Date 3/3/25
Page

Datasets

1. Healthcare Dataset

→ This synthetic healthcare dataset has been created to serve as a valuable resource for data science, machine learning & data analysis enthusiasts. The inspiration behind this dataset is rooted in the need for practical and diverse healthcare data for research purposes.

Attributes: Name, Age, Gender, Blood Type, Medical Condition, Date of Admission, Doctor, Hospital, Insurance Provider, Billing amount, Room Number, Admission Type, Discharge Date, Medication, Test Results.

No. of Rows = 55501
No. of unique values = 49992
No. of Columns = 15

2. Workout & Fitness Tracker Dataset

→ It is designed to help analyze and predict workout efficiency based on user activity, health metrics and lifestyle factors.

Attributes: User ID, Age, Gender, Height (cm), Weight (kg), Workout Type, Workout Duration (mins), calories burned, heart rate (bpm), steps taken, distance (km), workout intensity, sleep hours, water intake (liters), daily calories intake, resting heart rate (bpm), VO2 Max, Body Fat (%), Mood Before Workout, Mood After Workout.

No. of Rows = 10,000

Date __/__/__
Page _____

No. of columns = 21

3. Fraud Detection Transactions Dataset

This dataset is designed to help data scientists and machine learning enthusiasts develop robot fraud detection models.

Attributes : Transaction-id, User-id, transaction-amount, transaction-type, timestamp, account-balance, device-type, location, merchant-category, ip-address-flag, previous-fraudulent-activity, daily-transaction-count, avg-transaction-amount-7d, failed-transaction-count-7d, card-type, card-age, transaction-distance, authentication-method, risk-score, is-weekend, fraud-label.

No. of ~~Tota~~ rows = 50001
No. of columns = 21.

4 Diabetes Dataset

This dataset includes various health parameters, lifestyle habits and genetic predispositions that contribute to diabetes risk.

Attributes: Age, Pregnancies, BMI (Body Mass Index), glucose, Blood Pressure, HbA1c (Hemoglobin A1c level), LDL (low-density lipoprotein), HDL (High-Density lipoprotein), Triglycerides, Waist Circumference, Hip Circumference, WHR (Waist-to-Hip Ratio), Family History, Diet Type, Hypertension, Medication Use, Outcome

Date __/__/__
Page _____

No. of rows = 9539
No. of Columns = 17

5. Student Performance & Learning Style Dataset

This dataset provides insights into how different style habits, learning styles, & external factors influence student performance.

Attributes: student-id, age, gender, study-hours-per-week, preferred-learning-style, online-courses-completed, participation-in-discussions, assignment-completion-rate(%), Exam-score(%), Attendance-Rate(%), use-of-educational-tech, self-reported-stress-level, time-spent-on-social-media, sleep-hours-per-night, final-grade

No. of rows = 10001
No. of columns = 15

3/03/25

Date 10/3/24
Page

Lab-2

Linear Regression

→ Linear Regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables

→ Linear Regression is also a type of supervised machine-learning algorithm that learns from the labelled datasets and maps the data points with most optimised linear functions which can be used for prediction on new datasets.

→ It computes linear relationship between the dependent variable & one or more independent features by fitting a linear equation with observed data.

Advantages:

→ Simplicity & ease of implementation
→ Interpretability
→ Computationally Efficient
→ Baseline Model
→ Feature importance

Types of Linear Regression → Simple LR, Multiple LR

⇒ Single LR: It is the simplest form of LR & it involves only one independent variable & one dependent variable. The equation for simple LR is:

$y = \beta_0 + \beta_1 X$

$y \rightarrow$ dependent variable $\beta_0 \rightarrow$ Intercept

$X \rightarrow$ independent variable $\beta_1 \rightarrow$ slope

Date __/__/__
Page _____

Code:

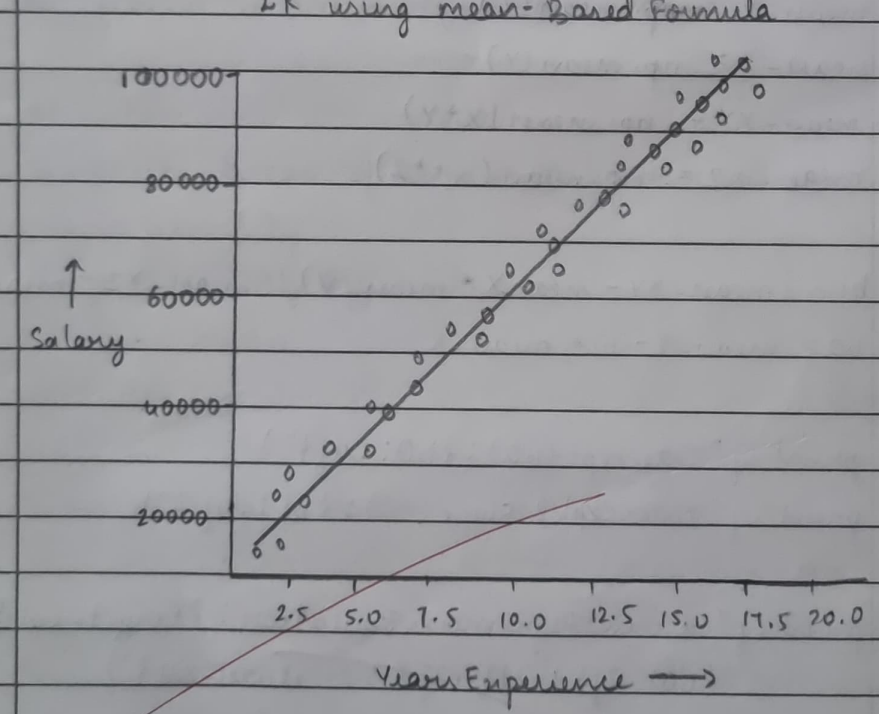
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("/experience-salary.csv")

feature_col = df.columns[0]
target_col = df.columns[1]

X = df[feature_col].values
Y = df[target_col].values

mean_X = np.mean(X)
mean_Y = np.mean(Y)
mean_XY = np.mean(X*Y)
mean_X2 = np.mean(X**2)

b1 = (mean_XY - mean_X * mean_Y)/(mean_X2 - mean_X**2)
b0 = mean_Y - b1 * mean_X

print(f"Intercept (b0): {b0:.4f}")
print(f"Intercept & Slope (b1): {b1:.4f}")

print(f"Linear Regression Equation: {target_col} =
      {b0:.2f} + {b1:.2f} * {feature_col}")

plt.scatter(X, Y, color="blue", marker="o", s=30,
            label="Actual Data")

plt.plot(X, b0+b1*X, color="red", label="Regression line
         (Mean-Based)")
```

Date __/__/____
Page _____

```
plt.xlabel(feature_col)
plt.ylabel(target_col)
plt.title("Linear Regression using mean-based
          formula")
plt.legend()
plt.grid(True)
plt.show()
```

Output:

Intercept(b0): 8775.4214
Slope(b1): 4301.3422
Linear Regression Eq: Salary = 8775.42 + 4301.34 * YearsExperience



10/2/26

Date 14/3/25
Page

Lab-3

Housing [Chapter-2: End to End ML]

The script focuses on housing price prediction using ML techniques in Python. It follows the end-to-end ML pipeline from data loading to model evaluation. Below is a structured breakdown:

1. Data loading & Exploration

→ Loads the california housing dataset from a csv file (housing.csv) which contains 20,640 records (rows) and 10 attributes (columns).
→ Display basic statistics using .head(), .info() & .describe()
→ Analyzes categorical feature distribution (ocean-proximity.value_counts())
→ Plots histograms to visualize numerical feature distributions.

2. Data Splitting

→ Uses random shuffling (split_train_test) and hash-based splitting (split_train_test_by_id) for reproducibility
→ Implements stratified sampling based on income categories to maintain proportional representation.

3. Data visualization

→ Scatter plots illustrate housing locations based on longitude & latitude
→ Color-coded visualizations highlight population density

Date __/__/__
Page _____

and median house value

→ Scatter matrix explores relationships between key features.

4. Feature Engineering

→ Creates new derived features:
- Rooms per household = total-rooms / households
- Bedrooms per room = total-bedrooms / total-rooms
- Population per household = population / households

→ Handles missing values using median imputation

5. Correlation Analysis

→ Computes Pearson's correlation coefficient between numerical features & median house value.

6. Data Preprocessing

→ Uses One Hot Encoder for categorical variables (ocean-proximity).

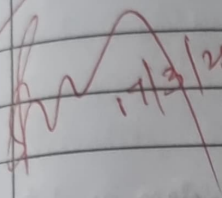
→ Applies pipelines to automate preprocessing:
- Numerical pipeline: Handles missing values, feature engineering & standardization.
- Categorical pipeline: Applies one-hot encoding

→ Usese ColumnTransformer to combine numerical & categorical processing.

Date __/__/__
Page _____

7. Model Training & Evaluation

→ Trains Linear Regression to predict house prices
→ Evaluates performance using root mean squared error (RSME)
→ Performs cross-validation for better performance estimation.



Date 24/3/25
Page

Lab 4
Decision Tree ID3 Classification

Pseudocode

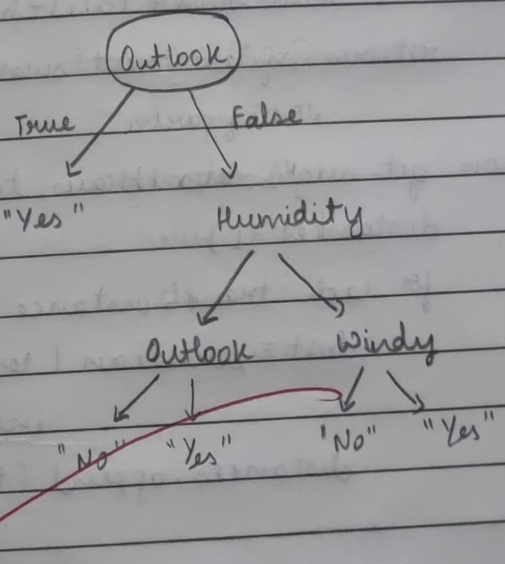
```
Function ID3(Data, Features, Target):
    if all instances in data have the same class:
        return the class label

    if features is empty:
        return the most common class in data

    BestFeature ← Feature with highest information
                  gain in features
    Tree ← Create a root node labeled BestFeature

    For each value v in BestFeature:
        Subset ← Subset of Data where
                 BestFeature = v
        If Subset is empty:
            Add a leaf node with the most
            common class in Data
        else:
            ChildNode ← ID3(Subset,
                        Features - {BestFeature},
                        target)
            Add childnode to Tree
    Return Tree.
```

Date __/__/__
Page _____

Entropy for each feature:
Outlook: 1.5774
Temperature: 1.5567
Humidity: 1.0000
Windy: 0.9852

Information Gain for each feature:
Outlook: 0.2467
Temperature: 0.0292
Humidity: 0.1518
Windy: 0.0481

Root Node: Outlook



Date 7/4/25
Page

Lab 5

KNN Algorithm

```
Function split(data, 0.2):
    split_index ← length(data) × (1-0.2)
    train = first split_index elements of data
    test = remaining elements of data
    return train, test

function euclidian(a,b):
    sum=0
    for i=1 to length(a):
        sum=sum + (a[i]-b[i])^2
    return square_root(sum).

function get_neighbours(train, test_instance, k):
    distances=[]
    for each train_instance in train:
        dist = euclidian(test_inst, train_inst.features)
        distances.append(train_instance, dist)
    distances.sort()
    neighbours ← first k elements of distances
    return neighbours

function predict(train, test_instance, k):
    neighbours ← get_neighbours(train, test_instance, 3)
    count occurences of each class in neighbours
    return class with max(count)
```

Date __/__/__
Page _____

```
function knn():
    train, test = split.dataset (dataset, 0.2)
    k = 3
    predictions = []

    for each test-instance in test-set:
        predicted-class = predict (train, test-
                                   instance.features, k)
        predictions.append (predicted-class)
```

- Dataset use is iris.csv
feature 1 = sepal length
feature 2 = sepal width

Predicted class for [6.7, 3.5, 1.7] : Iris-versicolor.

Date __/__/____
Page _____

SVM algorithm

```
function load_dataset(filename):
    dataset = []
    open(file)
    skip header
    for each row in file:
        features = first 2 feature values
        label = 1 if class is 'Iris-setosa',
                else 0
        dataset.append([features, label])
    return dataset

function train_test_split(dataset, test_ratio):
    split_index = length(dataset) × (1 - test_ratio)
    train_set = dataset[:split_index]
    test_set = dataset[split_index:]
    return train_set, test_set

function dot_product(a, b):
    return sum(a[i] × b[i]) for all

function train_svm(train, learning_rate,
                   lambda_param, epochs)
    w ← [0,0]
    b ← 0
    for epoch in epochs:
        for each data point (x,y) in train:
            if (y × (dot_product(w,x) + b) ≥ 1:
                update w = w - learning_
                    rate × 2 ×
                    lambda_param ×
                    w
```

Date __/__/__
Page _____

```
        else:
            update w = w - learning_rate * (2 *
                lambda_param * w - y * x)
            update b = b + learning_rate * y
        end if
    return w, b

function predict(x, w, b):
    return 1 if dot_product(w, x) + b >= 0 else
    -1

filename = "content/iris.csv"
dataset = load_dataset(filename)
train, test = train_test_split(dataset)
w, b
pb = train_sum(train)

plot_sum(train, w, b)

for each point in test:
    prediction = predict(point.features, w, b)
    print(Actual, Predicted)
```

Date 21/4/25
Page

Lab - 6

Random Forest

```
RandomForestClassifier(x_train, y_train, T, max_depth,
                       min_samples_split):
    trees = []
    n_samples, n_features = x_train.shape

    for t = 1 to T:
        bootstrap_idx = random.sample(range
                        (n_samples), n_samples)
        X_bootstrap = x_train[bootstrap_idx]
        Y_bootstrap = y_train[bootstrap_idx]

        tree = DecisionTree(max_depth = max_depth,
               min_samples_split = min_samples_
               split)
        tree.fit(X_bootstrap, Y_bootstrap)
        trees.append(tree)
    return trees.

Predict(x_test):
    predictions = []

    for tree in trees:
        tree_predictions = tree.predict(x_test)
        predictions.append(tree_predictions)

    final_predictions = majority_vote(predictions)
    return final_predictions

majority_vote(predictions):
    return mode(predictions)
```

Date 5/5/25
Page

Lab-7

Boosting Ensemble [Adaboost]

1. Initialize weights for each training sample:
   - Set each sample's weight to 1/N (where N is the no. of samples).

2. For each iteration (t=1 to T):
   a. Train a weak classifier (ex: a decision stump) using the weighted samples.

   b. Calculate the classifier's error:
      - Error = (sum of weights of misclassified samples)/(sum of all sample weights)

   c. Calculate the classifier's importance (alpha):
      - Alpha = 0.5 * log((1-Error)/Error)
      - If Error is high, alpha will be low, meaning less importance for this classifier.

   d. Update the sample weights:
      - increase the weight of misclassified sample.
      - decrease the weight of correctly classified samples
      - Normalize the weight to make sure they can sum to 1

3. After all iterations, combine the weak classifiers:
   - each classifier contributes based on its importance (alpha)

   - Final predictions = sign (sum of all classifiers)

Date __/__/__
Page _____

4. Use the combined classifier to make predictions on new data.

Date __/__/__
Page _____

K-Means Clustering Algorithm

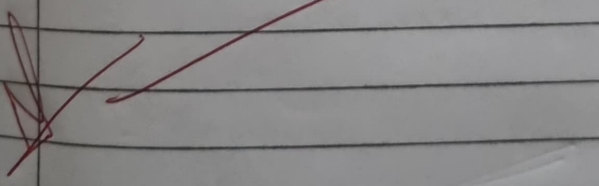
1. Initialize k cluster centroids (randomly selected data points).

2. Repeat until convergence:
   a. Assign step:
      - For each data point, calculate the distance to each centroid.
      - Assign each data point to the closest centroid (cluster).

   b. Update step:
      - For each cluster, compute the mean of all data points assigned to that cluster.
      - Update the centroid to this mean.

3. Repeat the process until the centroids do not change significantly (convergence).

4. The final centroids represent the cluster centers, and each data point belongs to one of the clusters.



Date 12/5/25
Page

logistic Regression

1. Initialize weights w & bias b to small values (e.g. zeros)

2. For i=1 to n-iter:
   a. Compute linear combination: $z = X \cdot w + b$
   b. Apply sigmoid function: $y\text{-pred} = 1/(1+\exp(-z))$
   c. Compute binary cross-entropy loss:
   $$loss = -(1/n) * \Sigma [y * \log(y\text{-pred}) + (1-y) * \log(1-y\text{-pred})]$$
   d. Compute gradients:
   $$dw = (1/n) * X.T * (y\text{-pred} - y)$$
   $$db = (1/n) * \Sigma (y\text{-pred} - y)$$
   e. Update weights and bias:
   $$w = w - \alpha * dw$$
   $$b = b - \alpha * db$$
3. Output the final weights & bias.

Date __/__/__
Page _____

PCA (Principal Component Analysis)

1. Standardize the dataset (mean = 0, std = 1)
   For each feature in X:
   Subtract mean & divide by standard deviation

2. Compute the covariance matrix of the standardized data
   $Cov = (X.T * X) / (n\text{-}samples - 1)$

3. Compute eigenvalues and eigenvectors of the covariance matrix
   eig-vals, eig-vecs = eig (cov)

4. Sort eigenvectors by decreasing eigenvalues (variance)
   sort eig-vecs according to sorted eig-vals in decreasing order.

5. Select the top k eigenvectors
   $W = eig\text{-}vecs[:, :k]$

6. Project the data onto the new subspace
   $X\text{-}reduced = X * W$.

Date __/__/__
Page

Multi-Linear Regression

1. Initialize weight vector w & bias b to zeros

2. For i=1 to n-iter:

   a. Compute predictions:

   $$y\text{-}pred = x \cdot w + b$$

   b. Compute loss (Mean Squared Error):

   $$loss = (1/n) * \Sigma (y\text{-}pred - y)^2$$

   c. Compute gradients:

   $$dw = (2/n) * X.T * (y\text{-}pred - y)$$

   $$db = (2/n) * \Sigma (y\text{-}pred - y)$$

   d. Update weights & bias:

   $$w = w - \alpha * dw$$

   $$b = b - \alpha * db$$

3. Return the final weights w & bias b.