```
import pandas as pd
import numpy as np
from scipy.stats import skew, kurtosis, norm, t
import matplotlib.pyplot as plt
import seaborn as sns


# Load Titanic dataset
df = pd.read_excel("/content/Titanic (1).xlsx")
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Next steps: Generate code with df View recommended plots New interactive sheet

```
# ----- TASK 1: DESCRIPTIVE STATISTICS -----

# 1. Mean age of passengers
mean_age = df['Age'].mean()
mean_age
```

```
np.float64(29.69911764705882)
```

```
# 2. Median fare paid
median_fare = df['Fare'].median()
median_fare
```

```
14.4542
```

```
# 3. Most common embarkation point (mode)
mode_embarked = df['Embarked'].mode()[0]
mode_embarked
```

```
# 4. Mean, Median, and Mode of Fare for each class
fare_stats_by_class = df.groupby('Pclass')['Fare'].agg(['mean', 'median', lambda x: x.mode()[0]])
fare_stats_by_class.columns = ['Mean Fare', 'Median Fare', 'Mode Fare']
fare_stats_by_class
```

What can I help you build?

Pclass	Mean Fare	Median Fare	Mode Fare
1	84.154687	60.2875	26.55
2	20.662183	14.2500	13.00
3	13.675550	8.0500	8.05

Next steps: Generate code with `fare_stats_by_class` View recommended plots New interactive sheet

```
# 5. Mean and Median of SibSp
sibsp_mean = df['SibSp'].mean()
sibsp_median = df['SibSp'].median()
sibsp_mean, sibsp_median
```

```
(np.float64(0.5230078563411896), 0.0)
```

```
# 6. Skewness of Fare
fare_skewness = skew(df['Fare'].dropna())
fare_skewness
```

```
np.float64(4.7792532923723545)
```

```
# 7. Kurtosis of Age
age_kurtosis = kurtosis(df['Age'].dropna())
age_kurtosis
```

```
np.float64(0.16863657224286044)
```

```
# 8. Skewness of Parch
parch_skewness = skew(df['Parch'].dropna())
parch_skewness
```

```
np.float64(2.7444867379203735)
```

```
# 9. Skewness & Kurtosis of Survived
survived_skew = skew(df['Survived'])
survived_kurtosis = kurtosis(df['Survived'])
survived_skew, survived_kurtosis
```

```
(np.float64(0.4777174662568536), np.float64(-1.7717860224331319))
```

```
# 10. Compare Skewness & Kurtosis of Fare vs Age
age_skew = skew(df['Age'].dropna())
fare_kurtosis = kurtosis(df['Fare'].dropna())
age_skew, fare_kurtosis
```

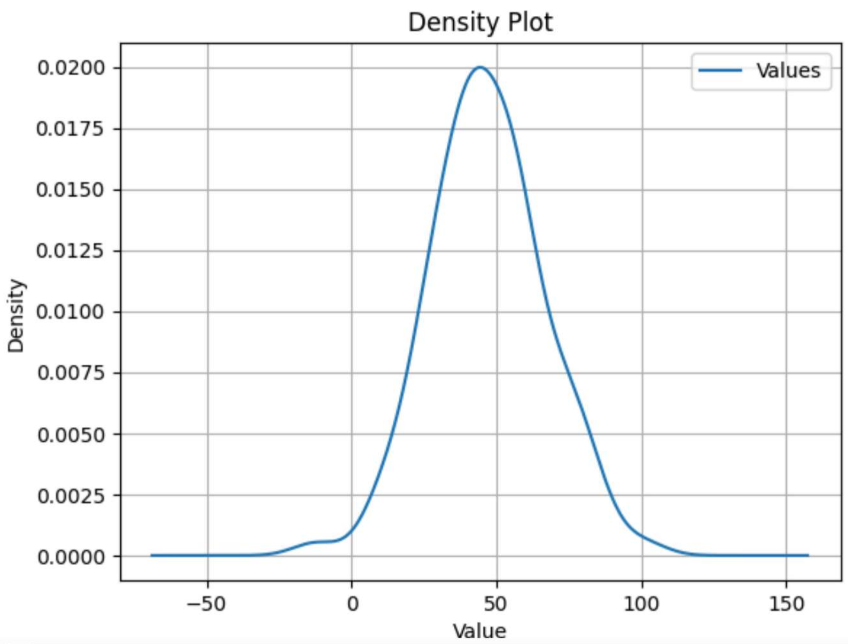
```
(np.float64(0.3882898514698657), np.float64(33.20428925264474))
```

```
# ----- TASK 2: STANDARDIZATION -----

# Step 1: Create Experience and Salary Lists
Exp = [1, 2, 3, 4, 5]
Salary = [1000, 2500, 4000, 5000, 7000]



# Step 2: Compute Mean and Standard Deviation
mean_exp = np.mean(Exp)
std_exp = np.std(Exp)
std_Exp = [(x - mean_exp) / std_exp for x in Exp]


mean_salary = np.mean(Salary)
std_salary = np.std(Salary)
std_Salary = [(x - mean_salary) / std_salary for x in Salary]
```

```
# Step 5: Create DataFrame
df_std = pd.DataFrame([Exp, Salary, std_Exp, std_Salary],
                      index=['Exp', 'Salary', 'Std_Exp', 'Std_Salary'])


# Step 6: Verify standardization properties
mean_std_exp = np.mean(std_Exp)
std_std_exp = np.std(std_Exp)
mean_std_salary = np.mean(std_Salary)
std_std_salary = np.std(std_Salary)
```

```
# ----- TASK 3: NORMAL DISTRIBUTION -----

from scipy.stats import norm

# Step 2: Generate Normal Data
data = norm.rvs(loc=50, scale=20, size=100)
```

```
# Step 3 & 4: Convert to DataFrame and plot density
df_norm = pd.DataFrame(data, columns=['Values'])
df_norm.plot(kind='density', title='Density Plot')
plt.xlabel('Value')
plt.grid(True)
plt.show()
```




```
# Step 5: Statistical Measures
normal_mean = df_norm['Values'].mean()
normal_median = df_norm['Values'].median()
```

```
# ----- TASK 4: HYPOTHESIS TESTING (Z TEST) -----

# 1. One-sample Z test
pop_mean = 168
sample_mean = 169.5
std_dev = 3.9
n = 36
z_score = (sample_mean - pop_mean) / (std_dev / np.sqrt(n))
z_critical_95 = norm.ppf(1 - 0.05 / 2)
reject_null = abs(z_score) > z_critical_95
```

```
# 2. Confidence Intervals for mean
sample_mean_2 = 32
std_dev_2 = 5.6
n2 = 40

z_80 = norm.ppf(1 - 0.2 / 2)
z_90 = norm.ppf(1 - 0.1 / 2)
z_98 = norm.ppf(1 - 0.02 / 2)

ci_80 = (sample_mean_2 - z_80 * std_dev_2 / np.sqrt(n2), sample_mean_2 + z_80 * std_dev_2 / np.sqrt(n2))
ci_90 = (sample_mean_2 - z_90 * std_dev_2 / np.sqrt(n2), sample_mean_2 + z_90 * std_dev_2 / np.sqrt(n2))
ci_98 = (sample_mean_2 - z_98 * std_dev_2 / np.sqrt(n2), sample_mean_2 + z_98 * std_dev_2 / np.sqrt(n2))
```

```
# ----- TASK 5: ONE SAMPLE T TEST -----

# 1. One sample T-test
sample_mean_3 = 140
pop_mean_3 = 100
std_dev_3 = 20
n3 = 30
t_stat = (sample_mean_3 - pop_mean_3) / (std_dev_3 / np.sqrt(n3))
t_critical = t.ppf(1 - 0.025, df=n3-1)
reject_null_t = abs(t_stat) > t_critical
```

```
# 2. 95% Confidence Interval
sample_mean_4 = 20
std_dev_4 = 3.5
n4 = 15
t_critical_95 = t.ppf(1 - 0.05 / 2, df=n4-1)
ci_95 = (sample_mean_4 - t_critical_95 * std_dev_4 / np.sqrt(n4), sample_mean_4 + t_critical_95 * std_dev_4 / np.sqrt(n4))
```

Start coding or generate with AI.