# Machine Learning Assignment

1) Write program to implement simple liner regression for predicting house price

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

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
```

```
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

## 2) Write program to implement Multiple linear regression for predicting house price

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics

# load the boston dataset
boston = datasets.load_boston(return_X_y=False)

# defining feature matrix(X) and response vector(y)
X = boston.data
y = boston.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=1)

# create linear regression object
reg = linear_model.LinearRegression()
```

```
# train the model using the training sets
reg.fit(X_train, y_train)

# regression coefficients
print('Coefficients: ', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))

# plot for residual error

## setting plot style
plt.style.use('fivethirtyeight')

## plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
            color = "green", s = 10, label = 'Train data')

## plotting residual errors in test data
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,
            color = "blue", s = 10, label = 'Test data')

## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

## plotting legend
plt.legend(loc = 'upper right')

## plot title
plt.title("Residual errors")

## method call for showing the plot
plt.show()
```

## 3) Write a Program to implement Decision tree whether or not to play tennis

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

1. **sklearn :**
    - In python, sklearn is a machine learning package which include a lot of ML algorithms.

    - Here, we are using some of its modules like train_test_split, DecisionTreeClassifier and accuracy_score.
2. **NumPy :**
    - It is a numeric python module which provides fast maths functions for calculations.
    - It is used to read data in numpy arrays and for manipulation purpose.
3. **Pandas :**
    - Used to read and write different files.
    - Data manipulation can be done easily with dataframes.

**Assumptions we make while using Decision tree :**

- At the beginning, we consider the whole training set as the root.
- Attributes are assumed to be categorical for information gain and for gini index, attributes are assumed to be continuous.
- On the basis of attribute values records are distributed recursively.
- We use statistical methods for ordering attributes as root or internal node.

**Pseudocode :**

1. Find the best attribute and place it on the root node of the tree.
2. Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.
3. Find leaf nodes in all branches by repeating 1 and 2 on each subset.

```
import numpy as np
import pandas as pd
```

[4]
```
#Loading the PlayTennis data
PlayTennis = pd.read_csv("playtennis.csv")
```

[5]
```
PlayTennis
```

| | outlook | temp | humidity | windy | play |
|---|---|---|---|---|---|
| **0** | sunny | hot | high | False | no |
| **1** | sunny | hot | high | True | no |
| **2** | overcast | hot | high | False | yes |
| **3** | rainy | mild | high | False | yes |
| **4** | rainy | cool | normal | False | yes |
| **5** | rainy | cool | normal | True | no |
| **6** | overcast | cool | normal | True | yes |
| **7** | sunny | mild | high | False | no |
| **8** | sunny | cool | normal | False | yes |
| **9** | rainy | mild | normal | False | yes |

```
[7]  from sklearn.preprocessing import LabelEncoder
     Le = LabelEncoder()

     PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])
     PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
     PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
     PlayTennis['windy'] = Le.fit_transform(PlayTennis['windy'])
     PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])
```

```
[8]  PlayTennis
```

| | outlook | temp | humidity | windy | play |
|---|---|---|---|---|---|
| **0** | 2 | 1 | 0 | 0 | 0 |
| **1** | 2 | 1 | 0 | 1 | 0 |
| **2** | 0 | 1 | 0 | 0 | 1 |
| **3** | 1 | 2 | 0 | 0 | 1 |
| **4** | 1 | 0 | 1 | 0 | 1 |
| **5** | 1 | 0 | 1 | 1 | 0 |
| **6** | 0 | 0 | 1 | 1 | 1 |
| **7** | 2 | 2 | 0 | 0 | 0 |

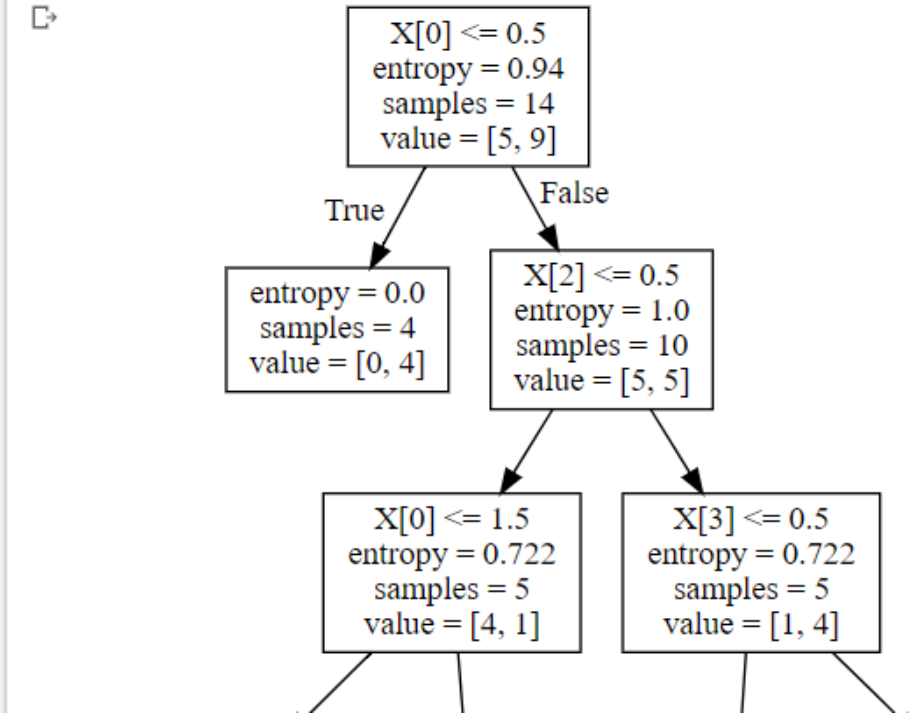
```
[9]  y = PlayTennis['play']
     X = PlayTennis.drop(['play'],axis=1)
```

```
[11] from sklearn import tree
     clf = tree.DecisionTreeClassifier(criterion = 'entropy')
     clf = clf.fit(X, y)
```

```
[12] # We can visualize the tree using tree.plot_tree
     tree.plot_tree(clf)
```

```
[Text(133.92000000000002, 195.696, 'X[0] <= 0.5\nentropy = 0.94\nsamples = 14\nvalue = [5, 9]'),
 Text(100.44000000000001, 152.208, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(167.40000000000003, 152.208, 'X[2] <= 0.5\nentropy = 1.0\nsamples = 10\nvalue = [5, 5]'),
 Text(100.44000000000001, 108.72, 'X[0] <= 1.5\nentropy = 0.722\nsamples = 5\nvalue = [4, 1]'),
 Text(66.96000000000001, 65.232, 'X[3] <= 0.5\nentropy = 1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(33.480000000000004, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(100.44000000000001, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(133.92000000000002, 65.232, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(234.36, 108.72, 'X[3] <= 0.5\nentropy = 0.722\nsamples = 5\nvalue = [1, 4]'),
```

```
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph
```




## playtennis.csv

| outlook | temp | humidity | windy | play |
|---|---|---|---|---|
| sunny | hot | high | FALSE | no |
| sunny | hot | high | TRUE | no |
| overcast | hot | high | FALSE | yes |
| rainy | mild | high | FALSE | yes |
| rainy | cool | normal | FALSE | yes |
| rainy | cool | normal | TRUE | no |
| overcast | cool | normal | TRUE | yes |
| sunny | mild | high | FALSE | no |
| sunny | cool | normal | FALSE | yes |
| rainy | mild | normal | FALSE | yes |
| sunny | mild | normal | TRUE | yes |
| overcast | mild | high | TRUE | yes |
| overcast | hot | normal | FALSE | yes |
| rainy | mild | high | TRUE | no |

4) In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values.

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list
df = pd.DataFrame(dict)

# using isnull() function
df.isnull()
```

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series True for NaN values
bool_series = pd.isnull(data["Gender"])

# filtering data
# displaying data only with Gender = NaN
data[bool_series]
```

In order to check null values in Pandas Dataframe, we use notnull() function this function return dataframe of Boolean values which are False for NaN values.

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np
```

```
# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe using dictionary
df = pd.DataFrame(dict)

# using notnull() function
df.notnull()
```

```
    # importing pandas package
    import pandas as pd

    # making data frame from csv file
    data = pd.read_csv("employees.csv")

    # creating bool series True for NaN values
    bool_series = pd.isnull(data["Team"])

    # filtering data
    # displayind data only with team = NaN
    data[bool_series]
```

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series True for NaN values
bool_series = pd.notnull(data["Gender"])

# filtering data
# displayind data only with Gender = Not NaN
data[bool_series]
```

Dropping Rows with at least 1 null value in CSV file

```
# importing pandas module
```

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# making new data frame with dropped NA values
new_data = data.dropna(axis = 0, how ='any')

new_data
```

**NOTE: ABOVE Assignment check employee.csv file**

## 5) Write python program to find all null values in given data set and remove them

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.DataFrame({"A":[12, 4, 5, None, 1],
                   "B":[None, 2, 54, 3, None],
                   "C":[20, 16, None, 3, 8],
                   "D":[14, 3, None, None, 6]})

# Print the dataframe
df
df = pd.DataFrame(dict)
```

**To remove null values used following function**

```
# using dropna() function
df.dropna()
```

## 6) Write python program for k-nearest neighbor algorithm

```
# Import necessary modules
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
             X, y, test_size = 0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
print(knn.predict(X_test))
```