IRIS FLOWER CLASSIFICATION

IMPORTING LIBRARIES

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from scipy.stats import chi2_contingency
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoos
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows',None)
```

LOADING DATA

In [ ]:

```
data = pd.read_csv('/kaggle/input/iris-flower-dataset/IRIS.csv')
```

In [ ]:

```
data.sample(5)
```

ANALYZING DATA

In [ ]:

```
data.shape
```

In [ ]:

```
def summary(df):
    sum = pd.DataFrame(df.dtypes, columns=['dtypes'])
    sum['missing#'] = df.isna().sum().values
    sum['missing%'] = (df.isna().sum().values*100)/len(df)
    sum['uniques'] = df.nunique().values
    sum['count'] = df.count().values
    #sum['skew'] = df.skew().values
    desc = pd.DataFrame(df.describe().T)
    sum['min'] = desc['min']
    sum['max'] = desc['max']
    sum['mean'] = desc['mean']
    return sum

summary(data).style.background_gradient(cmap='twilight_shifted_r')
```

In [ ]:

```
data.isnull().sum()
```

> - No null values

In [ ]:
```
sns.countplot(x =data['species'])
```

In [ ]:
```
data['species'].value_counts()
```

> - We can see its a perfectly balanced data

DATA PREPROCESSING

In [ ]:
```
le = LabelEncoder()
data['species'] = le.fit_transform(data['species'])
```

In [ ]:
```
y = data['species']
x = data.drop(columns=['species'])
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=
```

APPLYING MODELS

In [ ]:
```
dtree = DecisionTreeClassifier()
rf = RandomForestClassifier()
gb = GradientBoostingClassifier()
ada = AdaBoostClassifier()
knn = KNeighborsClassifier()
lr = LogisticRegression()
```

In [ ]:
```
dtree.fit(x_train, y_train)
rf.fit(x_train, y_train)
gb.fit(x_train, y_train)
ada.fit(x_train, y_train)
knn.fit(x_train, y_train)
lr.fit(x_train, y_train)
```

In [ ]:
```
pdtreetr = dtree.predict(x_train)
pdtreete = dtree.predict(x_test)

prftr = rf.predict(x_train)
prfte = rf.predict(x_test)

pgbtr = gb.predict(x_train)
pgbte = gb.predict(x_test)

padatr = ada.predict(x_train)
padate = ada.predict(x_test)

pknntr = knn.predict(x_train)
pknnte = knn.predict(x_test)
```

```
plrtr = lr.predict(x_train)
plrte = lr.predict(x_test)
```

In [ ]:

```
def acc_report(actual,predicted):
    acc_score=accuracy_score(actual,predicted)
    cm_matrix=confusion_matrix(actual,predicted)
    class_rep=classification_report(actual,predicted)
    print('the accuracy of tha model is ',acc_score)
    print(cm_matrix)
    print(class_rep)
```

In [ ]:

```
print(acc_report(y_train, pdtreetr))
print(acc_report(y_test, pdtreete))
```

In [ ]:

```
print(acc_report(y_train, prftr))
print(acc_report(y_test, prfte))
```

In [ ]:

```
print(acc_report(y_train, pgbtr))
print(acc_report(y_test, pgbte))
```

In [ ]:

```
print(acc_report(y_train, padatr))
print(acc_report(y_test, padate))
```

In [ ]:

```
print(acc_report(y_train, pknntr))
print(acc_report(y_test, pknnte))
```

In [ ]:

```
print(acc_report(y_train, plrtr))
print(acc_report(y_test, plrte))
```

DEPLOYMENT

In [ ]:

```
!pip install -q gradio
```

In [ ]:

```
import gradio as gr

def flower(sepal_length, sepal_width, petal_length, petal_width):
    test = np.array([[sepal_length, sepal_width, petal_length, petal_width]])
    result = lr.predict(test)
    results = ['setosa', 'versicolor', 'virginica']
    if result==0:
        return results[0]
    elif result==1:
        return results[1]
    else:
        return results[2]


demo = gr.Interface(
    fn=flower,
    inputs=['number','number', 'number', 'number'],
    outputs= ['text']
)
demo.launch()
```