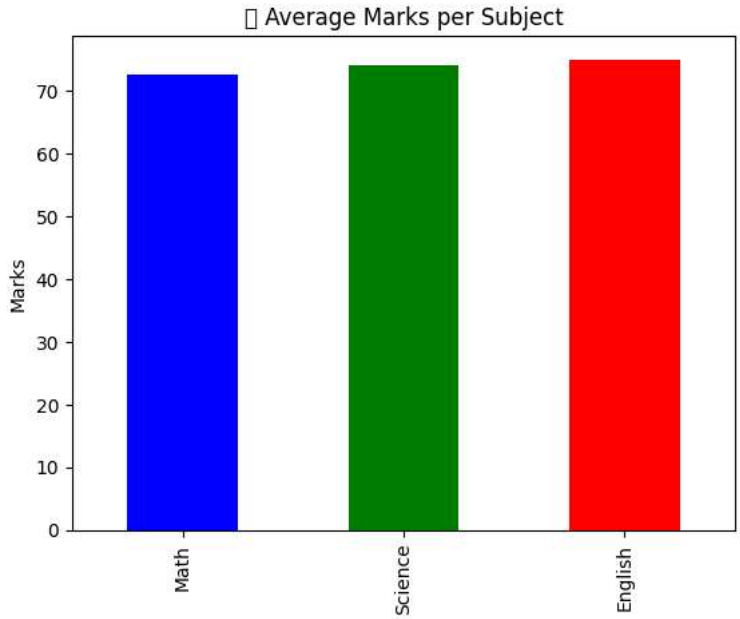
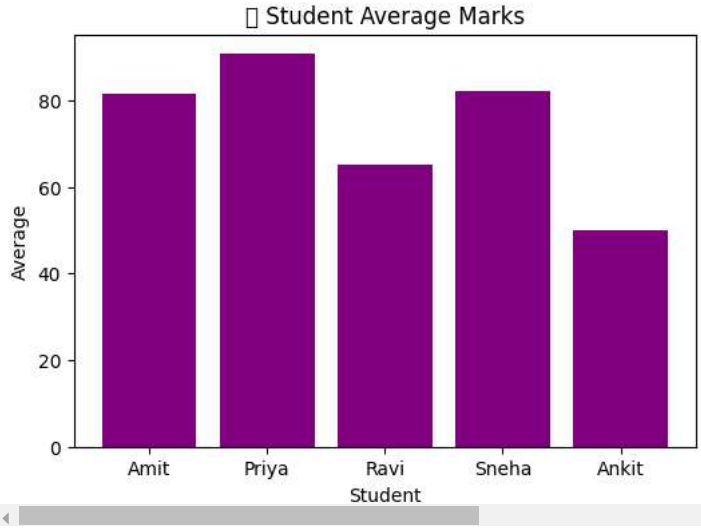
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd

# Create a dictionary with student data
data = {
    'Name': ['Amit', 'Priya', 'Ravi', 'Sneha', 'Ankit'],
    'Math': [78, 90, 65, 80, 50],
    'Science': [85, 88, 70, 82, 45],
    'English': [82, 94, 60, 84, 55]
}

# Convert dictionary to DataFrame
df = pd.DataFrame(data)

# Save to CSV
df.to_csv("students.csv", index=False)

print("✅ 'students.csv' created successfully!")
# Load the dataset
df = pd.read_csv("students.csv")

# Display the data
print("📄 Dataset:\n", df)

# Calculate average marks
df['Average'] = df[['Math', 'Science', 'English']].mean(axis=1)
print("\n📊 Average Marks:\n", df[['Name', 'Average']])

# Find highest scorer
topper = df.loc[df['Average'].idxmax()]
print("\n🏆 Topper:\n", topper)

# Find subject-wise average
subject_avg = df[['Math', 'Science', 'English']].mean()
print("\n📚 Subject-wise Average:\n", subject_avg)

# 📉 Plot subject-wise average
subject_avg.plot(kind='bar', color=['blue', 'green', 'red'])
plt.title("📊 Average Marks per Subject")
plt.ylabel("Marks")
plt.show()

# 📈 Plot each student’s average
plt.figure(figsize=(6,4))
plt.bar(df['Name'], df['Average'], color='purple')
plt.title("🎓 Student Average Marks")
plt.ylabel("Average")
plt.xlabel("Student")
plt.show()
```

```
📄 Dataset:
     Name  Math  Science  English
0   Amit    78       85       82
1  Priya    90       88       94
2   Ravi    65       70       60
3  Sneha    80       82       84
4  Ankit    50       45       55

📊 Average Marks:
     Name    Average
0   Amit  81.666667
1  Priya  90.666667
2   Ravi  65.000000
3  Sneha  82.000000
4  Ankit  50.000000

🏆 Topper:
 Name           Priya
Math               90
Science            88
English            94
Average     90.666667
Name: 1, dtype: object

📚 Subject-wise Average:
 Math       72.6
Science    74.0
English    75.0
dtype: float64
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning:
  fig.canvas.print_figure(bytes_io, **kw)
```




```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning:
  fig.canvas.print_figure(bytes_io, **kw)
```



```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Step 1: Load Dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
           'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df = pd.read_csv(url, header=None, names=columns)

print("✅ Data Sample:\n", df.head())
print("\n📊 Data Summary:\n", df.describe())

# Step 2: Visualize Relationships
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.show()

# Diabetes Outcome Count
sns.countplot(x='Outcome', data=df)
plt.title("Diabetes Outcome Distribution")
plt.xlabel("Outcome (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()

# Step 3: Prepare Data
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Scale Features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42)

# Step 4: Build the Model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Binary classification
])

# Step 5: Compile the Model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Step 6: Train the Model
history = model.fit(X_train, y_train,
                    epochs=50,
                    batch_size=16,
                    validation_split=0.2,
                    verbose=1)

# Step 7: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"🎯 Test Accuracy: {accuracy:.2f}")

# Step 8: Plot Training History
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Training vs Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()

# Step 9: Predictions (Optional)
```

```
y_pred = model.predict(X_test[:10])
print("\n🔍 Predictions vs Actual:")
for i in range(10):
    print(f"Predicted: {y_pred[i][0]:.2f} -> {1 if y_pred[i][0] >= 0.5 else 0} | Actual: {y_test.iloc[i]}")
```