Week-1: Find-S algorithm for finding the most specific hypothesis.

```
import io
import csv
import pandas as pd
from google.colab import files
uploaded = files.upload()

num_attributes=6
a=[]

print("The given Data set is")
with open('EXP.csv','r') as csvfile:
  reader=csv.reader(csvfile)
  for row in reader:
    a.append(row)
    print(row)

#setting initial Hypothesis
print("The initial hypothesis is")
hypothesis=['0']*num_attributes
print(hypothesis)

#Take the first 'yes' row attributes into Hypothesis
for j in range(0,num_attributes):
  hypothesis[j]=a[1][j]
print(hypothesis)

#Find the Maximum specific Hypothesis
print("Find S: Finding maximal Specific Hypothesis\n")
for i in range(1,len(a)):
  if a[i][num_attributes]=='Yes' or a[i][num_attributes]=='yes':
    for j in range(0,num_attributes):
      if a[i][j] != hypothesis[j]:
        hypothesis[j]='?'
      else:
        hypothesis[j]=a[i][j]
  print("For training instance No: {0} the hypothesis
is".format(i),hypothesis)

print("\n The Ma")
print(hypothesis)
```

Output:

```
The given Data set is
```

```
['sky', 'airtemp', 'Humadity', 'wind', 'water', 'Forecast',
'EnjoySport']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'cool', 'change', 'Yes']
```

```
The initial hypothesis is
['0', '0', '0', '0', '0', '0']
```

```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

```
Find S: Finding maximal Specific Hypothesis
```

```
For training instance No: 1 the hypothesis is ['Sunny', 'Warm',
'Normal', 'Strong', 'Warm', 'Same']
For training instance No: 2 the hypothesis is ['Sunny', 'Warm', '?',
'Strong', 'Warm', 'Same']
For training instance No: 3 the hypothesis is ['Sunny', 'Warm', '?',
'Strong', 'Warm', 'Same']
For training instance No: 4 the hypothesis is ['Sunny', 'Warm', '?',
'Strong', '?', '?']
```

```
The Ma
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Week-2: Decision tree based ID3 algorithm.

```
from google.colab import files
uploaded=files.upload()

import pandas as pd
df=pd.read_csv("week2.csv")
print(df)

def entropy(probs):
  import math
  return sum(-prob*math.log(prob,2)for prob in probs)

def entropy_of_list(a_list):
  from collections import Counter
  cnt=Counter(x for x in a_list)
  num_instances=len(a_list)
  probs=[x/num_instances for x in cnt.values()]
  return entropy(probs)
total_entropy= entropy_of_list(df['Play Tennis'])
print(total_entropy)

def information_gain(df,split_attribute_name,target_attribute_name,trac=0):
  df_split =df.groupby(split_attribute_name)
```

```
  for name,group in df_split:
    nobs=len(df.index)*1.0
    df_agg_ent=df_split.agg({target_attribute_name : [entropy_of_list,lambda
x: len(x)/nobs] })[target_attribute_name]
    avg_info=sum(df_agg_ent['entropy_of_list'] * df_agg_ent['<lambda_0>'])
    old_entropy=entropy_of_list(df[target_attribute_name])
    return old_entropy-avg_info

def id3DT(df,target_attribute_name,attribute_names,default_class=None):
  from collections import Counter
  cnt= Counter(x for x in df[target_attribute_name])
  if len(cnt)==1:
    return next(iter(cnt))
  elif df.empty or (not attribute_names):
    return default_class
  else:
    default_class=max(cnt.keys())
    gainz=[information_gain(df,attr,target_attribute_name) for attr in
attribute_names]
    index_of_max=gainz.index(max(gainz))
    best_attr=attribute_names[index_of_max]
    tree={best_attr:{}}
    remaining_attribute_names=[i for i in attribute_names if i != best_attr]
    for attr_val, data_subset in df.groupby(best_attr):
      subtree=id3DT(data_subset,target_attribute_name,remaining_attribute_name
s,default_class)
      tree[best_attr][attr_val]=subtree
    return tree

attribute_names=list(df.columns)
attribute_names.remove('Play Tennis')

from pprint import pprint
tree= id3DT(df,'Play Tennis',attribute_names)
print("The Resultant Decision Tree is ")
pprint(tree)
attribute=next(iter(tree))
print("Best Attribute: \n",attribute)
print("Tree Keys\n",tree[attribute].keys())

def classify(instance, tree, default=None): # Instance of Play Tennis with
Predicted

    #print("Instance:",instance)
    attribute = next(iter(tree)) # Outlook/Humidity/Wind
    print("Key:",tree.keys())  # [Outlook,Humidity,Wind ]
    print("Attribute:",attribute) # [Key /Attribute Both are same ]
```

```
    # print("Insance of Attribute :",instance[attribute],attribute)
    if instance[attribute] in tree[attribute].keys(): # Value of the attributs
in  set of Tree keys
        result = tree[attribute][instance[attribute]]
        print("Instance Attribute:",instance[attribute],"TreeKeys
:",tree[attribute].keys())
        if isinstance(result, dict): # this is a tree, delve deeper
            return classify(instance, result)
        else:
            return result # this is a label
    else:
        return default

tree1={'Outlook':['Rain','Sunny'],'Temperature':['Mild','Hot'],'Humidity':['No
rmal','High'],'Wind':['Strong','Weak'],'Play Tennis':['Yes','No']}
df2=pd.DataFrame(tree1)
df2['Predicted']=df2.apply(classify,axis=1,args=(tree,'No'))
print(df2)
```

Output:

```
     Outlook Temperature Humidity    Wind Play Tennis
0      Sunny         Hot     High    Weak          No
1      Sunny         Hot     High  Strong          No
2   Overcast         Hot     High    Weak         Yes
3       Rain        Mild     High    Weak         Yes
4       Rain        Cool   Normal    Weak         Yes
5       Rain        Cool   Normal  Strong          No
6   Overcast        Cool   Normal  Strong         Yes
7      Sunny        Mild     High    Weak          No
8      Sunny        Cool   Normal    Weak         Yes
9       Rain        Mild   Normal    Weak         Yes
10     Sunny        Mild   Normal  Strong         Yes
11  Overcast        Mild     High  Strong         Yes
12  Overcast         Hot   Normal    Weak         Yes
13      Rain        Mild     High  Strong          No
```

```
Key: dict_keys(['Outlook'])
Attribute: Outlook
Instance Attribute: Rain TreeKeys : dict_keys(['Overcast', 'Rain',
'Sunny'])
Key: dict_keys(['Wind'])
Attribute: Wind
Instance Attribute: Strong TreeKeys : dict_keys(['Strong', 'Weak'])
Key: dict_keys(['Outlook'])
Attribute: Outlook
Instance Attribute: Sunny TreeKeys : dict_keys(['Overcast', 'Rain',
'Sunny'])
Key: dict_keys(['Humidity'])
Attribute: Humidity
Instance Attribute: High TreeKeys : dict_keys(['High', 'Normal'])
  Outlook Temperature Humidity    Wind Play Tennis Predicted
```

```
0    Rain         Mild    Normal   Strong         Yes        No
1   Sunny          Hot      High     Weak          No        No
```

Week-3: Locally Weighted Regression algorithm.

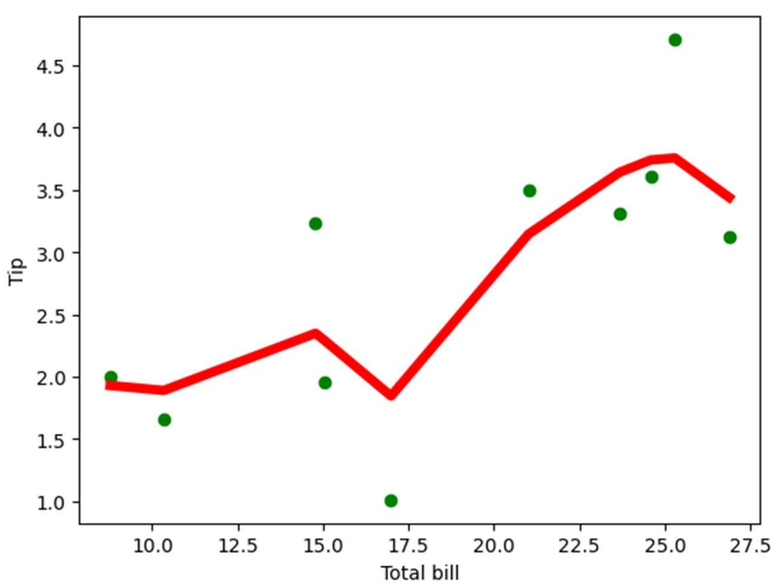
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from google.colab import files
uploaded=files.upload()

def kernal(point,xmat,k):
  m,n=np.shape(xmat)
  weights=np.mat(np.eye(m))
  print("WEIGHTS",weights)
  for j in range(m):
    diff = point - X[j]
    weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    print("WEIGHTS",weights)
  return weights

def localweight(point,xmat,ymat,k):
  print("XMAT",xmat)
  print("YMAT",ymat)
  print("K",k)
  wei = kernal(point,xmat,k)
  W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
  print("W",W)
  return W

def localWeightRegression(xmat,ymat,k):
  m,n = np.shape(xmat)
  ypred = np.zeros(m)
  print("M",m)
  for i in range(m):
    ypred[i]=xmat[i]*localweight(xmat[i],xmat,ymat,k)
    print("ypred",ypred[i])
  return ypred

data = pd.read_csv('week3.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
print("Mbill",mbill)
mtip = np.mat(tip)
print("Mtip",mtip)
```

```
m=np.shape(mbill)[1]
one=np.mat(np.ones(m))
X=np.hstack((one.T,mbill.T))
print("X",X)

ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='green')
ax.plot(xsort[:,1],ypred[SortIndex],color = 'red',linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output:

```
Mbill [[16.99 10.34 21.01 23.68 24.59 25.29  8.77 26.88 15.04 14.78]]
Mtip [[1.01 1.66 3.5  3.31 3.61 4.71 2.   3.12 1.96 3.23]]
```

```
X [[ 1.    16.99]
 [ 1.    10.34]
 [ 1.    21.01]
 [ 1.    23.68]
 [ 1.    24.59]
 [ 1.    25.29]
 [ 1.     8.77]
 [ 1.    26.88]
 [ 1.    15.04]
 [ 1.    14.78]]
```

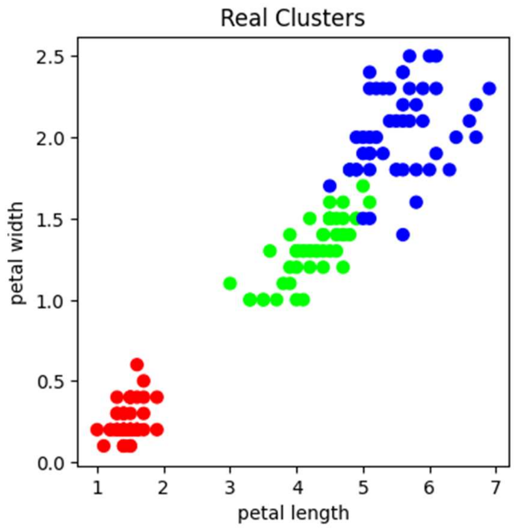


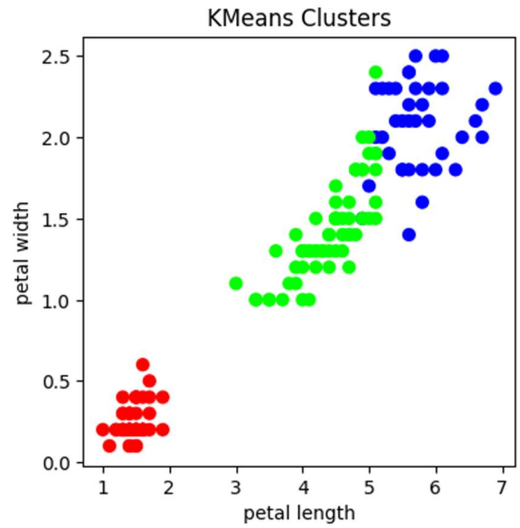
Week-4: EM algorithm & Clustering using K-means algorithm.

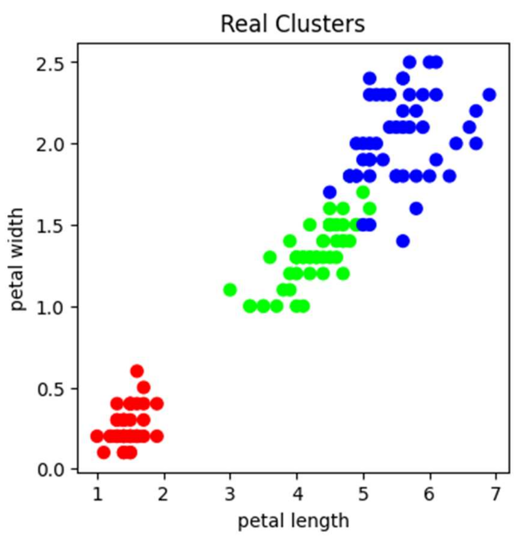
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score

# %matplotlib inline

iris=datasets.load_iris()

x=pd.DataFrame(iris.data)
x.columns=['Sepal_Length','Sepal_width','petal_length','petal_width']
print(x)

y=pd.DataFrame(iris.target)
y.columns=['Targets']
print(y)

kmeans=KMeans(n_clusters=3)
clusters=kmeans.fit_predict(x)
print(clusters)



from scipy.stats import mode
labels=np.zeros_like(clusters)
print(labels)



for i in range(3):
  cat=(clusters==i)
  labels[cat]=mode(iris.target[cat])[0]
  print(cat)


acc=accuracy_score(iris.target,labels)
print('Accuracy=',acc)

plt.figure(figsize=(10,10))
colormap=np.array(['red','lime','blue'])
```

```
plt.subplot(2,2,1)
plt.scatter(x.petal_length,x.petal_width,c=colormap[y.Targets],s=40)
plt.title('Real Clusters')
plt.xlabel('petal length')
plt.ylabel('petal width')

plt.subplot(2,2,2)
plt.scatter(x.petal_length,x.petal_width,c=colormap[labels],s=40)
plt.title('KMeans Clusters')
plt.xlabel('petal length')
plt.ylabel('petal width')




from sklearn import preprocessing

scaler=preprocessing.StandardScaler()
scaler.fit(x)
scaled_x = scaler.transform(x)
xs = pd.DataFrame(scaled_x,columns = x.columns)
from sklearn.mixture import GaussianMixture

gmm=GaussianMixture(n_components=3)
gmm_y=gmm.fit_predict(xs)

labels=np.zeros_like(clusters)

for i in range(3):
  cat=(gmm_y==i)
  labels[cat]=mode(iris.target[cat])[0]

acc=accuracy_score(iris.target,labels)
print("Accuracy using GMM= ",acc)


plt.subplot(2,2,3)
plt.scatter(x.petal_length,x.petal_width,c=colormap[y.Targets],s=40)
plt.subplots_adjust(hspace=0.4,wspace=0.4)
plt.title('Real Clusters')
plt.xlabel('petal length')
plt.ylabel('petal width')
```

Output:







Week-5: KNN algorithm.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
import pandas as pd
import numpy as np

iris=datasets.load_iris()
print("Iris Data set loaded...")

x_train, x_test, y_train, y_test=
train_test_split(iris.data,iris.target,test_size=0.2)
```

```
print("Dataset is split into Training and Testing...")
print("Size of training data and it's label",x_train.shape,y_train.shape)
print("Size of Test data and it's label",x_test.shape,y_test.shape)

for i in range(len(iris.target_names)):
  print("Label",i,"-",str(iris.target_names[i]))
classifier=KNeighborsClassifier(n_neighbors=1)

classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)

print("Result of Classification using K-NN with K=1")
for r in range(0,len(x_test)):
  print("Sample:",str(x_test[r]),"Actual-label:",str(y_test[r]),"Predicted-
Label:",str(y_pred[r]))
print("Classification Accuracy:",classifier.score(x_test,y_test))

from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix")
print(confusion_matrix(y_test,y_pred))
print("Accuracy Matrix")
print(classification_report(y_test,y_pred))
```

Output:

```
Dataset is split into Training and Testing...
Size of training data and it's label (120, 4) (120,)
Size of Test data and it's label (30, 4) (30,)
```

```
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
```

```
Result of Classification using K-NN with K=1
Sample: [5.7 4.4 1.5 0.4] Actual-label: 0 Predicted-Label: 0
Sample: [4.4 3.2 1.3 0.2] Actual-label: 0 Predicted-Label: 0
Sample: [6.1 2.6 5.6 1.4] Actual-label: 2 Predicted-Label: 2
Sample: [7.2 3.  5.8 1.6] Actual-label: 2 Predicted-Label: 2
Sample: [5.  3.5 1.6 0.6] Actual-label: 0 Predicted-Label: 0
Sample: [6.9 3.1 5.4 2.1] Actual-label: 2 Predicted-Label: 2
Sample: [5.  3.3 1.4 0.2] Actual-label: 0 Predicted-Label: 0
Sample: [6.  2.9 4.5 1.5] Actual-label: 1 Predicted-Label: 1
Sample: [6.9 3.1 5.1 2.3] Actual-label: 2 Predicted-Label: 2
Sample: [4.9 3.  1.4 0.2] Actual-label: 0 Predicted-Label: 0
Sample: [4.4 2.9 1.4 0.2] Actual-label: 0 Predicted-Label: 0
Sample: [6.3 3.4 5.6 2.4] Actual-label: 2 Predicted-Label: 2
Sample: [6.2 2.8 4.8 1.8] Actual-label: 2 Predicted-Label: 2
Sample: [6.2 2.9 4.3 1.3] Actual-label: 1 Predicted-Label: 1
Sample: [6.  2.2 4.  1. ] Actual-label: 1 Predicted-Label: 1
Sample: [6.1 3.  4.6 1.4] Actual-label: 1 Predicted-Label: 1
Sample: [7.4 2.8 6.1 1.9] Actual-label: 2 Predicted-Label: 2
Sample: [6.2 2.2 4.5 1.5] Actual-label: 1 Predicted-Label: 1
Sample: [7.7 3.  6.1 2.3] Actual-label: 2 Predicted-Label: 2
```

```
Sample:  [4.8 3.  1.4 0.1] Actual-label: 0 Predicted-Label: 0
Sample:  [6.8 3.  5.5 2.1] Actual-label: 2 Predicted-Label: 2
Sample:  [5.5 2.5 4.  1.3] Actual-label: 1 Predicted-Label: 1
Sample:  [6.4 3.1 5.5 1.8] Actual-label: 2 Predicted-Label: 2
Sample:  [7.9 3.8 6.4 2. ] Actual-label: 2 Predicted-Label: 2
Sample:  [5.6 2.5 3.9 1.1] Actual-label: 1 Predicted-Label: 1
Sample:  [6.7 3.  5.  1.7] Actual-label: 1 Predicted-Label: 1
Sample:  [6.5 2.8 4.6 1.5] Actual-label: 1 Predicted-Label: 1
Sample:  [5.7 2.9 4.2 1.3] Actual-label: 1 Predicted-Label: 1
Sample:  [5.1 3.8 1.6 0.2] Actual-label: 0 Predicted-Label: 0
Sample:  [6.3 2.5 4.9 1.5] Actual-label: 1 Predicted-Label: 2
Classification Accuracy: 0.9666666666666667
```

```
Confusion Matrix
[[ 8  0  0]
 [ 0 10  1]
 [ 0  0 11]]
Accuracy Matrix
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      0.91      0.95        11
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

Week-6: Back Propagation algorithm.

```
import numpy as np
x=np.array(([2,9],[1,5],[3,6]))
y=np.array(([92],[86],[89]))

y=y/100

def sigmoid(x):
  return 1/(1+np.exp(-x))

def derivatives_sigmoid(x):
  return x*(1-x)

epoch=10000
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bias_hidden=np.random.uniform(size=(1,hiddenlayer_neurons))
weight_hidden=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
```

```
bias_output=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
  hinp1=np.dot(x,wh)
  hinp=hinp1+bias_hidden
  hlayer_activation=sigmoid(hinp)
  outinp1=np.dot(hlayer_activation,weight_hidden)
  outinp=outinp1+bias_output
  output=sigmoid(outinp)

  E0=y-output
  outgrad=derivatives_sigmoid(output)
  d_output=E0*outgrad
  EH=d_output.dot(weight_hidden.T)

  hiddengrad=derivatives_sigmoid(hlayer_activation)
  d_hiddenlayer=EH*hiddengrad

  weight_hidden+=hlayer_activation.T.dot(d_output)*lr
  bias_hidden+=np.sum(d_hiddenlayer,axis=0,keepdims=True)*lr

  wh+=x.T.dot(d_hiddenlayer)*lr
  bias_output+=np.sum(d_output,axis=0,keepdims=True)*lr

print("Input:\n"+str(x))
print("Actual Output:\n"+str(y))
print("Predicted Output:\n",output)
```

Output:

```
Input:
[[2 9]
 [1 5]
 [3 6]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89183508]
 [0.88240132]
 [0.89476846]]
```

Week-8: Eligibility of a customer for a loan.

```
import pandas as pd
import numpy as np
import seaborn as sns
```

```python
import matplotlib.pyplot as plt

"""1.Gathering Data"""

from google.colab import files
uploaded=files.upload()

#  Create New Variable and stores the dataset values as Data Frame
# READING DATA
train=pd.read_csv("train.csv")
train.head()

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

train # We have 614 rows and 13 columns in the train dataset.

test  #We have 367 rows and 12 columns in test dataset.


print("Rows: ", len(train))

print("Columns: ", len(train.columns))


print(train.shape, test.shape)


train.columns  # print the list of columns

train_columns = train.columns # assign to a variable
train_columns # print the list of columns


train.describe()


train.info()

train['Loan_Status'].value_counts().plot.bar()

train['Gender'].value_counts(normalize=True).plot.bar(title='Gender')
plt.show()
train['Married'].value_counts(normalize=True).plot.bar(title='Married')
plt.show()
train['Self_Employed'].value_counts(normalize=True).plot.bar(title='Self_emplo
yee')
plt.show()
```

```
train['Credit_History'].value_counts(normalize=True).plot.bar(title='Credit_History')
plt.show()

train['Dependents'].value_counts(normalize=True).plot.bar(title='Dependents')
plt.show()
train['Education'].value_counts(normalize=True).plot.bar(title='Education')
plt.show()
train['Property_Area'].value_counts(normalize=True).plot.bar(title='Property_Area')
plt.show()

train['ApplicantIncome'].value_counts(normalize=True).plot.bar(title='ApplicantIncome')
plt.show()
train['CoapplicantIncome'].plot.box()
plt.show()
train['LoanAmount'].plot.box()
plt.show()

train.boxplot(column='ApplicantIncome', by = 'Education')
plt.suptitle(' ')
plt.show()

Gender=pd.crosstab(train['Gender'],train['Loan_Status'])
Gender.div(Gender.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsize=(4,4))

train.groupby('Loan_Status')['ApplicantIncome'].mean().plot.bar()

bins=[0,2500,4000,6000,81000]
group=['Low','Average','High','Very High']
train['Income_bin']=pd.cut(train['ApplicantIncome'],bins,labels=group)
Income_bin=pd.crosstab(train['Income_bin'],train['Loan_Status'])
Income_bin.div(Income_bin.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True)
plt.xlabel('ApplicantIncome')
P=plt.ylabel('Percentage')

matrix=train.corr()
f,ax=plt.subplots(figsize=(9,6))
sns.heatmap(matrix,vmax=.8,square=True,cmap="BuPu",annot=True)

train.isnull().sum()

train['Gender'].fillna(train['Gender'].mode()[0],inplace=True)
print(train['Gender'])
```

```
train['Married'].fillna(train['Married'].mode()[0],inplace=True)
print(train['Married'])

train['Dependents'].fillna(train['Dependents'].mode()[0],inplace=True)
print(train['Dependents'])

train['Self_Employed'].fillna(train['Self_Employed'].mode()[0],inplace=True)
print(train['Self_Employed'])

train['LoanAmount'].fillna(train['LoanAmount'].mode()[0],inplace=True)
print(train['LoanAmount'])

train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mean(),inplace=True
)
print(train['LoanAmount'])

train['Credit_History'].fillna(train['Credit_History'].mean(),inplace=True)
print(train['Credit_History'])

# let's check whether all the missing values are filled in the dataset.
train.isnull().sum()

X=train.drop('Loan_Status',1)
#X=train.drop('Loan_ID')
X=X.drop('Loan_ID',axis=1)
Y=train['Loan_Status']

X=pd.get_dummies(X)

print(X)
print(Y)


def explore_object_type(df ,feature_name):
    """
    if df[feature_name].dtype ==  'object':
        print(df[feature_name].value_counts())

# After defined a function, Let's call it. and check what's the output of our
created function.
# Now, Test and Call a function for gender only
explore_object_type(train, 'Gender')

# Here's one little issue occurred, Suppose in your datasets there are lots of
feature to defined like this above code.
# Solution is, Do you remember we have variable with name of
`loan_train_columns`, Right,  let's use it
```

```
# 'Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
'Property_Area', 'Loan_Status'

for featureName in train_columns:
    if train[featureName].dtype == 'object':
        print('\n"' + str(featureName) + '\'s" Values with count are :')
        explore_object_type(train, str(featureName))

# We need to fill null values with mean and median using missingno package
import missingno as msno
# list of how many percentage values are missing
train

train.isna().sum()
# round((loan_train.isna().sum() / len(loan_train)) * 100, 2)

msno.bar(train)

msno.matrix(train )

# As we can see here, there are too many columns missing with small amount of
null values so we use mean amd mode to replace with NaN values.
train['Credit_History'].fillna(train['Credit_History'].mode(), inplace=True) #
Mode
test['Credit_History'].fillna(test['Credit_History'].mode(), inplace=True) #
Mode


train['LoanAmount'].fillna(train['LoanAmount'].mean(), inplace=True) # Mean
test['LoanAmount'].fillna(test['LoanAmount'].mean(), inplace=True) # Mean

# convert Categorical variable with Numerical values.Loan_Status feature
boolean values,
# So we replace Y values with 1 and N values with 0 and same for other Boolean
types of columns
train.Loan_Status = train.Loan_Status.replace({"Y": 1, "N" : 0})
# test.Loan_Status = loan_test.Loan_Status.replace({"Y": 1, "N" : 0})

train.Gender = train.Gender.replace({"Male": 1, "Female" : 0})
test.Gender = test.Gender.replace({"Male": 1, "Female" : 0})

train.Married = train.Married.replace({"Yes": 1, "No" : 0})
test.Married = test.Married.replace({"Yes": 1, "No" : 0})

train.Self_Employed = train.Self_Employed.replace({"Yes": 1, "No" : 0})
test.Self_Employed = test.Self_Employed.replace({"Yes": 1, "No" : 0})

train['Gender'].fillna(train['Gender'].mode()[0], inplace=True)
test['Gender'].fillna(test['Gender'].mode()[0], inplace=True)
```

```python
train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
test['Dependents'].fillna(test['Dependents'].mode()[0], inplace=True)

train['Married'].fillna(train['Married'].mode()[0], inplace=True)
test['Married'].fillna(test['Married'].mode()[0], inplace=True)

train['Credit_History'].fillna(train['Credit_History'].mean(), inplace=True)
test['Credit_History'].fillna(test['Credit_History'].mean(), inplace=True)

# Here, Property_Area, Dependents and Education has multiple values so now we
can use LabelEncoder from sklearn package
from sklearn.preprocessing import LabelEncoder
feature_col = ['Property_Area','Education', 'Dependents']
le = LabelEncoder()
for col in feature_col:
    train[col] = le.fit_transform(train[col])
    test[col] = le.fit_transform(test[col])

"""**Finally, We have all the features with numerical values**

**3. Data Visualizations**
"""

# Commented out IPython magic to ensure Python compatibility.
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns
sns.set_style('dark')

train
train.plot(figsize=(18, 8))
plt.show()

plt.figure(figsize=(18, 6))
plt.subplot(1, 2, 1)


train['ApplicantIncome'].hist(bins=10)
plt.title("Loan Application Amount ")

plt.subplot(1, 2, 2)
plt.grid()
plt.hist(np.log(train['LoanAmount']))
plt.title("Log Loan Application Amount ")

plt.show()

plt.figure(figsize=(18, 6))
```

```
plt.title("Relation Between Applicatoin Income vs Loan Amount ")

plt.grid()
plt.scatter(train['ApplicantIncome'] , train['LoanAmount'], c='k', marker='x')
plt.xlabel("Applicant Income")
plt.ylabel("Loan Amount")
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(train['Loan_Status'], train['LoanAmount'])
plt.title("Loan Application Amount ")
plt.show()

plt.figure(figsize=(12,8))
sns.heatmap(train.corr(), cmap='coolwarm', annot=True, fmt='.1f',
linewidths=.1)
plt.show()

train['Total_Income']=train['ApplicantIncome']+train['CoapplicantIncome']
test['Total_Income']=test['ApplicantIncome']+test['CoapplicantIncome']
sns.distplot(train['Total_Income'])
# Let's check the distribution of Total Income.

train['EMI']=train['LoanAmount']/train['Loan_Amount_Term']
test['EMI']=test['LoanAmount']/test['Loan_Amount_Term']
sns.distplot(train['EMI'])

# Let's check the distribution of the EMI variable.

train['Balance Income'] = train['Total_Income']-(train['EMI']*1000)
test['Balance Income']=test['Total_Income']-(test['EMI']*1000)
sns.distplot(train['Balance Income'])
# Let's check the distribution of the Balance Income

# import ml model from sklearn pacakge

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

# Let's build the model
logistic_model = LogisticRegression()


train_features = ['Credit_History', 'Education', 'Gender']

x_train = train[train_features].values
y_train = train['Loan_Status'].values
```

```python
x_test = test[train_features].values
logistic_model.fit(x_train, y_train)

"""**6. Predict Model**"""

# Predict the model for testin data
predicted = logistic_model.predict(x_test)

# check the coefficeints of the trained model
print('Coefficient of model :', logistic_model.coef_)

# check the intercept of the model
print('Intercept of model',logistic_model.intercept_)

# Accuray Score on train dataset
# accuracy_train = accuracy_score(x_test, predicted)
score = logistic_model.score(x_train, y_train)
print('accuracy_score overall :', score)
print('accuracy_score percent :', round(score*100,2))

# predict the target on the test dataset
predict_test = logistic_model.predict(x_test)
print('Target on test data',predict_test)

x=len(predict_test)
print(x)
```

Output:

```
Coefficient of model : [[ 3.316164   -0.3059193   0.09398266]]
Intercept of model [-1.98307795]
accuracy_score overall : 0.8094462540716613
accuracy_score percent : 80.94
```

```
Target on test data [1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 1 0 1 1
1 1
 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1
1 1
 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1
1 0
 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
1 1
 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1
1 1
 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
1 1
```

```
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1
1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
367
```

Week-10: Predict the winning team in IPL.

```
from google.colab import files
uploaded=files.upload()

import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

#importing the csv file
import pandas as pd
data=pd.read_csv("matches.csv")
data.head()

# Let’s get a brief summary of the IPL dataset.

data.describe()

# Checking whether there are any null values present in the dataset.

data.isnull().sum()

"""# information about IPL Matches
data.info()
"""

# Checking the columns
data.columns

data.drop(['umpire2'],axis=1,inplace=True)

for col in data.columns:
    print(data[col].unique())

#finding the unique values of all columns
# we will get all team names

# As you can see, the values of umpire3 are null in almost all rows, so we are
dropping the column umpire3.
```

```
# And also dropped some rows containing the null values after removing the
umpire3 column.
data = data.iloc[:,:-1]
data.dropna(inplace=True)

# Now let’s look into the total teams listed in this dataset.
data["team1"].unique()

# Here you can see the name Delhi Daredevils and Delhi Capitals; Delhi
Daredevils is the old name of the Delhi Capitals.
# Similarly, Decan Chargers is the old name of Sunrisers Hyderabad. So we are
changing the old name to the newer one.

#for Delhi Capitals
data['team1']=data['team1'].str.replace('Delhi Daredevils','Delhi Capitals')
data['team2']=data['team2'].str.replace('Delhi Daredevils','Delhi Capitals')
data['winner']=data['winner'].str.replace('Delhi Daredevils','Delhi Capitals')

#for sunrisers Hyderabad
data['team1']=data['team1'].str.replace('Deccan Chargers','Sunrisers
Hyderabad')
data['team2']=data['team2'].str.replace('Deccan Chargers','Sunrisers
Hyderabad')
data['winner']=data['winner'].str.replace('Deccan Chargers','Sunrisers
Hyderabad')

"""VISUALIZATIONS"""

# Visualizations
# Number of IPL matches won by each team.

plt.figure(figsize = (10,6))
sns.countplot(y = 'winner',data = data,order=
data['winner'].value_counts().index)
plt.xlabel('Wins')

plt.title('Number of  IPL  matches won by each team')

# Total number of matches played in a different stadium

plt.figure(figsize = (10,6))
sns.countplot(y = 'venue',data = data,order =
data['venue'].value_counts().iloc[:10].index)
plt.xlabel('No of matches',fontsize=12)
plt.ylabel('Venue',fontsize=12)
plt.title('Total Number of matches played in different stadium')

# The decision was taken by the toss winning team.
```

```
plt.figure(figsize = (10,6))
sns.countplot(x = "toss_decision", data=data)
plt.xlabel('Toss Decision',fontsize=12)
plt.ylabel('Count',fontsize=12)
plt.title('Toss Decision')

team_wins_ser = data['winner'].value_counts()

team_wins_df = pd.DataFrame(columns=["team", "wins"])
for items in team_wins_ser.iteritems():
    temp_df1 = pd.DataFrame({
        'team':[items[0]],
        'wins':[items[1]]
    })
    team_wins_df = team_wins_df.append(temp_df1, ignore_index=True)

plt.title("Total Victories of IPL Teams")
sns.barplot(x='wins', y='team', data=team_wins_df, palette='Paired');

mvp_ser = data['player_of_match'].value_counts()

mvp_ten_df = pd.DataFrame(columns=["player", "wins"])
count = 0
for items in mvp_ser.iteritems():
    if count>9:
        break
    else:
        temp_df2 = pd.DataFrame({
            'player':[items[0]],
            'wins':[items[1]]
        })
        mvp_ten_df = mvp_ten_df.append(temp_df2, ignore_index=True)
        count += 1
plt.title("Top Ten IPL Players")
sns.barplot(x='wins', y='player', data=mvp_ten_df, palette='Paired');

toss_ser = data['toss_winner'].value_counts()

toss_df = pd.DataFrame(columns=["team", "wins"])

for items in toss_ser.iteritems():
    temp_df3 = pd.DataFrame({
        'team':[items[0]],
        'wins':[items[1]]
    })
    toss_df = toss_df.append(temp_df3, ignore_index=True)
```

```
plt.title("How IPL Teams fared in toss?")
sns.barplot(x='wins', y='team', data=toss_df, palette='Paired');

data.loc[data["winner"]==data["team1"],"team1_win"]=1
data.loc[data["winner"]!=data["team1"],"team1_win"]=0

#outcome variable team1_toss_win as a value of team1 winning the toss
data.loc[data["toss_winner"]==data["team1"],"team1_toss_win"]=1
data.loc[data["toss_winner"]!=data["team1"],"team1_toss_win"]=0

#outcome variable team1_bat to depict if team1 bats first
data["team1_bat"]=0
data.loc[(data["team1_toss_win"]==1) &
(data["toss_decision"]=="bat"),"team1_bat"]=1

prediction_df=data[["team1","team2","team1_toss_win","team1_bat","team1_win","
venue"]]

#finding the higly correlated features
correlated_features = set()
correlation_matrix = prediction_df.drop('team1_win', axis=1).corr()

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.9:
            column = correlation_matrix.columns[i]
            correlated_features.add(column)

prediction_df.drop(columns=correlated_features)

# Now let’s check the unique values presented in each feature.

x = ["city", "toss_decision", "result", "dl_applied"]
for i in x:
  print("------------")
  print(data[i].unique())
  print(data[i].value_counts())

# The output data is also a categorical value, so we are converting it into
numerical using LabelEncoder of sklearn.

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(x)
# Now let’s convert our data into a training set in order to create the model
and test set for evaluating the created model.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8)
```

```
teams_per_season = data.groupby('season')['winner'].value_counts()
teams_per_season

year = 2008
winteamseason = pd.DataFrame(columns=['year', 'team', 'wins'])
winteamseason

for items in teams_per_season.iteritems():
 if items[0][0]==year:
   print(items)
   win_series = pd.DataFrame({
       'year': [items[0][0]],
        'team': [items[0][1]],
        'wins': [items[1]]
         })
   winteamseason = winteamseason.append(win_series)
   year += 1
print(winteamseason)
```

Output:

```
season winner 2008 Rajasthan Royals 13 Kings XI Punjab 10 Chennai Super
Kings 9 Delhi Capitals 7 Mumbai Indians 7 .. 2019 Kings XI Punjab 6
Kolkata Knight Riders 6 Sunrisers Hyderabad 6 Rajasthan Royals 5 Royal
Challengers Bangalore 5 Name: winner, Length: 100, dtype: int64
```

```
((2008, 'Rajasthan Royals'), 13)
((2009, 'Delhi Capitals'), 10)
((2010, 'Mumbai Indians'), 11)
((2011, 'Chennai Super Kings'), 11)
((2012, 'Kolkata Knight Riders'), 12)
((2013, 'Mumbai Indians'), 13)
((2014, 'Kings XI Punjab'), 11)
((2015, 'Chennai Super Kings'), 10)
((2016, 'Sunrisers Hyderabad'), 11)
((2017, 'Mumbai Indians'), 12)
((2018, 'Chennai Super Kings'), 11)
((2019, 'Mumbai Indians'), 11)
   year                   team wins
0  2008       Rajasthan Royals   13
0  2009         Delhi Capitals   10
0  2010         Mumbai Indians   11
0  2011    Chennai Super Kings   11
0  2012  Kolkata Knight Riders   12
0  2013         Mumbai Indians   13
0  2014        Kings XI Punjab   11
0  2015    Chennai Super Kings   10
0  2016    Sunrisers Hyderabad   11
0  2017         Mumbai Indians   12
0  2018    Chennai Super Kings   11
0  2019         Mumbai Indians   11
```