EX NO: 10 DATE:	STATISTICAL HYPOTHESIS TESTING Z-TEST, T-TEST, F-TEST

AIM:

ALGORITHM:

PROGRAM:

#SAMPLE DATASET

```
class_A_scores = [78, 85, 90, 87, 76, 82, 89, 84, 91, 77] # Sample A
class_B_scores = [72, 70, 75, 78, 80, 74, 77, 79, 73, 76] # Sample B

import numpy as np
from scipy.stats import ttest_1samp, ttest_ind, ttest_rel, f
from statsmodels.stats.weightstats import ztest

# Sample dataset
class_A_scores = [78, 85, 90, 87, 76, 82, 89, 84, 91, 77]
class_B_scores = [72, 70, 75, 78, 80, 74, 77, 79, 73, 76]
population_mean = 80

print("=== Z-Test ===")
# One-sample Z-test
z_stat, p_val = ztest(class_A_scores, value=population_mean)
print("One-sample Z-test (Class A vs Population Mean):")
print("Z-statistic:", z_stat, "| P-value:", p_val)

# Two-sample Z-test
z_stat, p_val = ztest(class_A_scores, class_B_scores)
print("\nTwo-sample Z-test (Class A vs Class B):")
print("Z-statistic:", z_stat, "| P-value:", p_val)



print("\n=== T-Test ===")
# One-sample T-test
t_stat, p_val = ttest_1samp(class_A_scores, population_mean)
print("One-sample T-test (Class A vs Population Mean):")
print("T-statistic:", t_stat, "| P-value:", p_val)
```

```
# Independent two-sample T-test
t_stat, p_val = ttest_ind(class_A_scores, class_B_scores)
print("\nIndependent Two-sample T-test (Class A vs Class B):")
print("T-statistic:", t_stat, "| P-value:", p_val)

# Paired T-test (assume scores before and after training)
before_training = [60, 65, 68, 70, 75]
after_training = [70, 75, 78, 80, 85]
t_stat, p_val = ttest_rel(before_training, after_training)
print("\nPaired T-test (Before vs After Training):")
print("T-statistic:", t_stat, "| P-value:", p_val)
print("\n=== F-Test ===")
# F-test to compare variance between Class A and Class B
var1 = np.var(class_A_scores, ddof=1)
var2 = np.var(class_B_scores, ddof=1)
f_stat = var1 / var2
df1 = len(class_A_scores) - 1
df2 = len(class_B_scores) - 1
p_val = 1 - f.cdf(f_stat, df1, df2)
print("F-statistic (Class A vs Class B variances):", f_stat)
print("P-value (one-tailed):", p_val)
```

OUTPUT:

```
=== Z-Test ===
One-sample Z-test (Class A vs Population Mean):
Z-statistic: 2.2396703154756357 | P-value: 0.02511233407102513

Two-sample Z-test (Class A vs Class B):
Z-statistic: 4.219056529150508 | P-value: 2.453267629329014e-05

=== T-Test ===
One-sample T-test (Class A vs Population Mean):
T-statistic: 2.2396703154756357 | P-value: 0.05187118134045702

Independent Two-sample T-test (Class A vs Class B):
T-statistic: 4.219056529150508 | P-value: 0.0005159136080406046

Paired T-test (Before vs After Training):
T-statistic: -inf | P-value: 0.0

=== F-Test ===
F-statistic (Class A vs Class B variances): 2.953463203463203
P-value (one-tailed): 0.06119106487728909
```

RESULT:

EX NO: 11 **DATE:**	**LONGITUDINAL DEVELOPEMNT**

AIM:

ALGORITHM:

PROGRAM:

```
import pandas as pd

import numpy as np

import statsmodels.formula.api as smf


# Simulate longitudinal data

np.random.seed(1)

n_participants = 100

time_points = [0, 3, 6, 9, 12]


data = []


for pid in range(1, n_participants + 1):

    base_sleep = np.random.normal(8, 1)  # baseline sleep duration (in hours)

    for t in time_points:

        usage = np.random.normal(3 + 0.2 * t, 1)  # social media use slightly increases over time

        sleep_quality = base_sleep - 0.3 * usage + np.random.normal(0, 0.5)  # effect of usage on sleep

        data.append([pid, t, usage, sleep_quality])


df = pd.DataFrame(data, columns=['ParticipantID', 'Month', 'SocialMediaUsage', 'SleepHours'])


# Fit linear mixed model

model = smf.mixedlm("SleepHours ~ SocialMediaUsage + Month", df, groups=df["ParticipantID"])

result = model.fit()


print(result.summary())
```

OUTPUT:

```
           Mixed Linear Model Regression Results
=============================================================
Model:              MixedLM  Dependent Variable:  SleepHours
No. Observations:   500      Method:              REML
No. Groups:         100      Scale:               0.2572
Min. group size:    5        Log-Likelihood:      -522.1425
Max. group size:    5        Converged:           Yes
Mean group size:    5.0
-------------------------------------------------------------
                 Coef.  Std.Err.    z    P>|z| [0.025 0.975]
-------------------------------------------------------------
Intercept         8.230    0.132  62.253 0.000  7.971  8.489
SocialMediaUsage -0.312    0.027 -11.617 0.000 -0.365 -0.259
Month            -0.016    0.007  -2.229 0.026 -0.030 -0.002
Group Var         0.887    0.294
=============================================================
```

RESULT:

EX NO: 12 DATE:	DIMENSIONALITY REDUCTION & FEATURE INSIGHTS

AIM:

ALGORITHM:

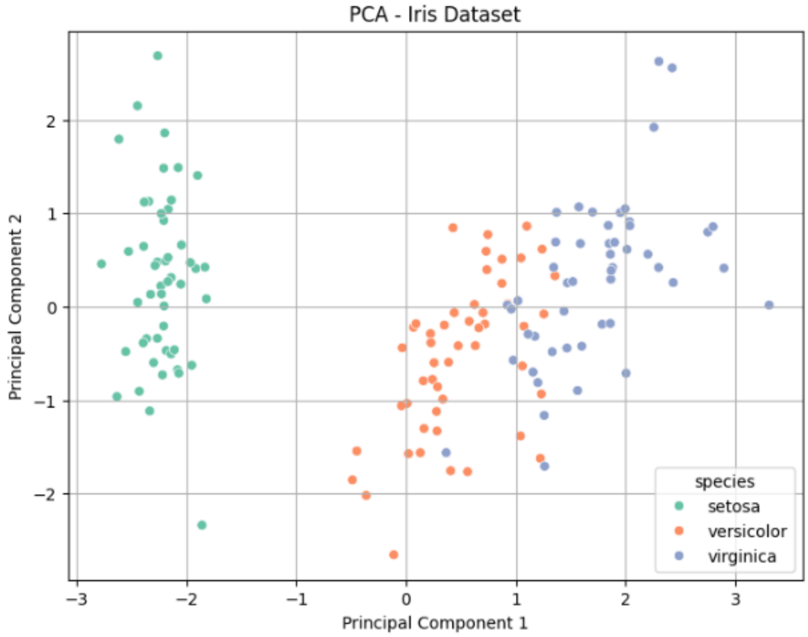
PROGRAM:

```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the dataset
df = sns.load_dataset("iris")
print("Original Dataset Shape:", df.shape)

# Drop the categorical target column for PCA
X = df.drop("species", axis=1)

# Step 1: Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 2: Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 dimensions for visualization
X_pca = pca.fit_transform(X_scaled)

# Step 3: Create a new DataFrame with principal components
pca_df = pd.DataFrame(data=X_pca, columns=["PC1", "PC2"])
pca_df["species"] = df["species"]

# Step 4: Visualize the reduced dimensions
plt.figure(figsize=(8,6))
sns.scatterplot(x="PC1", y="PC2", hue="species", data=pca_df, palette="Set2")
plt.title("PCA - Iris Dataset")
```

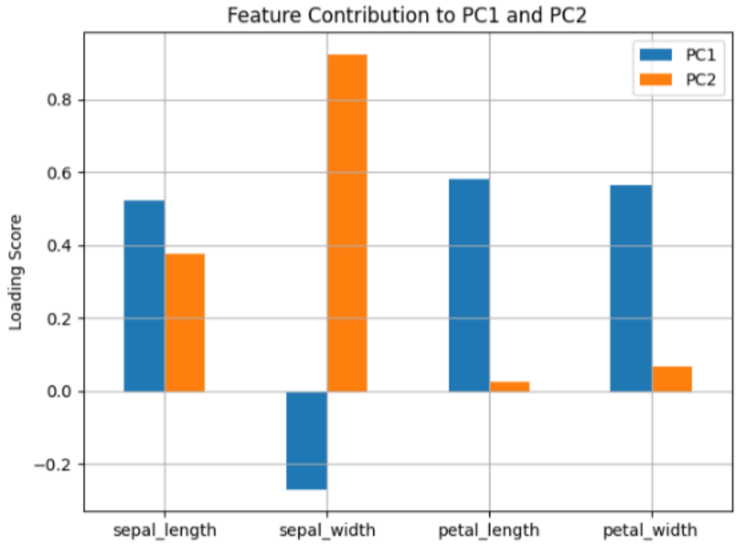
```
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()

# Step 5: Feature Contribution (Loadings)
loadings = pd.DataFrame(pca.components_.T,
                        columns=["PC1", "PC2"],
                        index=X.columns)

print("\n Feature Contribution to Principal Components:\n")
print(loadings)

# Step 6: Plot feature contributions
plt.figure(figsize=(8,5))
loadings.plot(kind='bar')
plt.title("Feature Contribution to PC1 and PC2")
plt.ylabel("Loading Score")
plt.grid(True)
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

OUTPUT:

```
Original Dataset Shape: (150, 5)
```




```
Feature Contribution to Principal Components:

                    PC1       PC2
sepal_length   0.521066  0.377418
sepal_width   -0.269347  0.923296
petal_length   0.580413  0.024492
petal_width    0.564857  0.066942
<Figure size 800x500 with 0 Axes>
```



RESULT: