Practical 1

Implement three different approaches to Load CSV file using Python.

import pandas as pd
df=pd.read_csv("C:/Users/admin/Downloads/nba.csv") print(df.head(5))

```
In [1]: runfile('C:/Users/admin/.spyder-py3/p1.py', wdir='C:/Users/
admin/.spyder-py3')
            Name            Team  Number  ... Weight            College
Salary
0  Avery Bradley  Boston Celtics     0.0  ...  180.0              Texas
7730337.0
1    Jae Crowder  Boston Celtics    99.0  ...  235.0          Marquette
6796117.0
2   John Holland  Boston Celtics    30.0  ...  205.0  Boston University
NaN
3    R.J. Hunter  Boston Celtics    28.0  ...  185.0      Georgia State
1148640.0
4  Jonas Jerebko  Boston Celtics     8.0  ...  231.0                NaN
5000000.0

[5 rows x 9 columns]
```

import numpy as np
data=np.genfromtxt("C:/Users/admin/Downloads/nba.csv", delimiter=',', dtype=None,names=True, encoding='utf-8')
print(pd.DataFrame(data).head(10))

```
            Name            Team  Number  ... Weight            College
Salary
0  Avery Bradley  Boston Celtics     0.0  ...  180.0              Texas
7730337.0
1    Jae Crowder  Boston Celtics    99.0  ...  235.0          Marquette
6796117.0
2   John Holland  Boston Celtics    30.0  ...  205.0  Boston University
NaN
3    R.J. Hunter  Boston Celtics    28.0  ...  185.0      Georgia State
1148640.0
4  Jonas Jerebko  Boston Celtics     8.0  ...  231.0
5000000.0
5   Amir Johnson  Boston Celtics    90.0  ...  240.0
12000000.0
6  Jordan Mickey  Boston Celtics    55.0  ...  235.0                LSU
1170960.0
7   Kelly Olynyk  Boston Celtics    41.0  ...  238.0            Gonzaga
2165160.0
8   Terry Rozier  Boston Celtics    12.0  ...  190.0         Louisville
1824360.0
9   Marcus Smart  Boston Celtics    36.0  ...  220.0     Oklahoma State
3431040.0

[10 rows x 9 columns]
```

```
def load_csv(filepath):
    data =  []
    col = []   checkcol =
False     with open(filepath)
as f:         for val in
f.readlines():
val = val.replace("\n","")
val = val.split(',')             if
checkcol is False:
col = valcheckcol = True
else:
data.append(val)
    df = pd.DataFrame(data=data, columns=col)
return df
myData = load_csv("C:/Users/admin/Downloads/nba.csv") print(myData.head(15))
```

```
               Name            Team  ...            College      Salary
0     Avery Bradley  Boston Celtics  ...              Texas   7730337.0
1       Jae Crowder  Boston Celtics  ...          Marquette   6796117.0
2      John Holland  Boston Celtics  ...  Boston University
3       R.J. Hunter  Boston Celtics  ...      Georgia State   1148640.0
4     Jonas Jerebko  Boston Celtics  ...                      5000000.0
5      Amir Johnson  Boston Celtics  ...                     12000000.0
6     Jordan Mickey  Boston Celtics  ...                LSU   1170960.0
7      Kelly Olynyk  Boston Celtics  ...            Gonzaga   2165160.0
8      Terry Rozier  Boston Celtics  ...         Louisville   1824360.0
9      Marcus Smart  Boston Celtics  ...     Oklahoma State   3431040.0
10  Jared Sullinger  Boston Celtics  ...         Ohio State   2569260.0
11    Isaiah Thomas  Boston Celtics  ...         Washington   6912869.0
12      Evan Turner  Boston Celtics  ...         Ohio State   3425510.0
13      James Young  Boston Celtics  ...           Kentucky   1749840.0
14     Tyler Zeller  Boston Celtics  ...     North Carolina   2616975.0

[15 rows x 9 columns]
```

Practical 2

Demonstrate statistical operations on pima-indians-diabetes.csv dataset available on Kaggle repository. For Example, checking dimensions of data, checking attribute type, Mean, Standard Deviation, Median, count, five-number summary.

```
import numpy as np import
pandas as pd
df = pd.read_csv("/content/diabetes.csv")
print(df.ndim) print(df.shape)
print(df.size) print(df.Pregnancies.dtype)
print(df.BMI.dtype)
```

```
2
(768, 9)
6912
int64
float64
```

print(df.info())

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

print(df.describe())

```
       Pregnancies     Glucose  ...         Age     Outcome
count   768.000000  768.000000  ...  768.000000  768.000000
mean      3.845052  120.894531  ...   33.240885    0.348958
std       3.369578   31.972618  ...   11.760232    0.476951
min       0.000000    0.000000  ...   21.000000    0.000000
25%       1.000000   99.000000  ...   24.000000    0.000000
50%       3.000000  117.000000  ...   29.000000    0.000000
75%       6.000000  140.250000  ...   41.000000    1.000000
max      17.000000  199.000000  ...   81.000000    1.000000

[8 rows x 9 columns]
```

df.head()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

data.mean()

```
prag       3.85
plas     120.89
pres      69.11
skin      20.54
test      79.80
mass      31.99
pedi       0.47
age       33.24
class      0.35
dtype: float64
```

data['skin'].mean()

```
20.536458333333332
```

data.median()

```
prag       3.00
plas     117.00
pres      72.00
skin      23.00
test      30.50
mass      32.00
pedi       0.37
age       29.00
class      0.00
dtype: float64
```

data['skin'].mode()

```
0    0
dtype: int64
```

data.std()

```
prag        3.37
plas       31.97
pres       19.36
skin       15.95
test      115.24
mass        7.88
pedi        0.33
age        11.76
class       0.48
dtype: float64
```

data.min()

```
prag       0.00
plas       0.00
pres       0.00
skin       0.00
test       0.00
mass       0.00
pedi       0.08
age       21.00
class      0.00
dtype: float64
```

data.max()

```
prag       17.00
plas      199.00
pres      122.00
skin       99.00
test      846.00
mass       67.10
pedi        2.42
age        81.00
class       1.00
dtype: float64
```

Five number summary:

print(f'min={min}\n1st quartile={quartiles[0]} \nmedian={quartiles[1]} \n3rd quartile={quartiles[2]} \nmax={max}')

```
min=0
1st quartile=1.0
median=3.0
3rd quartile=6.0
max=17
```

Practical 3

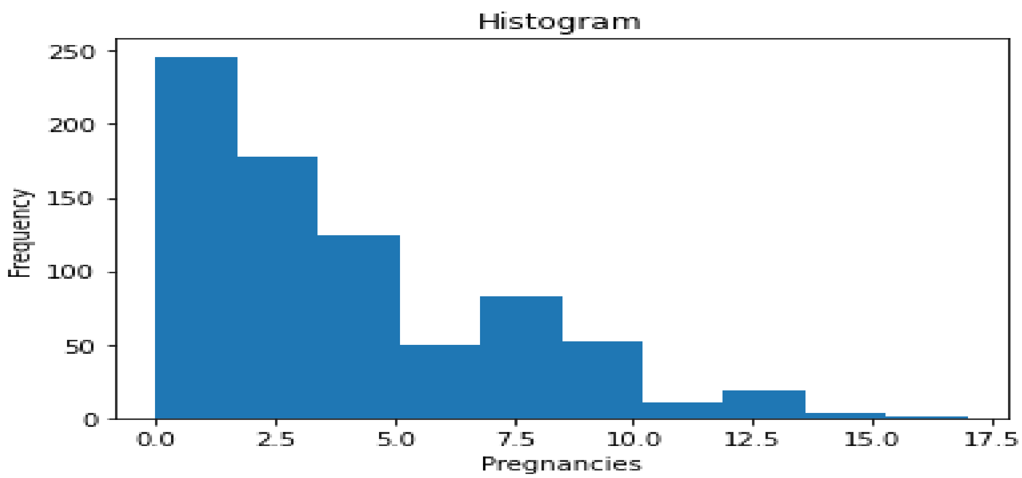
Write python script creating histogram of the attributes of pimaindians-diabetes.csv dataset available on Kaggle repository and plot different graphs.

import pandas as pd import
matplotlib.pyplot as plt import
%matplotlib inline

df = pd.read_csv("/content/diabetes.csv") df.head(10)

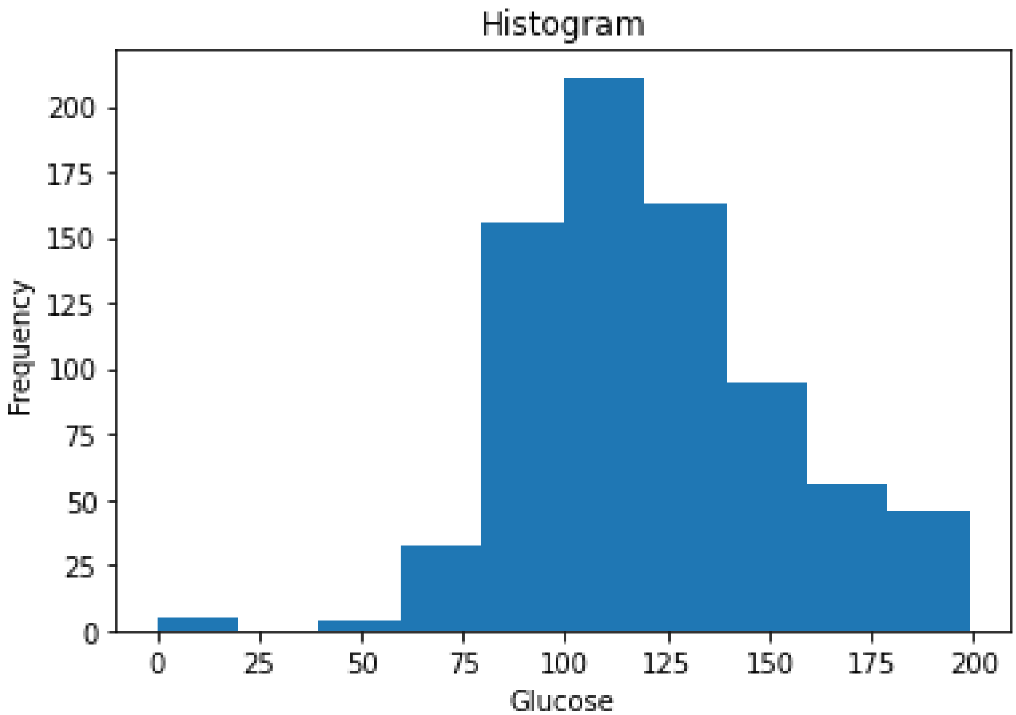
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

plt.hist(data['Pregnancies'])
plt.xlabel('Pregnancies')
plt.ylabel('Frequency')
plt.title('Histogram')

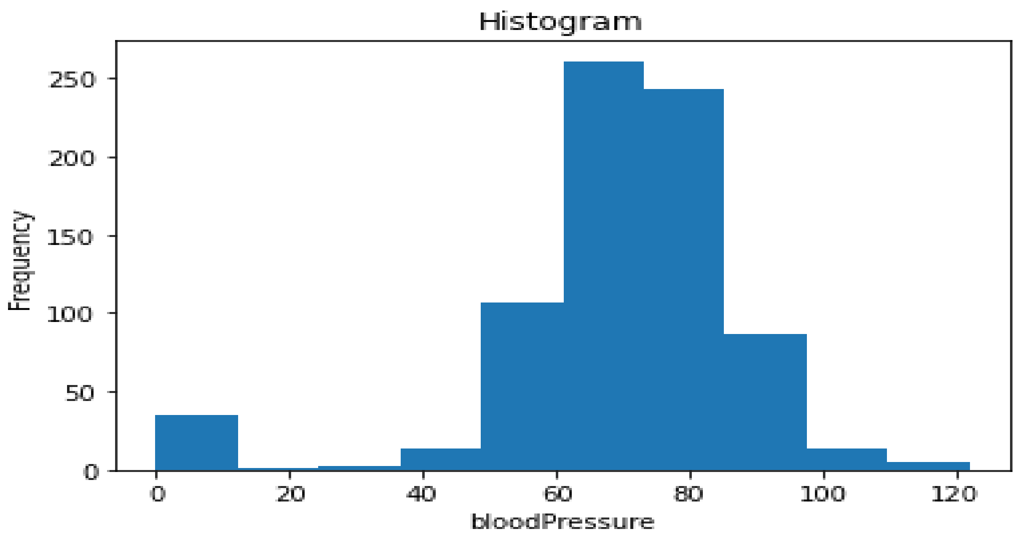


plt.hist(data['Glucose'])

```
plt.xlabel('Glucose')
plt.ylabel('Frequency')
plt.title('Histogram')
```

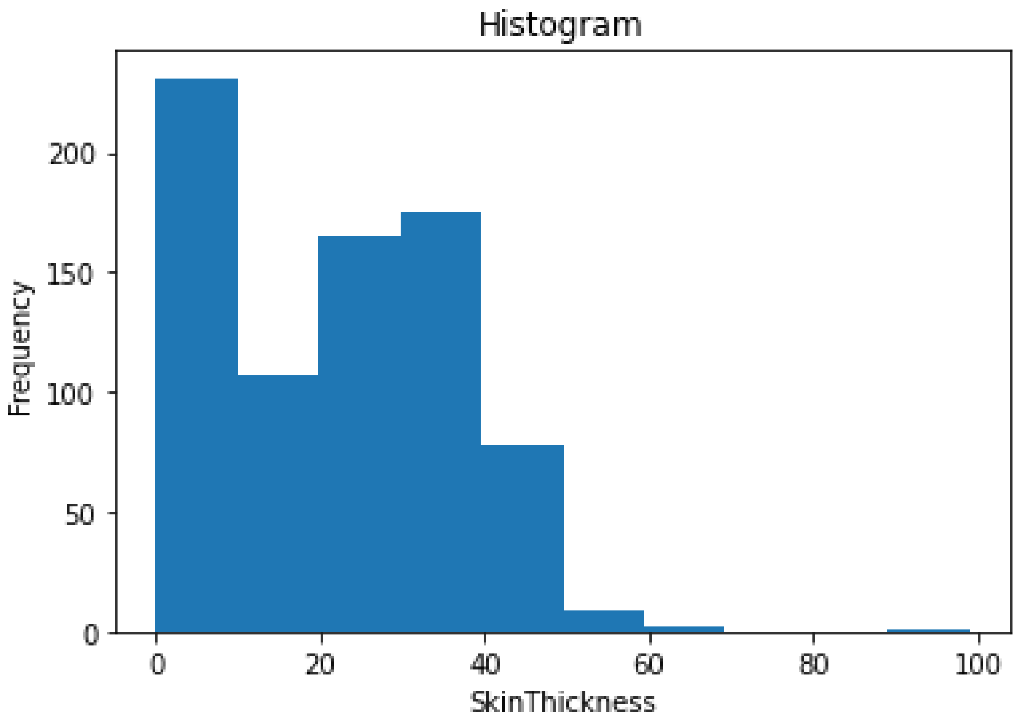



```
plt.hist(data['BloodPressure'])
plt.xlabel('bloodPressure')
plt.ylabel('Frequency')
plt.title('Histogram')
```

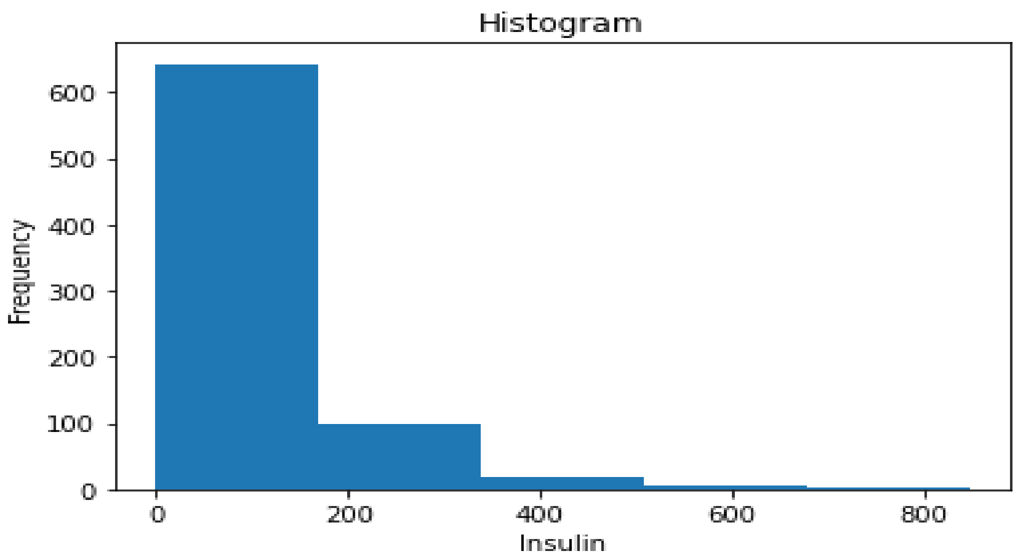



```
plt.hist(data['SkinThickness'])
plt.xlabel('SkinThickness')
```

plt.ylabel('Frequency')

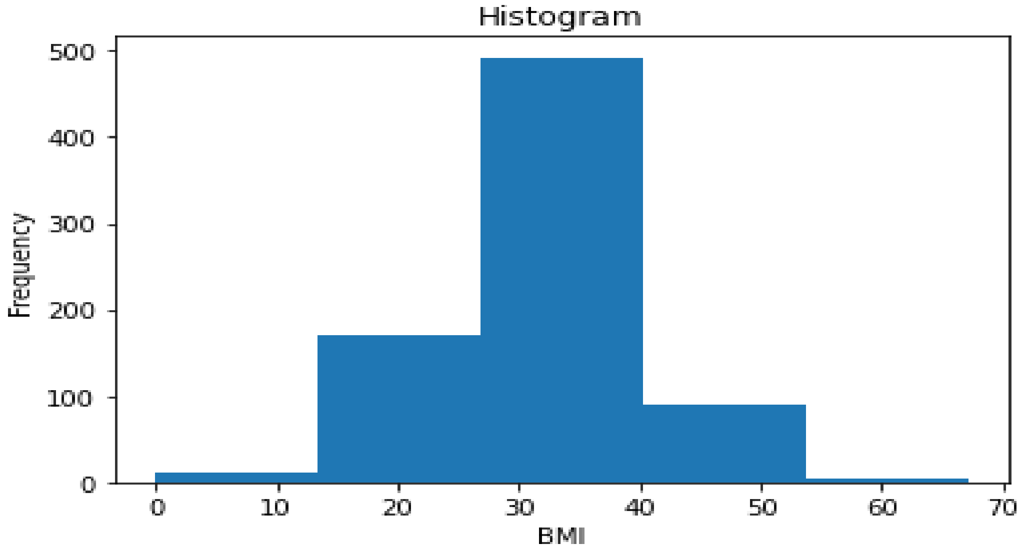


plt.hist(data['Insulin'], bins=5)
plt.xlabel('Insulin')
plt.ylabel('Frequency')
plt.title('Histogram')

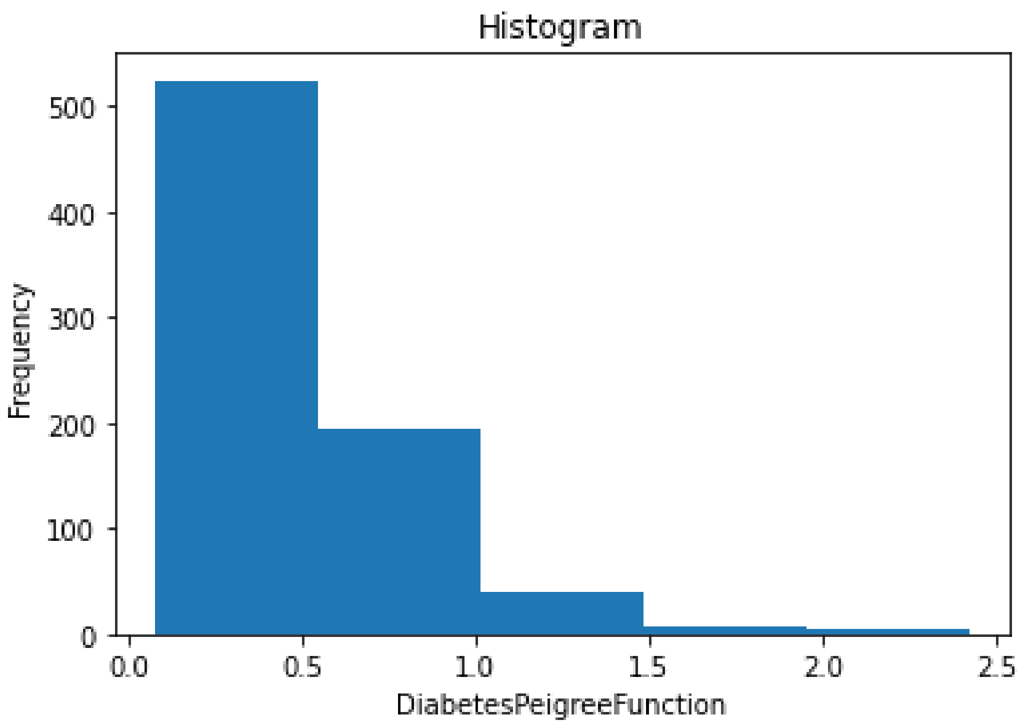


plt.hist(data['BMI'], bins=5)
plt.xlabel('BMI')
plt.ylabel('Frequency')

plt.title('Histogram')

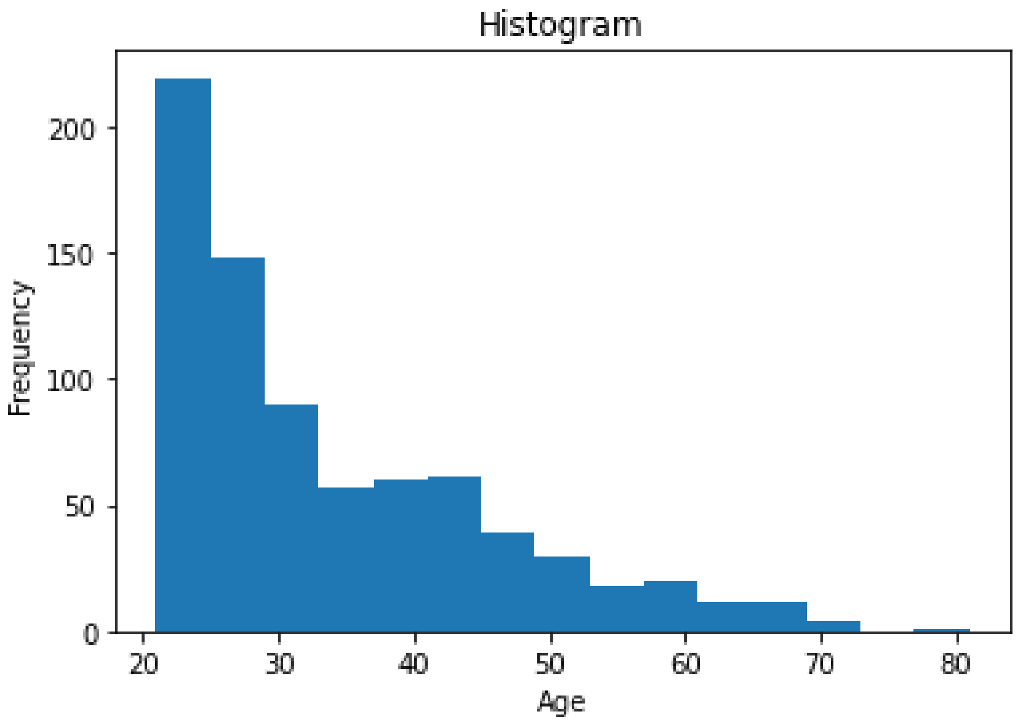


plt.hist(data['DiabetesPedigreeFunction'], bins=5)
plt.xlabel('DiabetesPeigreeFunction')
plt.ylabel('Frequency')
plt.title('Histogram')

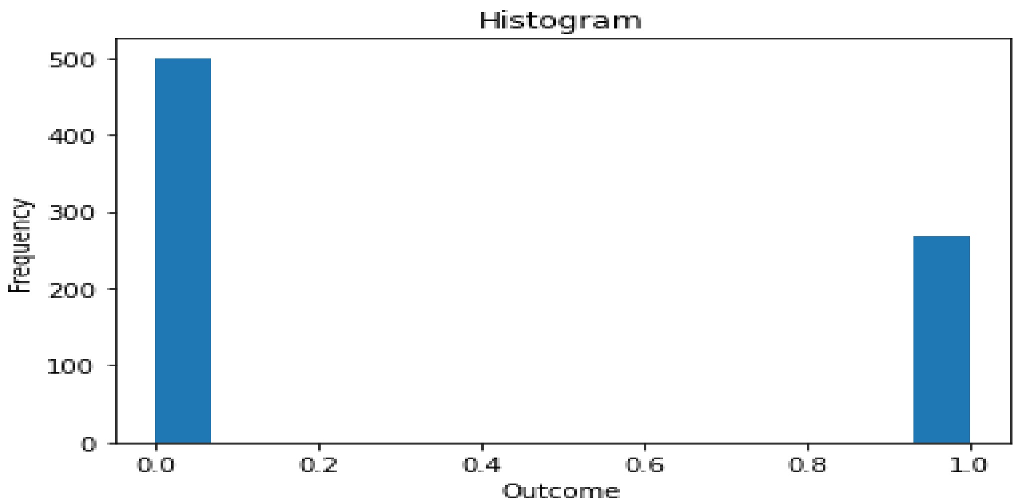


plt.hist(data['Age'], bins=15)
plt.xlabel('Age')
plt.ylabel('Frequency')

plt.title('Histogram')



plt.hist(data['Outcome'], bins=15)
plt.xlabel('Outcome')
plt.ylabel('Frequency')
plt.title('Histogram')



Practical 4

Implement binomial logistic regression in Python using multivariate flower dataset named 'iris'.

import numpy as np import
matplotlib.pyplot as plt import
pandas as pd
dataset = pd.read_csv('/content/IRIS.csv') dataset.describe()

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

X = dataset.iloc[:, [0,1,2, 3]].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
from sklearn.preprocessing import StandardScalersc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver='lbfgs', multi_class='auto')
classifier.fit(X_train, y_train)

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

y_pred = classifier.predict(X_test)

# Predict probabilities
probs_y=classifier.predict_proba(X_test)
Print results
probs_y = np.round(probs_y, 2)

res = "{:<10} | {:<10} | {:<10} | {:<13} | {:<5}".format("y_test", "y_pred", "Setosa(%)", "versicolor(%)", "virginica(%)\n") res
+= "-"*65+"\n"
res += "\n".join("{:<10} | {:<10} | {:<10} | {:<13} | {:<10}".format(x, y, a, b, c) for x, y, a, b, c in zip(y_test, y_pred, probs_y[:,0], probs_y[:,1], probs_y[:,2])) res += "\n"+"-"*65+"\n" print(res)

```
y_test     | y_pred     | Setosa(%)  | versicolor(%) | virginica(%)
-----------------------------------------------------------------
Iris-virginica | Iris-virginica | 0.0        | 0.03          | 0.97
Iris-versicolor | Iris-versicolor | 0.01       | 0.95          | 0.04
Iris-setosa | Iris-setosa | 1.0        | 0.0           | 0.0
Iris-virginica | Iris-virginica | 0.0        | 0.08          | 0.92
Iris-setosa | Iris-setosa | 0.98       | 0.02          | 0.0
Iris-virginica | Iris-virginica | 0.0        | 0.01          | 0.99
Iris-setosa | Iris-setosa | 0.98       | 0.02          | 0.0
Iris-versicolor | Iris-versicolor | 0.01       | 0.71          | 0.28
Iris-versicolor | Iris-versicolor | 0.0        | 0.73          | 0.27
Iris-versicolor | Iris-versicolor | 0.02       | 0.89          | 0.08
Iris-virginica | Iris-virginica | 0.0        | 0.44          | 0.56
Iris-versicolor | Iris-versicolor | 0.02       | 0.76          | 0.22
Iris-versicolor | Iris-versicolor | 0.01       | 0.85          | 0.13
Iris-versicolor | Iris-versicolor | 0.0        | 0.69          | 0.3
Iris-versicolor | Iris-versicolor | 0.01       | 0.75          | 0.24
Iris-setosa | Iris-setosa | 0.95       | 0.05          | 0.0
Iris-versicolor | Iris-versicolor | 0.02       | 0.72          | 0.26
Iris-versicolor | Iris-versicolor | 0.03       | 0.86          | 0.11
Iris-setosa | Iris-setosa | 0.94       | 0.06          | 0.0
Iris-setosa | Iris-setosa | 0.99       | 0.01          | 0.0
Iris-virginica | Iris-virginica | 0.0        | 0.17          | 0.83
Iris-versicolor | Iris-versicolor | 0.04       | 0.71          | 0.25
Iris-setosa | Iris-setosa | 0.98       | 0.02          | 0.0
Iris-setosa | Iris-setosa | 0.96       | 0.04          | 0.0
Iris-virginica | Iris-virginica | 0.0        | 0.35          | 0.65
Iris-setosa | Iris-setosa | 1.0        | 0.0           | 0.0
Iris-setosa | Iris-setosa | 0.99       | 0.01          | 0.0
Iris-versicolor | Iris-versicolor | 0.02       | 0.87          | 0.11
Iris-versicolor | Iris-versicolor | 0.09       | 0.9           | 0.02
Iris-setosa | Iris-setosa | 0.97       | 0.03          | 0.0
Iris-virginica | Iris-virginica | 0.0        | 0.21          | 0.79
Iris-versicolor | Iris-versicolor | 0.06       | 0.69          | 0.25
```

Practical 5

Implement Python script using Scikit learn library to build a Gaussian Naïve Bayes Model.

```
import pandas as pd import
numpy as np
from sklearn.impute import SimpleImputer from
sklearn import preprocessing from
sklearn.model_selection import train_test_split from
sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

adult_df = pd.read_csv('adult.data',header = None, delimiter=' *, *', engine='python')
adult_df.columns = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
                    'marital_status', 'occupation', 'relationship',
                    'race', 'sex', 'capital_gain', 'capital_loss',
'hours_per_week', 'native_country', 'income']
for value in ['workclass', 'education',
         'marital_status', 'occupation',
'relationship','race', 'sex',           'native_country',
'income']:
print( value,":", sum(adult_df[value] == '?'))
```

```
workclass : 1836
education : 0
marital_status : 0
occupation : 1843
relationship : 0
race : 0
sex : 0
native_country : 583
income : 0
```

```
for value
in['workclass','education','marital_sta
tus', 'occupation', 'relationship','race',
'sex','native_country', 'income']:
adult_df_rev[value].replace(['?'], [adult_df_rev.describe(include='all')[value][2]],inplac
e=True)

le = preprocessing.LabelEncoder() workclass_cat =
le.fit_transform(adult_df.workclass) education_cat =
le.fit_transform(adult_df.education) marital_cat   =
le.fit_transform(adult_df.marital_status)
occupation_cat=le.fit_transform(adult_df.occupation
relationship_cat=le.fit_transform(adult_df.relatioip)
race_cat = le.fit_transform(adult_df.race)
```

```
sex_cat=le.fit_transform(adult_df.sex)
native_country_cat = le.fit_transform(adult_df.native_country)

#initialize the encoded categorical columns
adult_df_rev['workclass_cat'] =
workclass_catadult_df_rev['education_cat'] =
education_catadult_df_rev['marital_cat'] =
marital_catadult_df_rev['occupation_cat'] =
occupation_catadult_df_rev['relationship_cat'] =
relationship_catadult_df_rev['race_cat'] = race_cat
adult_df_rev['sex_cat'] = sex_cat
adult_df_rev['native_country_cat'] = native_country_cat

#drop the old categorical columns
from dataframedummy_fields = ['workclass', 'education',
'marital_status',  'occupation', 'relationship', 'race', 'sex',
'native_country']
adult_df_rev = adult_df_rev.drop(dummy_fields, axis = 1)
adult_df_rev = adult_df_rev.reindex(['age', 'workclass_cat', 'fnlwgt', 'education_cat',
                                'education_num', 'marital_cat', 'occupation_cat',
                                'relationship_cat', 'race_cat', 'sex_cat', 'capital_gain',
                                'capital_loss', 'hours_per_week', 'native_country_cat',
  'income'], axis= 1)

adult_df_rev.head(1)
num_features = ['age', 'workclass_cat', 'fnlwgt', 'education_cat', 'education_num',
              'marital_cat', 'occupation_cat', 'relationship_cat', 'race_cat',
              'sex_cat', 'capital_gain', 'capital_loss', 'hours_per_week',
              'native_country_cat']

scaled_features = {}
scaler = preprocessing.StandardScaler()
for each in num_features:
adult_df_rev[[each]] = scaler.fit_transform(adult_df_rev[[each]].values)
features = adult_df_rev.values[:,:14] target = adult_df_rev.values[:,14]

features_train, features_test, target_train, target_test = train_test_split(features,target, test
_size = 0.33, random_state = 10)
clf = GaussianNB()
clf.fit(features_train, target_train)
```

```
target_pred = clf.predict(features_test)
accuracy_score(target_test, target_pred, normalize = True)
```

```
0.8014144798064397
```

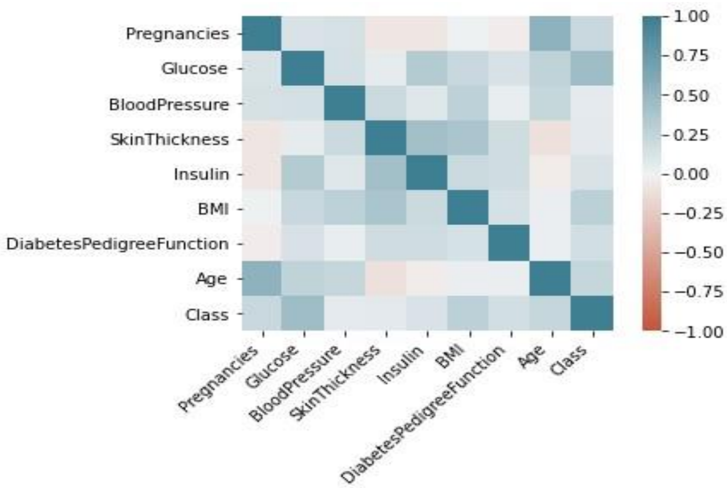
Practical 6

Implement Python script on Prima Indian Diabetes dataset using Decision tree classifier.

import pandas as pd from sklearn.tree import
DecisionTreeClassifier from
sklearn.model_selection import train_test_split from
sklearn import metrics
pima= pd.read_csv("/content/pima-indians-diabetes.csv") pima.head()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

import seaborn as snscorr = pima.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
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose', 'BloodPressure', 'DiabetesPedigreeFunction'] x = pima[feature_cols]
y = pima.Class

```
# split data
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3, random_state=1)
classifier = DecisionTreeClassifier() classifier = classifier.fit(X_train, Y_train)
# predict
y_pred = classifier.predict(X_test) print(y_pred)
```

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

```
# confusion matrix
from sklearn.metrics import
confusion_matrixconfusion_matrix(Y_test,y_
pred) print(confusion_matrix(Y_test, y_pred))

# accuracy
print("Accuracy:", metrics.accuracy_score(Y_test,y_pred))
```

```
[[113  33]
 [ 44  41]]
Accuracy: 0.6666666666666666
```

Practical 7

Implement Simple Linear Regression using your own set of data in Python.

import numpy as np import
matplotlib.pyplot as plt import
pandas as pd
sd=pd.read_csv('/content/Salary_Data.csv') sd.head()

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

x=sd.iloc[:,:-1].values
y=sd.iloc[:,-1].values

```
print(regressor.intercept_)
```

```
26777.391341197625
```

```
print(regressor.coef_)
```

```
[9360.26128619]
```

```
x_pred=regressor.predict(X_test) x_pred
```

```
array([ 40817.78327049, 123188.08258899,  65154.46261459,  63282.41035735,
       115699.87356004, 108211.66453108, 116635.89968866,  64218.43648597,
        76386.77615802])
```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 0)

from sklearn.linear_model import LinearRegression
regressor=LinearRegression() regressor.fit(X_train,y_train)
plt.scatter(X_train,y_train,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title('Salaray v/s Experience') plt.xlabel('Experience
(in Years)')
plt.ylabel('Salary') plt.show()


Salaray v/s Experience
Salary
120000
100000
80000
60000
40000
2
4
6
8
10
Experience (in Years)

Practical: 8

Implement K-mean clustering on simple digit dataset using Python.

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

data = pd.read_csv('IRIS.csv')
data
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

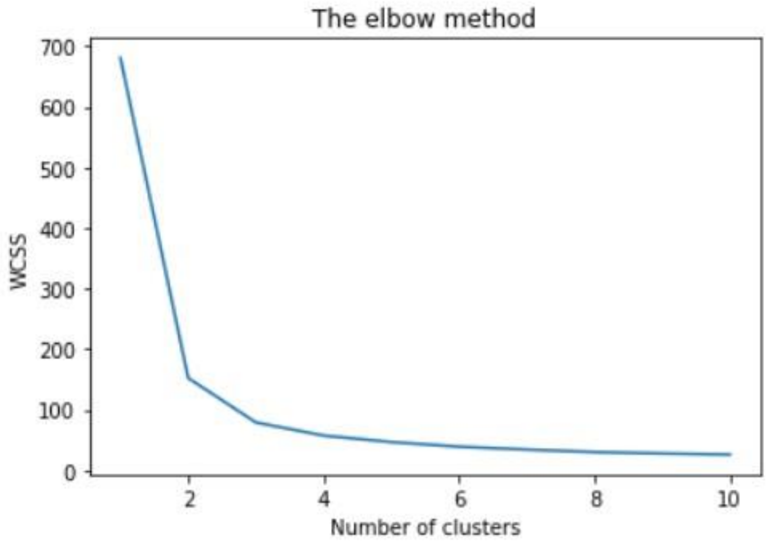
```
x = data.iloc[:, 0:4].values

from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
kmeans.fit(x)
wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
```

```
plt.ylabel('WCSS')
plt.show()
```

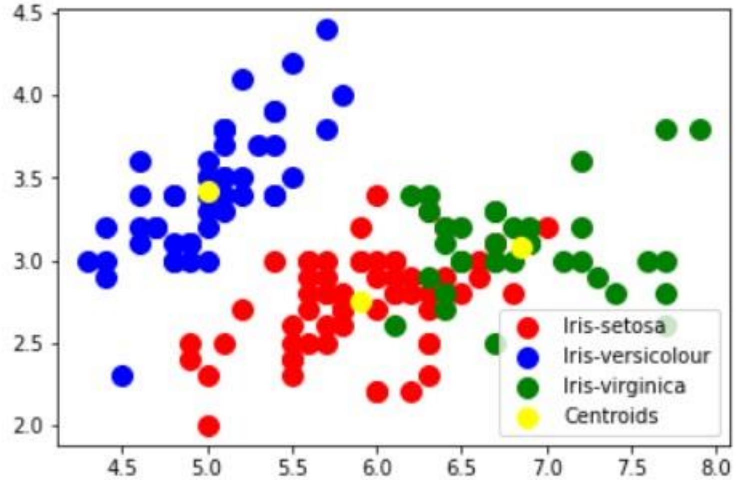



```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)

plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
```



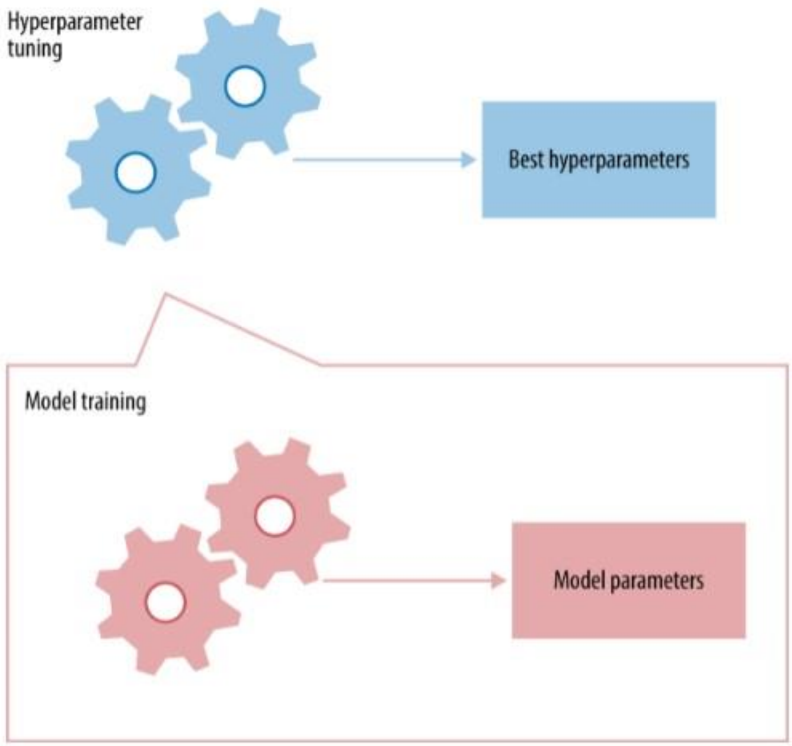
Practical: 9

Aim: Study Hyperparameters tuning and deep learning frameworks (TensorFlow and Keras).

Hyperparameter Tuning Mechanism

Hyperparameter settings could have a big impact on the prediction accuracy of the trained model. Optimal hyperparameter settings often differ for different datasets. Therefore they should be tuned for each dataset. Since the training process doesn't set the hyperparameters, there needs to be a meta process that tunes the hyperparameters. This is what we mean by hyperparameter tuning.

Hyperparameter tuning is a meta-optimization task. Each trial of a particular hyperparameter setting involves training a model—an inner optimization process. The outcome of hyperparameter tuning is the best hyperparameter setting, and the outcome of model training is the best model parameter setting.



Hyperparameter Tuning Algorithms

Conceptually, hyperparameter tuning is an optimization task, just like model training.

However, these two tasks are quite different in practice. When training a model, the quality of a proposed set of model parameters can be written as a mathematical formula (usually called the loss function). When tuning hyperparameters, however, the quality of those hyperparameters cannot be written down in a closed-form formula, because it depends on the outcome of a black box (the model training process).

This is why hyperparameter tuning is much harder. Up until a few years ago, the only available methods were grid search and random search. In the last few years, there's been increased interest in auto-tuning. Several research groups have worked on the problem, published papers, and released new tools.

Grid Search

Grid search, true to its name, picks out a grid of hyperparameter values, evaluates every one of them, and returns the winner. For example, if the hyperparameter is the number of leaves in a decision tree, then the grid could be 10, 20, 30, …, 100. For regularization parameters, it's common to use exponential scale: 1e-5, 1e-4, 1e-3, …, 1. Some guesswork is necessary to specify the minimum and maximum values. So sometimes people run a small grid, see if the optimum lies at either endpoint, and then expand the grid in that direction. This is called manual grid search.

Grid search is dead simple to set up and trivial to parallelize. It is the most expensive method in terms of total computation time. However, if run in parallel, it is fast in terms of wall clock time.

Random Search

Random search is a slight variation on grid search. Instead of searching over the entire grid, random search only evaluates a random sample of points on the grid. This makes random search a lot cheaper than grid search. Random search wasn't taken very seriously before. This is because it doesn't search over all the grid points, so it cannot possibly beat the optimum found by grid search. But then along came Bergstra and Bengio. They showed that, in surprisingly many instances, random search performs about as well as grid search. All in all, trying 60 random points sampled from the grid seems to be good enough.

In hindsight, there is a simple probabilistic explanation for the result: for any distribution over a sample space with a finite maximum, the maximum of 60 random observations lies within the top 5% of the true maximum, with 95% probability. That may sound complicated, but it's not. Imagine the 5% interval around the true maximum. Now imagine that we sample points from this space and see if any of them land within that maximum. Each random draw has a 5% chance of landing in that interval; if we draw n points independently, then the probability that all of them miss the desired interval is $(1 - 0.05)^n$.So the probability that at least one of them succeeds in hitting the interval is 1 minus that quantity. We want at least a 0.95 probability of success. To figure out the number of draws we need, just solve for n in the following equation:

$1 - (1 - 0.05)^n > 0.95$

We get n >= 60.

The moral of the story is: if at least 5% of the points on the grid yield a close-to-optimal solution, then random search with 60 trials will find that region with high probability. The condition of the if-statement is very important. It can be satisfied if either the close-to-optimal region is large, or if somehow there is a high concentration of grid points in that region. The former is more likely, because a good machine learning model should not be overly sensitive to the hyperparameters, i.e., the close-to-optimal region is large.

With its utter simplicity and surprisingly reasonable performance, random search is my go-to method for hyperparameter tuning. It's trivially parallelizable, just like grid search, but it takes much fewer tries and performs almost as well most of the time.

TensorFlow

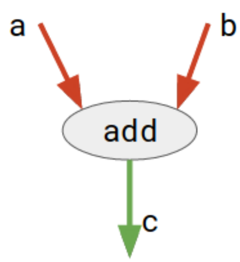
TensorFlow is an open-source software library. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well!

Let us first try to understand what the word TensorFlow actually mean!

TensorFlow is basically a software library for numerical computation using data flow graphs where:

- nodes in the graph represent mathematical operations.
- edges in the graph represent the multidimensional data arrays (called tensors) communicated between them. (Please note that tensor is the central unit of data in TensorFlow).

Consider the diagram given below:



Here, add is a node which represents addition operation. a and b are input tensors and c is the resultant tensor.

This flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API!

TensorFlow APIs

TensorFlow provides multiple APIs (Application Programming Interfaces). These can be classified into 2 major categories:

Low level API:

- complete programming control
- recommended for machine learning researchers
- provides fine levels of control over the models
- TensorFlow Core is the low level API of TensorFlow.

High level API:

- built on top of TensorFlow Core
- easier to learn and use than TensorFlow Core
- make repetitive tasks easier and more consistent between different users

- tf.contrib.learn is an example of a high level API.

Keras

Keras is a deep learning framework for Python that provides a convenient way to define and train almost any kind of deep learning model. Keras is a high-level neural networks API, written in Python which is capable of running on top of Tensorflow, Theano and CNTK. It was developed for enabling fast experimentation.

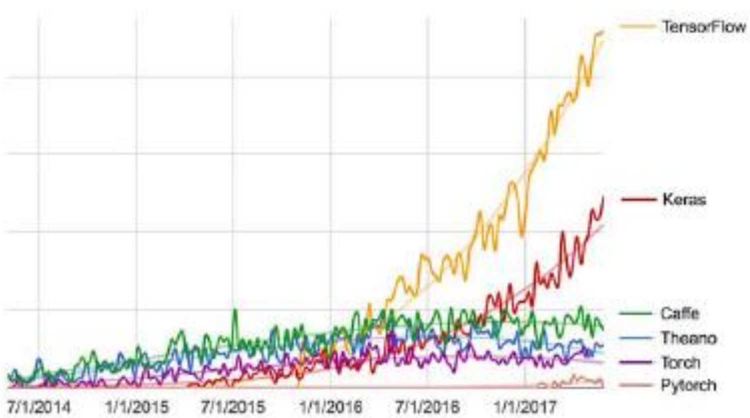
Being able to go from idea to result with the least possible delay is key to doing good research.

Keras has the following features :

- Allows for easy and fast prototyping
- Run seamlessly on CPU and GPU
- Supports both convolutional networks(for computer vision) and recurrent networks(for sequence and time-series), as well as the combination of two.
- It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing and so on. This means Keras is appropriate for building deep learning models, from generative adversarial networks to a neural Turing machine.

Keras is compatible with versions of Python from 2.7 to 3.6 till date.

Keras is used by around 200,000 users, ranging from academic researchers and engineers at both startups and large companies to graduate students and hobbyist. Keras is used at Google, Netflix, Uber, Microsoft, Square and many startups working on the wide variety of machine learning problems.

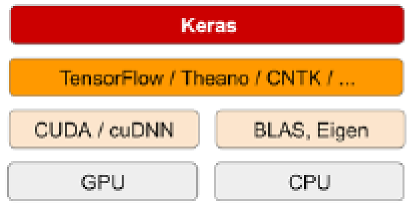


Keras recommend users to switch to tf.keras in Tensorflow 2.0, who use multi-backend keras with the tensorflow backend.

Guiding principles

- User Friendliness
- Modularity
- Easy Extensibility
- Work with Python

Keras doesn’t handle low-level operations such as tensor manipulations and differentiation. Instead, it relies on a specialized, well-optimized tensor library to do so which serves as the backend engine of Keras. We can use several backend engine for keras, and currently three existing backend implementations are the Tensorflow backend, the Theano backend, and the Microsoft Cognitive Toolkit (CNTK) backend.



Practical 10

Aim: Implement Convolution Neural network on FER-2013 dataset of Kaggle using Python.

```
importnumpyasnp
importpandasaspd

fromkeras.utilsimportto_categorical
fromkeras.callbacksimportEarlyStopping
fromkeras.modelsimportSequential
fromkeras.layersimportDense,Dropout,Activation,Flatten
fromkeras.layersimportConv2D,MaxPooling2D,BatchNormalization
fromkeras.lossesimportcategorical_crossentropy
fromsklearn.metricsimportaccuracy_score
fromkeras.optimizersimportAdam
fromkeras.regularizersimportl2
fromkeras.preprocessing.imageimportImageDataGenerator
fromsklearn.metricsimportclassification_report,confusion_matrix
importmatplotlib.pyplotasplt
importseabornassns

importos

data=pd.read_csv('../input/fer2013.csv')
data.shape
```

```
(35887, 3)
```

```
data.head(5)
```

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

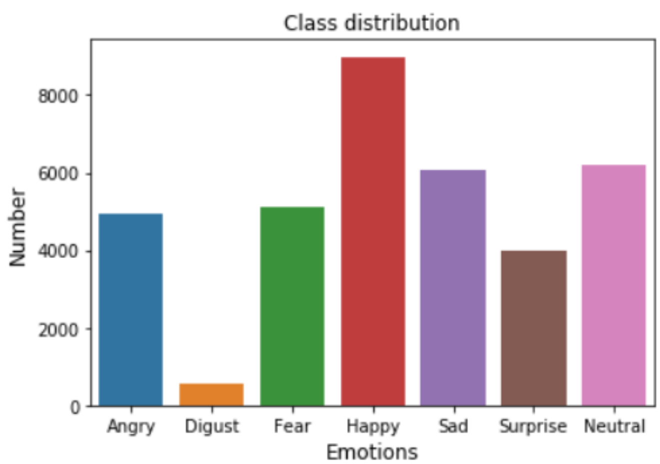
```
data.Usage.value_counts()
```

```
Training       28709
PublicTest      3589
PrivateTest     3589
Name: Usage, dtype: int64
```

```
emotion_map={0:'Angry',1:'Digust',2:'Fear',3:'Happy',4:'Sad',5:'Surprise',6:'Neutral'}
emotion_counts=data['emotion'].value_counts(sort=False).reset_index()
emotion_counts.columns=['emotion','number']
emotion_counts['emotion']=emotion_counts['emotion'].map(emotion_map)
emotion_counts
```

	emotion	number
0	Angry	4953
1	Digust	547
2	Fear	5121
3	Happy	8989
4	Sad	6077
5	Surprise	4002
6	Neutral	6198

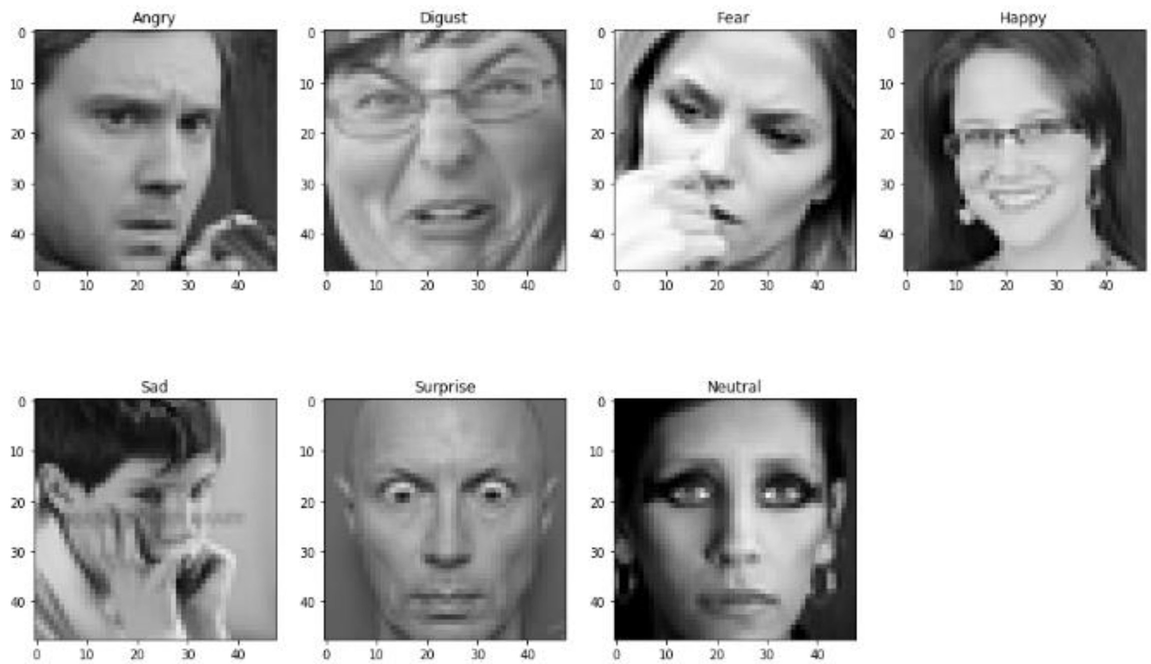
```
plt.figure(figsize=(6,4))
sns.barplot(emotion_counts.emotion,emotion_counts.number)
plt.title('Class distribution')
plt.ylabel('Number',fontsize=12)
plt.xlabel('Emotions',fontsize=12)
plt.show()
```



```
defrow2image(row):
pixels,emotion=row['pixels'],emotion_map[row['emotion']]
img=np.array(pixels.split())
img=img.reshape(48,48)
image=np.zeros((48,48,3))
image[:,:,0]=img
image[:,:,1]=img
image[:,:,2]=img
returnnp.array([image.astype(np.uint8),emotion])

plt.figure(0,figsize=(16,10))
foriinrange(1,8):
face=data[data['emotion']==i-1].iloc[0]
img=row2image(face)
plt.subplot(2,4,i)
plt.imshow(img[0])
plt.title(img[1])

plt.show()
```



Pre-processing data

```
data_train=data[data['Usage']=='Training'].copy()
data_val=data[data['Usage']=='PublicTest'].copy()
data_test=data[data['Usage']=='PrivateTest'].copy()
print("train shape: {}, \nvalidation shape: {}, \ntest shape: {}".format(data_train.shape,data_val.shape,data_test.shape))
```

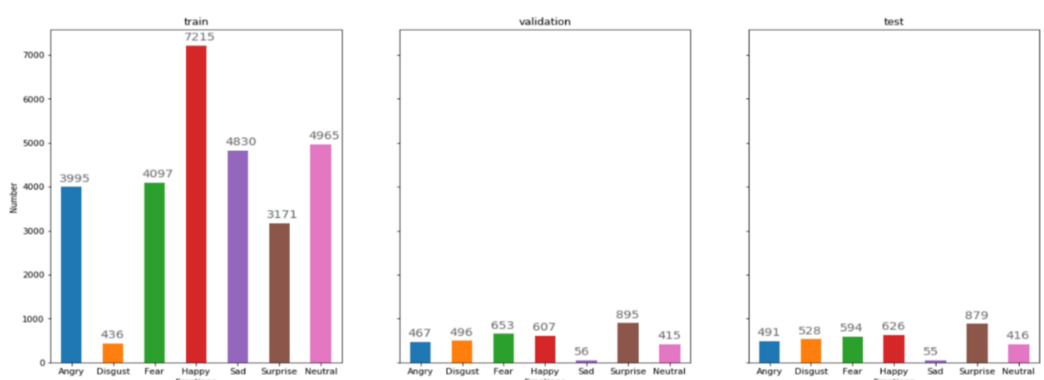
```
train shape: (28709, 3),
validation shape: (3589, 3),
test shape: (3589, 3)
```

```
emotion_labels=['Angry','Disgust','Fear','Happy','Sad','Surprise','Neutral']

defsetup_axe(axe,df,title):
df['emotion'].value_counts(sort=False).plot(ax=axe,kind='bar',rot=0)
axe.set_xticklabels(emotion_labels)
axe.set_xlabel("Emotions")
axe.set_ylabel("Number")
axe.set_title(title)

# set individual bar lables using above list
foriinaxe.patches:
# get_x pulls left or right; get_height pushes up or down
axe.text(i.get_x()-.05,i.get_height()+120, \
str(round((i.get_height()),2)),fontsize=14,color='dimgrey',
rotation=0)


fig,axes=plt.subplots(1,3,figsize=(20,8),sharey=True)
setup_axe(axes[0],data_train,'train')
setup_axe(axes[1],data_val,'validation')
setup_axe(axes[2],data_test,'test')
plt.show()
```



Building CNN Model

```
model=Sequential()
model.add(Conv2D(2*2*num_features,kernel_size=(3,3),input_shape=(width,height,1),data_for
mat='channels_last'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(2*2*num_features,kernel_size=(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(2*num_features,kernel_size=(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(2*num_features,kernel_size=(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(num_features,kernel_size=(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(num_features,kernel_size=(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(2*2*2*num_features))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(2*2*num_features))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(2*num_features))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(num_classes,activation='softmax'))

model.compile(loss='categorical_crossentropy',
optimizer=Adam(lr=0.001,beta_1=0.9,beta_2=0.999,epsilon=1e-7),
metrics=['accuracy'])
```

model.summary()

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 46, 46, 256)       2560
_________________________________________________________________
batch_normalization_1 (Batch (None, 46, 46, 256)       1024
_________________________________________________________________
activation_1 (Activation)    (None, 46, 46, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 256)       590080
_________________________________________________________________
batch_normalization_2 (Batch (None, 46, 46, 256)       1024
_________________________________________________________________
activation_2 (Activation)    (None, 46, 46, 256)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 256)       0
_________________________________________________________________
…………………
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
batch_normalization_6 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
activation_6 (Activation)    (None, 11, 11, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               819712
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
activation_7 (Activation)    (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_8 (Batch (None, 256)               1024
_________________________________________________________________
activation_8 (Activation)    (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_9 (Batch (None, 128)               512
_________________________________________________________________
activation_9 (Activation)    (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 7)                 903
=================================================================
Total params: 2,137,991
Trainable params: 2,134,407
Non-trainable params: 3,584
```

```
data_generator=ImageDataGenerator(
featurewise_center=False,
featurewise_std_normalization=False,
rotation_range=10,
width_shift_range=0.1,
height_shift_range=0.1,
zoom_range=.1,
horizontal_flip=True)
es=EarlyStopping(monitor='val_loss',patience=10,mode='min',restore_best_weights=True)
history=model.fit_generator(data_generator.flow(train_X,train_Y,batch_size),
steps_per_epoch=len(train_X)/batch_size,
epochs=num_epochs,
verbose=2,
callbacks=[es],validation_data=(val_X,val_Y))
```

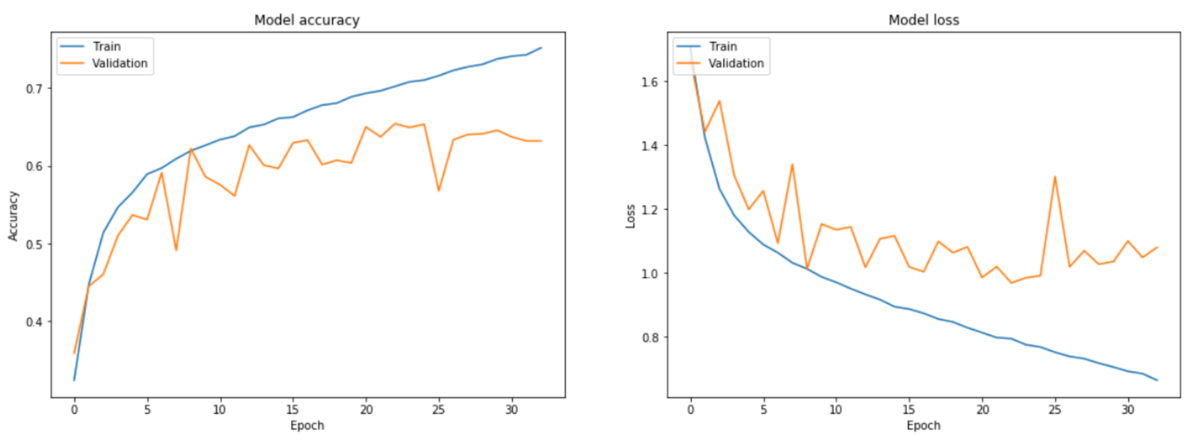
```
Epoch 1/50
 - 37s - loss: 1.7037 - acc: 0.3242 - val_loss: 1.6681 - val_acc: 0.3589
Epoch 2/50
 - 30s - loss: 1.4228 - acc: 0.4470 - val_loss: 1.4414 - val_acc: 0.4450
Epoch 3/50
 - 30s - loss: 1.2625 - acc: 0.5140 - val_loss: 1.5380 - val_acc: 0.4606
Epoch 4/50
 - 30s - loss: 1.1799 - acc: 0.5468 - val_loss: 1.3059 - val_acc: 0.5102
Epoch 5/50
...........
...........
...........
Epoch 30/50
 - 30s - loss: 0.7060 - acc: 0.7374 - val_loss: 1.0358 - val_acc: 0.6456
Epoch 31/50
 - 30s - loss: 0.6927 - acc: 0.7408 - val_loss: 1.0999 - val_acc: 0.6372
Epoch 32/50
 - 30s - loss: 0.6853 - acc: 0.7427 - val_loss: 1.0485 - val_acc: 0.6319
Epoch 33/50
 - 30s - loss: 0.6645 - acc: 0.7518 - val_loss: 1.0801 - val_acc: 0.6319
```

Vi

Visualize Training Performance:

```
fig,axes=plt.subplots(1,2,figsize=(18,6))
# Plot training & validation accuracy values
axes[0].plot(history.history['acc'])
axes[0].plot(history.history['val_acc'])
axes[0].set_title('Model accuracy')
axes[0].set_ylabel('Accuracy')
axes[0].set_xlabel('Epoch')
axes[0].legend(['Train','Validation'],loc='upper left')

# Plot training & validation loss values
axes[1].plot(history.history['loss'])
axes[1].plot(history.history['val_loss'])
axes[1].set_title('Model loss')
axes[1].set_ylabel('Loss')
axes[1].set_xlabel('Epoch')
axes[1].legend(['Train','Validation'],loc='upper left')
plt.show()
```



Evaluate Test Performance

```
test_true=np.argmax(test_Y,axis=1)
test_pred=np.argmax(model.predict(test_X),axis=1)
print("CNN Model Accuracy on test set: {:.4f}".format(accuracy_score(test_true,test_pred)))
```

```
CNN Model Accuracy on test set: 0.6662
```