03/03/25

ML Lab - 1

Sales Data Analysis

```
import pandas as pd

df = pd.read_csv('sales-data.csv')
print(df.head())

regional = df.groupby('Region')['Sales'].sum()
print(regional)
```

Stock Market Data Analysis

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

tickers = ["HDFC", "ICICI", "Kotak"]

start_date = "2024-01-01"
end_date = "2024-12-30"
data = yf.download(tickers, start_date, end_date)

for ticker in tickers:
    bank = data[ticker]
    daily_returns = bank['Close'].pct_change()

    bank_returns = bank.copy()
    bank_returns['Daily Returns'] = daily_returns
```

```
plt.figure(12,6)

plt.subplot(2,1,1)
bank['Close'].plot(title=f"{ticker} - Closing Price")

plt.tight_layout()
plt.show()
```

Output

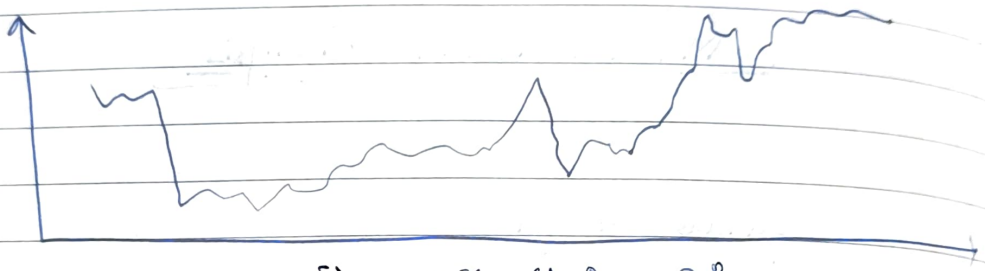


Fig: HDFC Closing Price

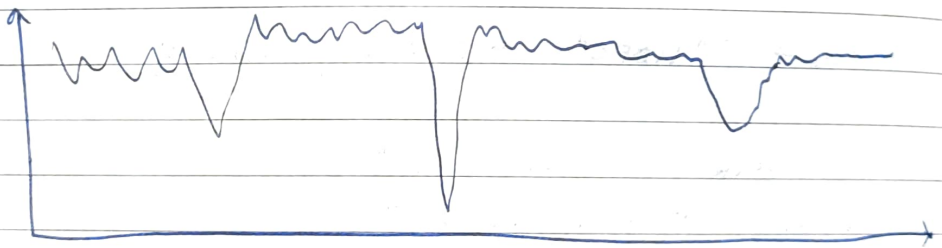


Fig: HDFC Daily Returns

Loading Data from sklearn

```
import pandas as pd
from sklearn.dataset import load_iris
iris = load_iris()

df = pd.DataFrame(iris.data)
df.head()
```

Initializing values directly

```
import pandas as pd

data = {
    'Name' : ['Alice', 'Bob']
    'Age' : [25, 30]
    'City' : ['New York', 'Los Angeles']
}

df = pd.DataFrame(data)
df.head()
```

Diabetes Dataset

```
import pandas as pd
from sklearn.dataset import load_diabetes
diabetes = load_diabetes()

df = pd.DataFrame(diabetes.data, column = iris.feature_names
df.head()
df.tail()
```

ML Lab - 2

Handling "Housing Dataset"

```
import pandas as pd

df = pd.read_csv('housing.csv')

print("All Columns")
print( df.info() )

print(" Descriptive Analysis")
print( df.describe() )

print ("Count of unique labels")

print( df['Ocean Proximity'].value_counts() )

print ("Column with missing value")
missing_values = df.isnull().sum()

column_missing = missing_values[missing_values > 0]

print( columns_with_missing_value )
```

Handling "Diabetes Dataset"

```
diabetes_cols = ['gender', 'polypharmacy', 'race']
```

for col in diabetes_cat:
    diabetes_df[col] = le.fit_transform(adult_df[col])

min-max scaling : scales data between 0 and 1
Use : For algorithms like neural network

Standardization : center data with mean
Use : For algo like SVM and regression

adult_cat = ['home_country', 'education', country', 'race']

10.03

17/3/25

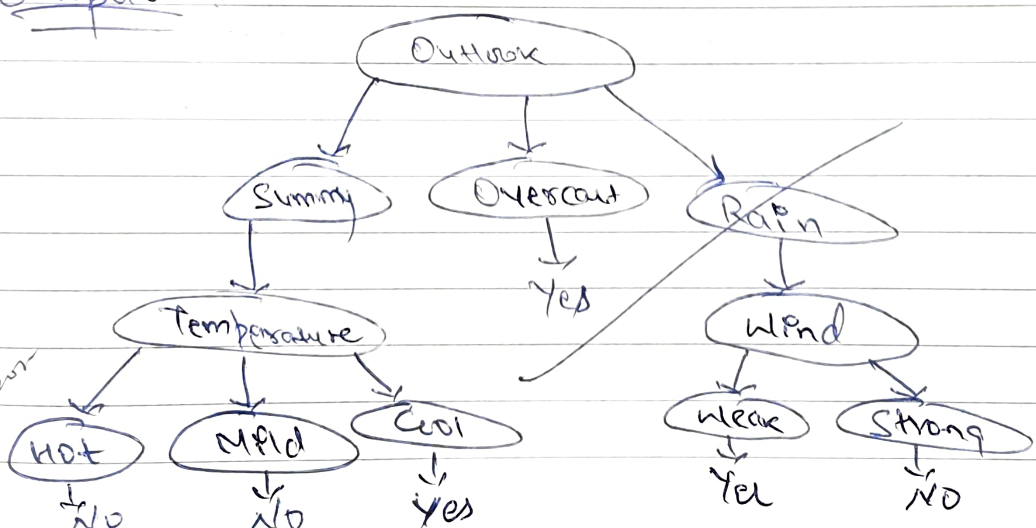
ML Lab - 3

ID3 Algorithm (Play Football or ...

```
import math
import pandas as pd

def entropy(data):
    counts = data.value_counts()
    prob = counts / len(data)
    return -sum(prob * prob.apply
        (lambda p: math.log2(p)
            if p>0 else 0))

def info-gain (data, feature, target):
    total-entropy = entropy (data[target])
    values = data[feature].unique()

    weighted-entropy = 0
    for value in values:
        subset = data[data[feature] == value]
        weighted-entropy += (len(subset)/
                len(data))
            * entropy(subset[target])

    return total-entropy - weighted-entropy
```

```
def best_feature(data, features, target):
    best_gain = -1
    best_feature = None

    for feature in features:
        gain = info_gain(data, feature, target)

        if gain > best_gain:
            best_gain = gain
            best_feature = feature

    return bestfeature


data = pd.read_csv('weather.csv')

target = 'Play'

features = ['Outlook', 'Temperature', 'Humidity', 'Wind']

tree = build_tree(data, features, target)
```

Output :-



17.03.2025

ML Lab-4

Linear Regression

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression

X, y = make_

def fit(X, y):
    n = len(X)

    if n != len(y)
        raise ValueError("length must be same")

    mean_x = sum(X) / n
    mean_y = sum(y) / n

    num = sum((X[i] - mean_x) * (y[i] - mean_y)
              for i in range(n))

    den = sum(X[i] - mean_x) ** 2 for i in
              range(n))

    slope = num / den
    intercept = mean_y - (slope * mean_x)

    return slope, intercept
```

```
def predict(x, slope, intercept):
    return [slope * x + intercept for x in x]

if __name__ == "__main__":
    X = [1, 2, 3, 4 . . . . , 9, 10]
    y = [2, 3, 5, 7, . . . . . , 23, 29]

    s, i = fit(X, y)

    pred = predict([11, 12, 13], s, i)

    print("Slope:", s)
    print("Intercept", i)
    print("Predictions", pred)
```

Output :-

Slope : 2.939393 . . . . . .
Intercept : − 3.26666 . . . . . .
Prediction for [11] : 29.066666 . . . . . .

Pg to O

Linear Multi Regression :-

```
import numpy as np

def fit(x, y):
    n = len(x)

    if n != len(y):
        raise ValueError("length must same")

    x_b = np.c_[np.ones((n,1)), x]

    theta_best = np.linalg.pinv(x_b.T.dot(x_b))
                 .dot(x_b.T).dot(y)

    return theta_best


def predict(x, theta):
    n = len(x)
    x_b = np.c_[np.ones((n,1)), x]
    return x_b.dot(theta)


if "__name__" == "__main__":

    X = np.array([[1,2],
                  [2,3],
                  :
                  [10,29]])
```

```
y = np.array([2, 3, 5, 4 ..... 29])

theta = fit(x, y)

new_data = np.array([[11, 31], ..... [13, 41]])
pred = predict(new_data, theta)

print("Coefficients:", theta)
print("Prediction:", pred)
```

Output:

```
Coefficient : [-5.02708986e-13,
               8.20414851e-14,
               1.00000000e+00]
```

Logistics Regression

```
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def fit(x, y, learning_rate = 0.01, iter = 1000):

    n_samples, n_features = X.shape
    weights = np.zeros(n_features)
    bias = 0
```

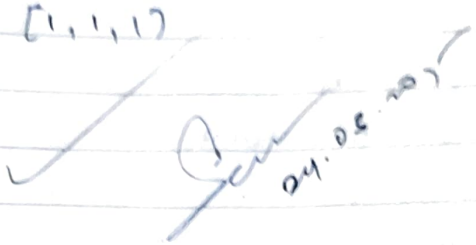
```
for _ in range(iter):

    linear_model = np.dot(X, weights) + bias
    y_predicted = sigmoid(linear_model)

    dw = (1/n_samples) * np.dot(X.T, (y_pre...
    db = (1/n_samples) * np.sum(y_pre - y)

    weights -= learning_rate * dw
    bias -= learning_rate * db

return weights, bias

if "__name__" == "__main__":

    X = np.array([[0.1, 0.2], ..... [1.0, 0.1]])

    y = np.array([.....])

    weights, bias = fit(X, y)

    pred = predict(new_data, weights, bias)

    print(weights)
    print(bias)
    print(pred)
```

Output

Weights : [1.04209544 0.20349434]

Bias : -0.08474891.....

Prediction : [1, 1, 1]



ML Lab - 5

① KNN Algorithm :-

```
import numpy as np
from collections import Counter
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

data = load_iris()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split()

model = KNN(k=3)
model.fit(X_train, y_train)
```

```
predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
```

Sample Output

Accuracy = ~~0.0~~ 0.89

Support Vector Machine :=

```
import numpy as np

class SVM:
    def __init__(self, learning_rate=0.001):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)
        self.w = np.zeros(n_features)
        self.b = 0

    def predict(self, X):
        approx = np.dot(X, self.w) - self.b
        return np.where(approx <= 0, -1, 1)
```

ML Lab - 6

Random forest

```
import numpy as np
from collections import Counter
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

class RandomForest:
    def __init__(self, n_estimators, sample_size=None):
        self.n_estimators = n_estimators
        self.sample_size = sample_size
        self.trees = []

    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree
                               in self.trees])
        return np.array([Counter(tree_preds[:, i])

data = load_iris()
X = data.data[data.target != 2]
y = data.target[data.target != 2]

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

rf = RandomForest(n_estimators=5)
```

```
df.fit(X_train, y_train)
predictions = df.predict(X_test)

print("Accuracy:", accuracy_score)
```

Sample Output :-

Accuracy - 0.88

ML lab - 7

Boosting Ensemble Learning

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

class AdaBoost:
    def __init__(self, n_estimators = 5):
        self.n_estimators = n_estimators
        self.alphas = []
        self.stumps = []

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y = np.where(y == 0, -1, 1)

        for _ in range(self.n_estimators):
            stump = self.build_stump(X, y, w)
            preds = stump['pred']
            err = np.sum(w * (pred != y))
            if err == 0:
                break

    def predict(self, X):
        clf_preds = [alpha * np.where(clf['polarity']
                    * X[:, clf['feat']] < clf['split'], -1, 1)
```

ML lab - 8

K-means Clustering

```
import numpy as np
import pandas as pd

def Kmeans (x, K=3, max_iter = 100):
    centroids = X [np.random.choice (len(x), K)]
    for _ in range (max_iter):
        dis = np.linalg.norm (X[:, np.newaxis]
              - cent, axis=2)

    labels = np.argmin (distances, axis=1)
    new_cent = np.array ([X[label == i].mean
               for i in range (K)])

    if np.allclose (cent, new_cent):
        break
    cent = new_cent

    return cent

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

X = df [[f1, f2]].values

labels, centroids = Kmeans (x, K=3)
```

ML lab -9

Principal Component Analysis

```
import numpy as np
import pandas as pd

def pca (x, n):

    X_mean = x - np.mean (x, axis =0)
    cov_mat = np.cov (x-mean, rowvar = F)
    eigen_val, eigen_vec = np.linalg.eigh

    idxs = np.argsort (eigen_val) [::-1]

    eigen_vec = eigen_vec [:, idxs]
    eigen_val = eigen_val [idxs]

    return np.dot (x-mean, eigen_vec)

df = pd.read_csv ("data.csv")
x = df [['y1', 'y2', 'y3']].values
X_pca = pca (x, n)
```