1. **Exploratory Data Analysis using Pandas & Seaborn**

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

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

print(df.head())
print(df.info())
print(df.describe())

sns.histplot(df['age'])
plt.show()

sns.boxplot(x=df['salary'])
plt.show()

sns.heatmap(df.corr(), annot=True)
plt.show()
```

2. **Data Cleaning and Preprocessing on Real Dataset**

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

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

df = df.drop_duplicates()
df = df.fillna(df.mean())

scaler = StandardScaler()
df[['age','salary']] = scaler.fit_transform(df[['age','salary']])

print(df.head())
```

3. **Implementing Linear and Logistic Regression**

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = df[['experience']]
y = df['salary']

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

model = LinearRegression()
model.fit(X_train, y_train)

print(model.predict(X_test))
```

```
//logestic regression code
from sklearn.linear_model import LogisticRegression

X = df[['age','salary']]
y = df['purchased']

model = LogisticRegression()
model.fit(X, y)

print(model.predict([[30,50000]]))
```

4.Decision Tree and Random Forest Classification

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X = df.drop('target', axis=1)
y = df['target']

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

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)

print(dt.score(X_test, y_test))
print(rf.score(X_test, y_test))
```

5. K-Means Clustering and Visualization

```
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt

X = df[['age','salary']]

kmeans = KMeans(n_clusters=3)
df['cluster'] = kmeans.fit_predict(X)

sns.scatterplot(x='age', y='salary', hue='cluster', data=df)
plt.show()
```

6. SVM and Hyperparameter Tuning

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

model = svm.SVC()

params = {
   'C':[1,10],
   'kernel':['linear','rbf']
}

grid = GridSearchCV(model, params)
grid.fit(X, y)

print(grid.best_params_)
```

7. Principal Component Analysis (PCA)

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled = scaler.fit_transform(df)

pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled)

print(pca.explained_variance_ratio_)
```

8. Building a complete ML pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([
   ('scaling', StandardScaler()),
   ('model', LogisticRegression())
])

pipe.fit(X, y)
print(pipe.predict(X[:5]))
```

9. Deploying an ML model using Flask

```
from flask import Flask, request
import pickle

app = Flask(__name__)
model = pickle.load(open("model.pkl","rb"))

@app.route('/predict', methods=['POST'])
def predict():
   data = request.json
   output = model.predict([data['values']])
   return str(output)

app.run()
```

10 Mini Project: ML application on a real-world dataset

✅1. Import Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import pickle
```

✅2. Load Dataset

```
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df.head())
```

✅3. Split Data

```
X = df.drop('target', axis=1)
y = df['target']

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

✅4. Feature Scaling

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

✅5. Model Training (Logistic Regression)

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

✅6. Model Evaluation

```
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

✅7. Save Model for Deployment

```
pickle.dump(model, open("iris_model.pkl", "wb"))
pickle.dump(scaler, open("scaler.pkl", "wb"))
```

✅8. Load Model (to test saved model)

```
loaded_model = pickle.load(open("iris_model.pkl", "rb"))
loaded_scaler = pickle.load(open("scaler.pkl", "rb"))

sample = [[5.1, 3.5, 1.4, 0.2]]   # Sample input

sample_scaled = loaded_scaler.transform(sample)
prediction = loaded_model.predict(sample_scaled)

print("Predicted Class:", prediction)
```

✅9. Flask API Code (Optional for Deployment)

Create: app.py

```
from flask import Flask, request, jsonify
import pickle

app = Flask(__name__)

model = pickle.load(open("iris_model.pkl", "rb"))
scaler = pickle.load(open("scaler.pkl", "rb"))

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json['data']
    data_scaled = scaler.transform([data])
    output = model.predict(data_scaled)[0]
    return jsonify({"prediction": int(output)})

app.run()
```

☐ 10. Body JSON Example (Postman or JavaScript)

```
{
  "data": [5.5, 2.3, 4.0, 1.3]
}
```