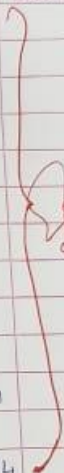
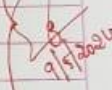

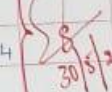


INDEX

Name POORVIKA S.K Std C

Roll No. Subject ML - LAB School/College

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

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
01.	31/3/24	Import and Export data	1-2	  9/5/24
02.	4/4/24	Framing the problem	3-6	
03.	4/4/24	Framing the problem and doing different operation	3-6	
04.	17/4/24	Simple linear and Multi Linear Regression	7-9	
05.	25/4/24	Decision Tree	10-11	
06.	9/5/24	Logistic Regression, KNN	12-14	  30/5/24
07.	23/5/24	1. Support vector machine 2. KMean Algorithm 3. principal component analysis	15-20	
08.	30/5/24	Implement RandomForest ensemble and Boosting ensemble method	21-24	

LAB 01

Build Support vector machine model for a given dataset.

import the dataset & libraries

Load the iris dataset

```
iris = load_iris()
```

```
data = pd.DataFrame(data = iris.data, columns = iris.feature_names)
```

```
data['target'] = iris.target
```

```
print(data.head())
```

	sepal length	width	petal length	width	target
0	5.1	3.5	1.4	0.2	0
1	5.1	3.5	1.4	0.2	0
2	4.9	3.0	1.4	0.2	0
3	4.7	3.2	1.3	0.2	0
4	4.6	3.1	1.6	0.2	0
5	5.0	3.6	1.4	0.2	0

plotting the data

```
H. scatter (data['sepal length (cm)'], data['sepal width (cm)'], c = data['target'])
```

```
H. show()
```

To Exporting data

df.to_csv("path/new-name.csv")

2 Discover and visualize the data to gain

start_train_set.shape start_test_set.shape

start_test_set.reset_index().to_feather
(fname = 'data/ot/start_test_set.f')

housing = start_train_set.copy(); housing

housing.plot(kind = 'scatter', x = 'longitude',
y = 'latitude')
plt.show()

3. Prepare the data for machine learning at

housing = start_train_set.drop("median
house_value", axis = 1)

housing_labels = start_train_set["median
value"].copy()

housing.shape, housing_labels.shape

Data cleaning

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy = 'median')

housing_num = housing.drop("ocean_proximity")
imputer.fit(housing_num)

imputer.statistics

housing_num.median().values

Date: / /
Page: 1

Build KNN Model

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

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

print(data.head())

plt.scatter(data['sepal length'], data['sepal width'], c=data['species'].astype('category').cat.codes)
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.title('Scatter plot of sepal length vs sepal width')
plt.show()

X = data.drop('species', axis=1)
y = data['species']

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

knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

```

petal length (cm) = 5.45
gini = 0.947
Samples = 120
value = [40, 41, 39]
class = versicolour

gini = 0.0	petal length (cm) <= 4.75
samples = 40	gini = 0.5
value = [40, 0, 0]	samples = 80
class = setosa	value = [0, 42, 38]
	class = versicolour

petal width (cm) <= 1.65
gini = 0.053
Samples = 37
value = [0, 36, 1]
class = versicolour

gini = 0.0	gini = 0.0
samples = 36	samples = 1
value = [0, 36, 0]	value = [0, 0, 1]
class = versicolour	class = virginica

Sp.1
25/4/24

Multi-linear Regression

Date: 1/1
Page: 3

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_boston

data_url = "https://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep='\\s+', skiprows=22, header=None)
```

```
x = np.hstack([raw_df.values[0:2, :], raw_df.values[11:12, :2]])
y = raw_df.values[0:2, 2]
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.4,
                                                    random_state=1)
```

```
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)
```

```
print('coefficients:', reg.coef_)
print('variance score: %f' % r2_score(y_test, reg.predict(x_test)))
```

```
plt.style.use('fivethirtyeight')
```

```
plt.scatter(reg.predict(x_train), y_train)
```


Step - 1 :

Examining the problem and looking at the big picture

Step - 2 :

Get the data

1. Downloading the data

```
import os
import tarfile
import urllib

Download_Root = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
Housing_Path = os.path.join("data", "01")
Housing_URL = Download_Root + "data.csv..."

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")
    fetch_housing_data()

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


Date: / /
Page: 17

1013_05
Decision Tree

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

iris_data = load_iris()

iris_df = pd.DataFrame(data=iris_data.data,
                        column=iris_data.feature_names)
iris_df['target'] = iris_data.target

print(iris_df.head())

x = iris_data.data
y = iris_data.target

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=1)

dt_classifier = DecisionTreeClassifier()

dt_classifier.fit(x_train, y_train)

plt.figure(figsize=(12, 8))
tree.plot_tree(dt_classifier, feature_names=iris_data.feature_names,
               class_names=iris_data.target_names,
               filled=True)
plt.show()

```

housing = pd.DataFrame(data, x, index = housing_num.index, columns = housing_rain.columns)
housing = housing.head()

4. Select and train a model

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(x = housing_prepared, y = housing_labels)
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)
print("Predictions:", lin_reg.predict(some_data_prepared))
print("Labels:", some_labels.tolist())
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
```

5. Fine-Tune Your Model

```
from sklearn.model_selection import GridSearchCV
```

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

- * Write a python program to import and Export data using Pandas library functions.

Importing data

```
import pandas as pd
# Read the csv file
airbnb_data = pd.read_csv("data/listings - austin.csv")
# view the first 5 rows
airbnb_data.head()
```

OUTPUT:-

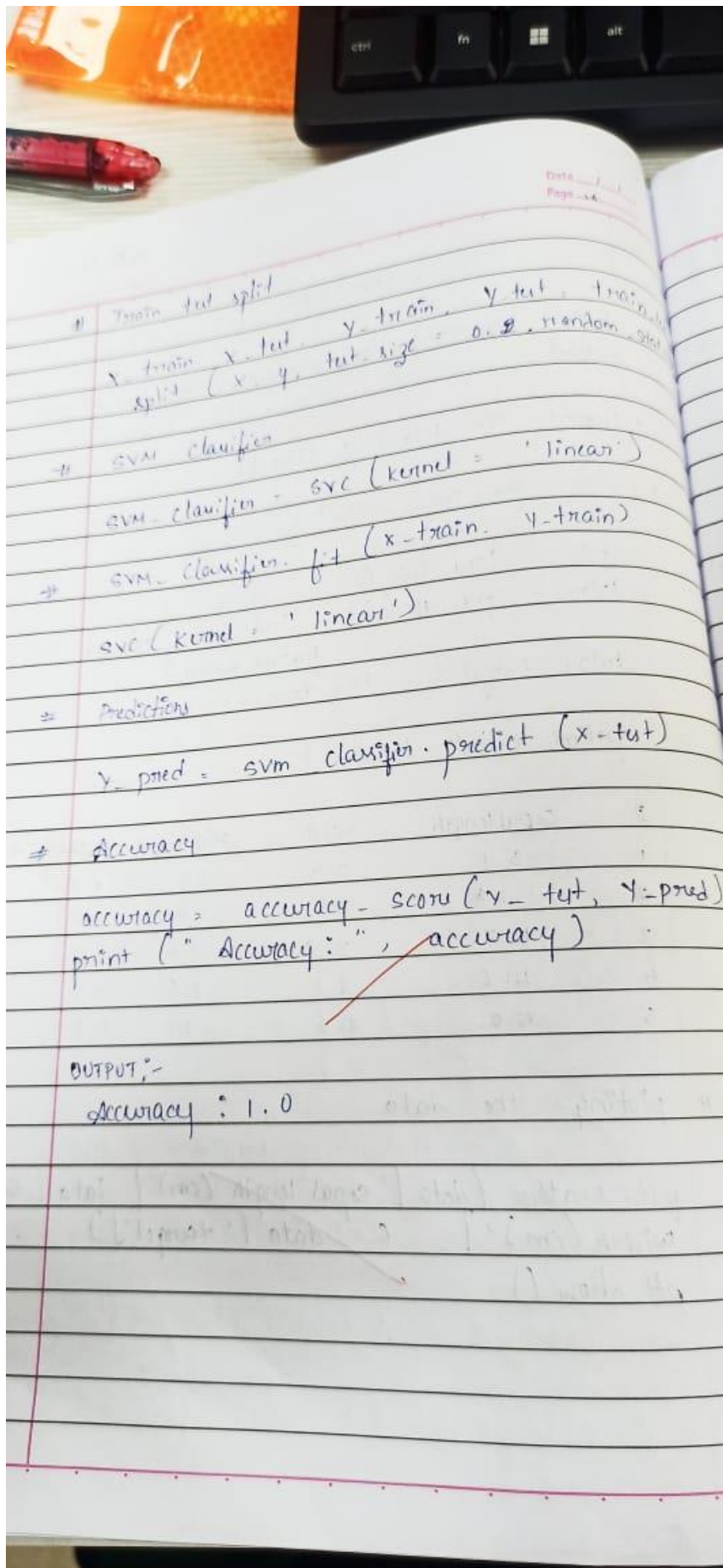
	id	name	host_id	host_name	latitude	price
0	2265	Zen-East	2466	Paddy	30.2752	179
1	5245	Eco-friendly	2465	Paddy	30.2715	114
2	5456	Walk to 6 th	8028	Sylvia	30.2607	108
3	5769	NW Austin Room	8186	Todd	30.45697	39
4	6413	Gem of a studio	13879	Todd	30.24895	109

Reading data from URL

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
```

```
col_names = ["sepal-length-in-cm", "sepal-width-in-cm", "petal-length-in-cm", "petal-width-in-cm"]
```

```
iris_data = pd.read_csv(url, names = col_names)
iris_data.head()
```

Date
Page 11

$y_{pred} = \text{knn-classifier.predict}(x_{test})$
 $\text{accuracy} = \frac{\text{score}(y_{test}, y_{pred})}{\text{accuracy}}$
 $\text{print}(\text{accuracy})$

Output:-

	Sepal length	P width	Petal length	P width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.0	1.4	0.2

accuracy : 1.0

only

Simple Linear Regression

```
def estimate_coef(x, y):
    n = np.size(x)
```

```
    m_x = np.mean(x)
```

```
    m_y = np.mean(y)
```

```
    SS_xy = np.sum(y*x) - n*m_y*m_x
```

```
    SS_xx = np.sum(x*x) - n*m_x*m_x
```

```
    b_1 = SS_xy / SS_xx
```

```
    b_0 = m_y - b_1*m_x
```

```
    return (b_0, b_1)
```

```
def plot_regression_line(x, y, b):
```

```
    plt.scatter(x, y, color="m",
                marker="o", s=30)
```

```
    y_pred = b[0] + b[1] * x
```

```
    plt.plot(x, y_pred, color="g")
```

```
    plt.xlabel('x')
```

```
    plt.ylabel('y')
```

```
def main():
```

```
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
    y = np.array([1, 3, 4, 5, 7, 8])
```

```
    b = estimate_coef(x, y)
```

```
    print("Estimated coefficients: \nb=0  
b1 = {}".format(b))
```


Date: / /
Page: 6

```

{ 'bootstrap': [False], 'n_estimators': [3, 10]
  'max_features': [2, 3, 4] }

forest = RandomForestRegressor()
grid_search = GridSearchCV(estimator=forest,
                             param_grid=param_grid,
                             scoring='neg_mean_squared_error',
                             cv=5,
                             return_train_score=True,
                             n_jobs=1)
grid_search.fit(X=train_data, y=train_labels)

grid_search.best_params_
grid_search.best_estimator_
cv_results = grid_search.cv_results_

for mean_score, params in zip(cv_results['mean_test_score'],
                               cv_results['params']):
    print(np.sqrt(-mean_score), params)
  
```

14/2024

```

Build k means algorithm to cluster a set
of data stored in a csv file

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

data = pd.read_csv('iris_data.csv')
print(data.head())
print(data.isnull().sum())

features = data.drop('species', axis=1)
target = data['species']

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
print(features_scaled[:5])

sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=1)
    kmeans.fit(features_scaled)
    sse.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), sse, marker='o')
plt.xlabel('Number of clusters')

```

Date: / /
Page: 26

```

z_pca = z @ pca_component
z_pca.rename({'pc1': 'pcas', 'pc2': 'pcas',
              axis = 1, inplace = True})
print(z_pca)

from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(z)
x_pca = pca.transform(z)

df_pca1 = pd.DataFrame(x_pca, columns =
                        ['pc{}'.format(i+1)
                        for i in range(n_components)])
print(df_pca1)

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')
plt.show()

```

22/5/2024

Implement Dimensionality reduction using Principal component analysis (PCA) method

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

```
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer(as_frame=True)
df = cancer.frame
```

```
print('Original dataframe shape:', df.shape)
```

```
X = df[cancer.feature_names]
```

```
print('Input Dataframe shape:', X.shape)
```

```
X_mean = X.mean()
```

```
X_std = X.std()
```

```
Z = (X - X_mean) / X_std
```

```
C = Z.cov()
```

```
eigenvalues, eigenvectors = np.linalg.eig(C)
```

```
print('Eigen values:', eigenvalues)
```

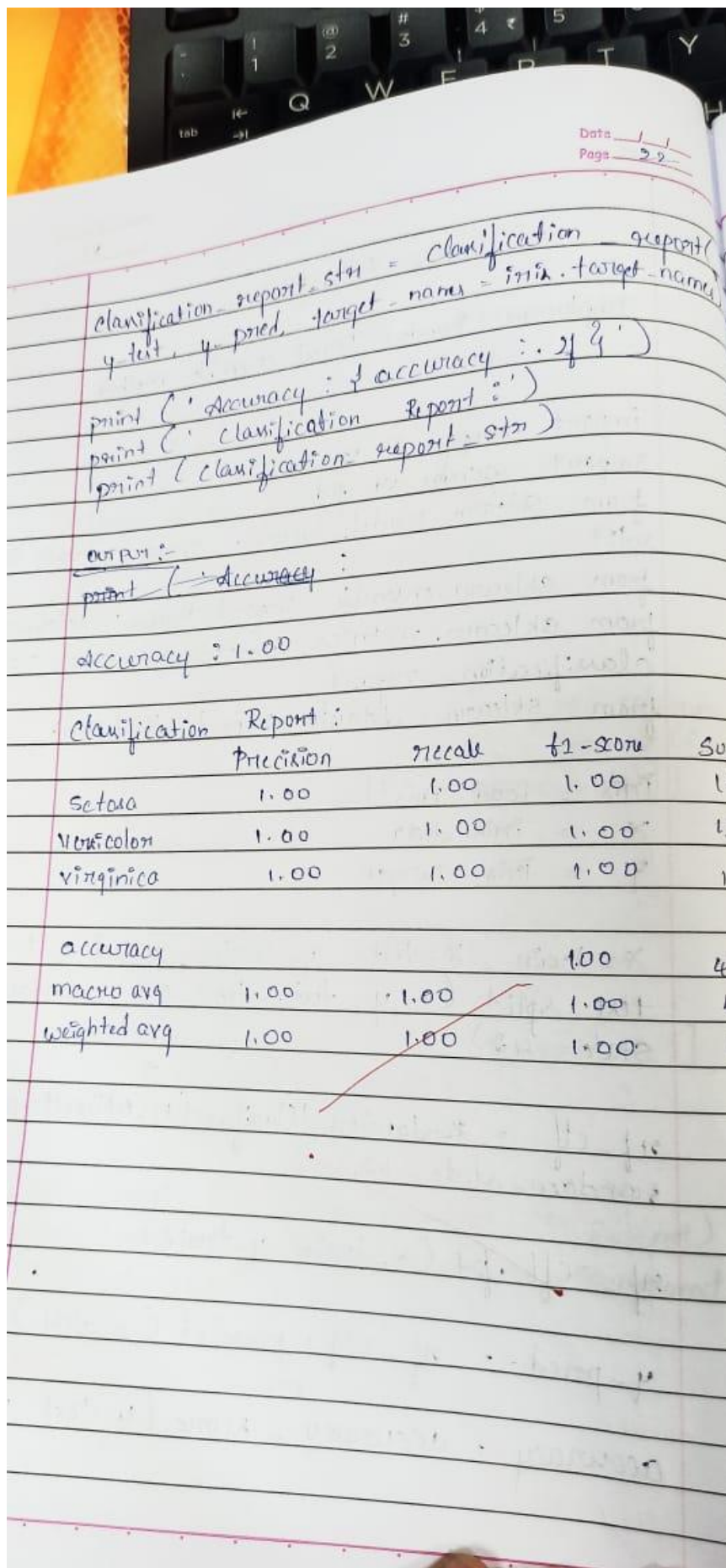
```
print('Eigen values shape:', eigenvalues.shape)
```

```
print('Eigen vector shape:', eigenvectors.shape)
```

```
idx = eigenvalues.argsort()[::-1]
```

```
eigenvalues = eigenvalues[idx]
```

```
eigenvectors = eigenvectors[:, idx]
```



reg.predict(x_train) - y_train
color = "green", s=10
label = 'Train data'

plt.scatter(reg.predict(x_test)
reg.predict(x_test) - y_test
color = "blue", s=10
label = 'Test data'

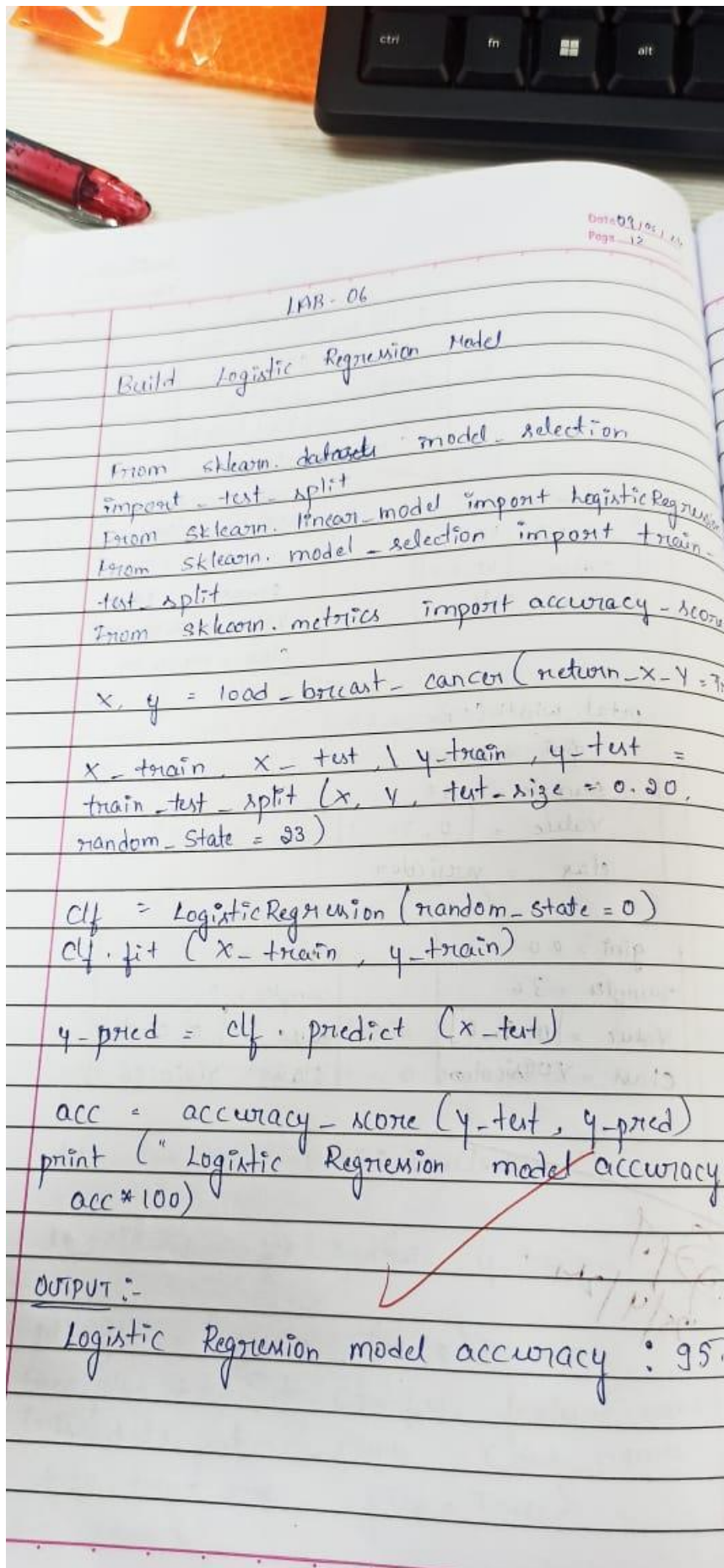
plt.hlines(y=0, xmin=0, xmax=50,
linewidth=2)

plt.legend(loc='upper right')

plt.title("Residual errors")

plt.show()

Spf.



Date / /
 Page 44

```

plt.ylabel('sum of squared distances (SSE)')
plt.show()

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(features_scaled)

label = kmeans.labels

plt.figure(figsize=(10,5))
sns.scatterplot(x=features_scaled[:,0], y=
features_scaled[:,1], hue=label, palette='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('original species visualization')
plt.show

```

LAR - OR

Implement Random Forest ensemble method

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

```
iris = load_iris()
```

```
X = iris.data
```

```
y = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

```
rf_clf = RandomForestClassifier(n_estimators = 100, random_state = 42)
```

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

```
y_pred = rf_clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
```


LAB - OR

Implement Random Forest ensemble method

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

```
iris = load_iris()
```

```
X = iris.data
```

```
y = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

```
rf_clf = RandomForestClassifier(n_estimators = 100, random_state = 42)
```

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

```
y_pred = rf_clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
```

LAR - OR

Implement Random Forest ensemble method

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

```
iris = load_iris()
```

```
X = iris.data
```

```
y = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

```
rf_clf = RandomForestClassifier(n_estimators = 100, random_state = 42)
```

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

```
y_pred = rf_clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
```

