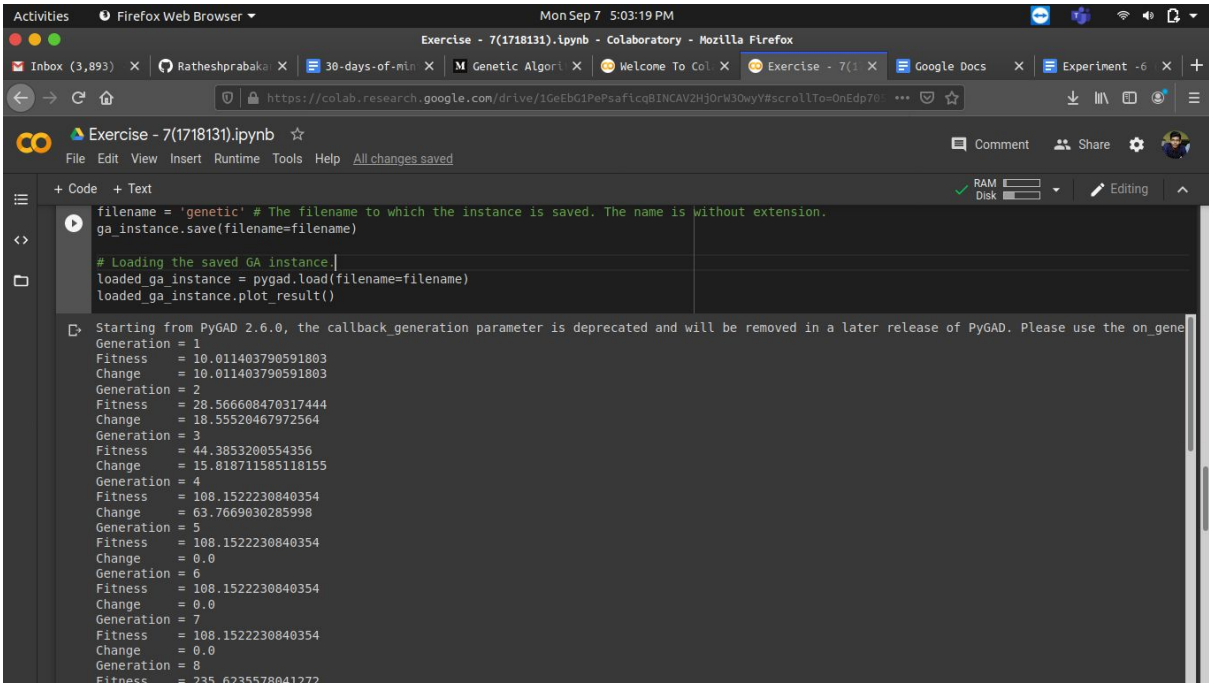
```
keep_parents = 7 # Number of parents to keep in the next population. -1
means keep all parents and 0 means keep nothing.

crossover_type = "single_point" # Type of the crossover operator.

# Parameters of the mutation operation.
mutation_type = "random" # Type of the mutation operator.
mutation_percent_genes = 10 # Percentage of genes to mutate. This
parameter has no action if the parameter mutation_num_genes exists or
when mutation_type is None.

last_fitness = 0
def callback_generation(ga_instance):
   global last_fitness
   print("Generation =
{generation}".format(generation=ga_instance.generations_completed))
   print("Fitness    =
{fitness}".format(fitness=ga_instance.best_solution()[1]))
   print("Change     =
{change}".format(change=ga_instance.best_solution()[1] - last_fitness))
   last_fitness = ga_instance.best_solution()[1]

# Creating an instance of the GA class inside the ga module. Some
parameters are initialized within the constructor.
ga_instance = pygad.GA(num_generations=num_generations,
                       num_parents_mating=num_parents_mating,
                       fitness_func=fitness_function,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       init_range_low=init_range_low,
                       init_range_high=init_range_high,
                       parent_selection_type=parent_selection_type,
                       keep_parents=keep_parents,
                       crossover_type=crossover_type,
                       mutation_type=mutation_type,
                       mutation_percent_genes=mutation_percent_genes,
                       callback_generation=callback_generation)

# Running the GA to optimize the parameters of the function.
ga_instance.run()
```

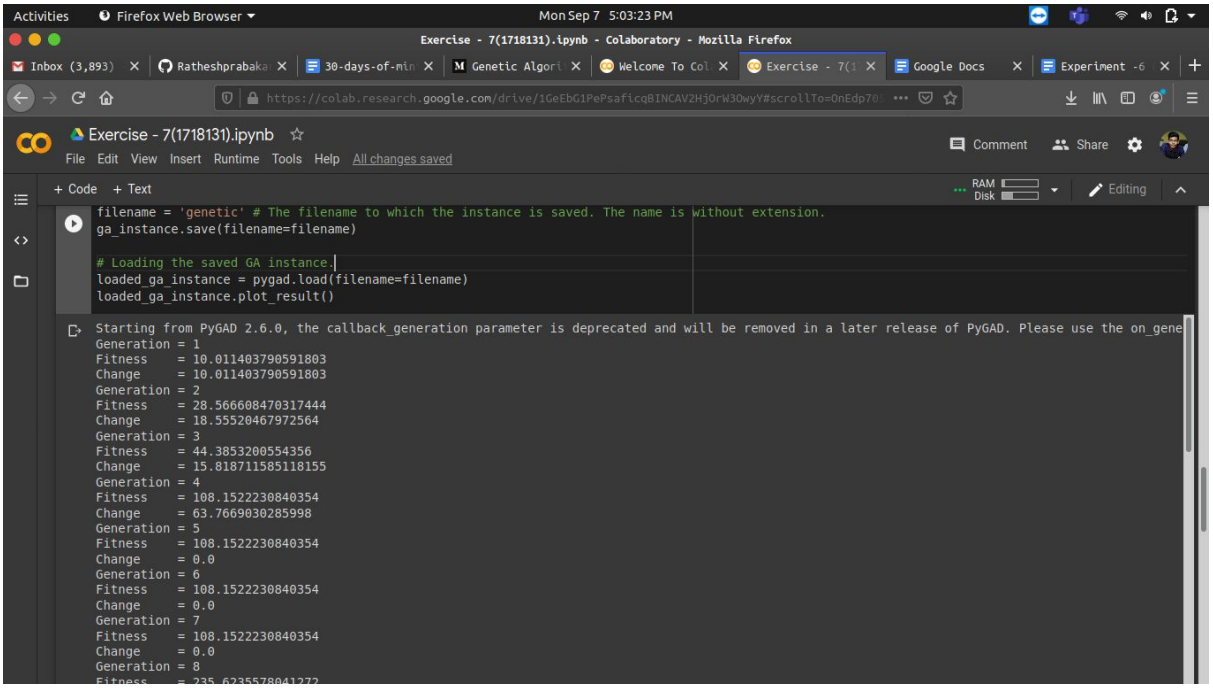
Genetic Algorithm

```
# After the generations complete, some plots are showed that summarize
the how the outputs/fitenss values evolve over generations.
ga_instance.plot_result()

# Returning the details of the best solution.
solution, solution_fitness, solution_idx = ga_instance.best_solution()
print("Parameters of the best solution :
{solution}".format(solution=solution))
print("Fitness value of the best solution =
{solution_fitness}".format(solution_fitness=solution_fitness))
print("Index of the best solution :
{solution_idx}".format(solution_idx=solution_idx))

prediction = numpy.sum(numpy.array(function_inputs)*solution)
print("Predicted output based on the best solution :
{prediction}".format(prediction=prediction))

if ga_instance.best_solution_generation != -1:
   print("Best fitness value reached after {best_solution_generation}
generations.".format(best_solution_generation=ga_instance.best_solution
_generation))

# Saving the GA instance.
filename = 'genetic' # The filename to which the instance is saved. The
name is without extension.
ga_instance.save(filename=filename)

# Loading the saved GA instance.
loaded_ga_instance = pygad.load(filename=filename)
loaded_ga_instance.plot_result()
```

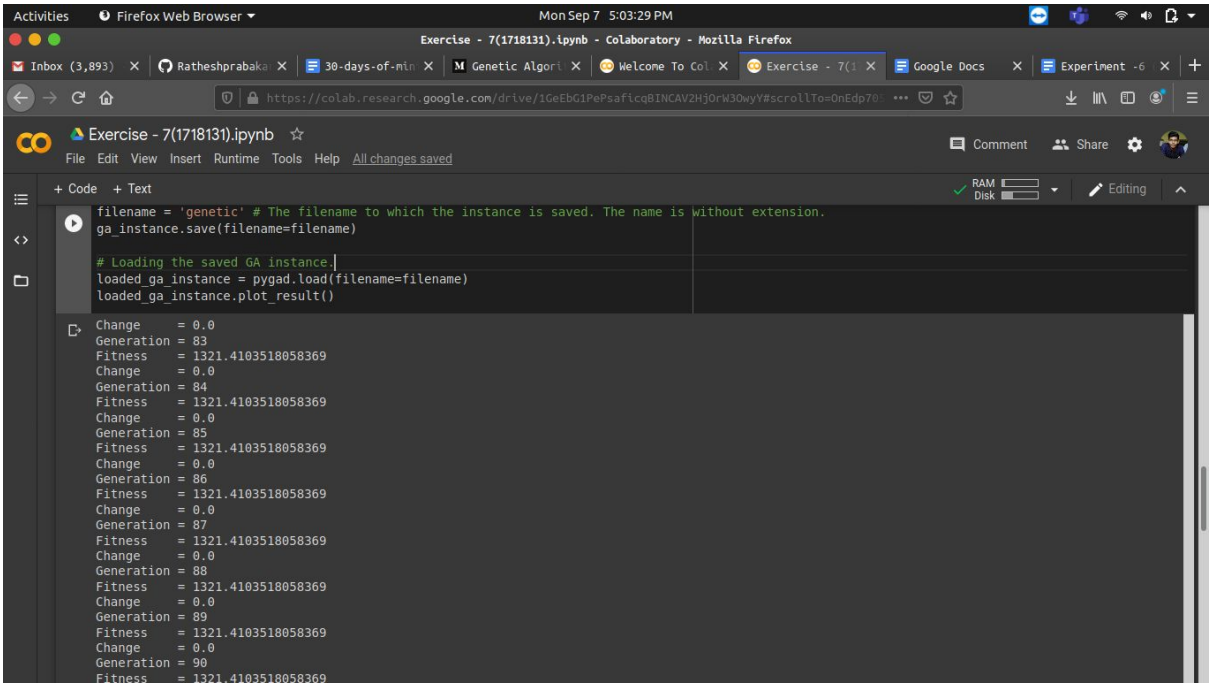
Genetic Algorithm

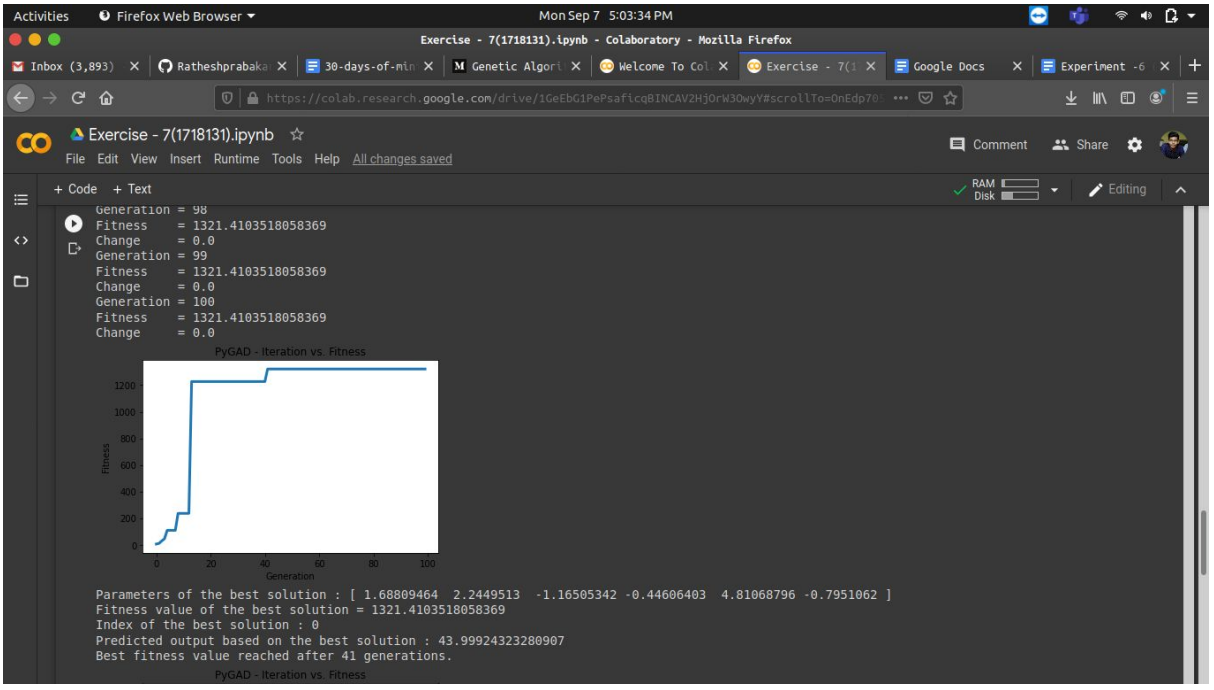
Output :



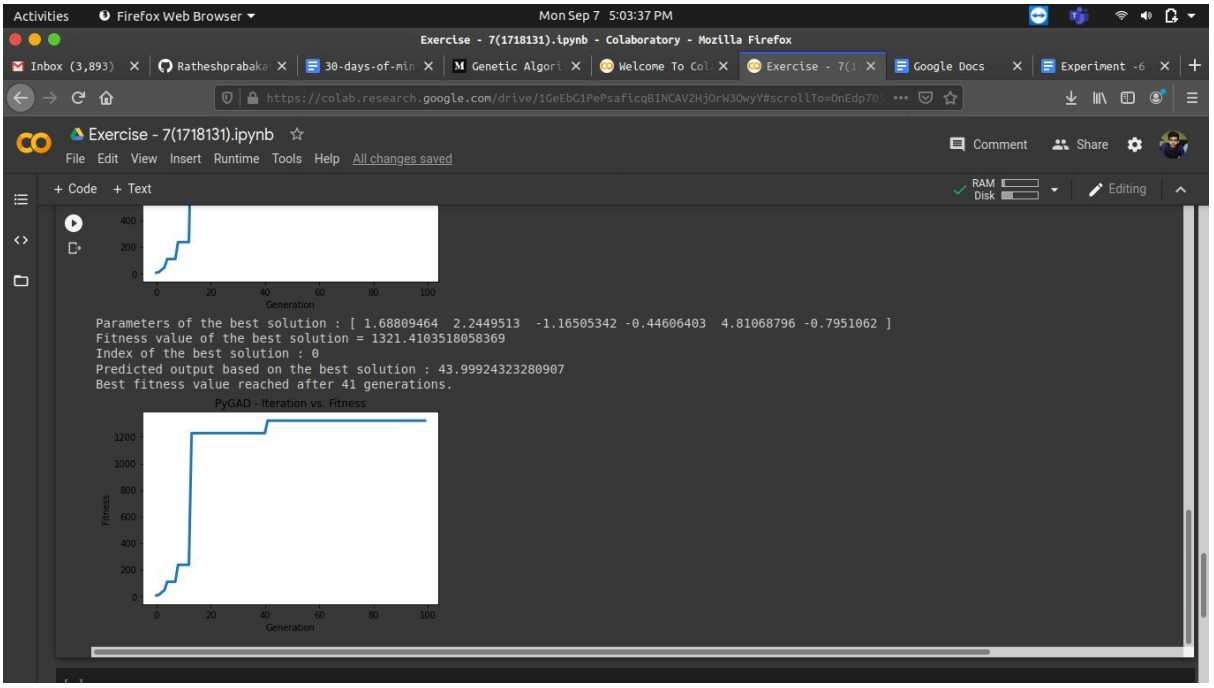


Genetic Algorithm





Genetic Algorithm



	Marks	Signature
Procedure		
Observation		
Viva		
Result		
Record		

Result :

Thus the values are predicted and printed successfully using **genetic algorithms.**

Evolutionary algorithm for Travelling salesman problem

Exercise No : 8
Date : 11.09.2020

Aim :

To write a python program to solve travelling salesman problems using an evolutionary algorithm.

Algorithm :

- Importing the necessary modules and packages includes numpy, random, operators, pandas, matplotlib, pyplot.
- Create a class to handle cities.
- Create a class fitness class and fitness functions.
- Create our initial population using a route generator.
- Then create the first population using a route generator.
- Then create an evolutionary algorithm.
- Create a selection function that will be used to make the list of parent routes.
- Creating a mating tool.
- Create a function to run mutation over the entire population.
- Put all steps together to create the next generation.
- Final step is to create the evolution algorithm.
- To run an evolutionary algorithm, create a list of cities and then run the algorithm.

Code :

```
import numpy as np, random, operator, pandas as pd, matplotlib.pyplot
as plt

class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, city):
        xDis = abs(self.x - city.x)
        yDis = abs(self.y - city.y)
        distance = np.sqrt((xDis ** 2) + (yDis ** 2))
        return distance

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness= 0.0
```

Evolutionary algorithm for Travelling salesman problem

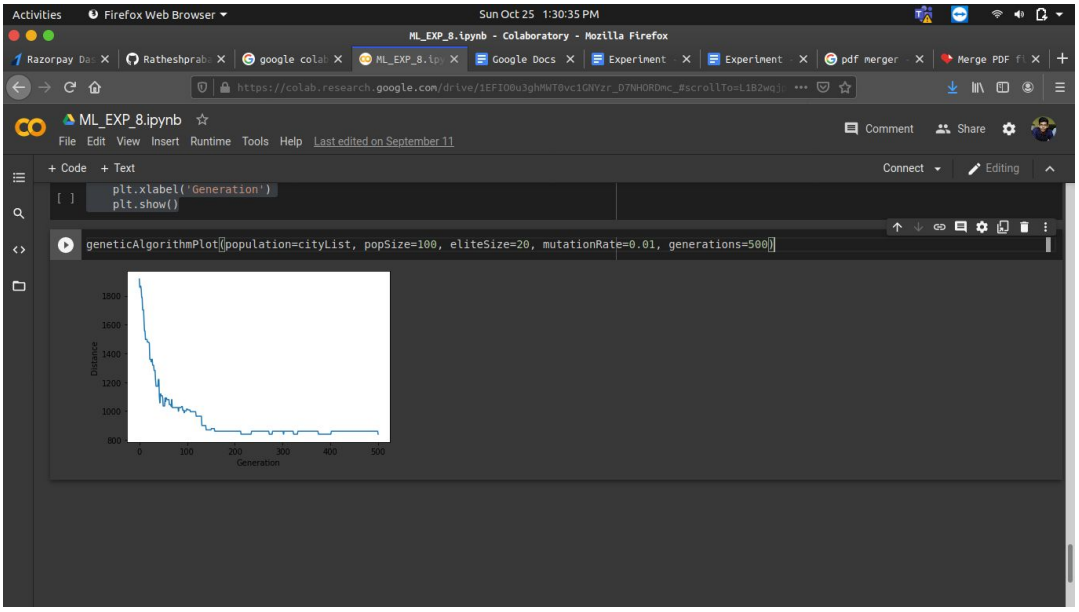
```
    def routeDistance(self):
        if self.distance ==0:
            pathDistance = 0
            for i in range(0, len(self.route)):
                fromCity = self.route[i]
                toCity = None
                if i + 1 < len(self.route):
                    toCity = self.route[i + 1]
                else:
                    toCity = self.route[0]
                pathDistance += fromCity.distance(toCity)
            self.distance = pathDistance
        return self.distance

    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
def createRoute(cityList):
    route = random.sample(cityList, len(cityList))
    return route
def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults

def breed(parent1, parent2):
    child = []
    childP1 = []
    childP2 = []
```

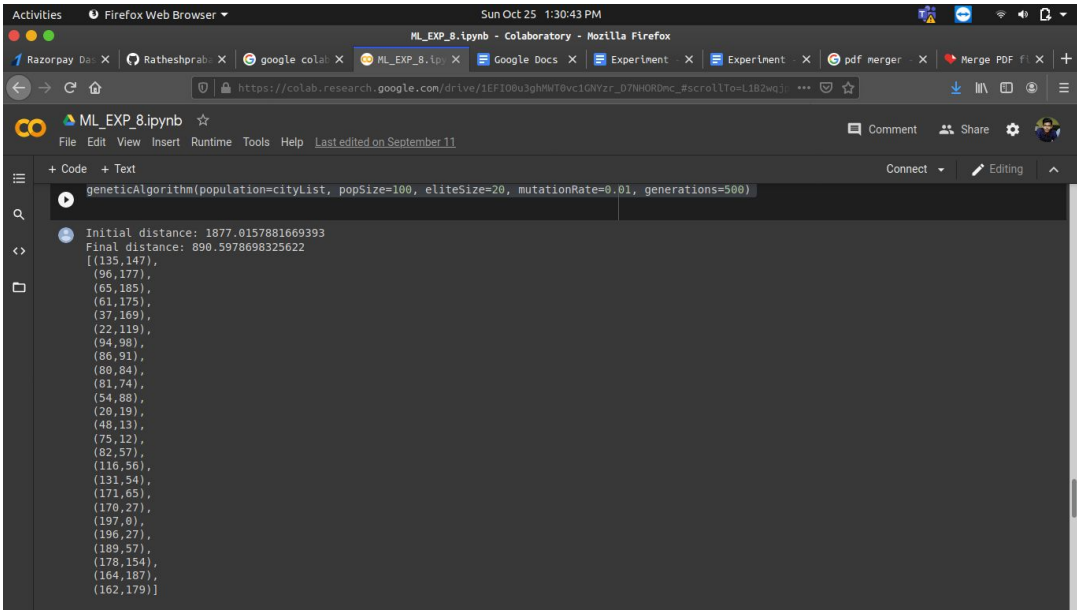
Evolutionary algorithm for Travelling salesman problem

```
    geneA = int(random.random() * len(parent1))
    geneB = int(random.random() * len(parent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)

    for i in range(startGene, endGene):
        childP1.append(parent1[i])

    childP2 = [item for item in parent2 if item not in childP1]

    child = childP1 + childP2
    return child
geneticAlgorithm(population=cityList, popSize=100, eliteSize=20,
mutationRate=0.01, generations=500)


def geneticAlgorithmPlot(population, popSize, eliteSize, mutationRate,
generations):
    pop = initialPopulation(popSize, population)
    progress = []
    progress.append(1 / rankRoutes(pop)[0][1])

    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)
        progress.append(1 / rankRoutes(pop)[0][1])

    plt.plot(progress)
    plt.ylabel('Distance')
    plt.xlabel('Generation')
    plt.show()
geneticAlgorithmPlot(population=cityList, popSize=100, eliteSize=20,
mutationRate=0.01, generations=500)
```

Evolutionary algorithm for Travelling salesman problem

Output :





Result :

Thus the travelling salesman problem is executed and verified using evolutionary algorithm.

GENETIC ALGORITHM WITH LOGISTIC REGRESSION

Exercise No : 9
Date : 12.09.2020

Aim :

To write a python code for applying the genetic algorithm with logistic regression for better feature selections on breast cancer data.

Algorithm :

- Import the important python libraries required for this algorithm (numpy, pandas).
- Import some other libraries for implementation of machine learning algorithms.
- Import the database from the python library sci-kit learn.
- Split dataset into test and train.
- Train the dataset using a logistic regression model.
- Now include a genetic algorithm in the process.
- Initialize the population randomly based on the data.
- Find the fitness value of each of the chromosomes.
- Select the best filled chromosomes as parents to pass the genes for the next generations and create a new population.
- Create a new set of chromosomes by combining the parents and adding them to a new population set.
- Perform mutation with alters one or more gene values in a chromosome in the new population set generated.
- Repeat step (iii) to (v) again for each generation.
- Finally train and predict the accuracy using genetic algorithms.

Code :

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot
%matplotlib inline

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

#import the breast cancer dataset
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer()
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
label=cancer["target"]

#splitting the model into training and testing set
```

GENETIC ALGORITHM WITH LOGISTIC REGRESSION

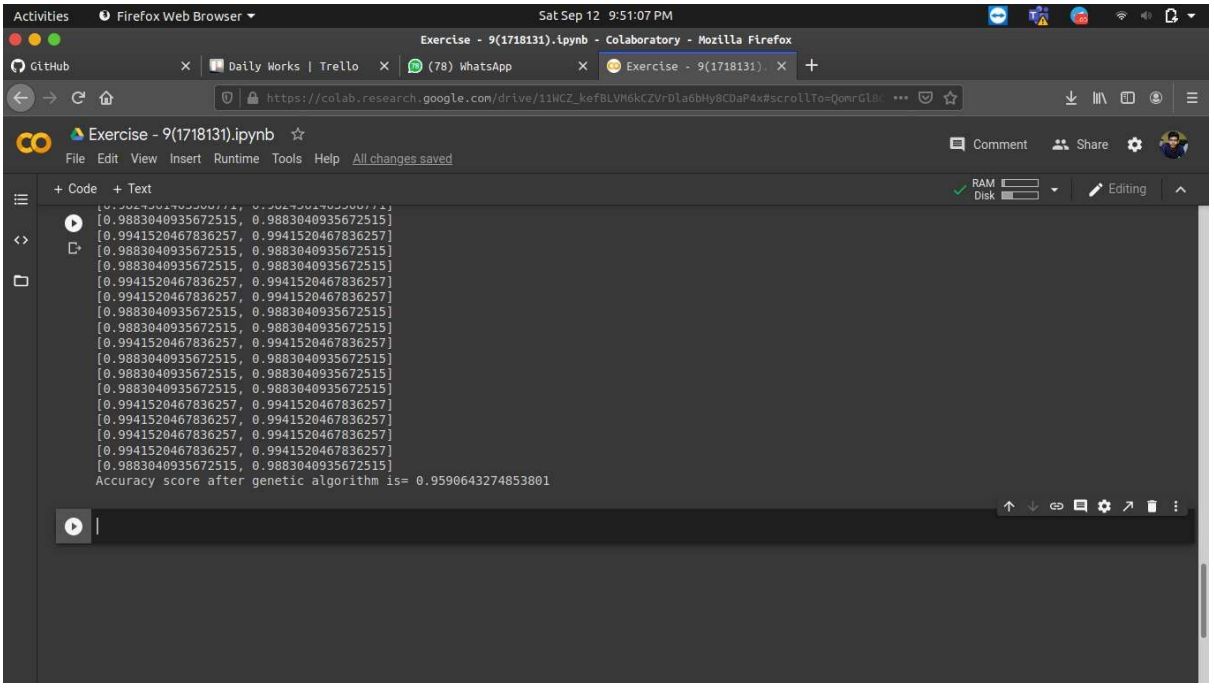
```
X_train, X_test, y_train, y_test = train_test_split(df,label,
test_size=0.30,random_state=0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#training a logistics regression model
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
print("Accuracy = "+ str(accuracy_score(y_test,predictions)))

#defining various steps required for the genetic algorithm
def initilization_of_population(size,n_feat):
    population = []
    for i in range(size):
        chromosome = np.ones(n_feat,dtype=np.bool)
        chromosome[:int(0.3*n_feat)]=False
        np.random.shuffle(chromosome)
        population.append(chromosome)
    return population

def fitness_score(population):
    scores = []
    for chromosome in population:
        logmodel.fit(X_train[:,chromosome],y_train)
        predictions = logmodel.predict(X_test[:,chromosome])
        scores.append(accuracy_score(y_test,predictions))
    scores, population = np.array(scores), np.array(population)
    inds = np.argsort(scores)
    return list(scores[inds][::-1]), list(population[inds,:][::-1])

def selection(pop_after_fit,n_parents):
    population_nextgen = []
    for i in range(n_parents):
        population_nextgen.append(pop_after_fit[i])
    return population_nextgen

def crossover(pop_after_sel):
    population_nextgen=pop_after_sel
    for i in range(len(pop_after_sel)):
```

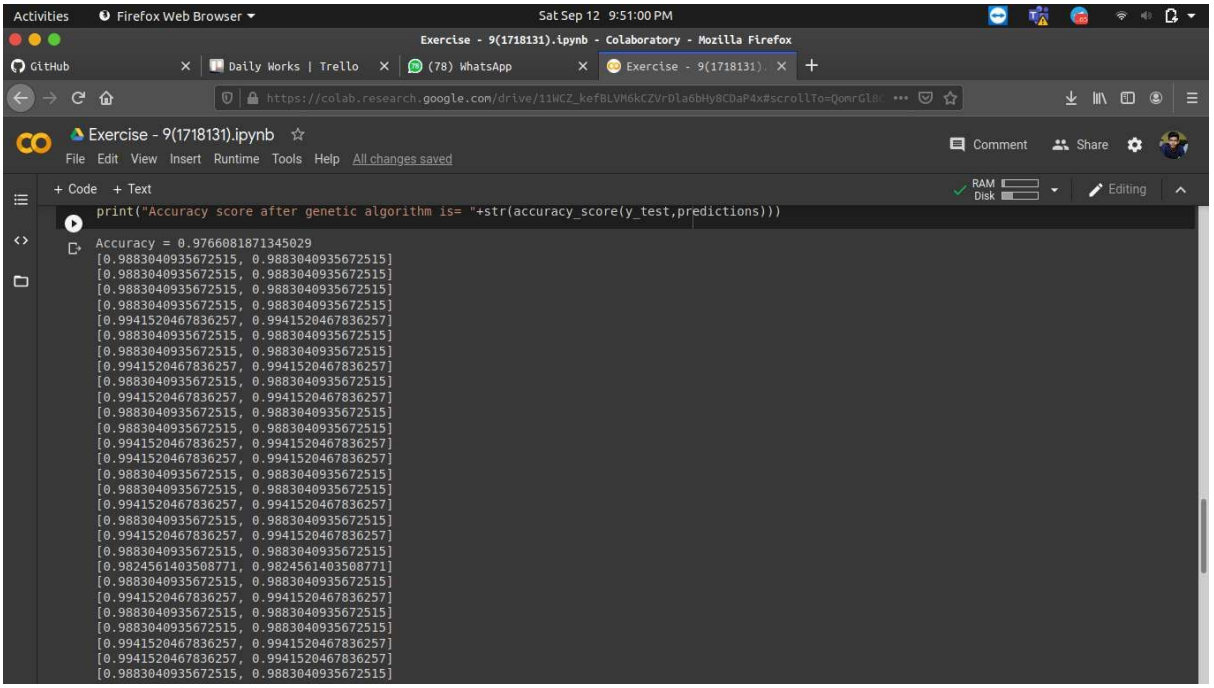
GENETIC ALGORITHM WITH LOGISTIC REGRESSION

```
        child=pop_after_sel[i]
        child[3:7]=pop_after_sel[(i+1)%len(pop_after_sel)][3:7]
        population_nextgen.append(child)
    return population_nextgen

def mutation(pop_after_cross,mutation_rate):
    population_nextgen = []
    for i in range(0,len(pop_after_cross)):
        chromosome = pop_after_cross[i]
        for j in range(len(chromosome)):
            if random.random() < mutation_rate:
                chromosome[j]= not chromosome[j]
        population_nextgen.append(chromosome)
    #print(population_nextgen)
    return population_nextgen

def
generations(size,n_feat,n_parents,mutation_rate,n_gen,X_train,X_test,
y_train, y_test):
    best_chromo= []
    best_score= []
    population_nextgen=initilization_of_population(size,n_feat)
    for i in range(n_gen):
        scores, pop_after_fit = fitness_score(population_nextgen)
        print(scores[:2])
        pop_after_sel = selection(pop_after_fit,n_parents)
        pop_after_cross = crossover(pop_after_sel)
        population_nextgen = mutation(pop_after_cross,mutation_rate)
        best_chromo.append(pop_after_fit[0])
        best_score.append(scores[0])
    return best_chromo,best_score

chromo,score=generations(size=200,n_feat=30,n_parents=100,mutation_rate
=0.10,n_gen=38,X_train=X_train,X_test=X_test,y_train=y_train,y_test=y_t
est)
logmodel.fit(X_train[:,chromo[-1]],y_train)
predictions = logmodel.predict(X_test[:,chromo[-1]])
print("Accuracy score after genetic algorithm is=
"+str(accuracy_score(y_test,predictions)))
```

GENETIC ALGORITHM WITH LOGISTIC REGRESSION

Output :





Result :

Thus the genetic algorithm with logistic regression for better feature selection on breast cancer data was executed and verified.

K - Means Algorithm

Exercise No : 10
Date : 18.09.2020

Aim :

Implement the K-Means algorithm on 2D dataset for the old faithful geyser in Yellowstone national park, Wyoming, USA, with different initializations of centroids.

Algorithm :

- Import the important python libraries pandas, standard scalar, K-Means and matplotlib.
- Import the dataset using standard scalar() fit_transform() function.
- Run the local implementation of k-means using k-means method.
- Calculate the centroids using cluster_centers.
- Finally plot the clustered data using plt.scatter pro plotting clusters & centroid point.

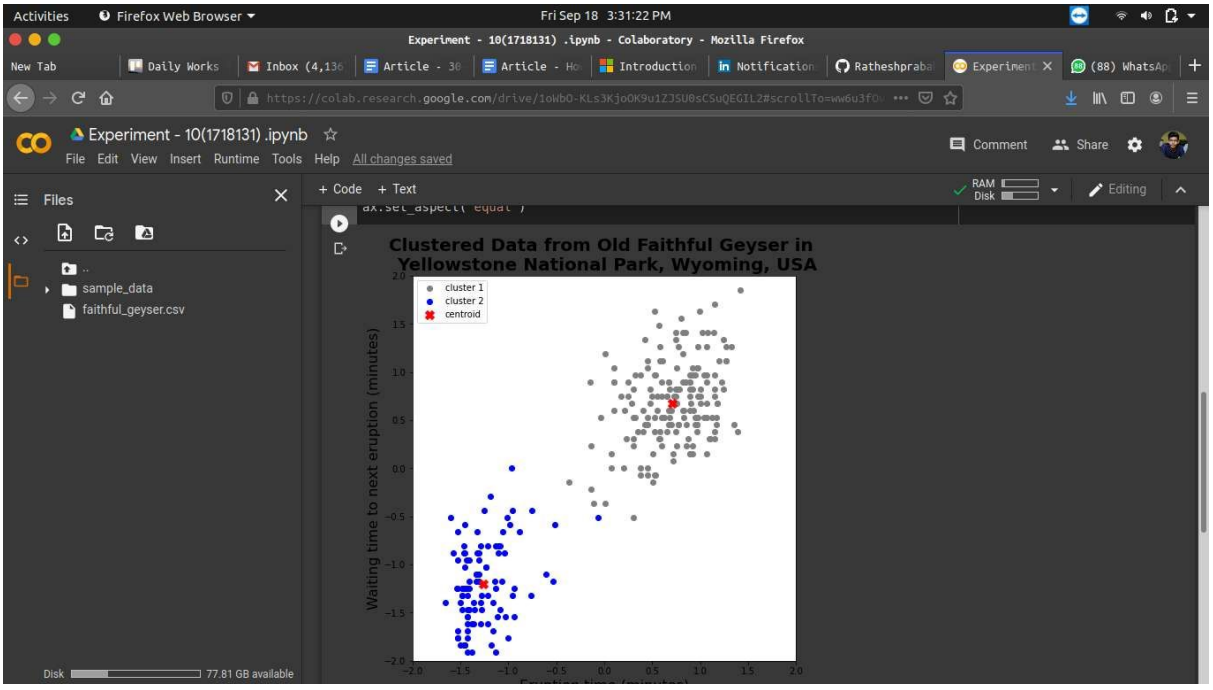
Code :

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Import the data
data_frame = pd.read_csv('faithful_geyser.csv')

# Standardize the data
X_std = StandardScaler().fit_transform(data_frame)

# Run local implementation of kmeans
km = KMeans(n_clusters=2, max_iter=100)
km.fit(X_std)
centroids = km.cluster_centers_

# Plot the clustered data
fig, ax = plt.subplots(figsize=(8, 8))
plt.scatter(X_std[km.labels_ == 0, 0], X_std[km.labels_ == 0, 1],
            c='grey', label='cluster 1')
plt.scatter(X_std[km.labels_ == 1, 0], X_std[km.labels_ == 1, 1],
            c='blue', label='cluster 2')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=100,
            c='r', label='centroid')
plt.legend()
plt.xlim([-2, 2])
plt.ylim([-2, 2])
```

K - Means Algorithm

```
plt.xlabel('Eruption time (minutes)',  fontsize=16)
plt.ylabel('Waiting time to next eruption (minutes)', fontsize=16)
plt.title('Clustered Data from Old Faithful Geyser in \n Yellowstone
National Park, Wyoming, USA',  fontsize=20, fontweight='bold')
ax.set_aspect('equal')
```

Output :



Result :

Thus the K means algorithm was executed and predicted successfully.

Extraction Maximization Algorithm

Exercise No : 11
Date : 22.09.2020

Aim :

To implement expectation maximization with Gaussian mixture model and to find the best mixture Gaussian mode.

Algorithm :

- Import libraries for plotting matrix math, normalization, probability density function computation & data processing.
- Select mean and standard deviation to generate data.
- Create a class Gaussian with members such as mean, standard deviation and probability density function.
- Create a class and perform an estimation step and assign each print to gaussian as well as how much to mic them.
- Perform an estimation step and assign each print to Gaussian 1 or 2 with a percentage.
- Perform a maximization step.
- Perform N iterations, then compute log-likelihood.
- Then find the least mixture Gaussian model.
- Finally show the mixture using sns.distplot() and pls.plot() functions.

Code :

```
# import libraries



# For plotting
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
%matplotlib inline
#for matrix math
import numpy as np
#for normalization + probability density function computation
from scipy import stats
#for data preprocessing
import pandas as pd
import pandas.util.testing as tm
from math import sqrt, log, exp, pi
from random import uniform
random_seed=36788765
np.random.seed(random_seed)
```

Extraction Maximization Algorithm

```
Mean1 = 2.0  # Input parameter, mean of first normal probability
distribution
Standard_dev1 = 4.0 #@param {type:"number"}
Mean2 = 9.0 # Input parameter, mean of second normal  probability
distribution
Standard_dev2 = 2.0 #@param {type:"number"}


# generate data
y1 = np.random.normal(Mean1, Standard_dev1, 1000)
y2 = np.random.normal(Mean2, Standard_dev2, 500)
data=np.append(y1,y2)


# For data visiualisation calculate left and right of the graph
Min_graph = min(data)
Max_graph = max(data)
x = np.linspace(Min_graph, Max_graph, 2000) # to plot the data


print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1,
Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2,
Standard_dev2))
sns.distplot(data, bins=20, kde=False);


class Gaussian:
   "Model univariate Gaussian"
   def __init__(self, mu, sigma):
       #mean and standard deviation
       self.mu = mu
       self.sigma = sigma


   #probability density function
   def pdf(self, datum):
       "Probability of a data point given the current parameters"
       u = (datum - self.mu) / abs(self.sigma)
       y = (1 / (sqrt(2 * pi) * abs(self.sigma))) * exp(-u * u / 2)
       return y
```

Extraction Maximization Algorithm

```
   def __repr__(self):
       return 'Gaussian({0:4.6}, {1:4.6})'.format(self.mu, self.sigma)


#gaussian of best fit
best_single = Gaussian(np.mean(data), np.std(data))
print('Best single Gaussian: μ = {:.2}, σ =
{:.2}'.format(best_single.mu, best_single.sigma))
#fit a single gaussian curve to the data
g_single = stats.norm(best_single.mu, best_single.sigma).pdf(x)
sns.distplot(data, bins=20, kde=False, norm_hist=True);
plt.plot(x, g_single, label='single gaussian');
plt.legend();


class GaussianMixture_self:
   "Model mixture of two univariate Gaussians and their EM estimation"


   def __init__(self, data, mu_min=min(data), mu_max=max(data),
sigma_min=1, sigma_max=1, mix=.5):
       self.data = data
       #todo the Algorithm would be numerical enhanced by normalizing
the data first, next do all the EM steps and do the de-normalising at
the end

       #init with multiple gaussians
       self.one = Gaussian(uniform(mu_min, mu_max),
                           uniform(sigma_min, sigma_max))
       self.two = Gaussian(uniform(mu_min, mu_max),
                           uniform(sigma_min, sigma_max))

       #as well as how much to mix them
       self.mix = mix


   def Estep(self):
       "Perform an E(stimation)-step, assign each point to gaussian 1
or 2 with a percentage"
       # compute weights
       self.loglike = 0. # = log(p = 1)
       for datum in self.data:
```

Extraction Maximization Algorithm

```
            # unnormalized weights
            wp1 = self.one.pdf(datum) * self.mix
            wp2 = self.two.pdf(datum) * (1. - self.mix)
            # compute denominator
            den = wp1 + wp2
            # normalize
            wp1 /= den
            wp2 /= den      # wp1+wp2= 1, it either belongs to gaussian 1
or gaussion 2
            # add into loglike
            self.loglike += log(den) #freshening up self.loglike in the
process
            # yield weight tuple
            yield (wp1, wp2)


    def Mstep(self, weights):
        "Perform an M(aximization)-step"
        # compute denominators
        (left, rigt) = zip(*weights)
        one_den = sum(left)
        two_den = sum(rigt)


        # compute new means
        self.one.mu = sum(w * d  for (w, d) in zip(left, data)) /
one_den
        self.two.mu = sum(w * d  for (w, d) in zip(rigt, data)) /
two_den

        # compute new sigmas
        self.one.sigma = sqrt(sum(w * ((d - self.one.mu) ** 2)
                                  for (w, d) in zip(left, data)) /
one_den)
        self.two.sigma = sqrt(sum(w * ((d - self.two.mu) ** 2)
                                  for (w, d) in zip(rigt, data)) /
two_den)
        # compute new mix
        self.mix = one_den / len(data)
```

Extraction Maximization Algorithm

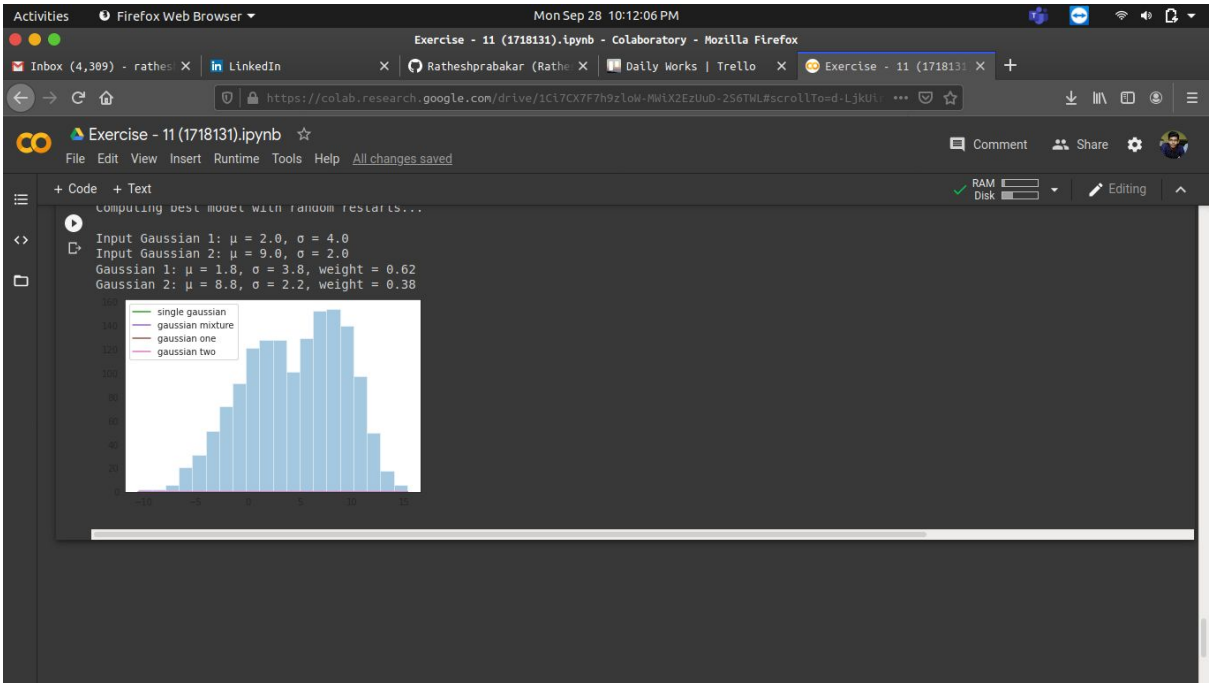
```
    def iterate(self, N=1, verbose=False):
        "Perform N iterations, then compute log-likelihood"
        for i in range(1, N+1):
            self.Mstep(self.Estep()) #The heart of the algorith, perform
E-stepand next M-step
            if verbose:
                print('{0:2} {1}'.format(i, self))
        self.Estep() # to freshen up self.loglike


    def pdf(self, x):
        return (self.mix)*self.one.pdf(x) + (1-self.mix)*self.two.pdf(x)

    def __repr__(self):
        return 'GaussianMixture({0}, {1}, mix={2.03})'.format(self.one,
                                                              self.two,
                                                              self.mix)


    def __str__(self):
        return 'Mixture: {0}, {1}, mix={2:.03})'.format(self.one,
                                                        self.two,
                                                        self.mix)
# See the algorithm in action
n_iterations = 20
best_mix = None
best_loglike = float('-inf')
mix = GaussianMixture_self(data)
for _ in range(n_iterations):
    try:
        #train!
        mix.iterate(verbose=True)
        if mix.loglike > best_loglike:
            best_loglike = mix.loglike
            best_mix = mix

    except (ZeroDivisionError, ValueError, RuntimeWarning): # Catch
division errors from bad starts, and just throw them out...
        print("one less")
        pass
```

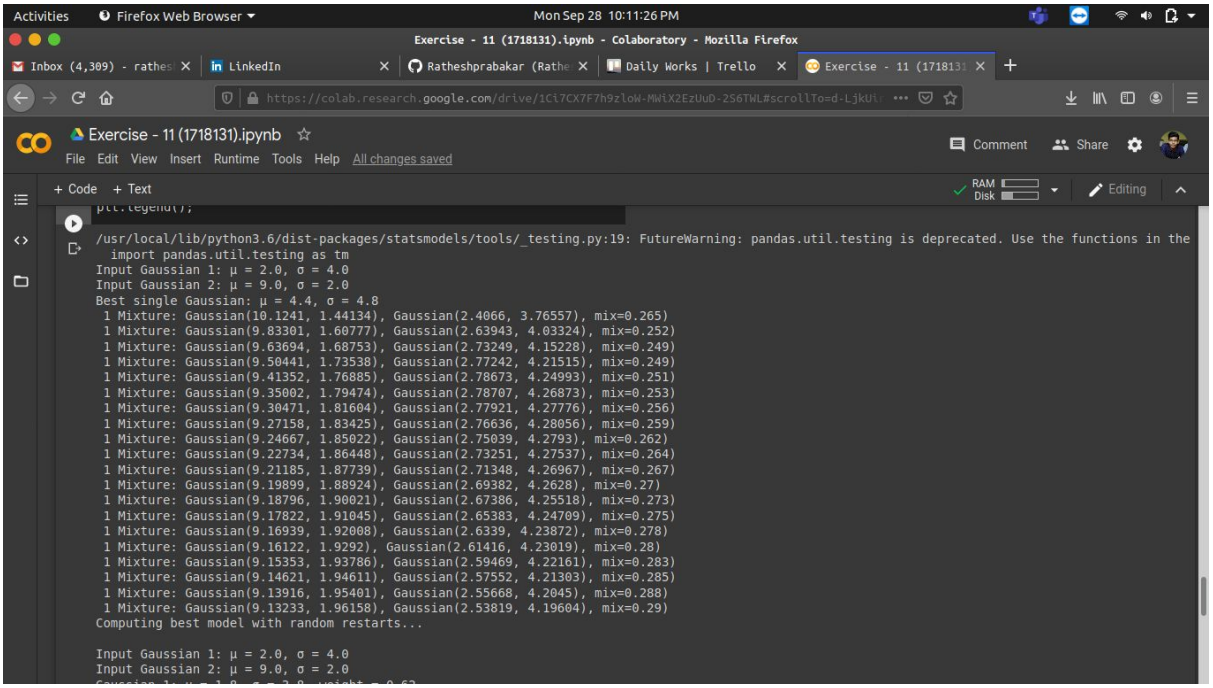
Extraction Maximization Algorithm

```
# Find best Mixture Gaussian model
n_iterations = 300
n_random_restarts = 4
best_mix = None
best_loglike = float('-inf')
print('Computing best model with random restarts...\n')
for _ in range(n_random_restarts):
   mix = GaussianMixture_self(data)
   for _ in range(n_iterations):
       try:
           mix.iterate()
           if mix.loglike > best_loglike:
               best_loglike = mix.loglike
               best_mix = mix
       except (ZeroDivisionError, ValueError, RuntimeWarning): # Catch
division errors from bad starts, and just throw them out...
           pass
#print('Best Gaussian Mixture : μ = {:.2}, σ = {:.2} with μ = {:.2}, σ
= {:.2}'.format(best_mix.one.mu, best_mix.one.sigma, best_mix.two.mu,
best_mix.two.sigma))


print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1,
Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2,
Standard_dev2))
print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format("1",
best_mix.one.mu, best_mix.one.sigma, best_mix.mix))
print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format("2",
best_mix.two.mu, best_mix.two.sigma, (1-best_mix.mix)))
#Show mixture
sns.distplot(data, bins=20, kde=False, norm_hist=True);
g_both = [best_mix.pdf(e) for e in x]
plt.plot(x, g_both, label='gaussian mixture');
g_left = [best_mix.one.pdf(e) * best_mix.mix for e in x]
plt.plot(x, g_left, label='gaussian one');
g_right = [best_mix.two.pdf(e) * (1-best_mix.mix) for e in x]
plt.plot(x, g_right, label='gaussian two');
plt.legend();
```

Extraction Maximization Algorithm

Output :





Result :

Thus the expectation maximization with 2 Gaussian mixture models has been implemented and verified.

Comparison of the accuracy of the classification algorithms

Exercise No : 12
Date :

Aim :

Write a python script to implement DCT, Naive bayes, KNN classification to predict whether the student will be placed or not. COmpare accuracy of various classification algorithms.

Algorithm :

- Open a new file and save the file with py extension.
- Import the csv file from the kaggle.
- Import the necessary modules.
- Drop the salary column and plot the graph and show it.
- Convert the string values into the numerical values.
- Split the test and train dataset.
- Import the decision tree classifier module and KNN classifier, Decision tree classifier.
- Fit the data with the model and train.
- Predict the output and print the accuracy.

Code :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

data=pd.read_csv('Placement_Data_Full_Class.csv')

data.info

data.dtypes

data=data.drop(['salary'],axis=1)
fig = plt.figure(figsize = (8,8))
ax = fig.gca()
data.hist(ax=ax)
plt.show()

fig = plt.figure(figsize=(10,7))
objList = data.select_dtypes(include = "object").columns
for pos in range(1,len(objList)+1):
   axes = fig.add_subplot(3,3, pos)
   axes.set_xlabel(objList[pos-1])
```

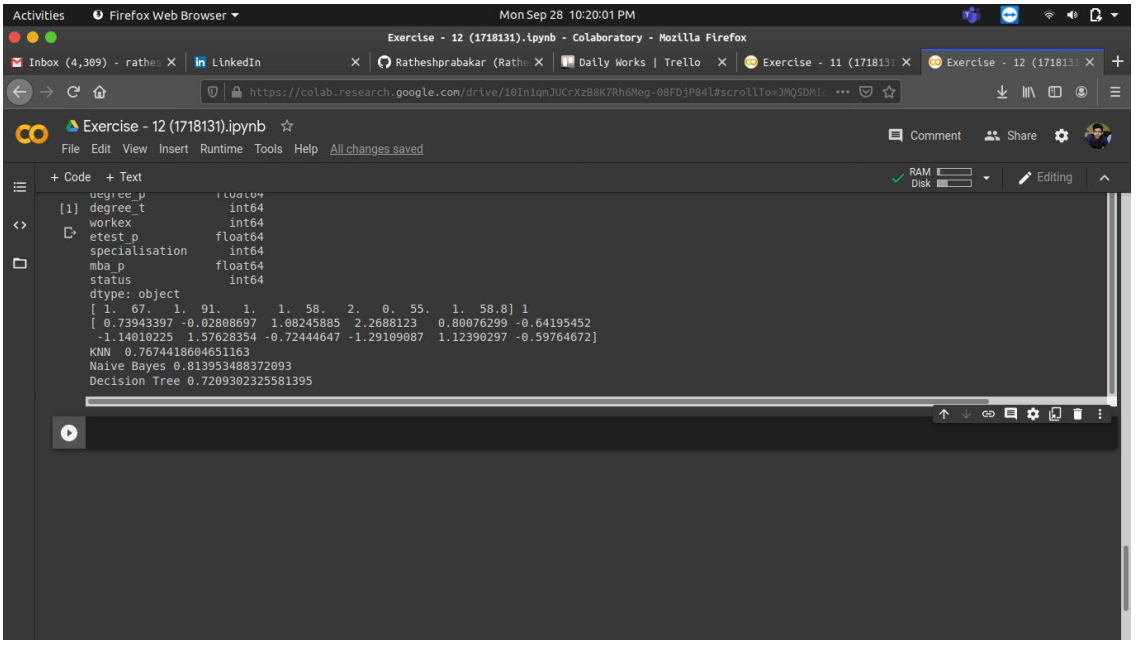
Comparison of the accuracy of the classification algorithms

```
   axes.hist(data[objList[pos-1]])
   axes.grid(True)
plt.tight_layout()
plt.show()

from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
objList = data.select_dtypes(include = "object").columns
print(objList)
for i in objList:
 data[i]=enc.fit_transform(data[i].astype(str))
print(data.dtypes)

data=data.drop(['sl_no'],axis=1)
X=data.iloc[:,:-1].values
y=data.iloc[:,-1].values
print(X[0],y[0])

from sklearn.preprocessing import StandardScaler
std=StandardScaler()
X=std.fit_transform(X)
print(X[0])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

from sklearn.model_selection import GridSearchCV
parameters_knn = {'n_neighbors':[3,5,8,10], 'p':[1,
2],'algorithm':('auto', 'ball_tree', 'kd_tree',
'brute'),'leaf_size':[20,30,40,50]}
parameters_dc =
{'criterion':['gini','entropy'],'max_depth':[4,5,6,7,8,9,10,11,12,15,20
,30,40,50,70,90,120,150]}



from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
model_nb=GaussianNB()
```

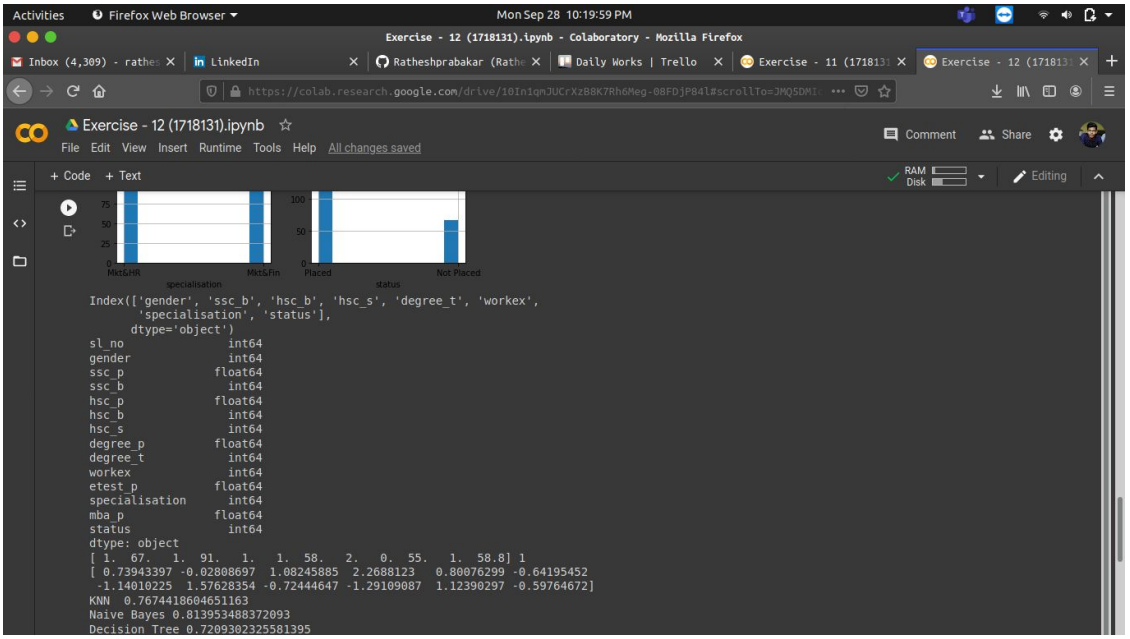
Comparison of the accuracy of the classification algorithms

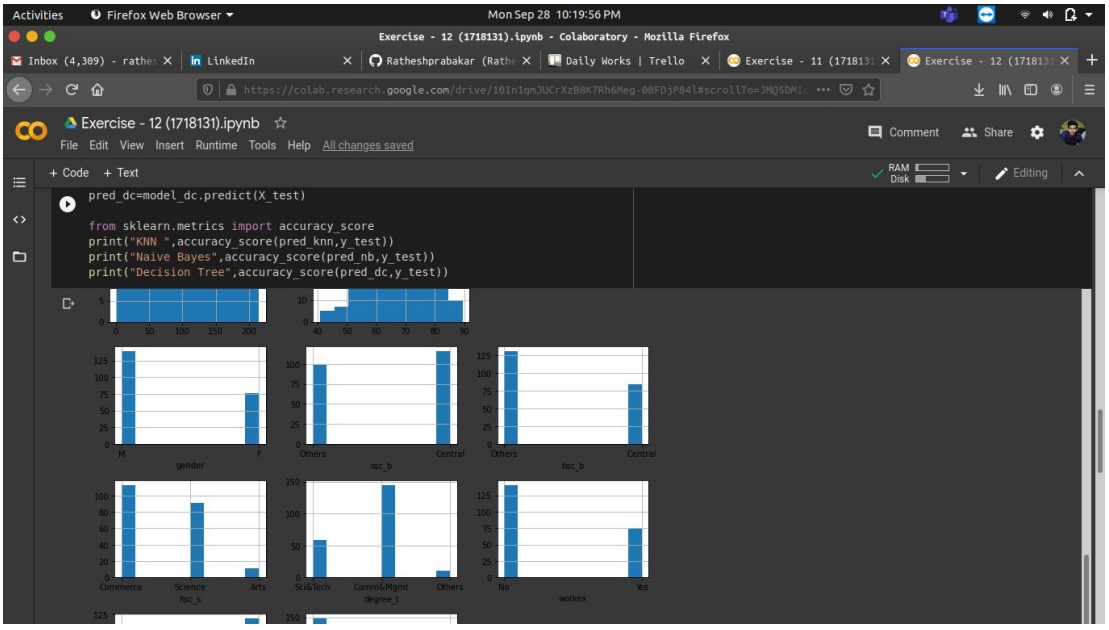
```
dc=DecisionTreeClassifier()
model_knn=GridSearchCV(knn,parameters_knn)
model_dc=GridSearchCV(dc,parameters_dc)
model_knn.fit(X_train,y_train)
model_nb.fit(X_train,y_train)
model_dc.fit(X_train,y_train)

pred_knn=model_knn.predict(X_test)
pred_nb=model_nb.predict(X_test)
pred_dc=model_dc.predict(X_test)

from sklearn.metrics import accuracy_score
print("KNN ",accuracy_score(pred_knn,y_test))
print("Naive Bayes",accuracy_score(pred_nb,y_test))
print("Decision Tree",accuracy_score(pred_dc,y_test))
```

Output :

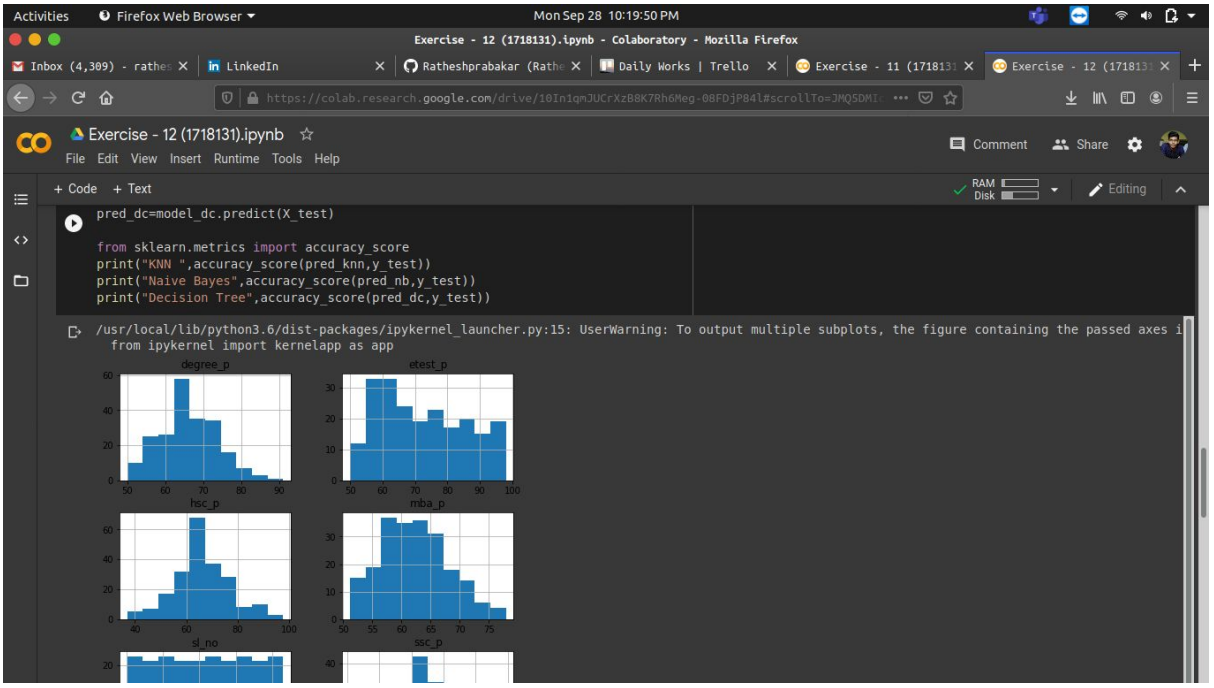


Comparison of the accuracy of the classification algorithms





Comparison of the accuracy of the classification algorithms



Result :

Thus the accuracy of classification algorithms are shown successfully.

Comparison of Feature selection algorithms (PCA, Recursive feature elimination)

Exercise No : 13
Date : 29.09.2020

Aim :

To implement univariate selection, recursive feature elimination, principle component analysis (PCA) in python using scikit-learn (sklearn) and compare them for pima Indian onset of diabetes dataset.

Algorithm :

- Open a new file and save the file with py extension.
- Import the necessary modules from univariate statistical tests, PCA, RFE.
- Load the dataset file.
- In a univariate statistical test, extract the feature, summarize the scores and selected features.
- In PCA, we are loading the data, extract using PFA and summarize the value.
- In RFE, We are doing the same with addition to that, we are calculating the feature ranking based upon our regression model.

Code :

```
# Feature Selection with Univariate Statistical Tests
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
# load data
filename = 'pima_dataset.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=f_classif, k=4)
fit = test.fit(X, Y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

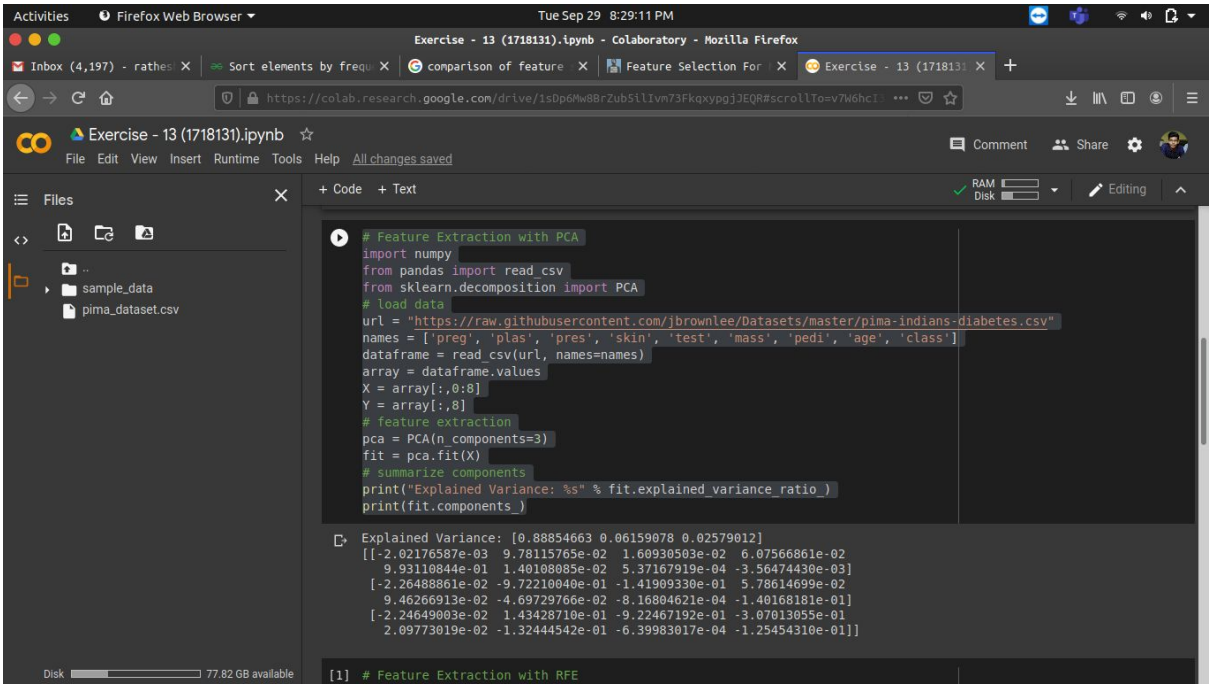
Comparison of Feature selection algorithms (PCA, Recursive feature elimination)

```
# Feature Extraction with PCA
import numpy
from pandas import read_csv
from sklearn.decomposition import PCA
# load data
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)


# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = LogisticRegression(solver='lbfgs')
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
```

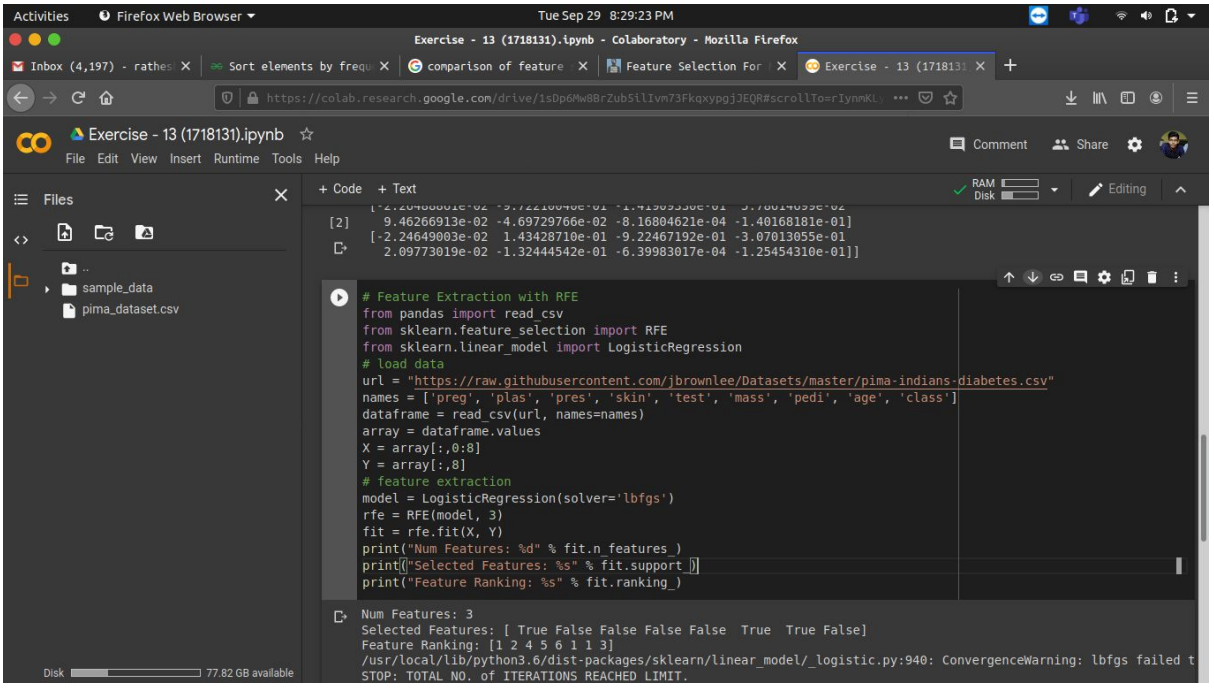
Comparison of Feature selection algorithms (PCA, Recursive feature elimination)

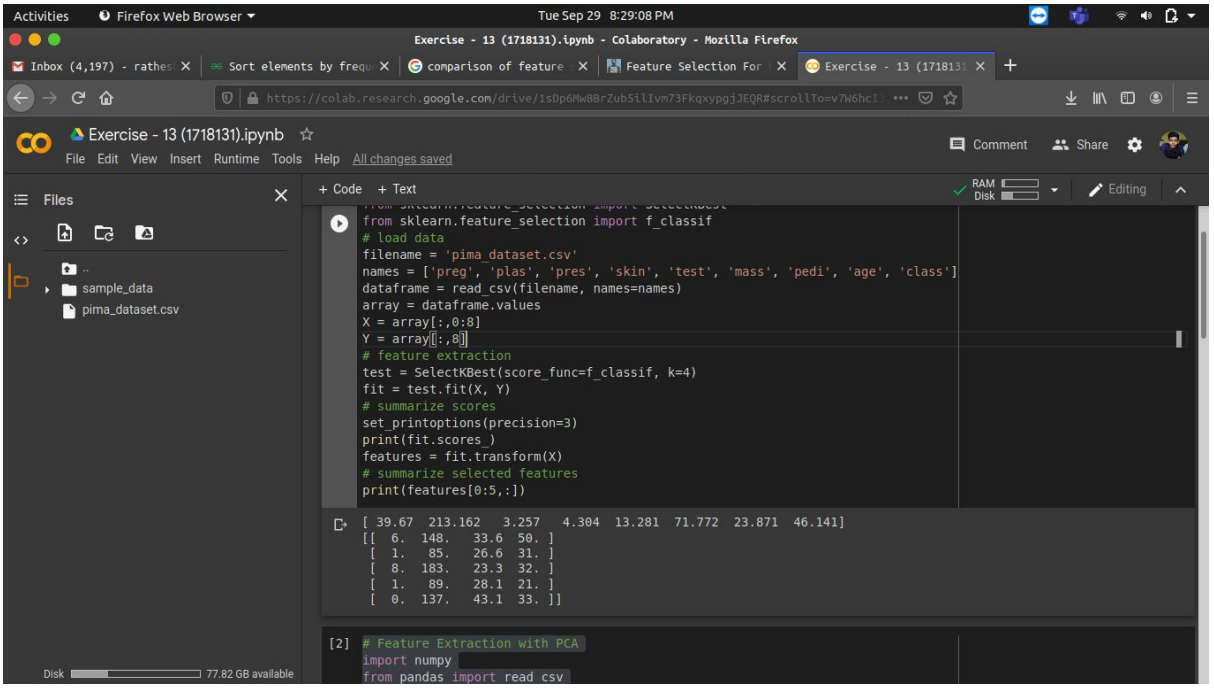
```
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

Output :



Comparison of Feature selection algorithms (PCA, Recursive feature elimination)





Result :

Thus the feature selection algorithms are implemented & analysed successfully.

Single Layer Perception

Exercise No : 14
Date : 06.10.2020

Aim :

To implement perception in iris-flower dataset strips. The last 50 rows of the dataset that belongs to the class 'iris-signical' and find optimal weights and classify all of these points.

Algorithm :

- Open a new file and save the file with py extension.
- Import the necessary modules.
- Load the iris dataset file.
- Assume the weights and make a total of 5 weights.
- Define the no of iterations to be after each iteration, the algorithm computes the class as 0 and 1.
- Update each point during each iteration.
- If a sample is misclassified then weights are updated by delta shifts.
- Plot the misclassified samples.
- Show the output.

Code :

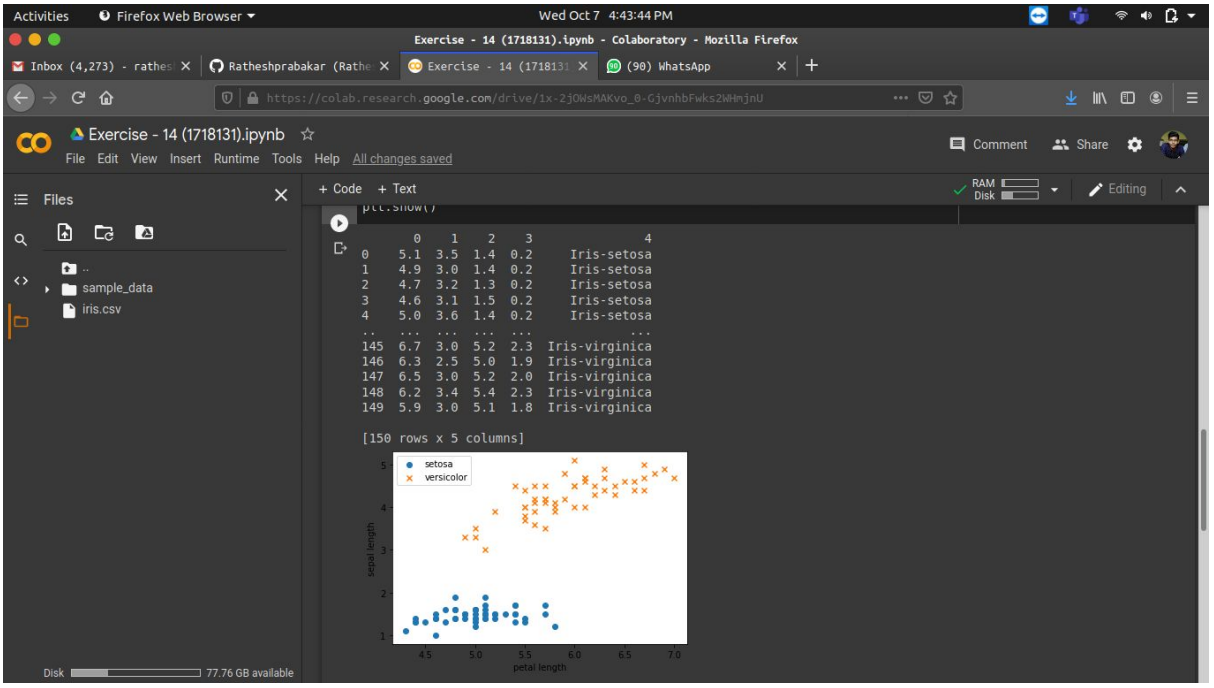
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
def load_data():
   data = pd.read_csv("iris.csv", header = None)
   print(data)

   data = data[:100]
   data[4] = np.where(data.iloc[:, -1]=='Iris-setosa', 0, 1)
   data = np.asmatrix(data, dtype = 'float64')
   return data
data = load_data()

plt.scatter(np.array(data[:50,0]), np.array(data[:50,2]), marker='o',
label='setosa')
plt.scatter(np.array(data[50:,0]), np.array(data[50:,2]), marker='x',
label='versicolor')
plt.xlabel('petal length')
plt.ylabel('sepal length')
plt.legend()
plt.show()

def perceptron(data, num_iter):
```

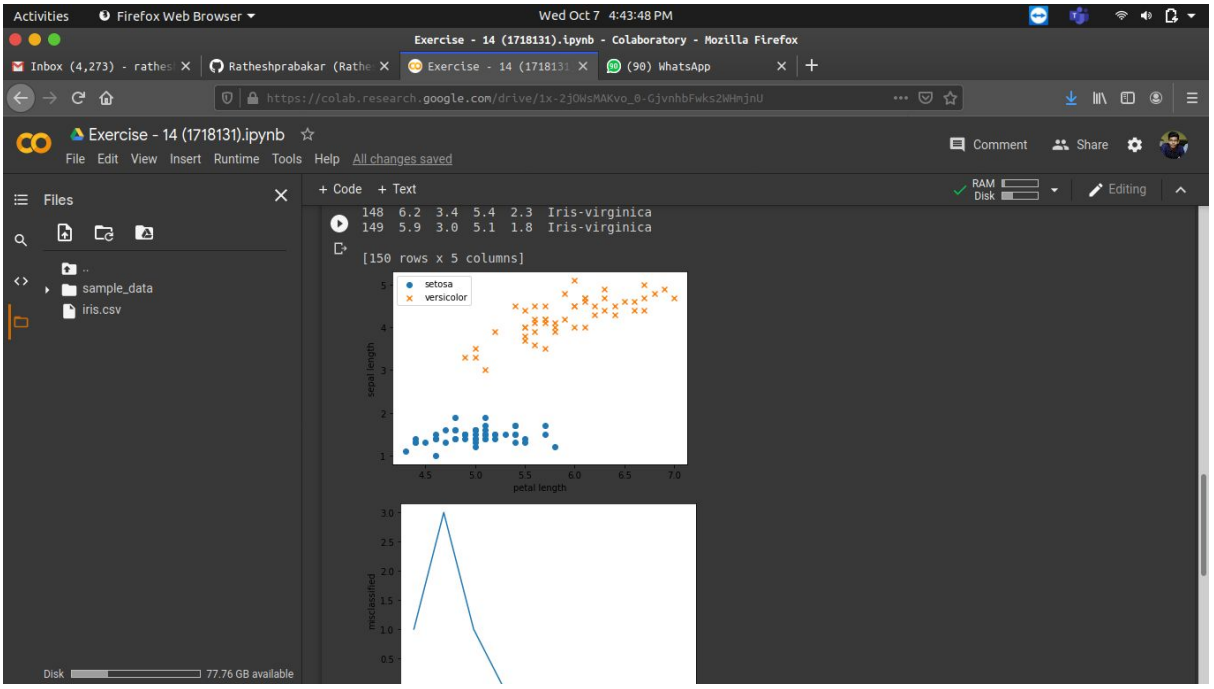
Single Layer Perception

```
    features = data[:, :-1]
    labels = data[:, -1]

    w = np.zeros(shape=(1, features.shape[1]+1))

    misclassified_ = []
     for epoch in range(num_iter):
        misclassified = 0
        for x, label in zip(features, labels):
            x = np.insert(x,0,1)
            y = np.dot(w, x.transpose())
            target = 1.0 if (y > 0) else 0.0

            delta = (label.item(0,0) - target)

            if(delta):
                misclassified += 1
                w += (delta * x)

        misclassified_.append(misclassified)
    return (w, misclassified_)

num_iter = 10
w, misclassified_ = perceptron(data, num_iter)

epochs = np.arange(1, num_iter+1)
plt.plot(epochs, misclassified_)
plt.xlabel('iterations')
plt.ylabel('misclassified')
plt.show()
```

Single Layer Perception

Output





Result :

Thus the optimal weight line that classified all of those points are found.

Multiple Layer Perception

Exercise No : 15
Date : 13.10.2020

Aim :

To implement MLP in an Iris flower dataset, find best parameters and calculate score for each one of samples.

Algorithm :

- Import the necessary libraries.
- Load the iris flower dataset.
- Plot the sepal example and petal sample.
- Separate the dataset into soap training samples and 20 test samples.
- For implementing MLP, do following 3 steps,
- Initialize the weights and bias with small randomized values.
- Propagate all values in the input layer until the output layer.
- Update weight and bias in inner layers.
- Find the least parameter using various lists with different parameter values.
- Display the results.
- Train the MLP and test the results.
- Display the accuracy and precision of the MLP.
- Find the score for each one of the samples and display the results.

Code :

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
combine = [train_df, test_df]

train_df = train_df.drop(['Ticket', 'Cabin'], axis=1)
test_df = test_df.drop(['Ticket', 'Cabin'], axis=1)
combine = [train_df, test_df]

for dataset in combine:
   dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.',
expand=False)

for dataset in combine:
   dataset['Title'] = dataset['Title'].replace(['Lady',
'Countess','Capt', 'Col',\
 'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
```

Multiple Layer Perception

```
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train_df = train_df.drop(['Name', 'PassengerId'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
combine = [train_df, test_df]

for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map( {'female': 1, 'male': 0}
).astype(int)

guess_ages = np.zeros((2,3))

for dataset in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = dataset[(dataset['Sex'] == i) & \
                                (dataset['Pclass'] ==
j+1)]['Age'].dropna()
            age_guess = guess_df.median()
            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5
    for i in range(0, 2):
        for j in range(0, 3):
            dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) &
(dataset.Pclass == j+1),\
                    'Age'] = guess_ages[i,j]
    dataset['Age'] = dataset['Age'].astype(int)

train_df['AgeBand'] = pd.cut(train_df['Age'], 5)

for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] =
1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] =
2
```

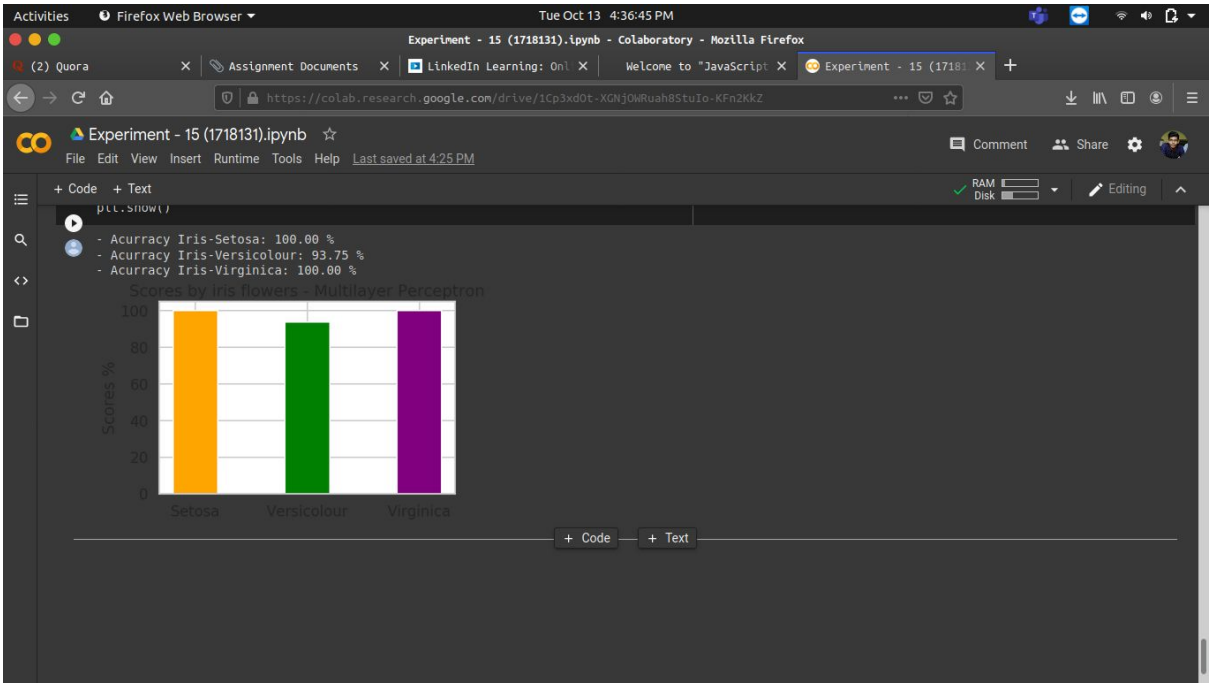
Multiple Layer Perception

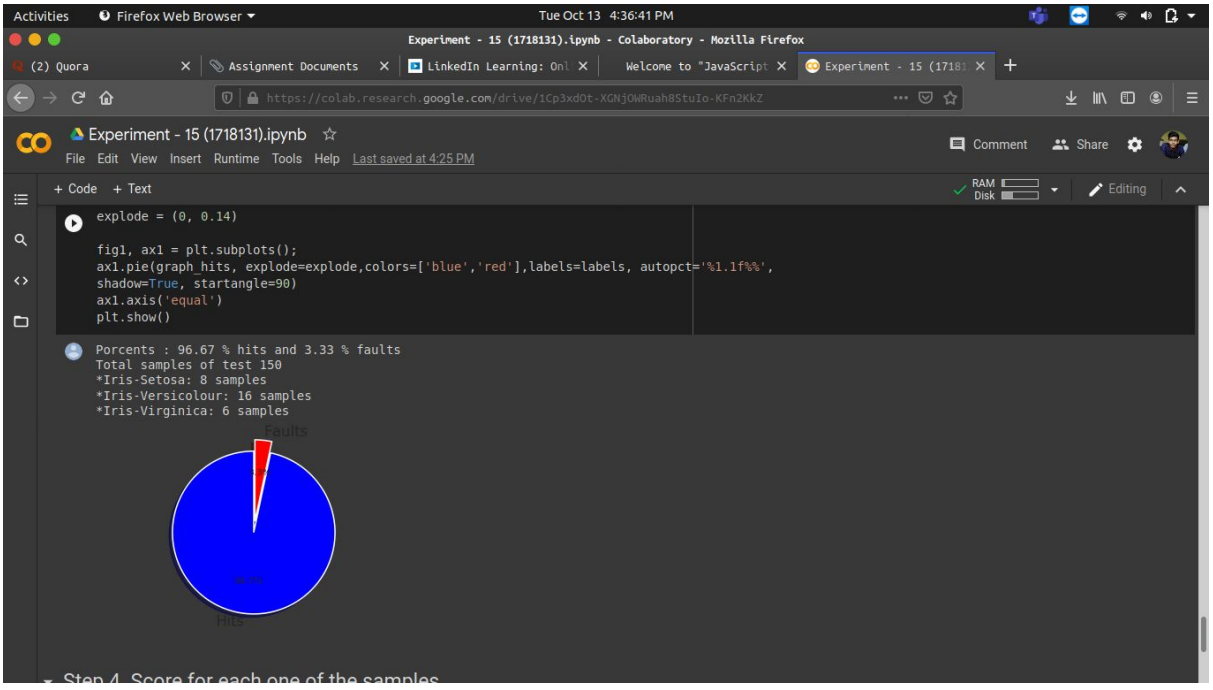
```
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] =
3
    dataset.loc[ dataset['Age'] > 64, 'Age']

train_df = train_df.drop(['AgeBand'], axis=1)
combine = [train_df, test_df]

for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

train_df = train_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
test_df = test_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
combine = [train_df, test_df]

for dataset in combine:
    dataset['Age*Class'] = dataset.Age * dataset.Pclass

freq_port = train_df.Embarked.dropna().mode()[0]

for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)

for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q':
2} ).astype(int)

test_df['Fare'].fillna(test_df['Fare'].dropna().median(), inplace=True)

train_df['FareBand'] = pd.qcut(train_df['Fare'], 4)

for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454),
'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31),
'Fare']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)
```

Multiple Layer Perception

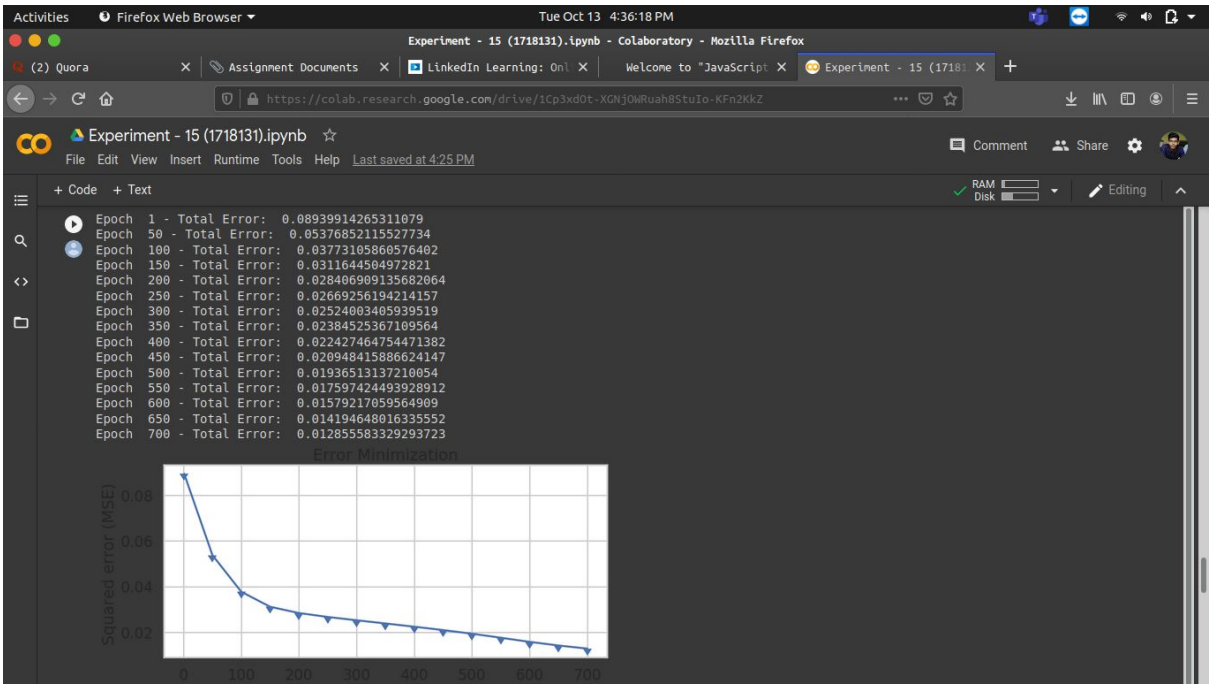
```
train_df = train_df.drop(['FareBand'], axis=1)
combine = [train_df, test_df]
```

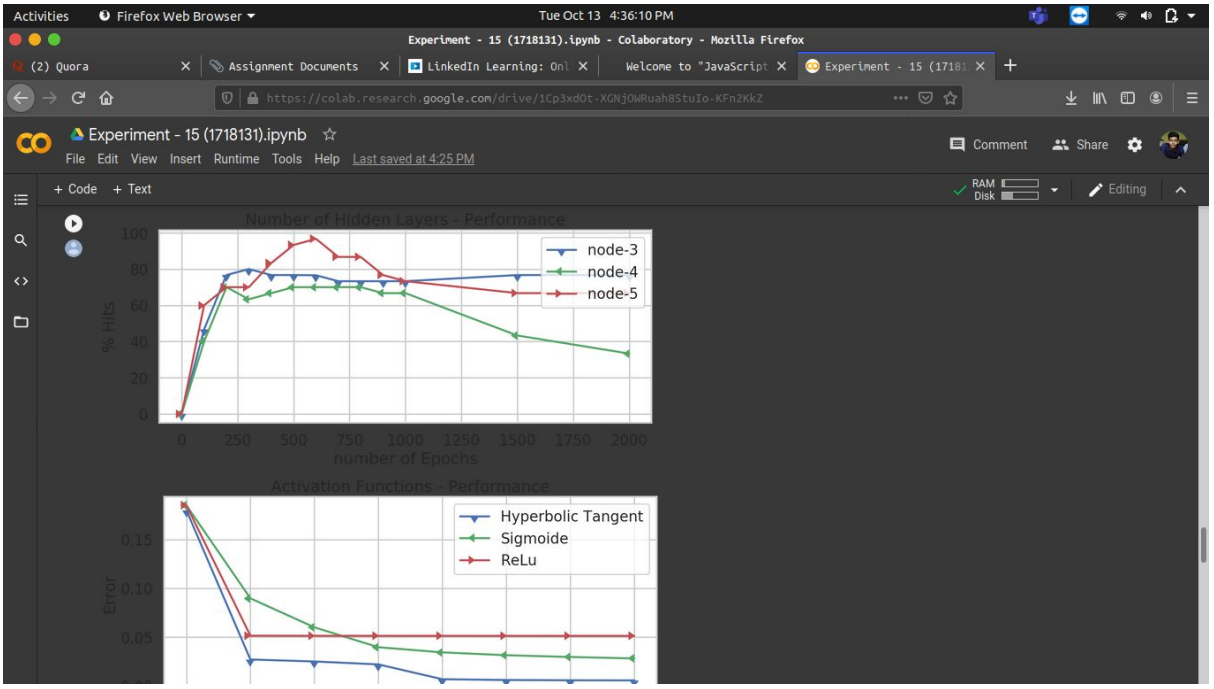
Output :



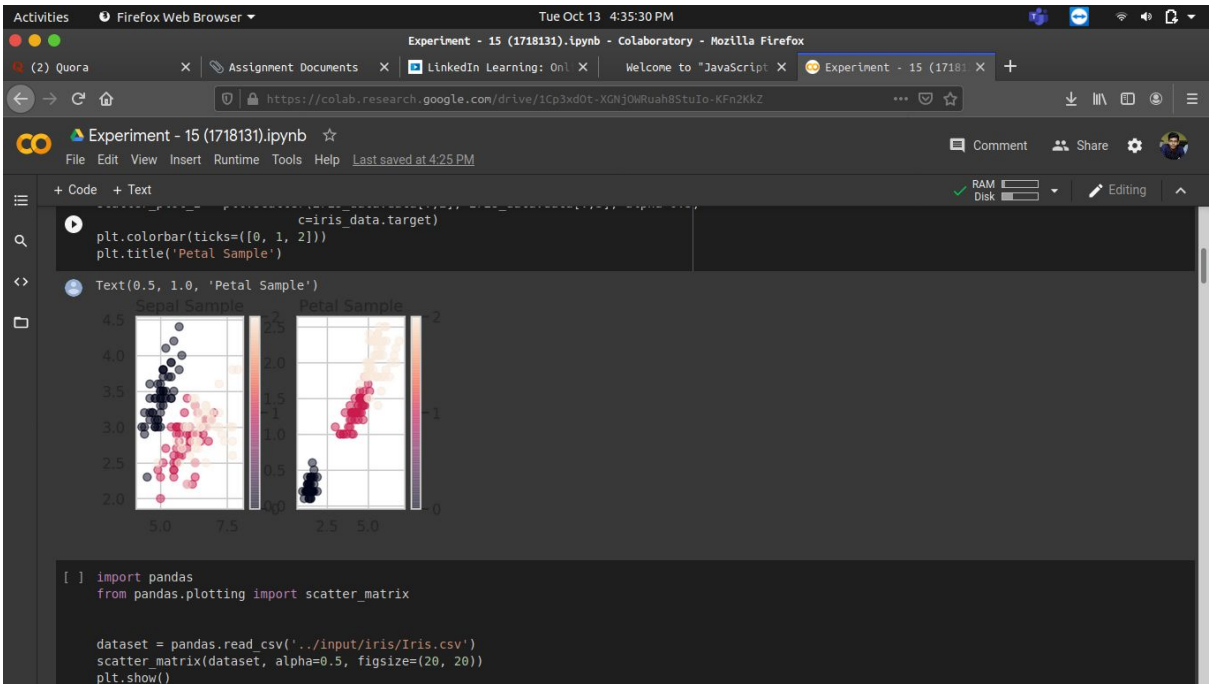


Multiple Layer Perception





Multiple Layer Perception



Result :

Thus the parameters and score of each one of samples in iris flower have been implemented and verified.

Historical Sales data analysis

Exercise No : 17
Date : 23.10.2020

Aim :

To forecast the total amount of products sold in every shop for the test set.

Algorithm :

- Import the necessary modules and packages.
- Merge and flatten the data frame.
- Generate table item_price_latest with the last item_price.
- Extract data columns to year, month, day.
- Aggregate on month level and create feature item_cnt_prev_month and item_cnt_month_mean.
- Generate table item_price_avg for each shop and month.
- Then create training data and test data.
- Use label encoder and train_test_split.
- Finally train the model with the historical data and predict the future output.

Code :

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import pandas as pd
import os
from datetime import datetime

# Import specific packages
import re
from collections import Counter
from scipy.sparse import csr_matrix
from itertools import compress
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import shuffle
from sklearn.preprocessing import LabelEncoder

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
```

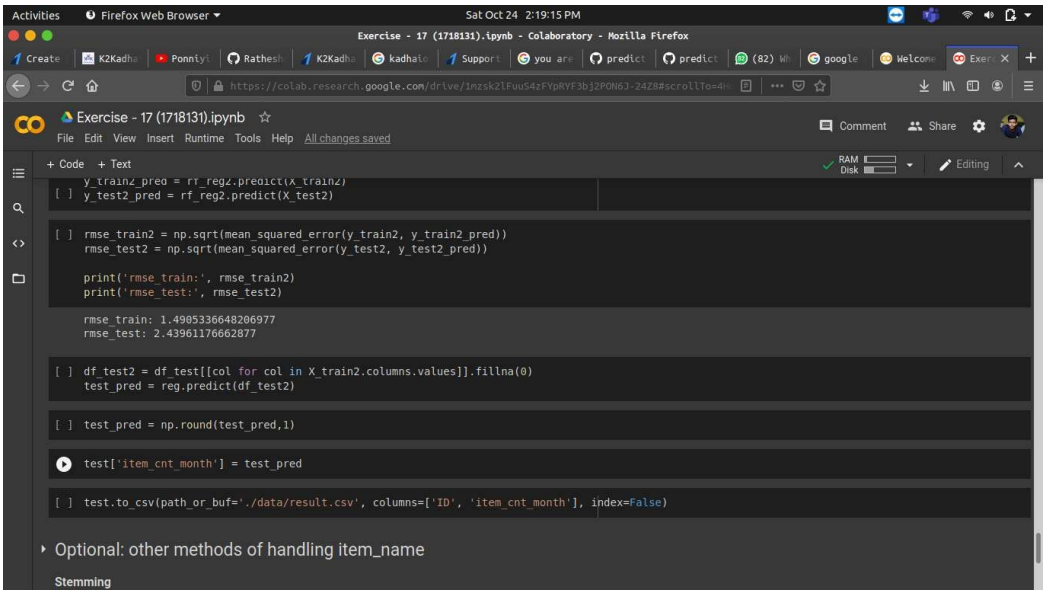
Historical Sales data analysis

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12


item_cat = pd.read_csv('./data/item_categories.csv')
items = pd.read_csv('./data/items.csv')
sales_train = pd.read_csv('./data/sales_train.csv')
test = pd.read_csv('./data/test.csv')
shops = pd.read_csv('./data/shops.csv')

feature_cnt = 25
tfidf = TfidfVectorizer(max_features=feature_cnt)
item_cat_name =
pd.DataFrame(tfidf.fit_transform(items['item_category_name']).toarray()
)

merge_dataframe(items, item_cat_name, 'item_cat_name')

tfidf = TfidfVectorizer(max_features=feature_cnt)
shop_name =
pd.DataFrame(tfidf.fit_transform(shops['shop_name']).toarray())

merge_dataframe(shops, shop_name, 'shop_name')

X = train[[col for col in train.columns.values if col not in
['item_name', 'item_category_name', 'shop_name', 'item_cnt_month',
'item_cnt_prev_month', 'item_cnt_month_mean']]].fillna(0)

y = train['item_cnt_month'].fillna(0)

list_training = list(X['date_block_num']<33)
list_testing = list(X['date_block_num']==33)

X_train2 = X[X['date_block_num']<33]
y_train2 = y[list_training].fillna(0)
X_test2 = X[X['date_block_num']==33]
y_test2 = y[list_testing].fillna(0)
```

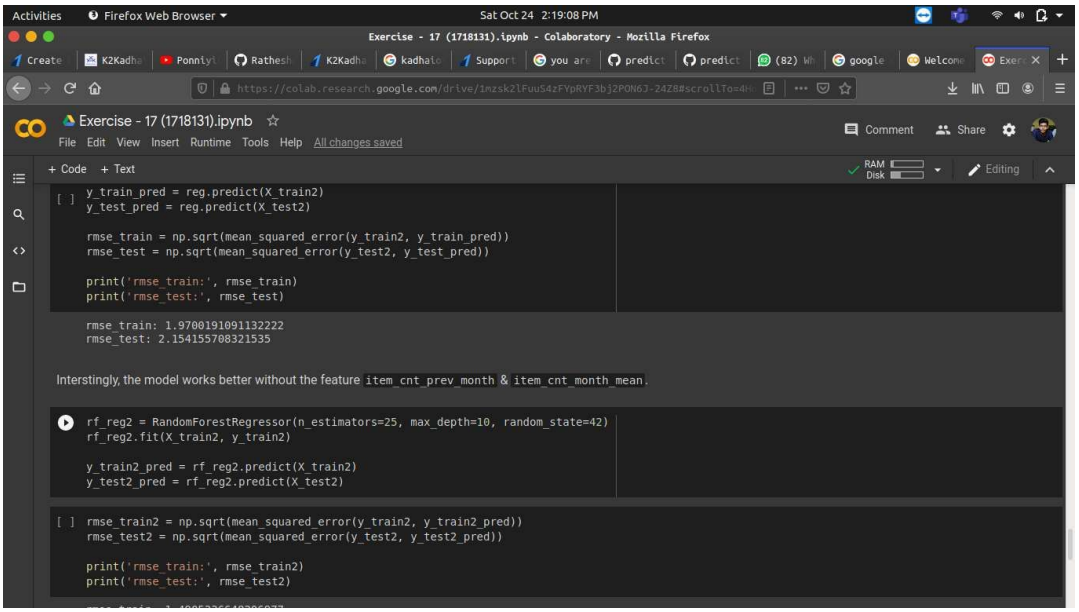
Historical Sales data analysis

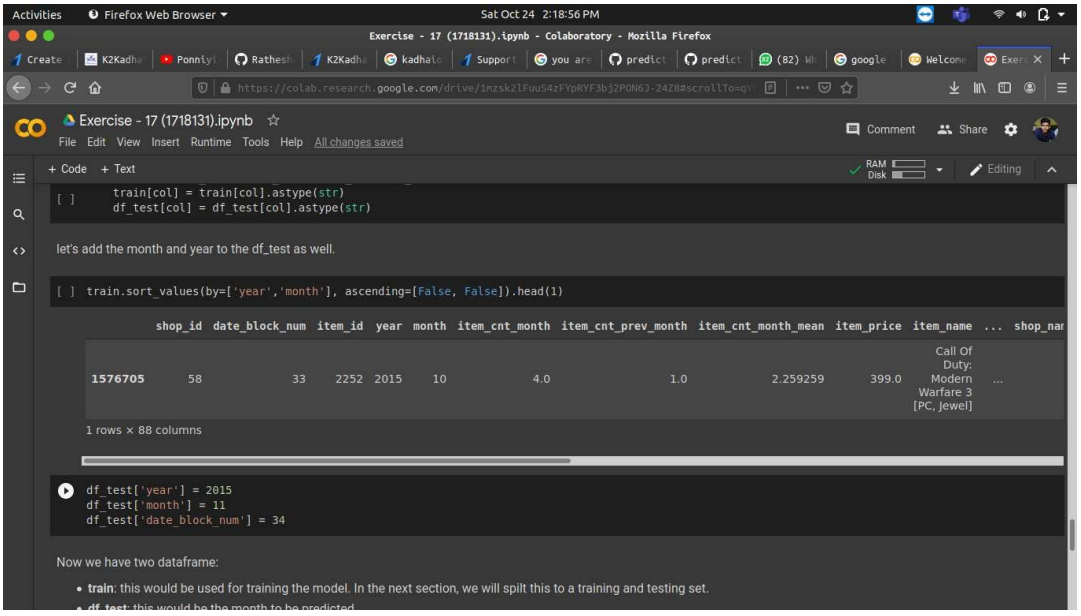
```
reg = ExtraTreesRegressor(n_estimators=25, n_jobs=-1, max_depth=15,
random_state=42)
reg.fit(X_train2, y_train2)

y_train_pred = reg.predict(X_train2)
y_test_pred = reg.predict(X_test2)

rmse_train = np.sqrt(mean_squared_error(y_train2, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test2, y_test_pred))

print('rmse_train:', rmse_train)
print('rmse_test:', rmse_test)

rmse_train2 = np.sqrt(mean_squared_error(y_train2, y_train2_pred))
rmse_test2 = np.sqrt(mean_squared_error(y_test2, y_test2_pred))

print('rmse_train:', rmse_train2)
print('rmse_test:', rmse_test2)

test.to_csv(path_or_buf='./data/result.csv', columns=['ID',
'item_cnt_month'], index=False)
```

Output :

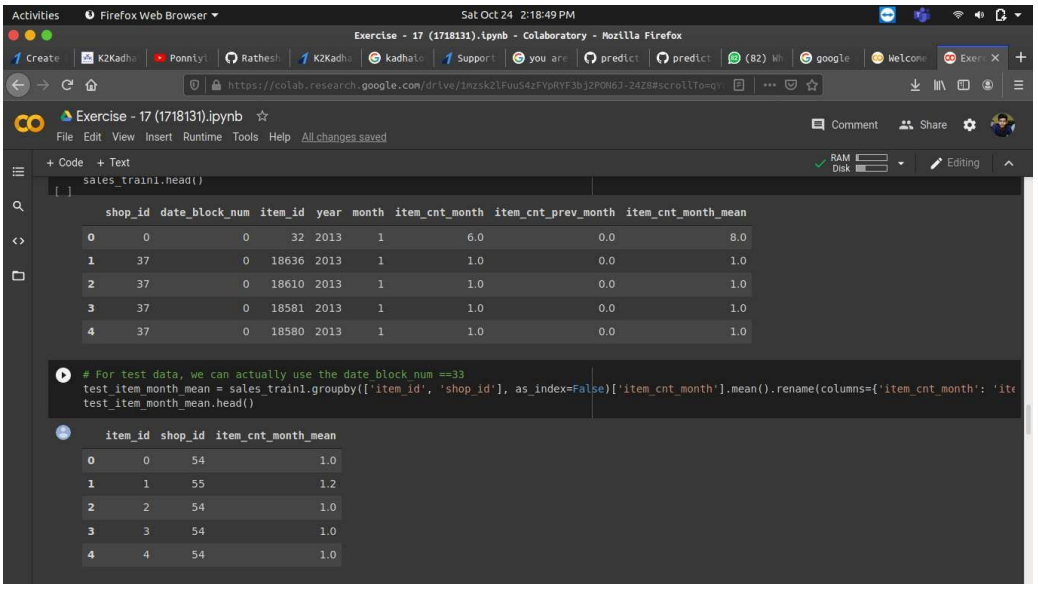


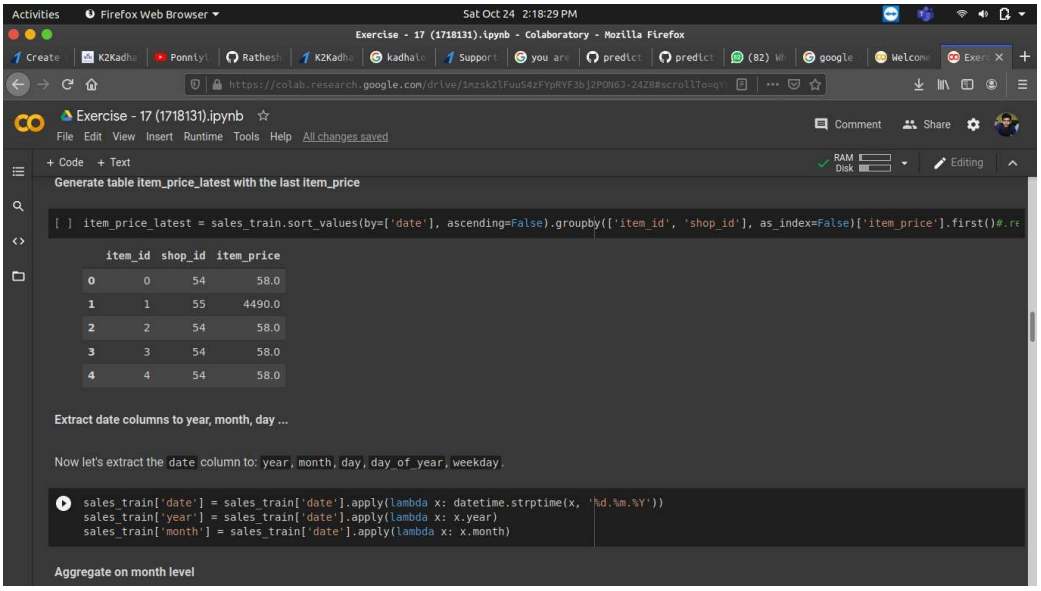
Historical Sales data analysis





Historical Sales data analysis





Result :

Thus the robust model has been created to forecast the total amount of product sold in every shop for the test.