```
import io, warnings
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
from google.colab import files
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from pandas.api.types import is_numeric_dtype
```

```
warnings.filterwarnings("ignore")
sns.set(style="whitegrid")

print("📁 Upload iris.csv")
uploaded = files.upload()
fname = list(uploaded.keys())[0]
df = pd.read_csv(io.BytesIO(uploaded[fname]))
```

```
📁 Upload iris.csv
Choose Files iris.csv
iris.csv(text/csv) - 3861 bytes, last modified: 11/8/2025 - 100% done
Saving iris.csv to iris (1).csv
```

```
candidates = [c for c in df.columns if c.lower() in ('species','class','targ
if candidates:
    target_col = candidates[0]
else:
    # if last column is non-numeric, treat as target; otherwise last column
    if not is_numeric_dtype(df.iloc[:, -1]):
        target_col = df.columns[-1]
    else:
        target_col = df.columns[-1]

X = df.drop(columns=[target_col]).values
y = df[target_col].values
```

```
le = LabelEncoder()
y_enc = le.fit_transform(y)
X_scaled = StandardScaler().fit_transform(X)

# --- train/test split ---
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_enc, test_si
                                                    random_state=42, stratif

# --- one-hot for regressor (MSE experiment) ---
n_classes = len(np.unique(y_train))
Y_train_oh = np.eye(n_classes)[y_train]
Y_test_oh  = np.eye(n_classes)[y_test]
```

```
hidden = (5,)
max_iter = 300
```

```
lr = 0.01

configs = [
    ("MLP-Clf (sigmoid, SGD)", MLPClassifier(hidden_layer_sizes=hidden, acti
                                             solver='sgd', learning_rate_init=
    ("MLP-Clf (tanh, SGD)", MLPClassifier(hidden_layer_sizes=hidden, activat
                                          solver='sgd', learning_rate_init=lr
    ("MLP-Clf (relu, Adam)", MLPClassifier(hidden_layer_sizes=hidden, activa
                                           solver='adam', learning_rate_init=
    ("MLP-Reg (MSE, relu, Adam)", MLPRegressor(hidden_layer_sizes=hidden, ac
                                               solver='adam', learning_rate_i
]

results = []
loss_curves = {}
weights = {}

for name, model in configs:
    if isinstance(model, MLPRegressor):
        model.fit(X_train, Y_train_oh)
        y_score = model.predict(X_test)
        y_pred = np.argmax(y_score, axis=1)
        loss = getattr(model, "loss_curve_", None)
        weights[name] = np.hstack([coef.ravel() for coef in model.coefs_])
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        loss = getattr(model, "loss_curve_", None)
        try:
            weights[name] = np.hstack([coef.ravel() for coef in model.coefs_
        except Exception:
            weights[name] = np.array([])

    acc  = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="macro", zero_division=0)
    rec  = recall_score(y_test, y_pred, average="macro", zero_division=0)
    f1   = f1_score(y_test, y_pred, average="macro", zero_division=0)

    results.append({"Model": name, "Accuracy": acc, "Precision": prec, "Reca
    loss_curves[name] = loss
```

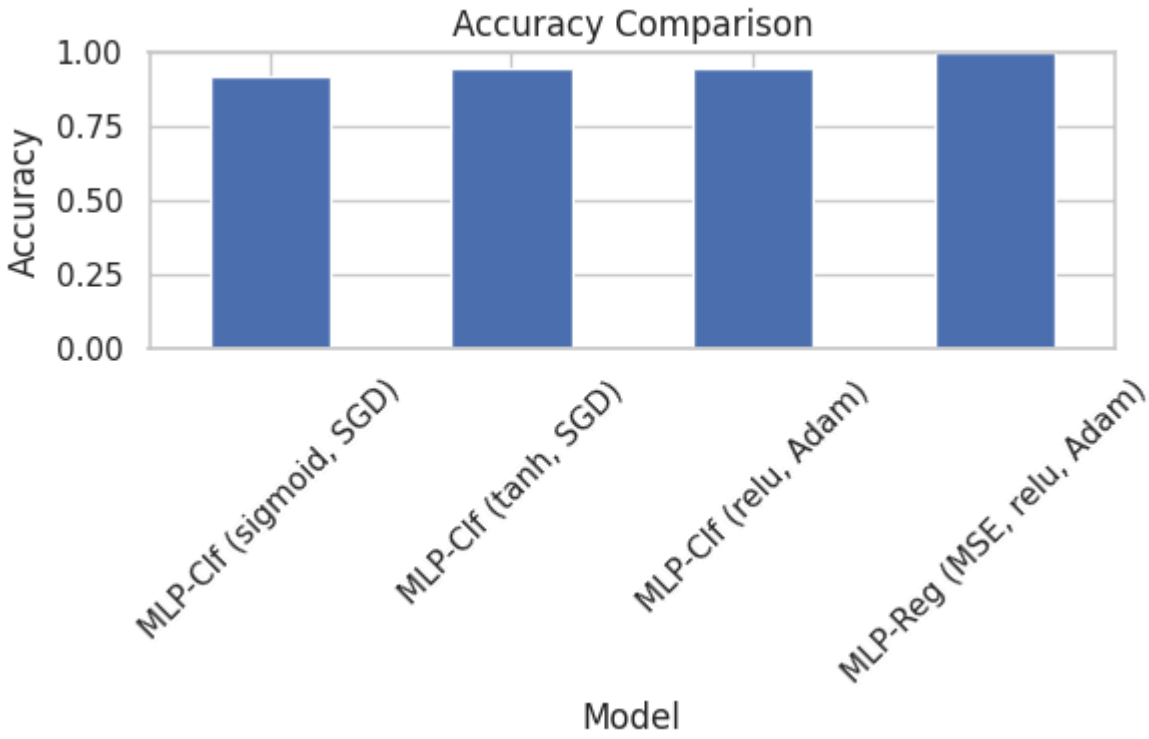
```
res_df = pd.DataFrame(results).set_index("Model")
pd.set_option("display.precision", 4)
print("\n=== Performance Metrics ===")
display(res_df)
```

```
=== Performance Metrics ===
```

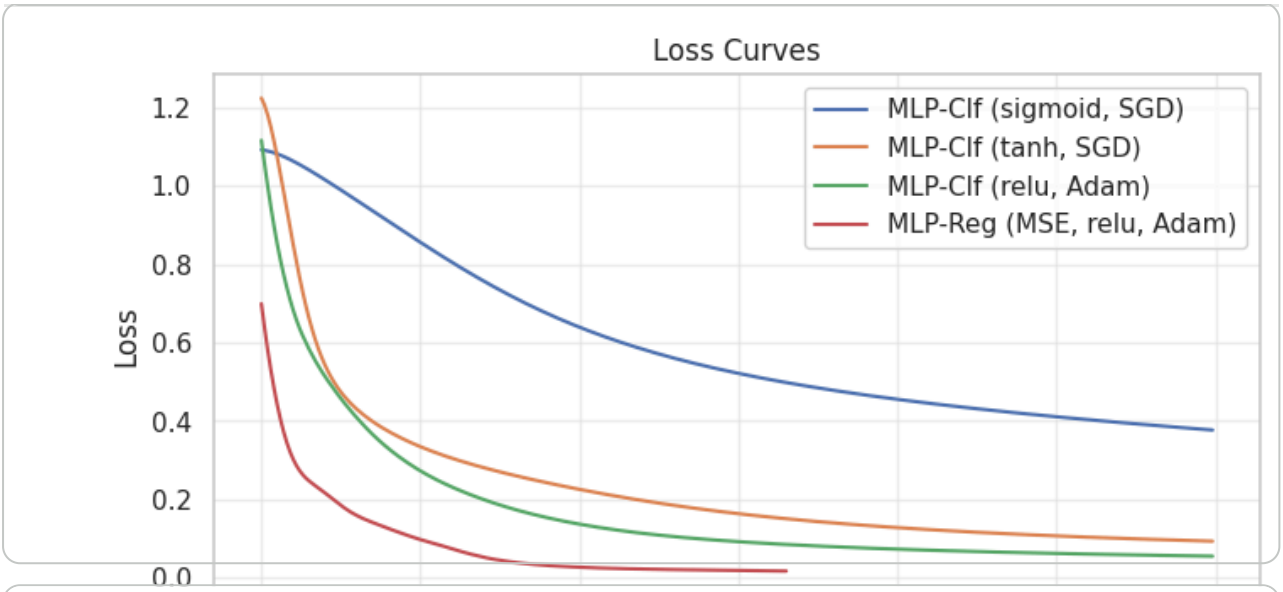
Model	Accuracy	Precision	Recall	F1-Score
MLP-Clf (sigmoid, SGD)	0.9211	0.9246	0.9231	0.9230
MLP-Clf (tanh, SGD)	0.9474	0.9487	0.9487	0.9487
MLP-Clf (relu, Adam)	0.9474	0.9487	0.9487	0.9487
MLP-Reg (MSE, relu, Adam)	1.0000	1.0000	1.0000	1.0000

Next steps: Generate code with `res_df` New interactive sheet

```
plt.figure(figsize=(6,4))
res_df["Accuracy"].plot(kind="bar", ylim=(0,1), rot=45)
plt.title("Accuracy Comparison"); plt.ylabel("Accuracy"); plt.tight_layout()
```




```
plt.figure(figsize=(8,4))
for name, lc in loss_curves.items():
    if lc is not None:
        plt.plot(lc, label=name)
plt.xlabel("Iteration"); plt.ylabel("Loss"); plt.title("Loss Curves"); plt.l
```




```
plt.figure(figsize=(8,4))
for name, w in weights.items():
    if w.size > 0:
        sns.kdeplot(w, label=name, fill=False)
plt.title("Weight Distributions"); plt.legend(); plt.show()
```