Name: Arham Sharif

Seat No.: EB21102022

Section: B

Subject: Data Warehouse & Data Mining

URL: https://www.kaggle.com/arhamsharif

DATA VISUALIZATION

EDA ON STUDENT PERFORMANCE DATASET

https://www.kaggle.com/code/arhamsharif/eda-on-student-performance-dataset

CODE

1. Import Libraries

```python
import pandas as pd
import matplotlib.pyplot as plt
```

2. Load Dataset from Raw URL

```python
url = "https://raw.githubusercontent.com/selva86/datasets/master/College.csv"
df = pd.read_csv(url)
```

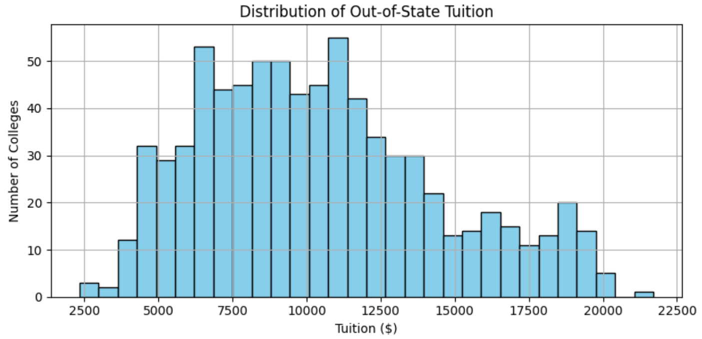
3. Basic Info

```python
print("Shape:", df.shape)
print("Columns:", df.columns.tolist())
print(df.head())
```

```
Shape: (777, 18)
Columns: ['Private', 'Apps', 'Accept', 'Enroll', 'Top10perc', 'Top25perc', 'F.Undergrad', 'P.Undergrad', 'Outstate', 'Room.Board', 'Books', 'Personal', 'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni', 'Expend', 'Grad.Rate']
  Private  Apps  Accept  Enroll  Top10perc  Top25perc  F.Undergrad  \
0     Yes  1660    1232     721         23         52         2885
1     Yes  2186    1924     512         16         29         2683
2     Yes  1428    1097     336         22         50         1036
3     Yes   417     349     137         60         89          510
4     Yes   193     146      55         16         44          249

   P.Undergrad  Outstate  Room.Board  Books  Personal  PhD  Terminal  \
0          537      7440        3300    450      2200   70        78
1         1227     12280        6450    750      1500   29        30
2           99     11250        3750    400      1165   53        66
3           63     12960        5450    450       875   92        97
4          869      7560        4120    800      1500   76        72

   S.F.Ratio  perc.alumni  Expend  Grad.Rate
0       18.1           12    7041         60
1       12.2           16   10527         56
2       12.9           30    8735         54
3        7.7           37   19016         59
4       11.9            2   10922         15
```

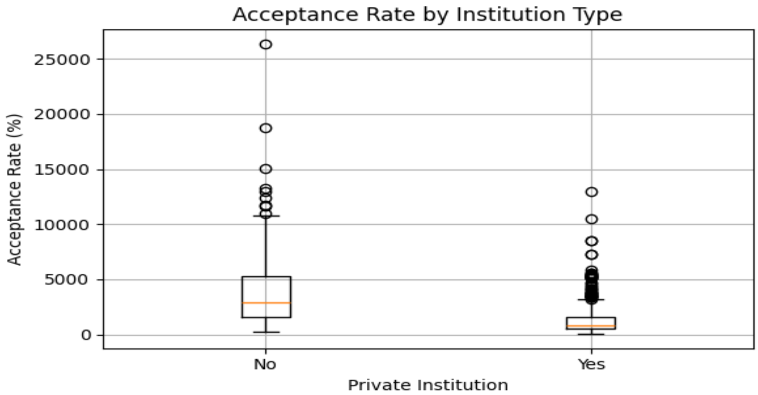
4. Histogram: Outstate Tuition

```python
plt.figure(figsize=(8, 4))
plt.hist(df["Outstate"], bins=30, color="skyblue", edgecolor="black")
plt.title("Distribution of Out-of-State Tuition")
plt.xlabel("Tuition ($)")
plt.ylabel("Number of Colleges")
plt.grid(True)
plt.tight_layout()
plt.show()
```



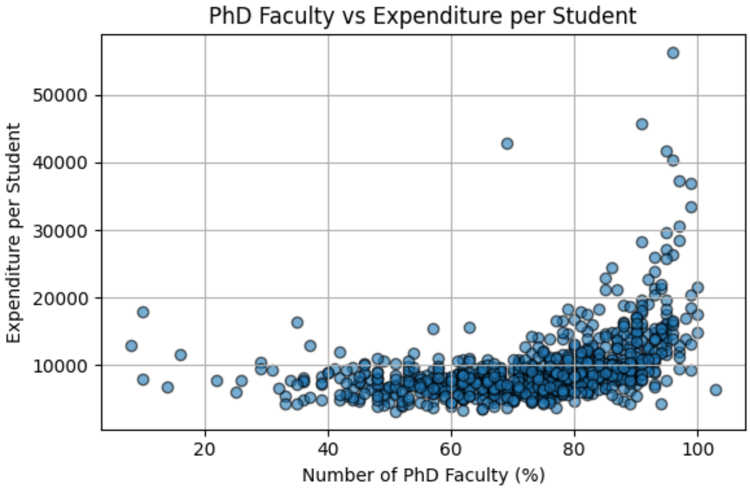
5. Boxplot: Accept Rate vs Private/Public

```python
plt.figure(figsize=(6, 4))
groups = df.groupby("Private")["Accept"]
plt.boxplot([group for _, group in groups], tick_labels=["No", "Yes"])
plt.title("Acceptance Rate by Institution Type")
plt.xlabel("Private Institution")
plt.ylabel("Acceptance Rate (%)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



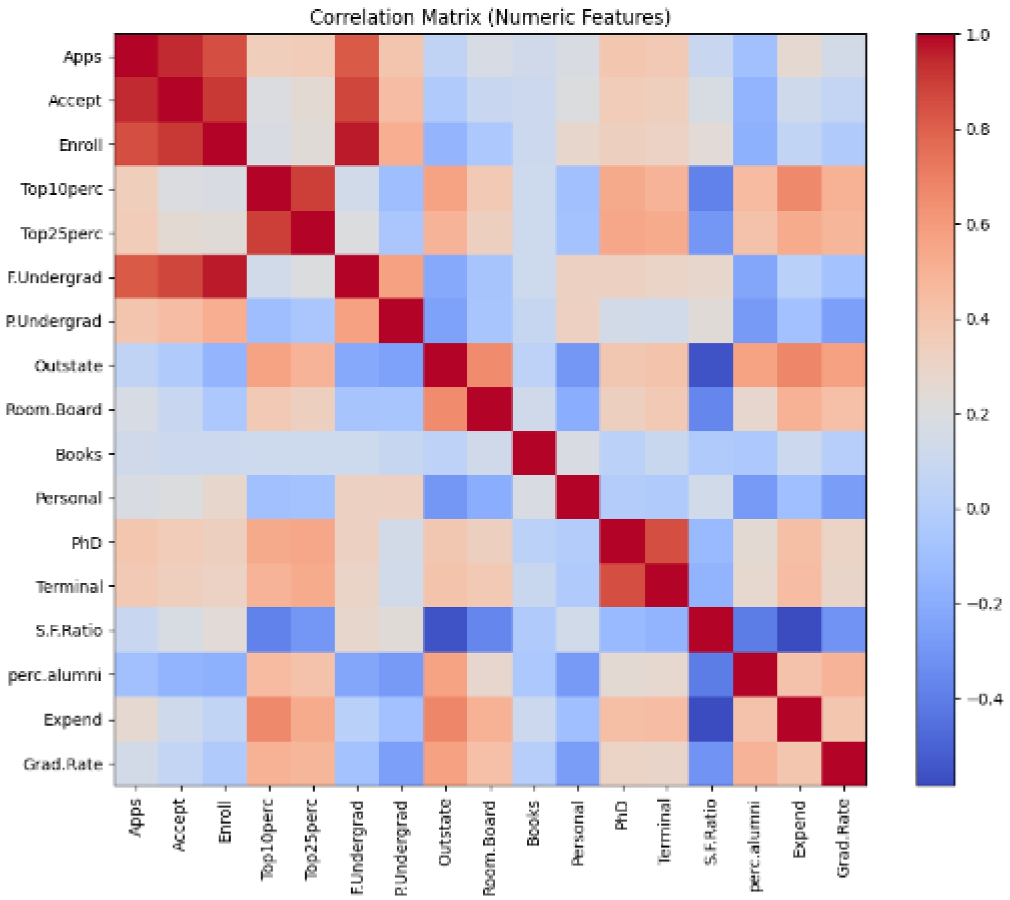
6. Scatter Plot: Faculty vs Expenditures

```python
plt.figure(figsize=(6, 4))
plt.scatter(df["PhD"], df["Expend"], alpha=0.6, edgecolor="black")
plt.title("PhD Faculty vs Expenditure per Student")
plt.xlabel("Number of PhD Faculty (%)")
plt.ylabel("Expenditure per Student")
plt.grid(True)
plt.tight_layout()
plt.show()
```



7. Correlation Heatmap with Matplotlib

```python
num_df = df.select_dtypes(include="number")
corr = num_df.corr()

plt.figure(figsize=(10, 8))
plt.imshow(corr, cmap="coolwarm", interpolation="nearest")
plt.colorbar()
plt.title("Correlation Matrix (Numeric Features)")
plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
plt.yticks(range(len(corr.columns)), corr.columns)
plt.tight_layout()
plt.show()
```



REINFORCEMENT LEARNING

BASIC REINFORCEMENT LEARNING WITH CARTPOLE

https://www.kaggle.com/code/arhamsharif/basic-reinforcement-learning-with-cartpole

CODE

---

1. Import Libraries

```python
import warnings

# Suppress the specific pkg_resources deprecation warning from pygame
warnings.filterwarnings(
    "ignore",
    message="pkg_resources is deprecated as an API*",
    category=UserWarning,
)

import numpy as np
np.bool8 = np.bool_  # Fix numpy bool8 attribute error
import gym
```

2. Create the CartPole environment with human rendering mode for visualization

```python
env = gym.make("CartPole-v1", render_mode="human")

episodes = 10  # Number of episodes to run
max_steps = 200  # Max steps per episode

for episode in range(episodes):
    # Reset environment to initial state
    obs = env.reset()

    # Handle new gym reset() returning (obs, info) tuple
    if isinstance(obs, tuple):
        state, _ = obs
    else:
        state = obs
```

```python
    total_reward = 0

    for step in range(max_steps):
        # No need to call env.render() here, render_mode="human" handles it

        # Select an action randomly from the action space
        action = env.action_space.sample()

        # Take a step in the environment with the selected action
        result = env.step(action)

        # Handle step() output differences between gym versions
        if len(result) == 5:
            next_state, reward, terminated, truncated, info = result
            done = terminated or truncated
        else:
            next_state, reward, done, info = result

        state = next_state
        total_reward += reward

        # Break loop if episode finished
        if done:
            break

    print(f"Episode {episode + 1}: Total Reward = {total_reward}")
```

```
Episode 1: Total Reward = 42.0
Episode 2: Total Reward = 26.0
Episode 3: Total Reward = 15.0
Episode 4: Total Reward = 12.0
Episode 5: Total Reward = 39.0
Episode 6: Total Reward = 16.0
Episode 7: Total Reward = 23.0
Episode 8: Total Reward = 13.0
Episode 9: Total Reward = 12.0
Episode 10: Total Reward = 10.0
```

3. Close environment window

```python
env.close()
```

CLASSIFICATION

IRIS DATASET CLASSIFICATION WITH MULTIPLE MODELS

https://www.kaggle.com/code/arhamsharif/iris-dataset-classification-with-multiple-models

CODE

1. Import Libraries

```python
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.inspection import DecisionBoundaryDisplay
```

2. Import Data from online URL

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

# The iris data doesn't have headers, so add them manually
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
df = pd.read_csv(url, header=None, names=column_names)
```

3. Preprocess / EDA / IDA

```python
print(df.head())
print(df['class'].value_counts())

# Use only two features for visualization
X = df[['sepal_length', 'sepal_width']].values
y = df['class'].values

# Encode target labels
le = LabelEncoder()
y = le.fit_transform(y)
```

```
   sepal_length  sepal_width  petal_length  petal_width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

4. Train-Test Split (80% Train / 20% Test)

```
# 4. Train-Test Split (80% Train / 20% Test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

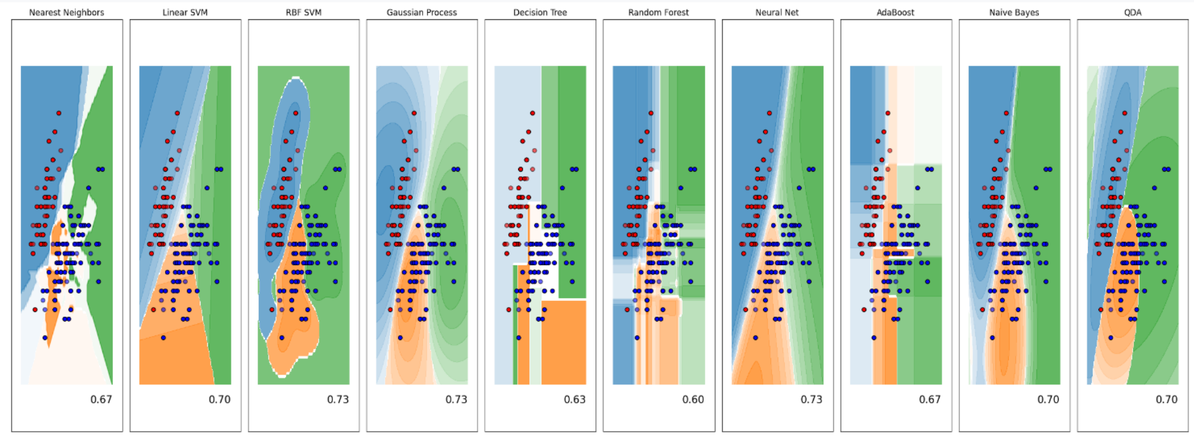
5. Execute / Feature Scaling inside pipeline

6. Define classifiers

```
names = [
    "Nearest Neighbors",
    "Linear SVM",
    "RBF SVM",
    "Gaussian Process",
    "Decision Tree",
    "Random Forest",
    "Neural Net",
    "AdaBoost",
    "Naive Bayes",
    "QDA",
]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025, random_state=42),
    SVC(gamma=2, C=1, random_state=42),
    GaussianProcessClassifier(1.0 * RBF(1.0), random_state=42),
    DecisionTreeClassifier(max_depth=5, random_state=42),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1, random_state=42),
    MLPClassifier(alpha=1, max_iter=1000, random_state=42),
    AdaBoostClassifier(random_state=42),
    GaussianNB(),
    QuadraticDiscriminantAnalysis(),
]
```

7. Testing and 8. Predict + Visualize decision boundaries

```
figure = plt.figure(figsize=(27, 9))
cm_bright = ListedColormap(["#FF0000", "#0000FF"])
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

# Plot input data
for i, (name, clf) in enumerate(zip(names, classifiers), start=2):
    ax = plt.subplot(1, len(classifiers) + 1, i)
    clf_pipeline = make_pipeline(StandardScaler(), clf)
    clf_pipeline.fit(X_train, y_train)
    score = clf_pipeline.score(X_test, y_test)

    DecisionBoundaryDisplay.from_estimator(
        clf_pipeline, X, alpha=0.8, ax=ax, eps=0.5
    )

    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, edgecolors="k")
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6, edgecolors="k")
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(name)

    ax.text(
        x_max - 0.5,
        y_min + 0.3,
        f"{score:.2f}",
        size=15,
        horizontalalignment="right",
    )

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



RECOGNIZING HANDWRITTEN DIGITS

https://www.kaggle.com/code/arhamsharif/recognizing-handwritten-digits-classification

CODE

1. Import Libraries

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score
```

2. Load Digits Dataset

```
digits = datasets.load_digits()
X = digits.data
y = digits.target

print(f"Dataset shape: {X.shape}")
print(f"Number of classes: {len(np.unique(y))}")
```

```
Dataset shape: (1797, 64)
Number of classes: 10
```

3. Train-Test Split (80% train, 20% test)

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

4. Define classifier pipeline (example: KNN with scaling)

```
clf = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=3))
```

5. Train classifier

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

This output uses HTML that may be stripped because the notebook is not trusted

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('kneighborsclassifier', KNeighborsClassifier(n_neighbors=3))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Pipeline
?Documentation for PipelineiFitted
▶ Parameters

StandardScaler
?Documentation for StandardScaler
▶ Parameters

KNeighborsClassifier
?Documentation for KNeighborsClassifier
▶ Parameters

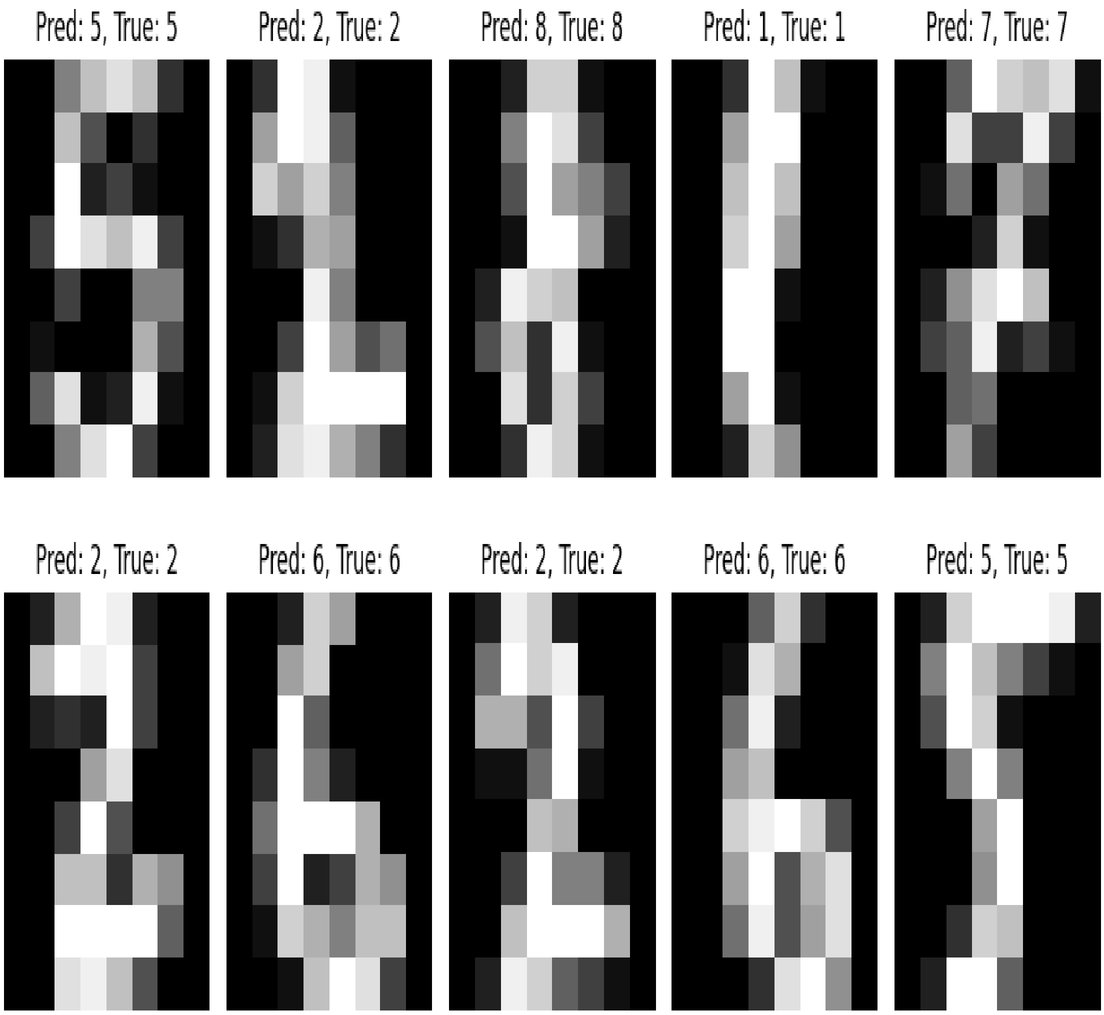
6. Test classifier

```
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.92      0.97      0.95        36
           2       0.90      1.00      0.95        35
           3       1.00      0.97      0.99        37
           4       0.97      0.94      0.96        36
           5       1.00      1.00      1.00        37
           6       0.97      1.00      0.99        36
           7       0.95      0.97      0.96        36
           8       0.97      0.89      0.93        35
           9       1.00      0.92      0.96        36

    accuracy                           0.97       360
   macro avg       0.97      0.97      0.97       360
weighted avg       0.97      0.97      0.97       360

Accuracy: 0.9667
```

7. Visualize some predictions

```
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for ax, image, pred, true in zip(axes.flatten(), X_test, y_pred, y_test):
    ax.imshow(image.reshape(8, 8), cmap='gray')
    ax.set_title(f"Pred: {pred}, True: {true}")
    ax.axis('off')
plt.tight_layout()
plt.show()
```



REGRESSION

BAYESIAN REGRESSION COMPARISON (DIABETES)

https://www.kaggle.com/code/arhamsharif/bayesian-regression-comparison-diabetes

CODE

1. Import Libraries

```
import matplotlib.pyplot as plt
from sklearn.linear_model import BayesianRidge, ARDRegression, LinearRegression
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
```

2. Import Data (Diabetes Dataset)

```
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

print(f"Dataset shape: {X.shape}")
print("Feature names:", diabetes.feature_names)
```

```
Dataset shape: (442, 10)
Feature names: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

3. Preprocess / Feature Scaling (Inside Pipelines or Manual)

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4. Train-Test Split

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

5. Define Regressors

```
ols = LinearRegression()
bayesian_ridge = BayesianRidge()
ard = ARDRegression()

regressors = {
    "Linear Regression (OLS)": ols,
    "Bayesian Ridge Regression": bayesian_ridge,
    "ARD Regression": ard,
}
```

6. Fit Models

```
for name, reg in regressors.items():
    reg.fit(X_train, y_train)
    print(f"{name} fitted successfully.")
```

```
Linear Regression (OLS) fitted successfully.
Bayesian Ridge Regression fitted successfully.
ARD Regression fitted successfully.
```

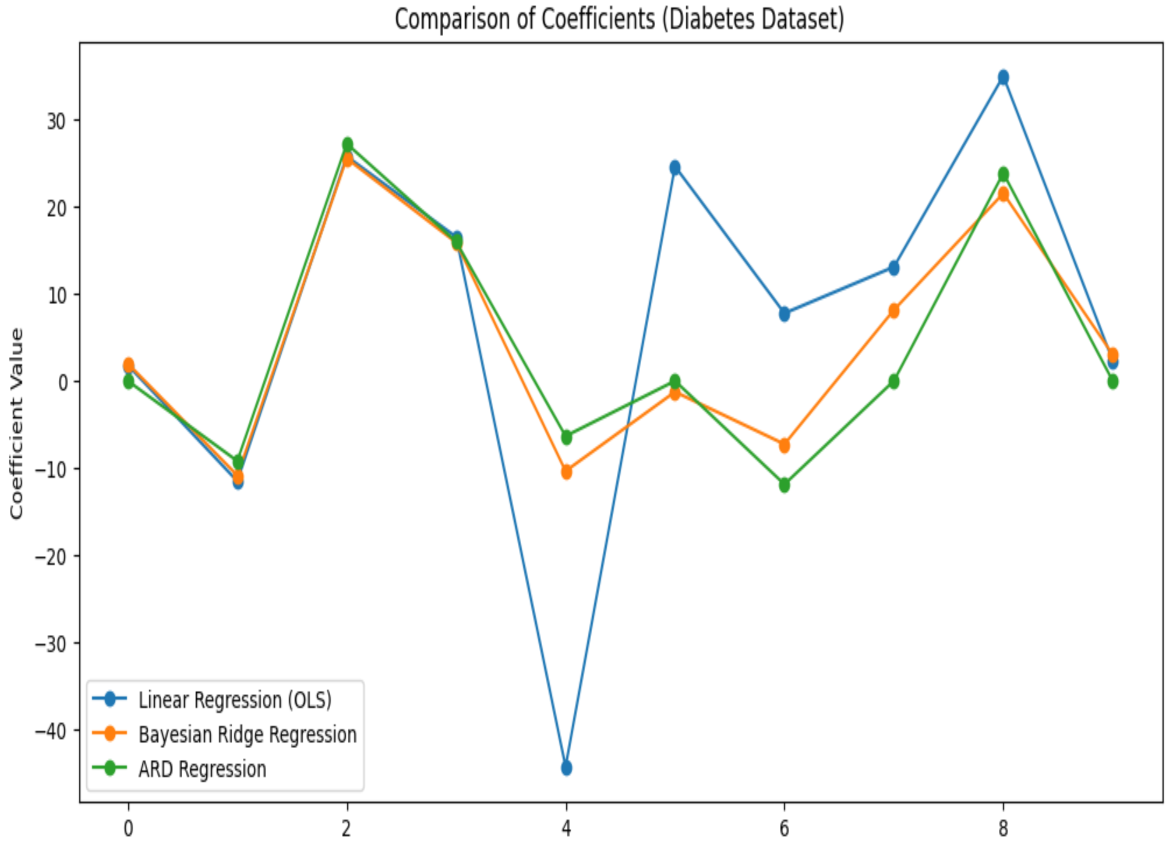
7. Test Models and Print Scores

```
for name, reg in regressors.items():
    y_pred = reg.predict(X_test)
    print(f"\n{name}")
    print(f"R^2 Score: {r2_score(y_test, y_pred):.4f}")
    print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.4f}")
```

```

Linear Regression (OLS)
R^2 Score: 0.4526
Mean Squared Error: 2900.1936

Bayesian Ridge Regression
R^2 Score: 0.4580
Mean Squared Error: 2871.7621

ARD Regression
R^2 Score: 0.4670
Mean Squared Error: 2823.7391
```

8. Visualize Coefficients

```
plt.figure(figsize=(12, 6))
for name, reg in regressors.items():
    plt.plot(reg.coef_, marker='o', label=name)

plt.title("Comparison of Coefficients (Diabetes Dataset)")
plt.xlabel("Feature Index")
plt.ylabel("Coefficient Value")
plt.legend()
plt.show()
```



ELASTICNET WITH AUTO PRECOMPUTED GRAM MATRIX

https://www.kaggle.com/code/arhamsharif/elasticnet-with-auto-precomputed-gram-matrix

CODE

1. Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import time
```

2. Load Dataset

```
housing = fetch_california_housing()
X = housing.data
y = housing.target

print(f"Dataset shape: {X.shape}")
print("Feature names:", housing.feature_names)
```

```
Dataset shape: (20640, 8)
Feature names: ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']
```

3. Preprocess / Feature Scaling

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4. Train-Test Split

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

5. Sample Weights (simulate weights based on distance from median target)

```
sample_weight = np.abs(y_train - np.median(y_train))
sample_weight /= sample_weight.max()
```

6. Fit ElasticNet Normally (Without Precomputed Gram)

```
start = time.time()
enet = ElasticNet(alpha=0.1, l1_ratio=0.5, fit_intercept=False, max_iter=10000)
enet.fit(X_train, y_train, sample_weight=sample_weight)
elapsed_normal = time.time() - start
print(f"ElasticNet fit without Gram took: {elapsed_normal:.4f} seconds")
```

```
ElasticNet fit without Gram took: 0.0535 seconds
```

7. Fit ElasticNet with Precomputed Gram (Automatically Handled by ElasticNet)

```
start = time.time()
enet_gram = ElasticNet(alpha=0.1, l1_ratio=0.5, fit_intercept=False, max_iter=10000, precompute=True)
enet_gram.fit(X_train, y_train, sample_weight=sample_weight)
elapsed_gram = time.time() - start
print(f"ElasticNet fit with Gram (auto-precomputed) took: {elapsed_gram:.4f} seconds")
```

```
ElasticNet fit with Gram (auto-precomputed) took: 0.0049 seconds
```

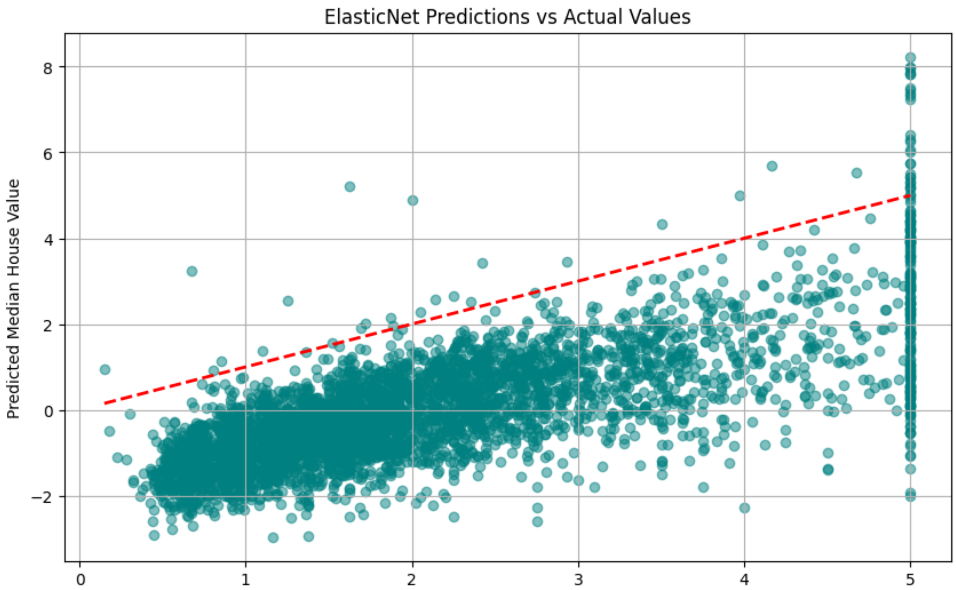
8. Predict & Evaluate

```
y_pred = enet.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (ElasticNet without Gram): {mse:.4f}")
```

```
Mean Squared Error (ElasticNet without Gram): 5.1821
```

9. Visualize Coefficients

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5, color='teal')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.title("ElasticNet Predictions vs Actual Values")
plt.xlabel("Actual Median House Value")
plt.ylabel("Predicted Median House Value")
plt.grid(True)
plt.show()
```



CLUSTERING

AGGLOMERATIVE CLUSTERING ON WINE DATASET

https://www.kaggle.com/code/arhamsharif/agglomerative-clustering-on-wine-dataset

CODE

1. Import Libraries

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
```

2. Load Wine Dataset

```python
wine = load_wine()
X = wine.data
y = wine.target

print("Dataset shape:", X.shape)
print("Feature names:", wine.feature_names)
print("Target classes:", wine.target_names)
```

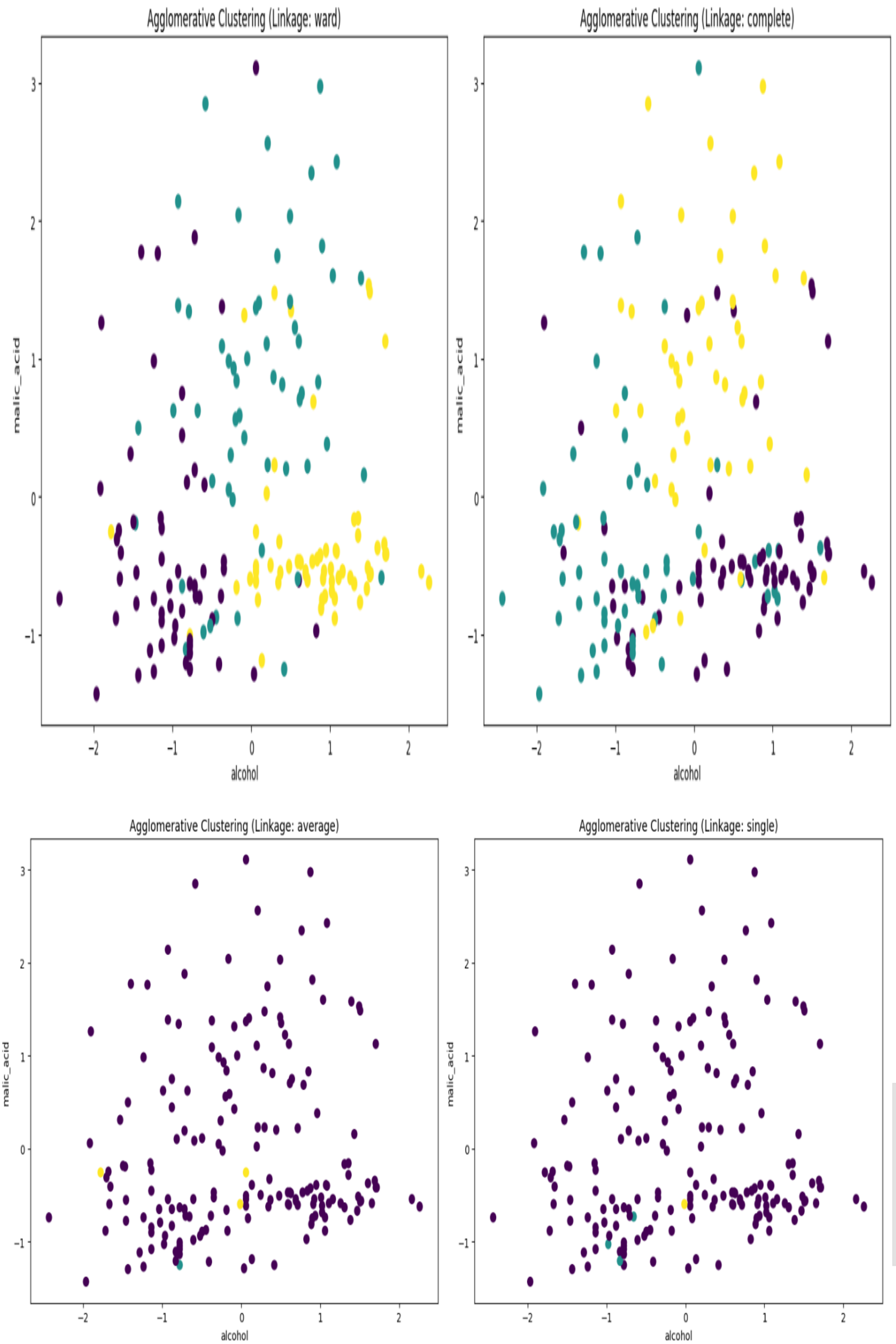
```
Dataset shape: (178, 13)
Feature names: ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
Target classes: ['class_0' 'class_1' 'class_2']
```

3. Feature Scaling (Clustering works better with scaled data)

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4. Apply Agglomerative Clustering with Different Linkage Methods

```python
linkages = ['ward', 'complete', 'average', 'single']

plt.figure(figsize=(16, 12))
for i, linkage in enumerate(linkages, 1):
    # 'ward' works only with Euclidean distance
    if linkage == 'ward':
        model = AgglomerativeClustering(n_clusters=3, linkage=linkage)
    else:
        model = AgglomerativeClustering(n_clusters=3, linkage=linkage, metric='euclidean')

    labels = model.fit_predict(X_scaled)

    # Plot clusters using 2 selected features for visualization
    plt.subplot(2, 2, i)
    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', s=50)
    plt.title(f"Agglomerative Clustering (Linkage: {linkage})")
    plt.xlabel(wine.feature_names[0])
    plt.ylabel(wine.feature_names[1])

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



Agglomerative Clustering (Linkage: ward)
Agglomerative Clustering (Linkage: complete)
Agglomerative Clustering (Linkage: average)
Agglomerative Clustering (Linkage: single)
malic_acid
alcohol

AGGLOMERATIVE CLUSTERING WITH & WITHOUT STRUCTURE

https://www.kaggle.com/code/arhamsharif/agglomerative-clustering-with-without-structure

CODE

1. Import Libraries

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons, make_blobs
from sklearn.cluster import AgglomerativeClustering
```

2. Generate Structured Data (Moons)

```
X_structured, _ = make_moons(n_samples=300, noise=0.05, random_state=42)
```

3. Generate Unstructured Data (Random Blobs)

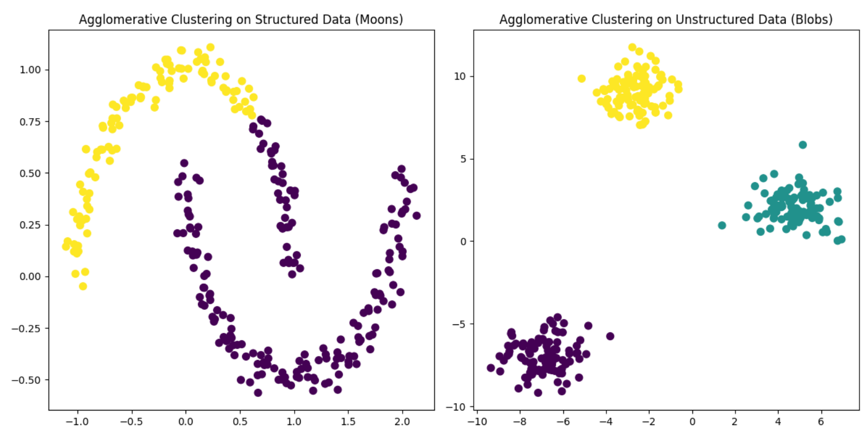
```
X_unstructured, _ = make_blobs(n_samples=300, centers=3, random_state=42)
```

4. Apply Agglomerative Clustering

```
clustering_structured = AgglomerativeClustering(n_clusters=2)
labels_structured = clustering_structured.fit_predict(X_structured)

clustering_unstructured = AgglomerativeClustering(n_clusters=3)
labels_unstructured = clustering_unstructured.fit_predict(X_unstructured)
```

5. Visualize Results

```
plt.figure(figsize=(12, 6))

# Structured Data (Moons)
plt.subplot(1, 2, 1)
plt.scatter(X_structured[:, 0], X_structured[:, 1], c=labels_structured, cmap='viridis', s=50)
plt.title("Agglomerative Clustering on Structured Data (Moons)")

# Unstructured Data (Blobs)
plt.subplot(1, 2, 2)
plt.scatter(X_unstructured[:, 0], X_unstructured[:, 1], c=labels_unstructured, cmap='viridis', s=50)
plt.title("Agglomerative Clustering on Unstructured Data (Blobs)")

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



DIMENSIONALITY REDUCTION

KERNEL PCA WITH RBF KERNEL ON NONLINEAR CIRCLES

https://www.kaggle.com/code/arhamsharif/kernel-pca-with-rbf-kernel-on-nonlinear-circles

CODE

1. Import Libraries

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.decomposition import KernelPCA
from sklearn.preprocessing import StandardScaler
```

2. Generate Nonlinear Data (Nested Circles)

```
X, y = make_circles(n_samples=400, factor=0.3, noise=0.05, random_state=42)
```

3. Standardize Features

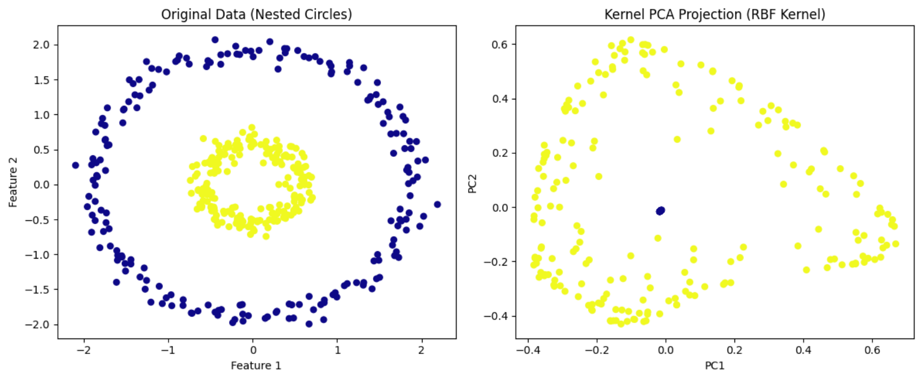
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4. Apply Kernel PCA

```
kpca = KernelPCA(n_components=2, kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X_scaled)
```

5. Plot Original Data

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, cmap='plasma', s=30)
plt.title("Original Data (Nested Circles)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")

# 6. Plot Transformed Data
plt.subplot(1, 2, 2)
plt.scatter(X_kpca[:, 0], X_kpca[:, 1], c=y, cmap='plasma', s=30)
plt.title("Kernel PCA Projection (RBF Kernel)")
plt.xlabel("PC1")
plt.ylabel("PC2")

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



PCA ON THE IRIS DATASET

https://www.kaggle.com/code/arhamsharif/pca-on-the-iris-dataset

CODE

1. Import Libraries

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

2. Load Iris Dataset

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names
```

3. Standardize the Data

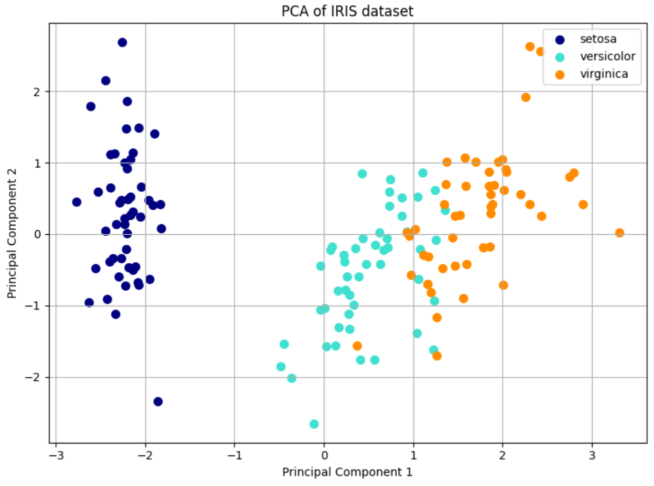
```
X_scaled = StandardScaler().fit_transform(X)
```

4. Apply PCA (reduce to 2 components for visualization)

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

5. Plot Results

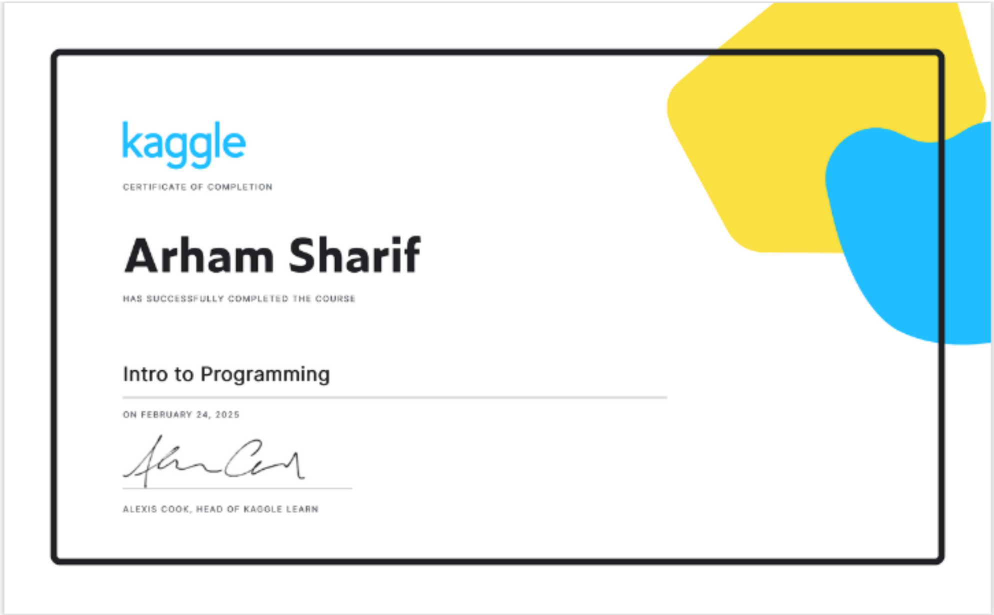
```
plt.figure(figsize=(8, 6))
colors = ['navy', 'turquoise', 'darkorange']
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1],
                color=color, lw=2, label=target_name)

plt.legend()
plt.title('PCA of IRIS dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.tight_layout()
plt.show()
```



CERTIFICATES

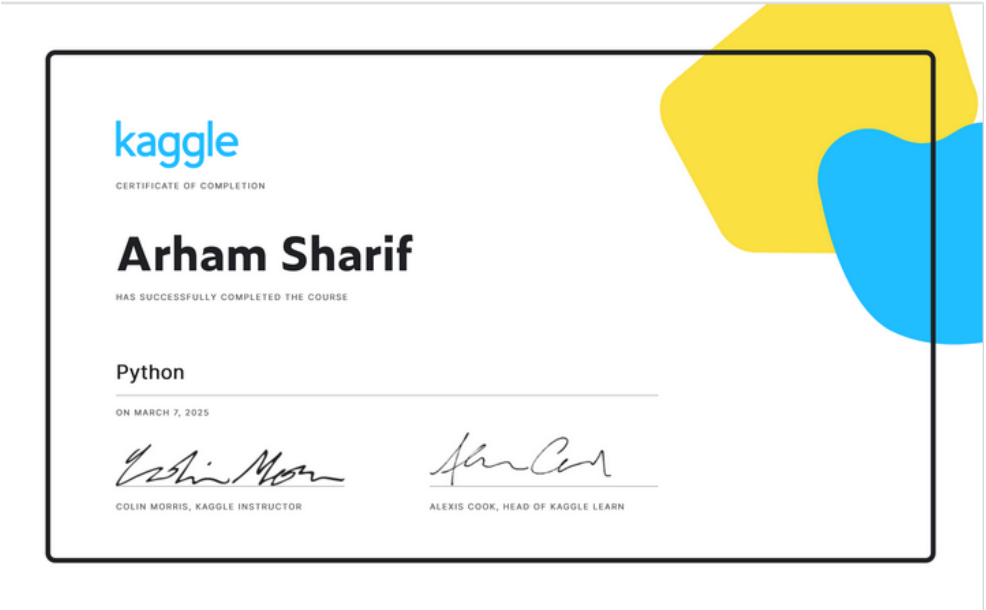
INTRODUCTION TO PROGRAMMING

https://www.kaggle.com/learn/certification/arhamsharif/intro-to-programming



PYTHON

https://www.kaggle.com/learn/certification/arhamsharif/python

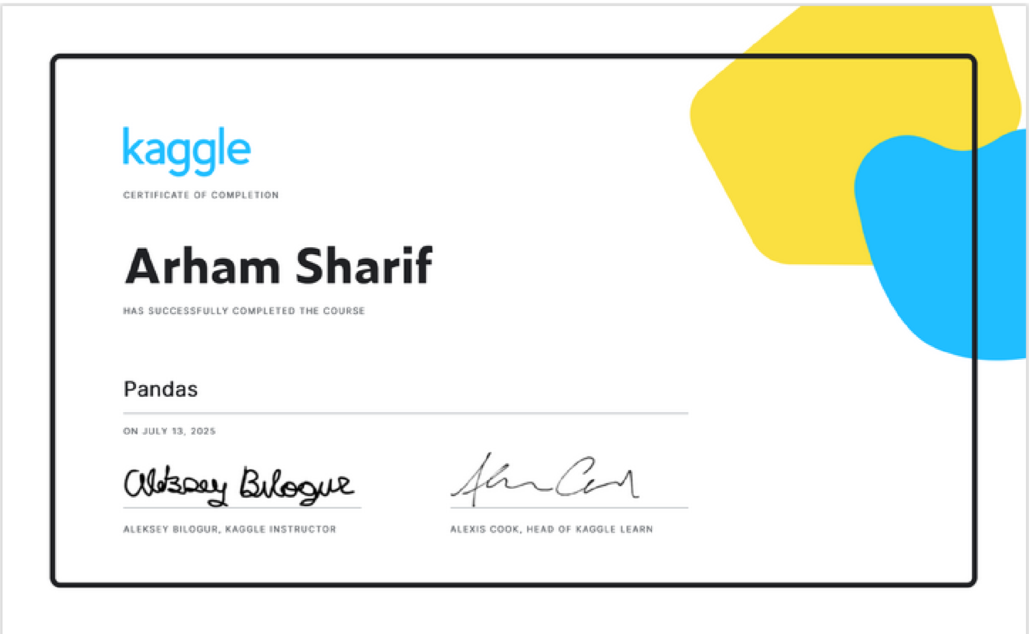


DATA VISUALIZATION

https://www.kaggle.com/learn/certification/arhamsharif/data-visualization

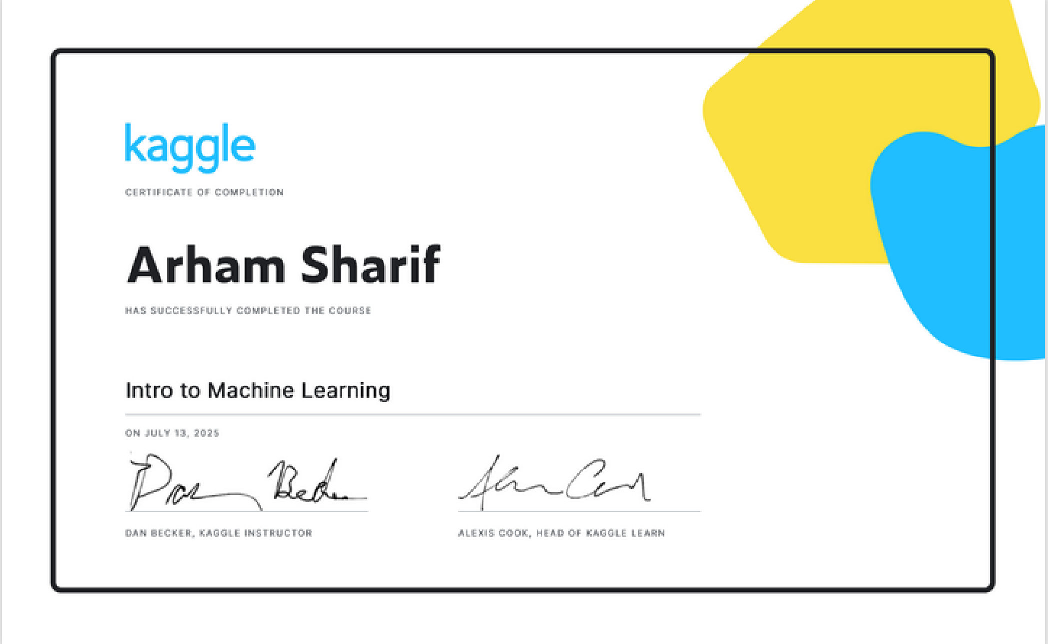
PANDAS

https://www.kaggle.com/learn/certification/arhamsharif/pandas



INTRO TO MACHINE LEARNING

https://www.kaggle.com/learn/certification/arhamsharif/intro-to-machine-learning



INTRO TO GAME AI AND REINFORCEMENT LEARNING

https://www.kaggle.com/learn/certification/arhamsharif/intro-to-game-ai-and-reinforcement-learning