INDEX

Name Nidhi. A Std M.Sc Sec 6D

Roll No. Subject School/College

School/College Tel. No. Parents Tel. No.

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1	3/2/25	Lab-0 - Basic		
2.	3/3/25	Lab - 1 - Diabetes dataset		
3.	10/2/25	Lab -2 - ID3		
4.	17/3/25	Lab - 3 - End to end project		
5.	24/3/25	Lab-4 - linear, multinear, logisti		
6.	7/4/25	Lab-5 - KNN & SVM		
7.	14/4/25	Lab-6 - Random Forest		
8	21/4/25	Lab - 7 - Boosting ensemble		
9.	21/4/25	Lab-8 - K-means clustering		
10	21/4/25	Lab-9 - PCA		

Date __/__/__
Page _____

Lab - 0

1. Method - 1: Initialising value directly into D.F

```
import pandas as pd
data = { 'Name': ['Alice', 'Bob', 'Charlie', 'David'],
         'Marks': [25, 30, 45, 35],
         'USN': ['1BM177', '1BM255', '1BM240',
                 '1BM203'] }
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

OUTPUT

	Name	Marks	USN
1	Alice	25	1BM177
2	Bob	30	1BM255
3	Charlie	45	1B240
4	David	35	1BM203

2. Method 2: Importing datasets from sklearn.datasets

```
from sklearn.datasets import load_diabetes
diab = load_diabetes()
df = pd.DataFrame(diab.data, cols = features_name)
df['target'] = diab.target
print("Sample data:")
print(df.head())
```

Sample data:

age	sex	bmi	bp	S1	S2	S3	S4	S5	S6	tar
0.038	0.050	0.061	0.02	-004	-003	-0.64	0.03	0.01	151	
-0.001	-0.044	-0.051	0.02	-0.08	-0.19	0.48	0.80	0.90	75	
0.085	0.050	0.044	-0.00	-0.04	0.18	0.34	0.40	-0.40	141	
0.005	-0.044	-0.011	-0.03	0.01	0.34	-0.18	0.44	0.44	206	

Date __/__/__
Page

3) Method 3 Importing datasets from a specific .csv file

```
file_path = 'data.csv'
df = pd.read_csv(file_path)
print("Sample data")
print(df.head())
print("\n")
```

Sample data.

ID	Name	Age	City
1	Alice	25	new york
2	Bob	30	los angeles
3	Charlie	45	chicago
4	David	20	houston
5	Eva	10	washington dc

4) Method 4. Downloading datasets from Kaggle.

```
df = pd.read_csv("/content/Dataset of diabetes
print(df.head())
```

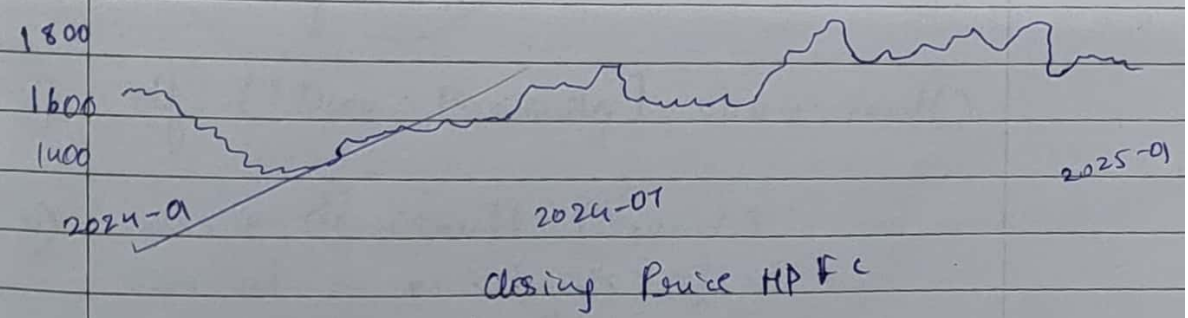
ID	No_Patient	Gender	AGE	Urea	Cr	HbA1c
0	502	F	50	4.7	46	4.9
1	735	M	26	4.5	62	4.9
2	420	F	50	4.7	46	4.9
3	680	F	50	4.7	46	4.9
4	504	M	33	7.1	46	4.9

Date __/__/__
Page _____

TO DO

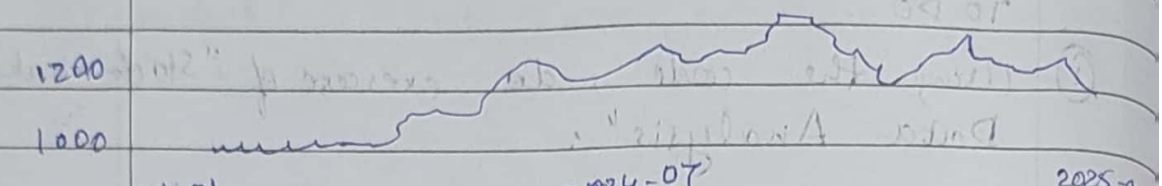
① Using the code, do exercise of "Stock Market Data Analysis":

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS",
           "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01",
                   end="2024-12-30", group_by='ticker')

for ticker in tickers:
    data.loc[:, (ticker, 'Daily Return')] =
        data[ticker]['Close'].pct_change()

plt.figure(figsize=(12, 16))
for i, ticker in enumerate(tickers):
    plt.subplot(3, 1, i+1)

plt.tight_layout()
plt.show()

plt.tight()
plt.show()
```



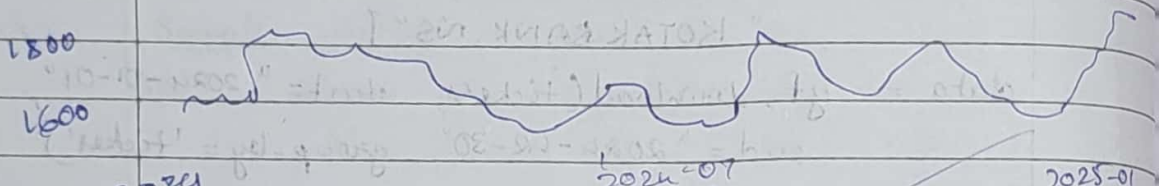
Closing Price HDFC

Date __/__/__
Page



ICICI closing price

KMB-1



KOTAK BANK
CLOSING PRICE

03.03 work

Lab 1

~~LAB 1~~

Diabetes

Print(df.isnull().sum())

o/p Ip 0

:

Class 0

dtype: int64

Adult income

age 0

:

income 0

dtype: int64

There are no missing values.

2. Gender & Class (Diabetes)

↓ ↓

ordinal encoders onehot encoders

ordinal_encoder = OrdinalEncoder(categories=["F", "M"]] handle_unknown = "use_encoded_value" unknown_value = -1)

Write python code consider filename as housing.csv

(i) to read csv

df = pd.read_csv(housing.csv)

ii) display all columns

df.info()

iii) statistical information

df.describe()

iv) df["Ocean Proximity"].value_count()

v) miss = df.isnull().sum()

miss_col = miss[miss > 0]

print(miss_col)

Date / /
Page

Q i) Which had missing value? How did you handle them

Diabetes:

columns: BMI BP, Glucose

They were replaced with median

ii) What categorical cols did you identify. How was it encoded?

Diabetes:

Gender Ethnicity,

Encoding Method.

Gender: Binary (1 - Male 0 - Female)

AdultIncome:

Sex Work

iii) Min max scaling: Rescales data to fixed rays.

standardization: Rescales data to have a mean of 0 & std

→ min max is used when a fixed rays is given sfdization does not need 0,

10.03.25 Lab - 2 Date __/__/__ Page

Iterative Dichotomiser (ID3) Algorithm.

Algorithm

```
def ID3 (D, A):
    if D is pure or A is empty:
        return a leaf node with the majority
        class in D.
    else:
        A_best = argmax (InformationGain (D, A))
        root = Node (A_best)
        for v in values (A_best):
            D_v = subset (D, A_best, v)
            child = ID3 (D_v, A - {A_best})
            root.add_child (v, child)
        return root.
```

ID3 algorithm is a decision tree algorithm used in machine learning and mining for classification tasks.

1. Prepare the Data:
   - Each instance has a set of features (attributes)
   - Each instance has a target class that you want to predict

2. Select the Best Attribute (Features)
   Check if the dataset is pure:
   - If all instances in the dataset have the same class label, return that class as the result (leaf node)

3. Check if the dataset is empty:
   If empty, return most frequent class

Date __/__/__
Page _____

4. Check if no more features to split
   - If no, return the most freq.

5. Calculate IG:

$$IG(D,A) = Entropy(D) - \sum \frac{|D_v|}{|D|} \times Entropy(D_v)$$

$$Entropy = p(c_i) \times \log(p(c_i))$$

6. Choose feature with highest IG
   → is the splitting criterion

7. Split the dataset based on the selected feature.

8. Recursively apply ID3 to each subset

9. Prune the tree by removing branches that hold no value.

Example)

Weather	Play outside
Sunny	Yes
Rainy	No
Overcast	Yes
Sunny	Yes
Rainy	No

$$E(PlayOutside) = -\left(\frac{3}{5}\log\frac{3}{5} + \frac{2}{5}\log\left(\frac{2}{5}\right)\right)$$

$$= 0.971$$

Weather	Ye	No	Entropy
Sunny	2	0	0.
Rainy	0	2	0
Overcast	1	0	0

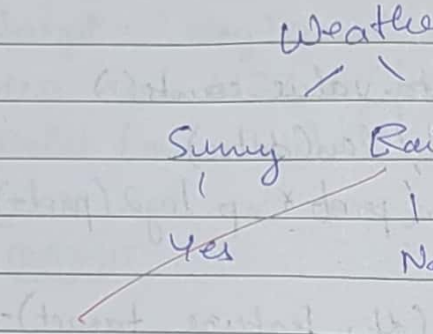
Date __/__/__
Page

Weighted entropy

$$E(\text{Weather}) = \left(\frac{2}{5} \times 0\right) + \left(\frac{2}{5} \times 0\right) + \left(\frac{1}{5} \times 0\right) = 0$$

Information Gain = $0.970 - 0 = 0.970$.

Weather is the best attribute to split on



SK
10/3/25

17/03/25

Implementation of ID-3

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

data = pd.read_csv("weather.csv")
df = pd.DataFrame(data)

def entropy(data):
    class_counts = data.value_counts()
    prob = class_counts/ len(data)
    return -np.sum(prob * np.log2(prob))

def information_gain(df, feature, target):
        total_entropy = entropy(df[target])
        feature_value = df[feature].unique()
        weighted_entropy = 0
        for value in feature_values:
            subset = df[df[feature] == value]
            weighted_entropy += (len(subset) /
                                 len(df)) *entropy(subset[
    return total_entropy - weighted_entropy.

def id3(df, features, target):
        if len(df[target].unique()) == 1:
            return df[target].iloc[0]
        if not features:
            return df[target].mode()[0]
    gains = {feature: information_gain(df, feature,
                target) for feature in features}
    best_feature = max(gains, key = gains.get)
    tree = {best_feature: {}}
```

Date __/__/__
Page 10/11

```
    for value in df[best_failure].unique():
        subset = df[df[best_failure] == value]
        tree[best_feature][value] = id3(subset, [f
            for f in features if f != best_feature],
                target)
    return value tree
```

```
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']
target = 'PlayTennis'
desc_tree = id3(df, features, target)
print(desc_tree)
```

OUTPUT

```
{'Outlook': {'Sunny':
  {Humidity:{'High': 'No', 'Low': 'Yes'}},
  'Overcast': 'Yes', 'Rain': {'Wind': {'Weak': 'Yes',
    'Strong': 'No'}}}}
```

17/03/25

Date
Page

Lab - 3 : End to End ML Projects

① Get Data

```
from sklearn.datasets import fetch_california_housing
cal_house = fetch_california_housing(as_frame=True)
print (cal_house.DESCR)
print cal_house.head()
```

② Discover & Visualise the data to Gain Insights

```
import matplotlib.plt as plt
import seaborn as sns

print (cal_house.frame.isnull().sum())

plt.figsize (figsize = (8, 6))
sns.histplot (cal_house.frame['MedHouseVal']
              , kde = True)
plt.title ('Dist')
plt.show()
```

③ Prepare the data for ML Algo

```
from sklearn.model_selection import test_train_split
from sklearn.preprocessing import StandardScaler

X = cal_house.frame.drop ('MedHouseVal', axis=1)
y = cal_house.frame ['MedHouseVal']

X_train, X_test, y_train, y_test = train_test_split
                    X, y, test_size = 0.2, random_state=42
```

Date __/__/__
Page _____

```
scaler = StandardScaler()
X_train_select = scaler.fit_transform(X_train)
X_test_scaler = scaler.transform(X_test)
```

④ Select a Model and Train it.

```
from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error,
    r2_score
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)
y_pred_linear = linear_model.predict(X_test_scaled)
mse_linear = mean_square_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)
print(f"mes_liner}, {r2_linear}").
```

⑤ Fine Tune Your Model

```
from sklearn.metrics import root_mean_square
_error
import numpy as np
y_pred = model.predict(X_test)
rmse = root_mean_squared_error(y_test, y_pred)
print(f"rmse}")
```

Date / /
Page

Lab - 4.

Simple Linear Regression

```
function [m, b] = reg(x, y, r, num)
    m = 0
    b = 0
    n = length(x)

    for i ← 1 to num:
        pred = []
        for j ← 0 to n-1
            pre = m * x[j] + b
            append pre to pred

        errors = []
        for j ← 0 to n-1
            error = pred[j] - y[j]
            errors.append(error)

        cost = (1/n) * sum(error^2 for error in errors)

        m_gradient = (2/n) * sum(errors[j] * x[j] for j ← 0 to n-1)

        b_gradient = (2/n) * sum(error)

        m = m - m_gradient * r
        b = b - b_gradient * r

    return m, b
```

Date __/__/__
Page _____

logistic linear Regression

```
function logistic_regression(X, y, lr, num):
    m = length(X[0])    # features
    n = length(X)
    weights = array of 0 to (with m
    bias = 0

    for i 1 to num:
        pred = []
        for j ← 0 to n-1:
            lin = dot_product(weights, X[j]
                  + bias
            pred = sigmoid(lin)
            predict.append(pre)

        errors = []
        for j ← 0 to n-1
            error = pred[j] - y[j]
            errors.append(errors)

    cost = -(1/n) * sum(y[j] * log(prediction[j]
           + (1-y[j]) * log(1-pred[j])
           for j from 0 to n-1)

    weight_grad = (1/n) * dot_product(errors, x)
    bias_grad = (1/n) * sum(errors)

    weights = weight - lr * weight_grad
    bias = bias - lr * bias_grad

return weights, bias
```

Date / /
Page

Algo
Linear Regression

1. Initialise m & b = 0
2. Make prediction mx + b.
3. Find error by y - pred.
4. Update m & b using gradient descent
5. Repeat 4 steps for num iteration

Algo Logistic Regression

1. m is the length of features initialized to
2. n = no. of datapoints
3. Initialize weight & bias to 0.
4. Make prediction weight * X + bias
5. Use sigmoid function for pred.
6. Find errors using pred - y[i]
7. Update weight & bias using gradient descent
8. Repeat steps & return weight & bias

Algo Multilinear Regression

Same as linear but with multiple features.

Date __/__/__
Page _____

Multilinear Regression

```
function multilinear_reg (x, y, r, num):

    m = length (X[0])
    weights = array of 0 of length m
    bias = 0
    n = length (x)

    for i ← 1 to num:
        pred = []
        for j ← 0 to n-1
            pr = dot-product (weight, x[j]) +
                 bias
            pred.append(pr)

        errors = []
        for j ← 0 to n-1
            error = pred[j] - y[j]
            errors.append(error)

        cost = 1/n * sum( y[j] * log(pred[j]) +
                          (1-y[j])) errors^2 in
                                error in errors)
        weight_grad = 1/n * dot prod(error, x)

        bias_grad = 1/n * dot sum (error)

        weights = weight - r * weight_grad
        bias = bias - r * bias_grad

    return weigh & bias.
```

7/4/25 Lab-5

Date __/__/__
Page

K-Neighbour

K-Nearest Neighbour

Algorithm.

1. Choose the number of neighbours (K).
2. Calculate the distance:
   For each instance in training dataset, calculate dist b/w training dataset with test data. dist = $\sqrt{(x-x')^2+(y-y')^2}$
3. Sort distances in ascending order
4. Select k nearest neighbours
5. For classification
   Take a majority vote among the 'k' neighbours to determine predicted label
6. For regression.
   Take an average of k neighbours to return predicted value
7. Return the prediction.

Pseudocode

```
function kNN(train, test, k):
    distances = []
    for each (features, label) in train:
        distance = cal_dis(features, test)
        distances.append((dist, label))

    distances.sort(key = lambda x:(1O])

    k_n_n = distance[:k]
```

```
if classification:
    votes = {}
    for neighbor in k_nearest:
        label = neighbor[1]
        if label in votes:
            votes[label] += 1
        else
            votes[label] = 1
    predicted_label = max(votes, key=votes.get)
    return predicted_label

else:
    total = 0
    for neighbor in k_nearest:
        total += neighbor[1]
    predicted_val = total/k
    return predicted_val.

function cal_dist(features1, features2):

    return sqrt(sum((f1-f2)**2 for f1, f2 in
                zip(features1, features)))
```

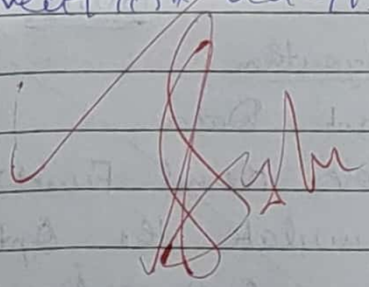
SVM

Algorithm

1. Input Data
2. Choose Kernel Function
3. Formulate the Optimization Problem.
4. Solve Optimization Problem.
5. Make Predictions
6. Output the model.

Date __/__/__
Page

```
function svm (train, labels, kernel, C):

    n = length (train)
    alpha = array of zeros of size n
    b = 0
    w = array of zeros of no. of features
    for i from 0 to n-1:
        if alpha[i] > 0:
            b = labels[i] - dot_prod(w, training[i])
            break
    return (w, b)

function predict (model, test):
    (w, b) = model
    des_val = dot_prod(w, test) + b
    if des_val >= 0:
        return +1;
    else
        return -1

function dot_prod (vect1, vect2):
    result = 0
    for i from 0 to len(vect1) - 1:
        res += vect1[i] * vect2[i]
    return res
```



2/4/25 Lab 6 Date __/__/__ Page _____

i) Random Forest Ensemble Method

1. Load dataset D with features X and target Y

2. Preprocess the data:
   - Handle missing value
   - Encode categorical variables
   - Split dataset into training set D_train and test set D_test.

3. Initialise Randomforest with parameter:
   - n_estimators (no. of trees)
   - max_depth (max depth of trees)
   - random_state (seed for replicability)

4. For each tree t=1 to n_estimators:
   - Randomly select a bootstrap sample of data points from D_train
   - For each node in the tree:
     i) Randomly select a subset of features
     ii) Calculate Gini Impurity or Entropy for each feature and split based on the best value
     - Gini Impurity: Gini = 1 - Σ(p_i)^2
     - Entropy: Entropy(S) = -Σ(p_i * log2(p_i))
     - Split data at the best feature & continue until max depth is reach or stopping condition are met.

5. After all trees are built, make predictions:
   a) For each test instance x_test, aggregate

predictions from all trees using majority voting

i) Prediction = mode (prediction from all tree)

6. Evaluate the model:
   Calculate accuracy, precision, recall, F1 score.

7. Return Model performance.

21/4/26 Lab-7 Date __/__/__ Page ____

Boosting Ensemble Method.

1. Load dataset D with features X and target Y.
2. Preprocess the data:
   a. Handle missing values
   b. Encode categorical variables
   c. Split dataset into training set D_train and test set D_test.
3. Intialiase the weight of all training sample equally.
   $w1 = w2 = \dots = W_N = 1/N$
   where N is the no of training sample
4. For each iteration t=1 to n_estimator.
   a. Train a weak classifier h_t using the weighted dataset D_train with sample weights w_i.
   b. Calculate the error of the classifier
      $\varepsilon_t = (\sum w_i * I(h_t(x_i) \neq y_i)) / \sum w_i$
      where I is the indicator function
   c) Compute the classifier weight (α_t):
      $\alpha_t = 0.5 * \log((1-\varepsilon_t)/\varepsilon_t)$
   d) Update the weights of miscalculated points:
      $w_i = w_i * \exp(\alpha_t * I(h_t(x_i) \neq y_i))$
   e) Normalize the weight so that they sum to 1:
      $w_i = w_i / \sum w_i$

Date __/__/__
Page _____

5. After all iteration, combine weak classifier:
   a) Final prediction for test point
   $x = sign(\sum \alpha_t * h_t(x))$

6. Evaluate the model
   a) Calculate accuracy, precision, recall, F1-score

7. Return model performance

Date __/__/__
Page _____

Lab - 8

K-means Clustering Algorithm.

1. Load dataset D with data-points x_i, where $x_i \in R^d$ (d-dimensional space)

2. Preprocess the data:
   a. Handling the missing value
   b. Stardize the data

3. Intialize k centroids randomly from the dataset:
   C-1, C-2, ..., C-k

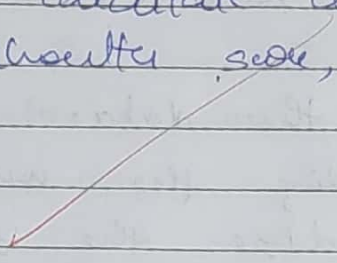
4. Repeat until convergence (centroids no loger change):
   a. For each data point x_i =
      i) Calculate the distance to each centroid C_j:
         distance $(x_i, C_j) = \| x_i - C_j \|_2$
      ii) Assign point x_i to the closest centroid:
         Cluster (x_i) = argmin_j distance (x_i, C_j)
   b. Update centroids by calculating the mean of points in each cluster:
      $C_j = (1/|S_j|) * \sum_{x_i \in S_j} x_i$
      where S_j is the set of points assigned to centroid C_j.

5. Convergence:- If centroid do not change significantly b/w iterations, stop.

Date __/__/__
Page

6. Output:
   a. Final centroids C-1, C-2, ..., C-k
   b. Cluster assignments for each data point

7. Optionally, calculate clustering metrics (eg., silhouette score, inertia).



Date / /
Page

Lab 9

Principle Component Analysis

1. Load data D with data points x_i, where x_i ∈ R^d (d-dimensional space)

2. Pre process the data:

   a. Handle missing values
   b. Standardize the data by subtracting mean of each feature
      $x_j = (x_j - \mu_j) / \sigma_j$
      where μ_j is the mean of feature j & σ_j is the standard deviation

3. Compute the covariance matrix Σ of standardize data:
   $\Sigma = (1/N) * X^T * X$
   where X is the data matrix (each row is a data point)

4. Perform eigenvalue decomposition on the covariance matrix Σ:
   $\Sigma * v = \lambda * v$
   where λ are the eigenvalue, and v are the eigenvectors

5. Sort the eigenvalues in descending order and select the top k eigenvectors

6. Project the data onto the k eigenvectors (PCA):
   X_new = X * V

Date __/__/__
Page

where V is the matrix of the top k eigenvectors

Output:

Transformed dataset X_new (reduced dimensionality)

Optionally, visualize the reduced data (if k=2 or k=3).