INDEX

NAME: Shikha Singh STD.: VI SEC.: D ROLL NO.: 202

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	21/3/24	week 1	1	
2	28/3/24	week 2	2-4	
3	4/4/24	week 3	5-8	
4	18/4/24	week 4	9-12	
5	25/4/24	week 5	13-14	
6	9/5/24	week 6	15-16	
7	9/5/24	week 7	16-17	
8	23/5/24	week 8	17-20	
9	30/5/24	week 9	20-24	

Bafna Gold
Date: Page: 1

Week-1

PANDAS (Reading CSV)

```
import pandas as pd
df2 = pd.read_csv("/content/sample_data/california_housing_test.csv")
df2.head()
```

Output:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-122.05	37.37	27.0	3885.0	661.0

population	households	median_income	median_house_value
1537.0	606.0	6.6085	344700.0

```
df2.to_csv('new.csv')
```

```
col_names = ['sepal_length_in_cm', 'sepal_width_in_cm',
             'petal_length_in_cm', 'petal_width_in_cm', 'class']

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url, names = col_names)
df.head()
```

OUTPUT:

sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm
5.1	3.5	1.4

petal_width_in_cm	class
0.2	Iris-setosa

Week-2

1. Performance measure
2. Get the data
2.1 Download the data

→ Step 1: Use the data

```
import os
import tarfile
import urllib
```

```
Download_Root = "https://raw.githubusercontent.com/ageron/
                 handson-ml/"
Housing_PATH = os.path.join("datasets", "housing")
HOUSING_URL = Download_ROOT + "datasets/housing/
                housing.tgz"
def fetch_housing_data(housing_url = HOUSING_URL,
                housing_path = HOUSING_PATH):
    os.makedirs(name=housing_path, exist_ok=TRUE)
    tgz_path = os.path.join(housing_path, "
                housing.tgz")
    urllib.request.urlretrieve(url= housing_url,
                filename=tgz_path)
    housing_tgz = tarfile.open(name=tgz_path)
    housing_tgz.extractall(path = housing_path)
    housing_tgz.close()
fetch_housing_data()
```

```
import pandas as pd

def load_housing_data(housing_path = Housing_path):
    data_path = os.path.join(housing_path, "housing.
                csv")
```

return pd.read_csv(data_path)

2) Take a quick look at the data structure
housing = load_housing_data()
housing.head()
housing.info()
housing["ocean_proximity"].value_counts()
housing.describe()

3) PLOTTING GRAPH
- import matplotlib.pyplot as plt
- import seaborn as sns
- housing.hist(bins = 50, figsize = (20, 15))
- plt.show()

4) Create a Test set

import numpy as np
def split_train_test(data, test_ratio = 0.2):
shuffled_indices = np.random.permutation(len(data))
test_set_size = int(len(data) * test_ratio)
test_indices = shuffled_indices[:test_set_size]
train_indices = shuffled_indices[test_set_size:]
return data.iloc[train_indices], data.iloc[test_indices]

train_set, test_set = split_train_test(data = housing)
len(train_set), len(test_set)
from zlib import

from zlib import crc32
def test_set_check(identifier, test_ratio = .2):
total size = 2**32
hex_repr = crc32(np.int64(identifier)) & 0xffffff

```
h_test. = here_repr <(test_ratio ~~x total_size~~)
  return h_test,
  [test_set_check(i) for i in range(10)]
```

Q.5) Coordinates to Index

```
def ~~from~~ from_z_to_N(z):
  if z >= 0:
     n = 2*z
  else:
     n = -2*z -1
  return n

def cantor_pairing(n1, n2):
  n= (((n1+n2)*(n1+n2+1))/2)+n2
  return n

def lat_lon_to_index( lat, lon):
  lat, lon = int = int(lat *100), int (lon*100)
  lat, lon = from_z_to_N (lat), from_z_to_N
                                       (lon)
  index = cantor_pairing(lat, lon)
  return np.int64(index)

housing['id'] = housing.apply( lambda row): lat_lon_to_
                                           index
  (row['latitude'], row['longitude']), axis=1)

housing['id'].value_counts(),
```

WEEK-3

~~Week-3~~

~~Part-A~~

Step 3

prepare the data for ML Algo

Data cleaning

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = 'median')
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
```

Handling text and categorical ~~attributes~~

```
housing_cat = housing['ocean_proximity']
housing_cat.head(10)
housing_cat['ocean_proximity'].value_counts()
```

Customer Transformers

```
class combined_attributes_adder(BaseEstimator,
                                TransformerMixin):
  def init(self, add_bedrooms_per_room=True):
    self.add_bedrooms_per_room = add_bedrooms_per_room
def fit(self, X, y=None):
        return self
def transform(self, X, y=None):
    rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
    population_per_household = X[:, population_ix] /
                X[:, household_ix]
    if self.add_bedrooms_per_room:
      bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
      return np.c_[X, rooms_per_household, population_per
          household, bedrooms_per_room]
    else:
```

return np.c_[X, rooms_per_household, Population_per_household]

Transformation pipelines

num_pipeline = pipeline([('imputer', SimpleImputer(strategy='median')), ('attribs_adder', CombinedAttributesAdder()), ('std_scaler', StandardScaler())])

housing_num_tr = num_pipeline.fit_transform(housing_num)

housing_num_tr.shape

Select and Train a Model

def display_scores(scores):
print("Scores:", scores)
print("Mean", scores.mean())
print("Standard deviation:", scores.std())

Fine Tune Model

param_grid = [{'n_estimators': [3,10,30], 'max_features': [2,4,6,8]}, {'bootstrap': [False], 'n_estimators': [3,10], 'max_features': [2,3,4]}]

Evaluate your system on the test set.

final_model = grid_search.best_estimator_
X_test = strat_test.set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

```
final_mse = mean_squared_error(y_true=y_test, y_predict
            =final_prediction)
final_rmse = np.sqrt(final_mse)
final_rmse
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from pandas.core.common import linear_regression
```

```
df_sal = pd.read_csv('/content/drive/MyDrive/Salary_
    Data.SalaryData.csv')
df_sal.head()
plt.title('Salary_distribution plt')
sns.distplot(df_sal['Salary'])
plt.show()
plt.scatter(df_sal['YearsExperience'], plt.sal['Salary'],
plt.title('Salary vs Experience')
plt.xlabel('years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()
```

```
# Splitting variables
x = df_sal.iloc[:, :1]
y = df_sal.iloc[:,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)

y_pred_test = regressor.predict(x_test)
y_pred_train = regressor.predict(x_train)
plt.scatter(x_train, y_train, color = 'Hyderabad')
plt.plot(x_train, y_pred_train, color = 'Arabian')
plt.title('Salary vs Experience(Training set))
plt.xlabel('years of Experience')
plt.ylabel('Salary')
plt.legend(['x_train/pred(y_test)', 'x_train/y_train
    'Sal/Exp', loc = 'best', facecolor='white')
plt.box(False)
plt.show()
```

```
print('coefficient: {regressor.coef_}')
print('Intercept: {regressor.intercept_}')
```

Output:

coefficient: [9312.57512673]

Intercept: [26780.09915063]

Date: Page:

Week-4 (Decision Tree)

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
% matplotlib inline
import sklearn.datasets as datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
        roc_auc_score, roc_curve
from sklearn.tree import plot_tree

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV,
        RandomizedSearchCV

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

url = "https://archive.ics.uci.edu/ml/machine-learning-
    databases/iris/iris.data"
df = pd.read_csv(url, header = None, names = ['sepal
    length(cm)', 'sepal width(cm)', 'petal length(cm)',
    'petal width(cm)', 'Species'])
df.head()
df.shape
df.info()
df.isnull().sum()
df.describe().T
cols = df.columns[0:-1]
for i in cols:
    sns.boxplot(y=df[i])
    plt.show()
```

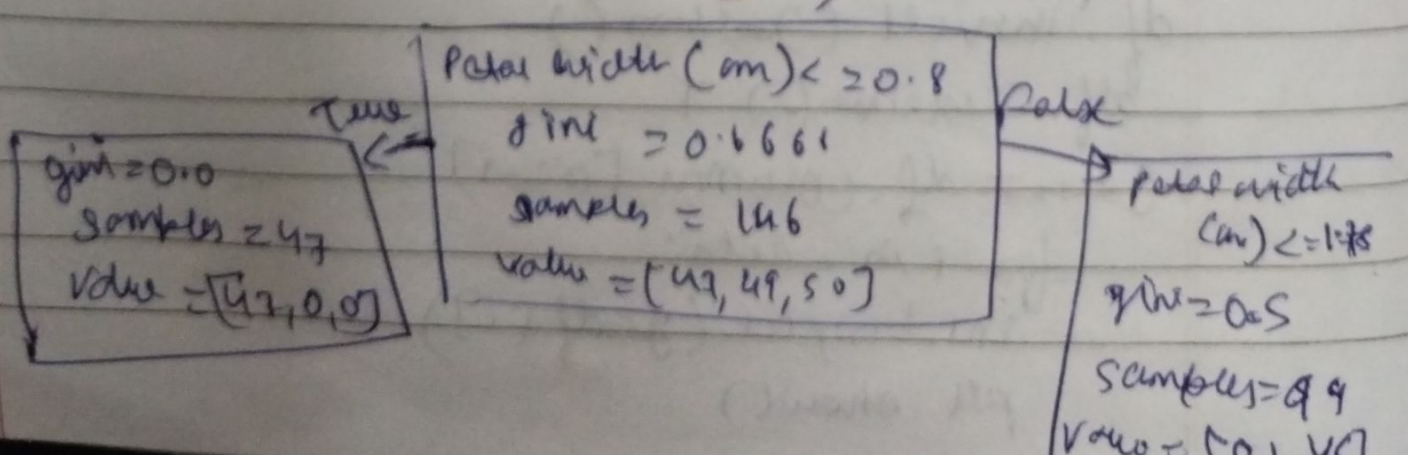
```
q1 = df['sepal width (cm)'].quantile(0.25)
q3 = df['sepal width (cm)'].quantile(0.75)
iqr = q3 - q1
df = df[(df['sepal width (cm)'] >= q1 - 1.5 *
  iqr) & (df['sepal width (cm)'] <= q3 + 1.5 *
                                    iqr)]
df.shape
sns.boxplot(y = df['sepal width (cm)'])
plt.show()
x = df.drop("species", axis = 1)
y = df["species"]
x_train, x_test, y_train, y_test = train_test_split
   (X, y, test_size = 0.3, random_state = 1)

dt = DecisionTreeClassifier(max_depth = 3, max
    samples_leaf = 10, random_state = 1)
dt.fit(x, y)
```

~~from pandas~~ :

```
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

features = x.columns
dot_data = export_graphviz(dt, outfile = None,
          feature_names = features)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```



dt = Decision_Tree classifier (random_state = 1)
dt.fit (x_train, y_train)
y_pred = dt.predict (x_test)
y_prob = dt.predict_proba (x_test)

print ('Accuracy of Desion Tree_Train :', accuracy_score (y_pred, train, y_train))
print ('Accuracy of Decision Tree_Test :', accuracy_score (y_pred, y_test))

print (classification_report (y_test, y_pred))

dt = DecisionTreeClassifier (random_state = 1)
params = {'max_depth' : [2,3,4,5],
'min_samples_split' : [2,3,4,5],
'min_samples_leaf' : [1,2,3,4,5]}
gsearch = GridSearchCV (dt2, param_grid = params, cv=3)
gsearch.fit(x, y)
gsearch.best_params_

dt = DecisionTreeClassifier (** gsearch.best_params_, random_state = 1)
dt.fit (x_train, y_train)
y_pred_train = dt.predict (x_train)
y_prob_train = dt.predict_proba (x_train)[:, 1]
y_pred = dt.predict (x_test)
y_prob = dt.predict_proba (x_test)[:, 1]

print ('confusion matrix - Train :', '\n',
confusion_matrix (y_train, y_pred_train))

print ('\n', 'confusion matrix - Test :', '\n',

```
confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

print('Accuracy of Decision Tree-Train:',
      accuracy_score(y_pred_train, y_train))
print('Accuracy of Decision Tree-Test:',
      accuracy_score(y_pred, y_test)).
```

Accuracy of Decision Tree-Train : 0.9901960-
Accuracy of Decision Tree-Test: 0.954545

Week-5

```
import pandas as pd
from matplotlib import pyplot as plt
% matplotlib inline

df = pd.read_csv("insurance_data.csv")
df.head()

plt.scatter(df.age, df.bought_insurance, marker='+',
                                   color='red')
from sklearn.model_selection
import train_test_split
x_train, x_test, y_train, y_test = train_test_split
   (df[['age']], df.bought_insurance, train_size=0

from sklearn.linear_model.
import LogisticRegression
model = LogisticRegression()
model.fit(x_train, y_train)

y_predicted = model.predict(x_test)
model.predict_proba(x_test)
model.score(x_test, y_test)
                              → accuracy → 1.0

model.coef_
model.intercept_   ] L-R

import math
def sigmoid(x):
return 1/(1+math.exp(-x))
```

```
def predictori_function(age):
z= 0.042*age -1.5396 # 0.041 30133 ~0.042
        and -1.507 26963 ~-1.53
y = sigmoid(z)
return y
```

age 35
prediction function(age) → 0.485

age =43
predictori_function(age) → 0.569

25-4-2024

Week-6

Build KNN classification model for a given dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')

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

df.shape
x = df.drop('Outcome', axis=1).values
y = df['Outcome'].values

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split
    (X, Y, test_size = 0.4, random_state = 42, stratify=y)

from sklearn.neighbors import KNeighborsClassifier

neighbors = np.arange(1,9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors = k)

knn.fit(x_train, y_train)
train_accuracy[i] = knn.score(x_train, y_train)
```

```
test_accuracy[i] = knn.score(x_test, y_test)

plt.title('K-NN varying number of neighbors')
plt.plot(neighbors, test_accuracy

for(i,k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(x_train, y_train)

->
plt.scatter(neighbor, train_accuracy, color=k)
plt.show()

knn = KNeighborsClassifier(n_neighbor=7)

knn.fit(x_train, y_train)
knn.score(x_test, y_test)
```

Output:

0.730519

Week-7

SVM Model

```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
print(cancer.target)
from sklearn.model_selection
import train_test_split

X_train, X_test, y_train, y_test := train_test_split
(cancer.data, cancer.target, test_size=
    0.3, random_state =109)

from sklearn import SVM
clf = svm.SVC(kernel='linear')
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test

from sklearn import metrics
print("Accuracy:", metrics.accuracy_score
        (y_test, y_pred))
```

Output:

Accuracy: 0.9649

Week-8 BackPropagationNeuralNetwork

```
db=np.loadtxt('.../input/dukes...')
np.random.shuffle(db)
y = db[:,0]
x = np.delete(db,[0], axis=1)
x_train, x_test, y_train, y_test = train_test_split
                (x, y, test_size =0.1)
print(np.shape(x_train), np.shape(x_test))

hidden_layer = np.zeros(72)
weights = np.random.random((len[x[0],72))
output_layer = np.zeros(2)
hidden_weights = np.random.random((72,2))

def sum_function(weights, index_col_x):
    result=0
    for i in range(0, len(x)):
        result += x[i] * weights[i][index_hidd_col]
    return result

def activate_layer(layer, weight, x):
    for i in range(0, len(layer)):
        layer[i] = (1.7159 * np.tanh(2.0 *
                    sum_function(weight, i, x))/3.0)

def bpropagation(hidden_layer, output_layer, one_hot
    encoding, learning_rate, x):
    output_derivation = np.zeros(2)
    output_gradient = np.zeros(2)
```

```
for i in range(0, len(output_layer)):
    output_derivative[i] = (1.0 - output_layer[i]) * output_layer[i]

for i in range(0, len(output_layer)):
    output_gradient[i] = output_derivative
        [i] * (one_hot_encoding[i] - output_layer[i])

hidden_derivative = np.zeros(72)
hidden_gradient = np.zeros(72)
for i in range(0, len(hidden_layer)):
    hidden_derivative[i] = (1.0 - hiddenlayer[i] * (
                1.0 + hiddenlayer[i])

for i in range(0, len(hidden_layer)):
    sum = 0
    for j in range(0, len(output_gradient)):
        sum += output_gradient[j] * hidden_weight[i,j]

hidden_gradient[i] = sum * hidden_derivative[i]

recalculate(learning_rate, weight_hidden_gradient, x).

Random boosting and Ada Boosting
dataset = pd.read_csv(" ")
corr_matrix = input_x.corr().abs()
upper = corr_matrix.where(np.triu
    np.ones(corr_matrix.shape), k=1).astype(np.bool

to_drop = [column for column in upper.columns
    if any(upper[column] > 0.95)]
print(to_drop)
x_train, x_test, y_train, y_test =
```

```
Train Test split (X, y, Test size = 0.1)
n_classes = 7
n_estimators = 100
Random_seed = 13
names = ['RandomForestClassifier', 'AdaBoostClassifier']

models = [RandomForestClassifier(n_estimators=
                                 n_estimators),
AdaBoostClassifier(DecisionTreeClassifier
    (max_depth = None), n_estimators=
                        n_estimators)]
for counter, model in enumerate
(models): model.fit(X_train, y_train)

y_pred = model.Predict(X_test)
```

Accuracy RandomForestClassifier: 0.768439

Accuracy AdaBoostClassifier: 0.7341337

Bafna Gold
Date: Page:

Week 9 - Build K-Means to cluster a set of stored data in .csv file

```
iris = pd.read_csv("../IRIS.csv")
x = iris.iloc[:,[0,1,2,3]].values
iris_outcome = pd.crosstab(index=iris["Species"]
                           columns="count")
iris_outcome
sns.FacetGrid(iris, hue="species").map(sns.
    distplot,"petal_length").add_legend()

sns.FacetGrid(iris, hue="species").map(sns.
    distplot,"petal_width").add_legend()
sns.FacetGrid(iris, hue="species").map(sns.distplot,
    "sepallength").add_length()
plt.show()

sns.boxplot(x="Species",y="petal_length",
                           data=iris)
plt.show()
sns.set_style("whitegrid")
sns.pairplot(iris, hue="Species",size=3);
plt.show()

kmeans = Kmeans(n_clusters=3,
    init='k-means++',max_iter=300,
    n_init=10,random_state=0)

y_means = k_means.fit_predict(X)
```

Program: 11 PCA

```
df = pd.read_csv("... data.csv")
df_feature = df.drop(['diagnosis'], axis=1)

from sklearn.preprocessing
import standard scalar
standardized = standard scaler()
standardized.fit(df_features)
scaled_data = standardized.transform(df_feat

from sklearn.decomposition
import PCA
pca = PCA(n_components = 3)
pca.fit(scaled_data)
xpca = pca.transform(scaled_data)
scaled_data.shape
xpca.shape
def diag(x):
  if(x=='M'):
    return 1
else:
    return 0
df_diag = df['diagnosis'].apply(diag)
x_pca[:1]
df_pc = pd.DataFrame(pca.components_
    columns = df_features.columns)
plt.figure(figsize=(15,8))
sns.heatmap(df_pc, cmap='viridis')
plt.title('Principal components correlation
                with the features')
```