

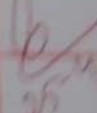
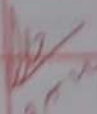
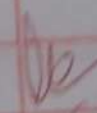

INDEX

NAME: Shreeram

STD

SEC: 6D

ROLL NO: 23248

S.No.	Date	Title	Page No.	Teacher's Sign Remarks
1.	21-3	Import & export data using Pandas library functions		 25/4
2.	28-03	Demonstrate various data preprocessing techniques of datasets		 25/4
3.	4-4	Implement linear and Multidimensional regression algo using appropriate dataset		 25/4
4.	18-4	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new		 25/4

Lab-1 - code-1

1. Write a python program to import and export data using Pandas library functions.

```
→ import pandas as pd
airbnb-data = pd.read_csv("listing-austin.csv")
airbnb-data.head()

url = "http://archive.ics.uci.edu/ml/machine-learning-
-database1/iris/iris.data"

col-names = ["sepal-length-in-cm", "sepal-width-in-
cm", "petal-length-in-cm",
"petal-width-in-cm", "class"]

iris-data = pd.read_csv(url, names = col-names)
iris-data.head()
iris-data.to_csv("cleaned-iris-data.csv")
```

Output:

id	name	host_id	host-name	neighborhood	price
0	2265 Zen-East	2462	Paddy	78702	179
1	5245 Eco-friendly	2466	Paddy	78702	114

	sepal-length-in-cm	sepal-width	petal length	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

class
iris-setosa
iris-setosa

Week 2

2. Demonstrate various data pre-processing techniques for a given dataset.

→ Get the data:

Download the data:

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

```
Download-Root = "https://raw.githubusercontent.com/  
agreen101/handson-ml2/master/"
```

```
HOUSING_PATH = os.path.join("data", "os")
```

```
HOUSING_URL = DOWNLOAD-ROOT + "data-sets/housing.tgz"
```

```
def fetch-housing-data(housing-url, housing-path):  
    os.makedirs(name=housing-path, exist_ok=True)  
    tgz-path = os.path.join(housing-path, "housing.tgz")  
    housing = load-housing-data()  
    housing.head()  
    housing.info()
```

3> Discover and visualize the Data to gain insights.

```
strat_train_set.shape, strat_test_set.shape
```

```
strat_test_set.reset_index().to_feather("frames
```

```
"data/01/strat-test-sets.f")
```

```
housing = strat_train_set.copy(); housing.shape
```

* Now for visualizing geographical Data


```
housing.plot(kind="scatter", x="longitude", y="latitude",
             alpha=0.1)
plt.show
```

Now for correlation

```
corr_matrix = housing.corr()
corr_matrix[["median-house-value"]]
sort_values(ascending=False)
```

from pandas plotting import scatterMatrix.

```
attributes = ["median-house-value", "median-
income"]
```

```
scatter_matrix(frame=attributes, figsize=(12,8))
plt.show()
```

```
housing["rooms-per-household"] = housing["totalrooms"]
housing["household"]
```

4. > Prepare Data for Machine Learning Algorithms

```
from sklearn.input import simple.Imputer
imputer = simple.Imputer(strategy="median")
housing_num = housing.drop("ocean-proximity", axis=1)
imputer.fit(housing_num)
imputer.statistics housing_num.median().values
```

```
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
ordinal_encoder.fit(housing_num)
```

```
attr_addr = combined Attribute Addr(["add-bedroom",
per-room, etc])
```

```
from sklearn.compose import ColumnTransformer
num_attributes = housing_num.columns.tolist()
```

5) select and train a model

from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()

lin_reg.fit(x=housing_prepared,
y=housing_labels)

housing_predictions = ~~linear~~ lin_reg.predict(
housing_prepared)

lin_mse = mean_squared_error(housing_labels,
housing_predictions)

lin_rmse = np.sqrt(lin_mse)

lin_rmse

tree_reg = DecisionTreeRegressor()

tree_reg.fit(x=housing_prepared, y=housing_labels)

forest_reg = RandomForestRegressor()

forest_reg.fit(x=housing_prepared,
y=housing_labels)

6) Time to train the models

grid_search.best_params

grid_search.best_estimator_.get_params()

full_pipeline.named_transformers["cat"]

final_model = grid_search.best_estimator_

x_test_prepared = full_pipeline.transform(
x=x_text)

final_rmse = np.sqrt(final_mse)

final_rmse

squared_errors = (y_test - final_predictions)²

Simple Linear Regression and Multiple Linear Regression.

```
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 random_state
from pandas.core.common import
from sklearn.linear_model import LinearRegression
```

```
df_sal = pd.read_csv(r"C:\salary_data.csv")
```

```
df_sal.head()
```

```
df_sal.describe()
```

```
plt.title("salary distribution plot")
```

```
sns.distplot(df_sal["salary"])
```

```
plt.show()
```

```
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)
```

```
regression = LinearRegression()
```

```
regressor.fit(X_train, y_train)
```

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

```
y_pred_train = regressor.predict(X_train)
```



```
print(f'coefficient : {regressor.coef - }')  
print(f'Intercept : {regressor.intercept - }')  
df_start = pd.read_csv(r'c:\50-startups.csv')  
df_start df_start.head()  
df_start.describe()
```

```
sns.distplot(df_start['profit'])  
plt.show()
```

```
regressor = LinearRegression()
```

```
regressor.fit(x_train, y_train)
```

```
y_pred = regressor.predict(x_test)
```

```
np.set_printoptions(precision = 2)
```

```
result = np.concatenate((y_pred.reshape  
(len(y_pred), 1), y_test.reshape  
(len(y_test), 1), 1)
```

```
result.
```

Je
25-4-23

Week-4

4. Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import seaborn as sns
```

```
iris_data = load_iris()
print(iris_data, desc)
```

```
X = iris_data.data
Y = iris_data.target
```

```
print("shape of X: ", X.shape)
print("shape of Y: ", Y.shape)
```

```
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=42)
```

```
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, Y_train)
DecisionTreeClassifier()
```



```
from sklearn.model_selection import cross_val_score
```

```
scores = cross_val_score(clf, x, y, cv=5)
```

```
accuracy =
```

```
accuracy = scores.mean()
```

```
print("Mean Accuracy Accuracy:", accuracy)
```

Output:-

Mean Accuracy: 0.96666

De
25-4-22

Lab 5

2. Build Logistic Regression model for a given dataset.

```
→ import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
df = pd.read_csv("insurance-data.csv")
```

```
print(df.head(1))
```

```
x_train, x_test, y_train, y_test = train_test_split(
    df[["age"]], df["bought_insurance"],
    test_size=0.2)
```

```
print("X-test:")
```

```
print(X_Test)
```

```
model = LogisticRegression()
```

```
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_test)
```

```
print(y_pred)
```

```
print(model.predict_proba(x_test))
```

```
print(model.score(x_test, y_test))
```

Output:-

Accuracy=0.5

```
import math
```

```
def sigmoid(z)
```

```
    return 1 / (1 + Math.exp(-z))
```

```
def predict(age):
```

```
    z = 0.042 * age - 1.5
```

```
    y = sigmoid(z)
```

```
print(predo(35))  
print(pred(43))
```

Output:-

prediction : array([0, 0, 1, 0, 0, 0, 1, 0])

score : 0.8888

linear Reg Score : 0.584321

predictions : 0.485

0.5685.

✓
9-5-24

Lab-6 and Lab-7
KNN classification model

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("iris.csv")
df.head()
```

```
classes = df["species"].unique()
```

```
colors = ['r', 'g', 'b']
```

```
for i, cls in enumerate(classes):
```

```
    class_data = df[df["species"] == cls]
```

```
    plt.scatter(class_data["sepal.length.cm"], class_data["petal.length"], c=colors[i], label=cls)
```

```
plt.xlabel("sepal length [cm]")
```

```
plt.ylabel("petal length [cm]")
```

```
plt.legend()
```

```
plt.show()
```

```
for i, cls in enumerate(classes):
```

```
    class_data = df[df["species"] == cls]
```

```
    plt.scatter(class_data["sepal.length.cm"], class_data["petal.length"], c=colors[i], label=cls)
```

```
plt.xlabel("Sepal width (cm)")
plt.ylabel("Petal width (cm)")
plt.legend()
plt.show()
```

```
y = df["species"]
x = df.drop(["species"], axis=1)
```

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

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
y2 = knn.predict(X_train)
```

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_pred, y_test)
print(f"Training Accuracy: {score}")
score2 = accuracy_score(y2, y_train)
print(f"Training Accuracy: {score2}")
```

Output:-

Training Accuracy = 1.0
 Test Accuracy = 1.0

8-6-24

Lab-6

```
from sklearn.svm import SVC  
model = SVC(kernel='linear', random_state=0, C=1.0)  
model.fit(X_train, y_train)  
y2 = model.predict(X_train, y_train)
```

```
from sklearn.metrics import accuracy_score  
score1 = accuracy_score(y_pred, y_test)  
print(f"Testing Accuracy : {score1}")  
score2 = accuracy_score(y2, y_train)  
print(f"Training Accuracy : {score2}")
```

Output = Training Accuracy = 1.0

Training Accuracy = 1.0

OK
6-6-20

Lab-8

Build Artificial Neural Network model with back propagation on a given dataset.

⇒

```
import numpy as np
```

```
X = np.array([[2,5], [1,5], [3,6]], dtype=float)
```

```
Y = np.array([92, 86, 85], dtype=float)
```

```
X = X / np.amax(X, axis=0)
```

```
Y = Y / 100
```

```
epoch = 5000
```

```
lr = 0.1
```

```
input_layer_neurons = 2
```

```
hidden_layer_neurons = 3
```

```
output_neurons = 1
```

```
W_h = np.random.uniform(size=(input_layer_neurons,  
                                hidden_layer_neurons))
```

```
b_h = np.random.uniform(size=(1, hidden_layer_neurons))
```

```
W_out = np.random.uniform(size=(hidden_layer_neurons,  
                                output_neurons))
```

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

```
def sigmoid(x):
```

```
    return 1/(1+np.exp(-x))
```

```
def derivative_sigmoid(x):
```

```
    return x*(1-x)
```

```
for i in range(epoch):
```

```
    hinp = np.dot(x, wh)
```

```
    hinp = hinp + bb
```

```
    hlayer_act = sigmoid(hinp)
```

```
    outinp1 = np.dot(hlayer_act, word)
```

```
    outinp = outinp1 + bout
```

```
    output = sigmoid(outinp)
```

```
    EO = y - output
```

```
    outgrad = derivative_sigmoid(output)
```

```
    d_output = EO * outgrad
```

```
    EH = d_output * dot(word, T)
```

```
    hiddengrad = derivative_sigmoid(hlayer_act)
```

```
    d_hidden_layer = EH * hiddengrad
```

```
    bout += hlayer_act.T.dot(d_output) * lr
```

```
    wh += x.T.dot(d_hidden_layer) * lr
```

```
    print("Input: \n", str(x))
```

```
    print("Actual output: \n", str(y))
```

```
    print("Predicted output: \n", output)
```

Output :

Input : $\begin{bmatrix} [0.66667 & 1 &] \\ [0.33333 & 0.66666] \\ [1 & 0.666667] \end{bmatrix}$

Actual output : $\begin{bmatrix} [0.92] \\ [0.86] \\ [0.89] \end{bmatrix}$

Predicted output : $\begin{bmatrix} [0.8913061] \\ [0.8851982] \\ [0.88962179] \end{bmatrix}$

Ans
6-6-2

Output :-

Testing Accuracy = 1.0

Training Accuracy = 1.0

(b)

```
from sklearn.ensemble import AdaBoostClassifier
adb = AdaBoostClassifier()
adb_model = adb.fit(X_train, y_train)
y_pred = adb.predict(X_test)
y2 = adb.predict(X_train)
```

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_pred, y_test)
print(f"Testing Accuracy : {score}")
score2 = accuracy_score(y2, y_train)
print(f"Training Accuracy : {score2}")
```

Output :- Testing Accuracy: 0.97777

Training Accuracy: 1.00.

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_pred, y_test)
print(f"Testing Accuracy : {score}")
score2 = accuracy_score(y2, y_train)
print(f"Training Accuracy : {score2}")
```

Outputs:-

Testing Accuracy = 1.0

Training Accuracy = 1.0

12
66

Lab - 10

10. Build a k-means Algo to cluster a set of data stored in .csv file.

```
→ import matplotlib.pyplot as plt
from sklearn import dataset
from sklearn import KMeans
import pandas as pd
import numpy as np

iris = dataset.load_iris()
x = pd.DataFrame(iris.data, columns=["sepal-length",
                                     "sepal-width", "petal-length", "petal-width"])
y = pd.DataFrame(iris.target, columns=["target"])

model = KMeans(n_clusters=3)

model.fit(x)

plt.figure(figsize=(14, 14))
colormap = np.array(["red", "blue", "black"])
plt.subplot(2, 2, 1)
plt.scatter(x["petal-length"], x["petal-width"],
            c=colormap[y["target"]], 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[model.labels_], s=40)

plt.title("K-means clustering")
```



```
plt.xlabel("petal length")  
plt.ylabel("petal width")  
plt.show()
```

Lab 11

Implement dimensionality reduction using PCA method

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

```
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
scaler.fit(df)
```

```
scaled_data = scaler.transform(df)
```

```
from sklearn.decomposition import PCA  
pca = PCA(n_components=2)  
pca.fit(scaled_data)
```

```
x_pca.shape
```

```
plt.figure(figsize=(8,6))
```

```
plt.scatter(x_pca[:,0], x_pca[:,1], c=cancer["target"],  
            cmap="plasma")
```

```
plt.xlabel("First principal component")
```

```
plt.ylabel("Second principal component")
```