

Project Name

EDA and Statistical Analysis of Credit Card Dataset

By:

Ruchir Tidke

Sudarshan Pagar

Ashish Bikkad

Shubham Rajput

Piyush Khupse

Overview

This Statistics and EDA project is designed to train and test you on basic Data Exploratory and Statistical techniques used in the industry today. Apart from bringing you to speed with basic descriptive and inferential methods, you will also deep dive into a dataset and perform thorough cleaning and analysis in order to draw useful business insights from the data. This will expose you to what data scientists do most often—Exploratory Data Analysis.

Goals

1. Using the core statistical theoretical concepts and knowledge to solve real time problem statements.
2. Visualize a real time industry scenario where one can use these statistical concepts.
3. Detailed data analysis and number crunching using statistics
4. Exhaustive report building using EDA and visualization techniques to help the business take decisions using insights from the data

Specifications

Part -I is concept based and walks you through various concepts of descriptive statistics, probability distributions and inferential statistics including confidence intervals and hypothesis testing.

Part -II on the other hand is dataset based and explore various data cleaning options, data analysis options and using EDA to derive deep and meaningful insights for the business

PART-A (Concept Based)--25 points

The following are the ages of CEOs of 42 Top Fortune 500 Companies when they took over the post of CEO

57 61 57 57 58 57 61 54 68

51 49 64 50 48 65 52 56 46

54 49 50 47 55 55 54 42 51

56 55 54 51 60 62 43 55 56

61 52 69 64 46 54

Use this data for answering following questions where relevant.

Q1. Compute the mean, median and the mode of the data

Code :

```
mean=np.mean(age_c)
```

Mean = 54,

```
median=np.median(age_c)
```

Median = 55,

```
mode=stats.mode(age_c)
```

Mode = 54, count[5]

Q2. Compute the range , variance and standard deviation of CEO ages

Code:

```
range=np.max(age_c)-np.min(age_c)
```

range = 27

```
var=np.var(age_c)
```

var = 38.51473922902495

```
std=np.std(age_c)
```

std = 6.2060244302633025

Q3. Find the mean deviation for the data. The mean deviation is defined as below.

$$\text{Mean deviation} = \frac{\sum |X - \bar{X}|}{n}$$

Code:

```
r=[]

for i in age_c:

    r.append(abs(i - mean))

str(res)
```

CEO_ages

Mean Deviations

```
[2.095238095238095, 6.095238095238095, 2.095238095238095, 2.095238095238095,
3.095238095238095, 2.095238095238095, 6.095238095238095, 0.9047619047619051,
13.095238095238095, 3.904761904761905, 5.904761904761905, 9.095238095238095,
4.904761904761905, 6.904761904761905, 10.095238095238095, 2.904761904761905,
1.095238095238095, 8.904761904761905, 0.9047619047619051, 5.904761904761905,
4.904761904761905, 7.904761904761905, 0.0952380952380949,
0.0952380952380949, 0.9047619047619051, 12.904761904761905,
3.904761904761905, 1.095238095238095, 0.0952380952380949,
0.9047619047619051, 3.904761904761905, 5.095238095238095, 7.095238095238095,
11.904761904761905, 0.0952380952380949, 1.095238095238095,
6.095238095238095, 2.904761904761905, 14.095238095238095, 9.095238095238095,
8.904761904761905, 0.9047619047619051]
```

Q4. Calculate the Pearson coefficient of skewness and comment on the skewness of the data

[A measure to determine the skewness of a distribution is called the Pearson coefficient of skewness. The formula is

$$Skewness = \frac{3(\bar{X} - MD)}{s}$$

where MD is the median and s the standard deviation

The value of the coefficient of skewness usually ranges from -3 to 3. When the distribution is symmetric, the coefficient is zero; when the distribution is positively skewed, the coefficient is positive, and when the distribution is negatively skewed the coefficient is negative.]

Code:

```
x_bar = np.mean(l_ceo_age)

MD = np.median(l_ceo_age)
```



```
sd = np.std(l_ceo_age)
```

```
skew = 3*(x_bar - MD)/sd
```

Skewness in range -3 to 3 hence it is distribution is symmetric: -0.04603821479029574

Q5. Count the number of data values that fall within two standard deviations of the mean. Compare this with the answer from Chebyshev's Theorem.

Code:

```
u=mean+(2*std)
```

```
l=mean-(2*std)
```

```
c=[]
```

```
for i in age_c:
```

```
    if i<=u and i>=l:
```

```
        c.append(i)
```

```
print(len(c))
```

```
39
```

```
round(len(c)/len(age_c)*100)
```

```
93
```

According to chebyshev's theorem and empirical rules approximately 95% of the data lie within two standard deviations of the mean, that is, in the interval with endpoints $\bar{x} \pm 2s$ for samples and with endpoints $\mu \pm 2\sigma$ for populations.

AS we can see than 39 data values lie within two standard deviations which is approximately 93%.

It is similar to the answer from chebyshev's theorem which states that 95% of the data lie within two standard deviations of the mean.

Q6. Find the three quartiles and the interquartile range (IQR).

Code:

```
q1 = np.percentile(age_c, 25)
```

```
q3= np.percentile(age_c, 75)
```

```
print('Quartile 1 :',q1,'Quartile 2:',median,'Quartile 3:',q3,'Inter Quartile Range :',q3-q1)
```

```
iqr=scipy.stats.iqr(age_c)
```

```
print('Inter quartile range using scipy:',iqr)
```

```
Quartile 1: 51.0 Quartile 2: 55.0 Quartile 3: 57.75
```

```
Inter Quartile Range: 6.75
```

```
Inter quartile range using scipy: 6.75
```

Q7. Are there any outliers in the data set?

Code:

```
mx_w=q3+(1.5*iqr)

mi_w=q1-(1.5*iqr)

#print(mx_w,mi_w)

for i in age_c:

    if i<=mi_w or i>=mx_w:

        print(i)
```

68

69

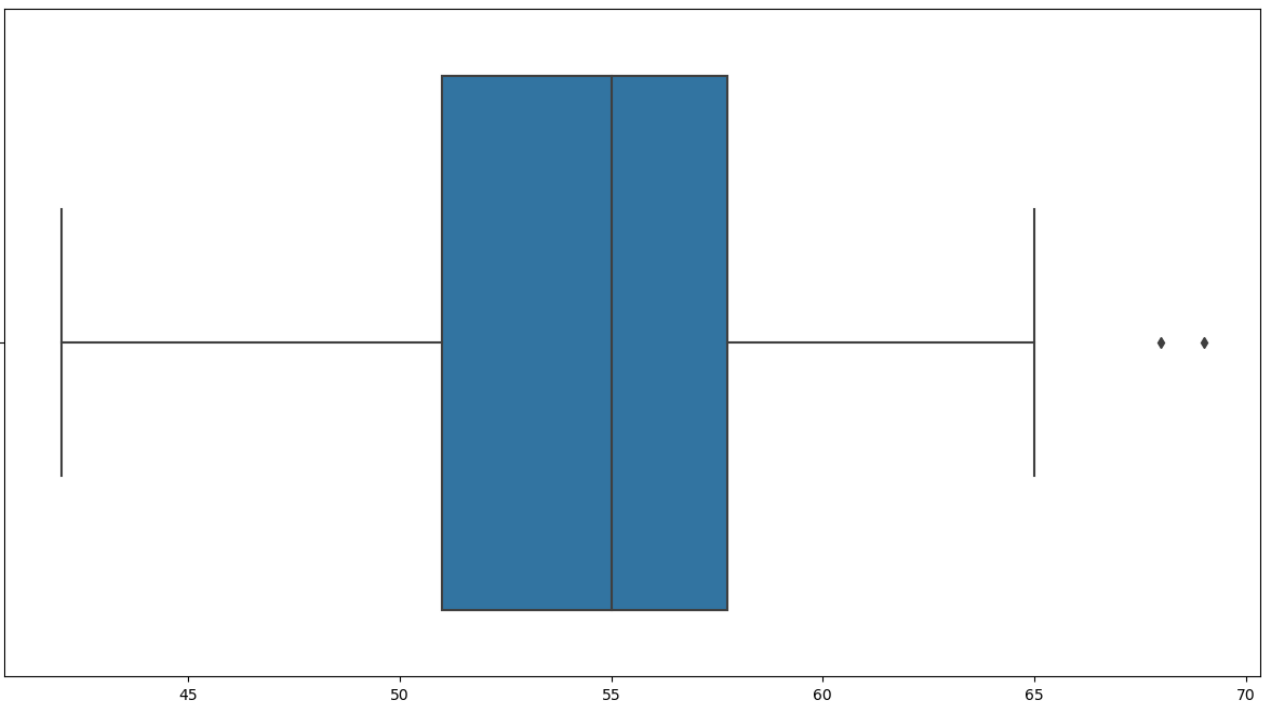
Q8. Draw a boxplot of the dataset to confirm .

Code:

```
plt.figure(figsize=(15,8))

sns.boxplot(age_c)

plt.show()
```



Q9. Find the percentile rank of the datapoint 50.

Code:

```
stats.percentileofscore(age_c, 50)
22.61904761904762
```

Q10. What is the probability that a person becoming a CEO is below 50 years of age ?

Code:

```
l=[]
for i in age_c:
    if i<50:
        l.append(i)
p=len(l)/len(age_c)
p = 0.19047619047619047
```

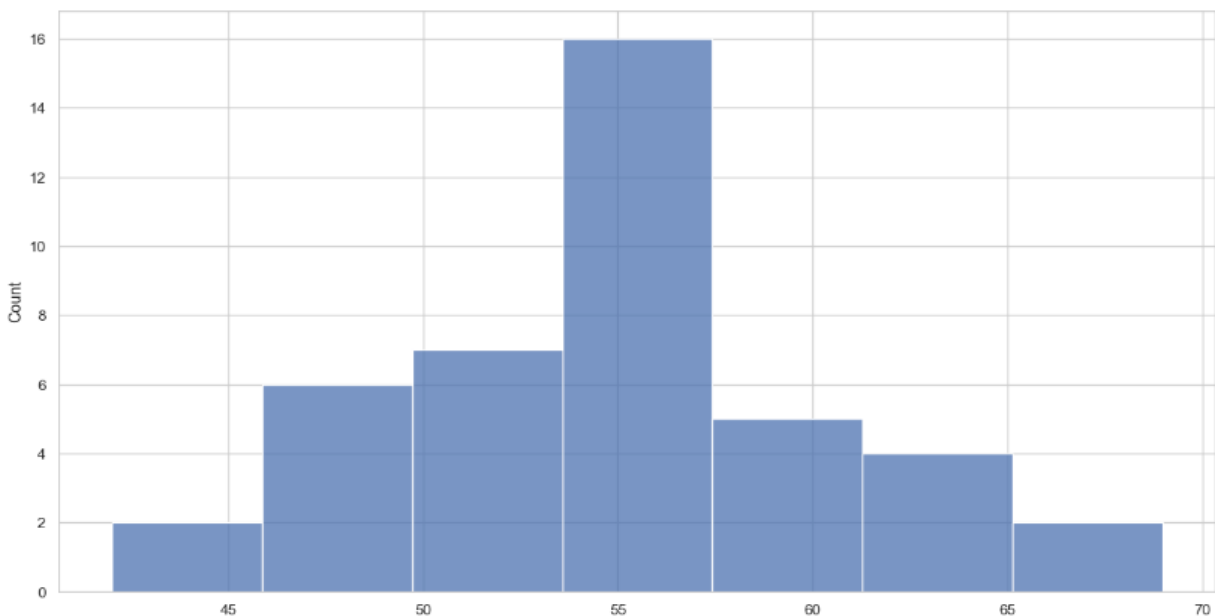
Q11. Create a frequency distribution for the data and visualize it appropriately

Code:

```
plt.figure(figsize=(15,8))
sns.histplot(se,bins=10)
plt.title('Frequency distribution for the data ')
plt.show()
```

```
1 plt.figure(figsize=(15,8))
2 print(age_c.count)
3 sns.histplot(age_c)
4 plt.show()
```

<built-in method count of list object at 0x0000000125200E80>

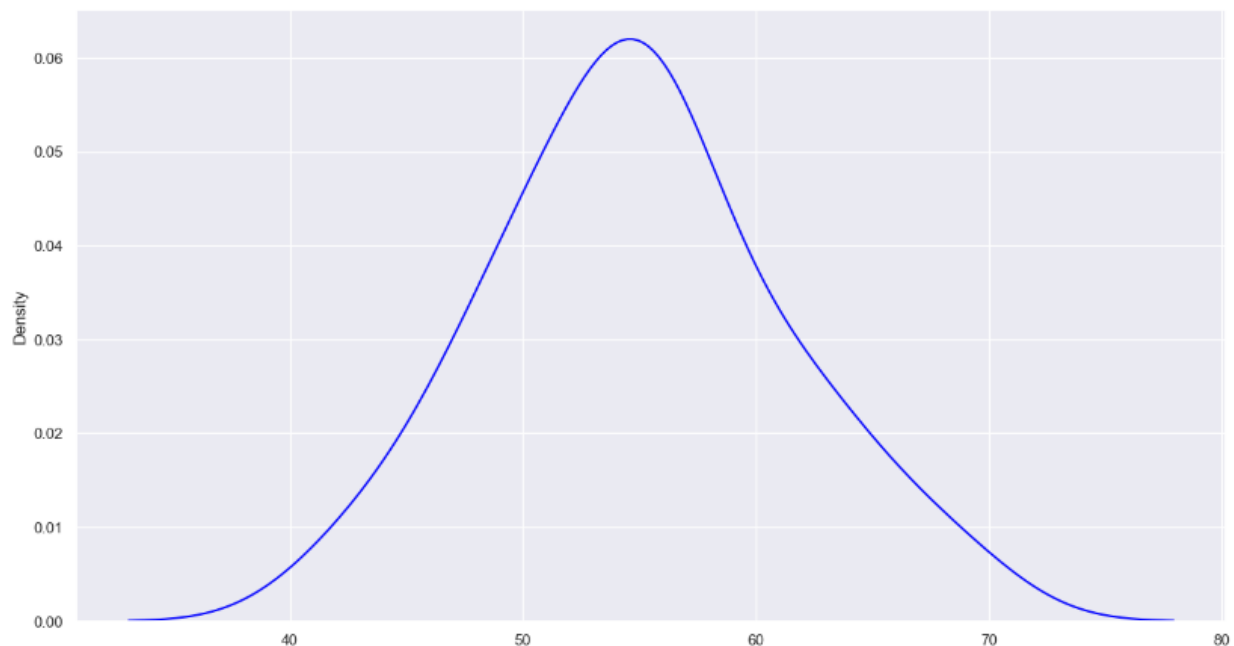


Q12. Create a probability distribution of the data and visualize it appropriately.

Code:

```
plt.figure(figsize=(15,8))
sns.kdeplot(age_c,color='yellow')
plt.show()
```

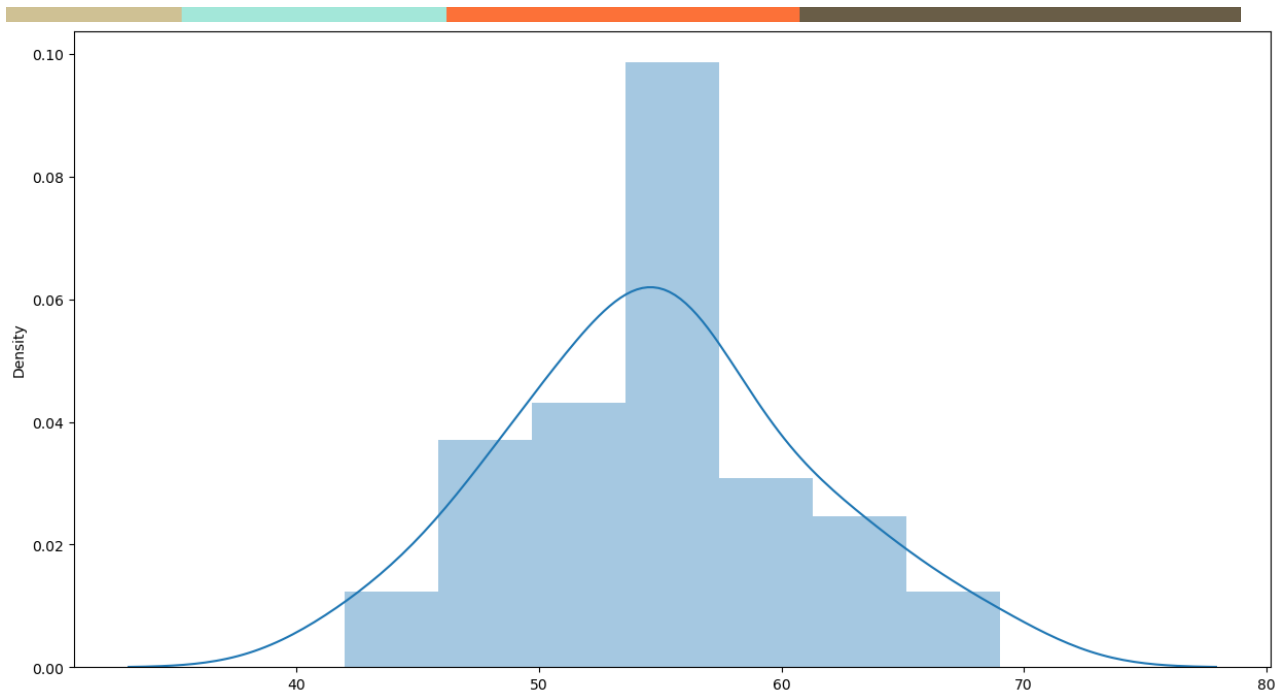
```
1 plt.figure(figsize=(15,8))
2 sns.set()
3 sns.kdeplot(age_c,color='blue')
4 plt.show()
```



Q13. What is the shape of the distribution of this dataset? Create an appropriate graph to determine that. Take 100 random samples with replacement from this dataset of size 5 each. Create a sampling distribution of the mean age of customers. Compare with other sampling distributions of sample size 10, 15, 20, 25, 30. State your observations. Does it corroborate the Central Limit Theorem?

Code:

```
plt.figure(figsize=(15,8))
sns.distplot(age_c)
plt.show()
```

Code:

```
import random
```

```
n1=5
```

```
n2=10
```

```
n3=15
```

```
n4=20
```

```
n5=25
```

```
n6=30
```

```
n_s=100
```

```
plt.figure(figsize=(15,8))
```

```
sns.distplot(age_c)
```

```
plt.title("Population Distribution", fontsize=15)
```

```
plt.show()
```

```
sample_means = []
```

```
for i in range(n_s):
```

```
    sample = np.random.choice(age_c, size=n1, replace=True)
```

```
    sample_mean = np.mean(sample)
```

```
    sample_means.append(sample_mean)
```

```
sample_means2 = []
```

```
for i in range(n_s):
```

```
    sample2 = np.random.choice(age_c, size=n2, replace=True)
```

```
    sample_mean2 = np.mean(sample2)
```

```

sample_means2.append(sample_mean2)

sample_means3 = []
for i in range(n_s):
    sample3 = np.random.choice(age_c, size=n3, replace=True)
    sample_mean3 = np.mean(sample3)
    sample_means3.append(sample_mean3)

sample_means4 = []
for i in range(n_s):
    sample4 = np.random.choice(age_c, size=n4, replace=True)
    sample_mean4 = np.mean(sample4)
    sample_means4.append(sample_mean4)

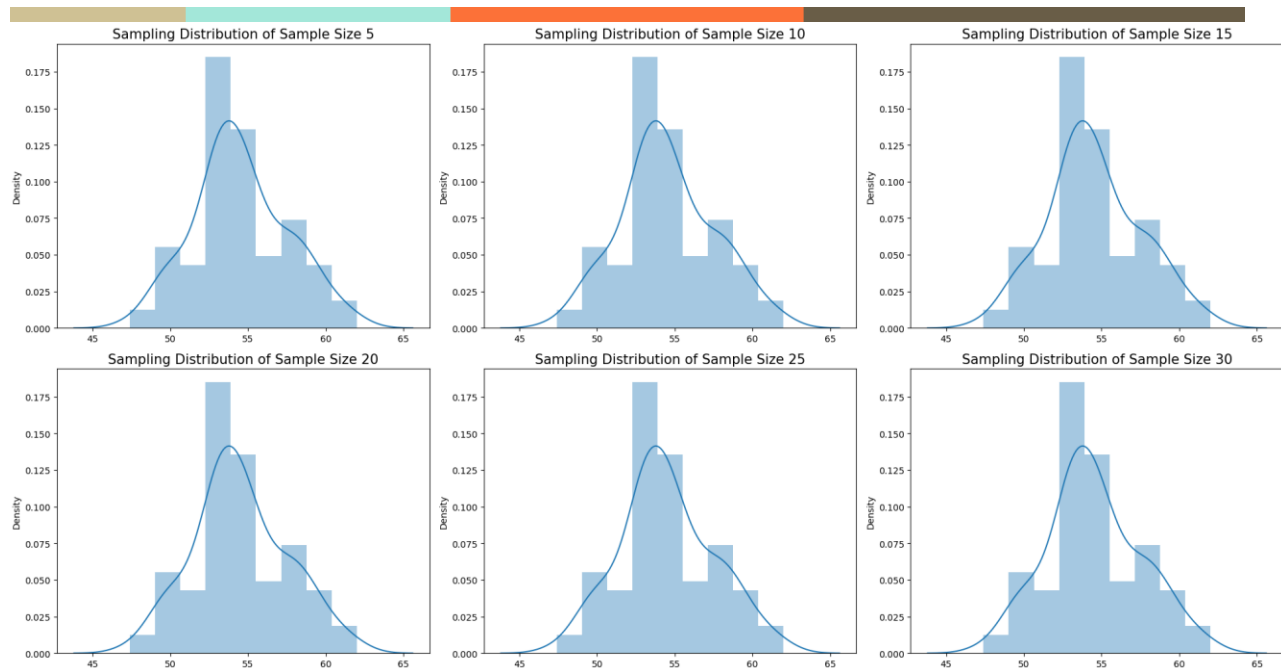
sample_means5 = []
for i in range(n_s):
    sample5 = np.random.choice(age_c, size=n5, replace=True)
    sample_mean5 = np.mean(sample5)
    sample_means5.append(sample_mean5)

sample_means6 = []
for i in range(n_s):
    sample6 = np.random.choice(age_c, size=n6, replace=True)
    sample_mean6 = np.mean(sample6)
    sample_means6.append(sample_mean6)

l_s=[sample_means,sample_means2,sample_means3,sample_means4,sample_means5,sample_means6]
l_n=['5','10','15','20','25','30']
plt.figure(figsize=(20,15))
j=1

for i in range(len(l_s)):
    plt.subplot(3,3,j)
    sns.distplot(sample_means)
    plt.title("Sampling Distribution of Sample Size "+str(l_n[i]), fontsize=15)
    j+=1
plt.tight_layout()
plt.show()

```



Q14. Treat this dataset as a binomial distribution where p is the probability that a person becomes a CEO above 50 years age. What is the probability that out of a random sample of 10 CEOs of Fortune 500 companies exactly 6 are above 50 years of age?

Code:

```
from scipy.stats import binom
```

```
l1=[]
```

```
for i in age_c:
```

```
    if i>50:
```

```
        l1.append(i)
```

```
p_50=len(l1)/len(age_c) # probability of a CEOs being above 50 years of age
```

```
n = len(age_c) # total number of CEOs
```

```
k = 6 # number of CEO'ss above 50 years of age in the sample of 10 CEO's
```

```
p_6=binom.pmf(k, n, p_50)
```

```
print(' the probability that out of a random sample of 10 CEOs exactly 6 are above 50 years of age is',p_6)
```

The probability that out of a random sample of 10 CEOs exactly 6 are above 50 years of age is
3.7518048711673905e-17

Q15. A study claims that 25% of all Fortune 500 companies becoming a CEO are above 60 years of age. Using the Normal approximation of a Binomial distribution, find the probability that in a random sample of 300 Fortune 500 companies exactly 75 CEOs will be above 50 years of age.

[Note that the normal distribution can be used to approximate a binomial distribution if $np \geq 5$ and $nq \geq 5$ with the following correction for continuity $P(X=z) = P(z-0.5 < X < z+0.5)$]

Code:

```
from scipy.stats import norm
n = 300 # sample size
p = 0.25 # probability of a CEOs being above 50 years of age
k = 75 # number of CEOs above 50 years of age in the sample of 300
# mean and standard deviation of the binomial distribution
mean = n*p
std = (n*p*(1-p))**0.5
# using the normal approximation with correction for continuity
norm.cdf(k+0.5, mean, std) - norm.cdf(k-0.5, mean, std)
The probability that in a random sample of 300 Fortune 500 companies exactly 75 CEOs will be above
50 years of age: 0.053152928600730176
```

Q16. Compute a 95% Confidence Interval for the true mean age of the population of CEOs for the given dataset using appropriate distribution. (State reasons as to why did you use a z or t distribution)

Code:

```
from scipy.stats import t
import numpy as np

n = len(age_c) # sample size
mean = np.mean(age_c) # sample mean
std = np.std(age_c, ddof=1) # sample standard deviation with ddof=1 for unbiased estimator
alpha = 0.05 # significance level

# using t-distribution with n-1 degrees of freedom
t_critical = t.ppf(1-alpha/2, n-1)

# margin of error
margin_of_error = t_critical * std / (n**0.5)

# lower and upper bounds of the confidence interval
lower_CI = mean - margin_of_error
upper_CI = mean + margin_of_error

(lower_CI, upper_CI)
(52.94738608388915, 56.86213772563466)
```

Q17. A data scientist wants to estimate with 95% confidence the proportion of CEOs of Fortune 500 companies are above 60 years in the population.

Another recent study showed that 25% of CEOs interviewed were above 60. The data scientist wants to be accurate within 2% of the true proportion. Find the minimum sample size necessary.

Code:

```
ME=2
sd=np.std(age_c)
z_alpha_by_2=(norm.isf(0.05/2))

n = round(( z_alpha_by_2 *sd /ME)**2)
n = 37
```

Q18. The same data scientist wants to estimate the true proportion of CEOs ascending to the post and above 60 years. She wants to be 90% confident

and accurate within 5% of true proportion. Find the minimum sample size necessary.

Code:

```
alpha = 0.1
me= 5
z_alpha_by_2 = norm.isf(alpha/2)
n = round(( z_alpha_by_2 *sd/me)**2)
n = 4
```

Q19. A researcher claims that currently 25% of all CEOs are above 60 years .Test his claim with an $\alpha = 0.05$ if out of a random sample of 30 CEOs only 10 are above 60 years.

Code:

```
from scipy import stats
from statsmodels.stats.proportion import proportions_ztest

sample_size = 30
num_CEOs = 10
```

```
# Calculate proportion of EV owners in sample
prop = num_CEOs / sample_size
z=stats.norm.ppf(0.05)
# Perform proportion test
z_score, p_value = proportions_ztest(num_CEOs, sample_size, 0.25)
print(z)
# Print results
print("Z-score:", z_score)
print("P-value:", p_value)
-1.6448536269514729
Z-score: 0.968245836551854
P-value: 0.33292160806556603
```

Q20. Assume you are a data scientist for the Fortune 500 companies. You are asked to research the question whether the CEO ages of UK are on average older than the CEO ages of Americans. you take a random sample of 40 CEO ages from America and UK and the data is as follows:

UK

47 49 73 50 65 70 49 47 40 43

46 35 38 40 47 39 49 37 37 36

40 37 31 48 48 45 52 38 38 36

44 40 48 45 45 36 39 44 52 47

USA

47 57 52 47 48 56 56 52 50 40

46 43 44 51 36 42 49 49 40 43

39 39 22 41 45 46 39 32 36 32

32 32 37 33 44 49 44 44 49 32

a. What are your hypotheses?

μ_1 = Average ages of ceos in uk

μ_2 = Average ages of ceos in usa

$H_0: \mu_1 \leq \mu_2$ or $\mu_1 - \mu_2 \leq 0$

$H_1: \mu_1 > \mu_2$ or $\mu_1 - \mu_2 > 0$

b. What significance level will you use?

We will use Significance level of 5%

c. What statistical test will you use?

We will use Z-test for the given datasets.

d. What are the test results? (Assume $s_1 = 8.8$ and $s_2 = 7.8$.)

`stats.levene(UK,USA)`

`LeveneResult (statistic=0.020083022437076308, pvalue=0.8876708478111702)`

`n1=len(UK)`

`s1=8.8`

`x1_b=np.mean(UK)`

`n2=len(USA)`

`s2=7.8`

`x2_b=np.mean(USA)`

`a=0.05`

`test_s,p_val=weightstats.ztest(x1=UK1,x2=USA,value=0,alternative='larger')`

`print('TestStatistic:',test_s,'Pvalue:',p_val)`

`z_c=stats.norm.isf(a)`

`z_c = 1.6448536269514729`

After Performing the test it can be observed that Pvalue is more than alpha(0.05)

e. What is your decision?


We cant find enough evidence to reject null hypothesis.

f. What can you conclude?

So we Fail to reject the hypothesis that the CEO ages of UK are on average older than the CEO ages of Americans

g. Do you feel that using the data given really answers the original question asked?

The given data does not provide conclusive evidence to answer the



original Question of whether the CEO ages of UK are on average older than the CEO ages of Americans.

- h. What other data might be used to answer the question?
- A larger sample size from the population and the standard deviation of population might be useful for answering the question.

PART-B (Dataset Based)--25 points

Topic - Credit Card Fraud Detection

Introduction

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Dataset Description

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.

Data Dictionary

- a) It contains only numeric input variables. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.
- b) Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.
- c) The feature 'Amount' is the transaction Amount,
- d) Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Questions -

1. Import the dataset and view the first 10 rows of it.

```
df=pd.read_csv('creditcard.csv')
```

```
df.head(10)
```

```
df=pd.read_csv('creditcard.csv')
df.head(10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050

10 rows x 31 columns

2. Display shape/dimension of the dataset.

df.shape

df.shape

(284807, 31)

3. Check for the missing values.Display number of missing values per column.

df.isna().sum().sum()

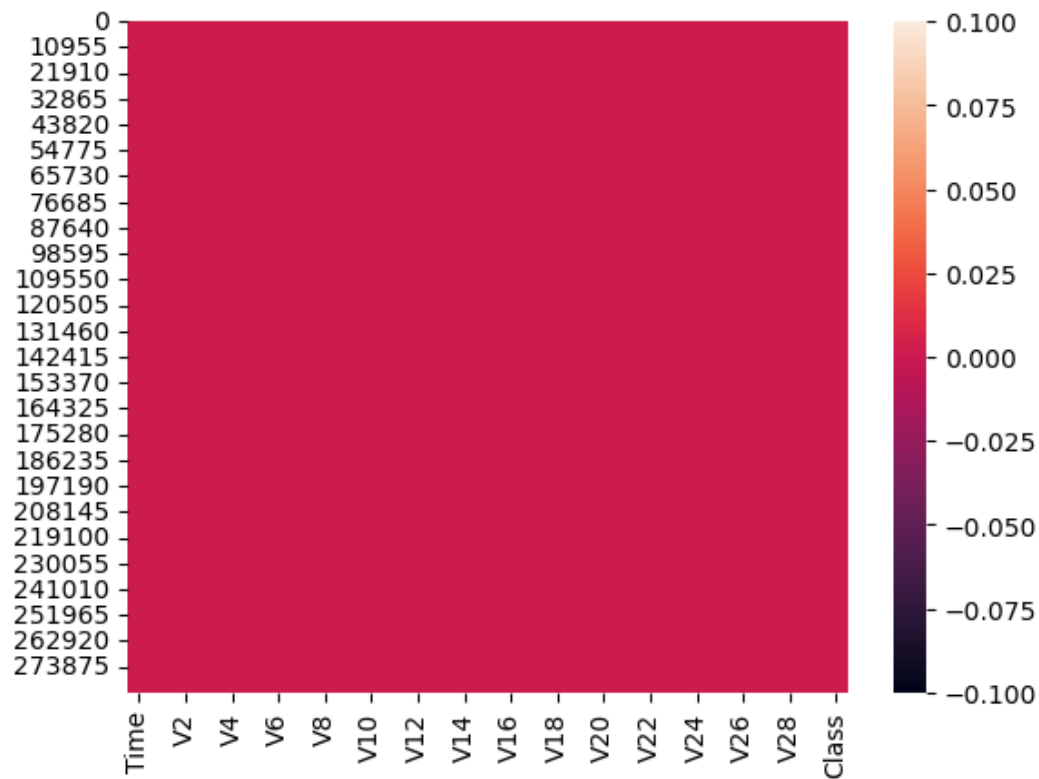
```
df.isna().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Visual Representation of null values:

```
1 sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```



4. Check the datatype, number of non-null values and name of each variable in the dataset.

```
df.info()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Time    284807 non-null   float64
1   V1       284807 non-null   float64
2   V2       284807 non-null   float64
3   V3       284807 non-null   float64
4   V4       284807 non-null   float64
5   V5       284807 non-null   float64
6   V6       284807 non-null   float64
7   V7       284807 non-null   float64
8   V8       284807 non-null   float64
9   V9       284807 non-null   float64
10  V10      284807 non-null   float64
11  V11      284807 non-null   float64
12  V12      284807 non-null   float64
13  V13      284807 non-null   float64
14  V14      284807 non-null   float64
15  V15      284807 non-null   float64
16  V16      284807 non-null   float64
17  V17      284807 non-null   float64
18  V18      284807 non-null   float64
19  V19      284807 non-null   float64
20  V20      284807 non-null   float64
21  V21      284807 non-null   float64
22  V22      284807 non-null   float64
23  V23      284807 non-null   float64
24  V24      284807 non-null   float64
25  V25      284807 non-null   float64
26  V26      284807 non-null   float64
27  V27      284807 non-null   float64
28  V28      284807 non-null   float64
29  Amount   284807 non-null   float64
30  Class    284807 non-null   int64
dtypes: float64(30), int64(1)
```

```
convert_dict = {'Class': object,
                }
```

```
df = df.astype(convert_dict)
print(df.dtypes)
```

```
Time      float64
V1         float64
V2         float64
V3         float64
V4         float64
V5         float64
V6         float64
V7         float64
V8         float64
V9         float64
V10        float64
V11        float64
V12        float64
V13        float64
V14        float64
V15        float64
V16        float64
V17        float64
V18        float64
V19        float64
V20        float64
V21        float64
V22        float64
V23        float64
V24        float64
V25        float64
V26        float64
V27        float64
V28        float64
Amount     float64
Class      object
dtype: object
```

5. Check if there are any non-real characters in the dataset.

```
r=df.applymap(np.isreal)
```

`r.isna().sum()`

```
r=df.applymap(np.isreal)
r.isna().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

6. Check the descriptive statistics of the dataset.

`df.describe()`

```
df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918840e-15	5.682688e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.688953e-15	-1.893285e-16	-3.147640e-15
std	47488.145955	1.958898e+00	1.651309e+00	1.516255e+00	1.415889e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903848e-01	-8.488401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84892.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-01
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+02

8 rows x 10 columns

7. Check the number of fraudulent transactions in the dataset and visualize using pie chart and bar chart.

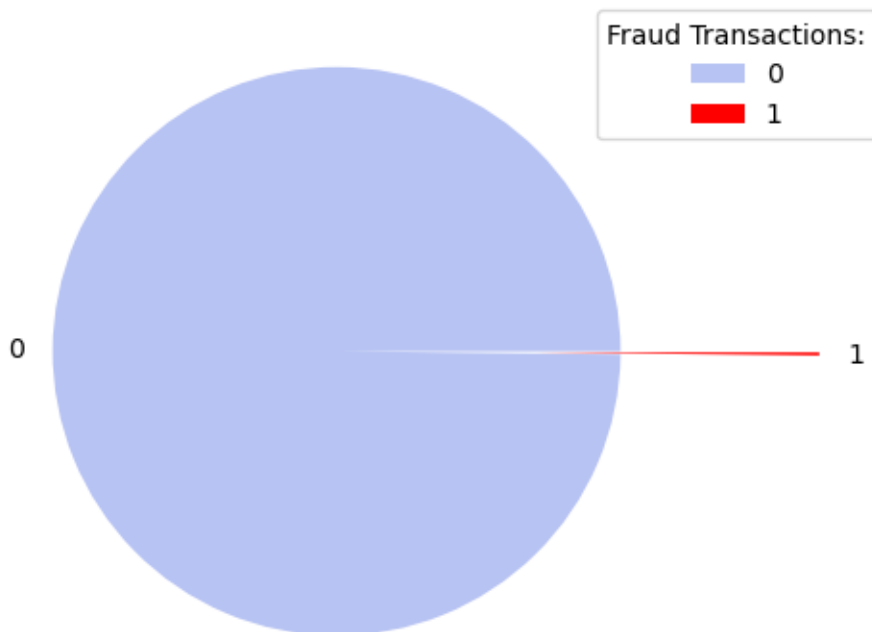
```
colors = ['#B7C3F3','red']
```

```
plt.pie(x=df['Class'].value_counts(),labels=df['Class'].value_counts().index,explode
```

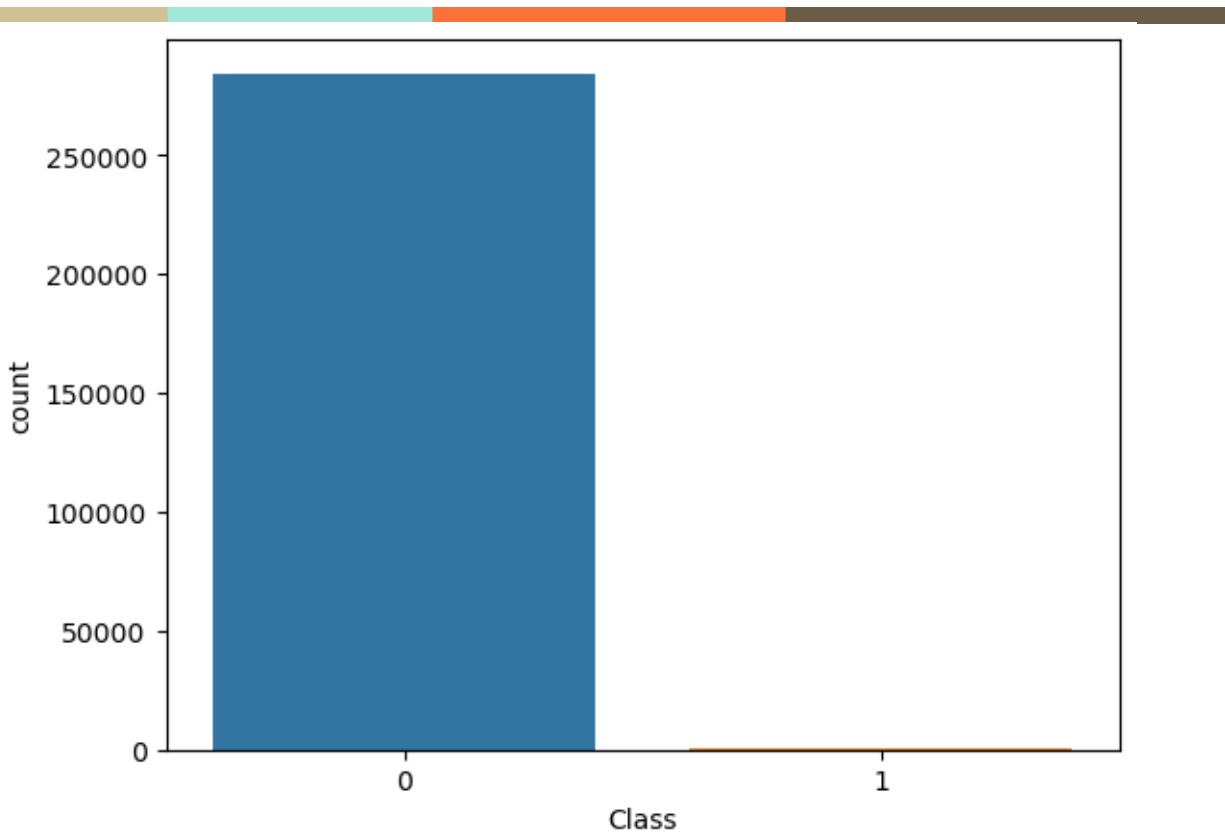
```
e=(0.7,0),colors=colors)
```

```
plt.legend(title = "Fraud Transactions:")
```

```
print(df['Class'].value_counts())
```



```
sns.countplot(x='Class',data=df)
```



8. Check the maximum and minimum fraudulent amount.

```
fraud=df[df['Class']==1]
```

```
fraud['Amount'].max()  
2125.87
```

```
fraud['Amount'].min()  
0.0
```

9. Check the number of transactions where the transaction amount is zero and consider as a

fraud transaction.

```
fraud[fraud['Amount']==0]
```


fraud[fraud['Amount']!=0]

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
541	406.0	-2.312227	1.951992	-1.609851	3.997908	-0.522188	-1.426545	-2.537387	1.391657	-2.770089	...	0.517232	-0.035049	-0.4
8842	12093.0	-4.696795	2.693867	-4.475133	5.467685	-1.556758	-1.549420	-4.104215	0.553934	-1.498468	...	0.573898	-0.080163	0.3
23308	32686.0	0.287953	1.728735	-1.652173	3.813544	-1.090927	-0.984745	-2.202318	0.555088	-2.033892	...	0.262202	-0.633528	0.0
42756	41233.0	-10.645800	5.918307	-11.671043	8.807369	-7.975501	-3.586806	-13.616797	8.428189	-7.368451	...	2.571970	0.206809	-1.6
69980	53658.0	-1.739341	1.344521	-0.534379	3.195291	-0.416196	-1.261961	-2.340991	0.713004	-1.416265	...	0.383180	-0.213952	-0.3
93486	64443.0	1.079524	0.872988	-0.303850	2.755399	0.301688	-0.350284	-0.042848	0.246625	-0.779176	...	-0.023255	-0.158601	-0.0
93788	64585.0	1.080433	0.962831	-0.278065	2.743318	0.412384	-0.320778	0.041290	0.176170	-0.966952	...	-0.008996	-0.057036	-0.0
141257	84204.0	-0.937843	3.462889	-6.445104	4.932199	-2.233983	-2.291561	-5.695594	1.338825	-4.322377	...	1.066550	-0.521657	-0.3
141258	84204.0	-0.937843	3.462889	-6.445104	4.932199	-2.233983	-2.291561	-5.695594	1.338825	-4.322377	...	1.066550	-0.521657	-0.3
143333	85285.0	-7.030308	3.421991	-9.525072	5.270891	-4.024630	-2.866682	-6.989195	3.791551	-4.622730	...	1.103398	-0.541855	0.0

10. Check the distribution of columns. List down columns that are normally distributed. List down columns that are not normally distributed.

```
plt.figure(figsize=(15,8))
```

```
j=1
```

```
for i in df.columns:
```

```
    if df[i].dtypes == 'float64' or df[i].dtypes == 'int64':
```

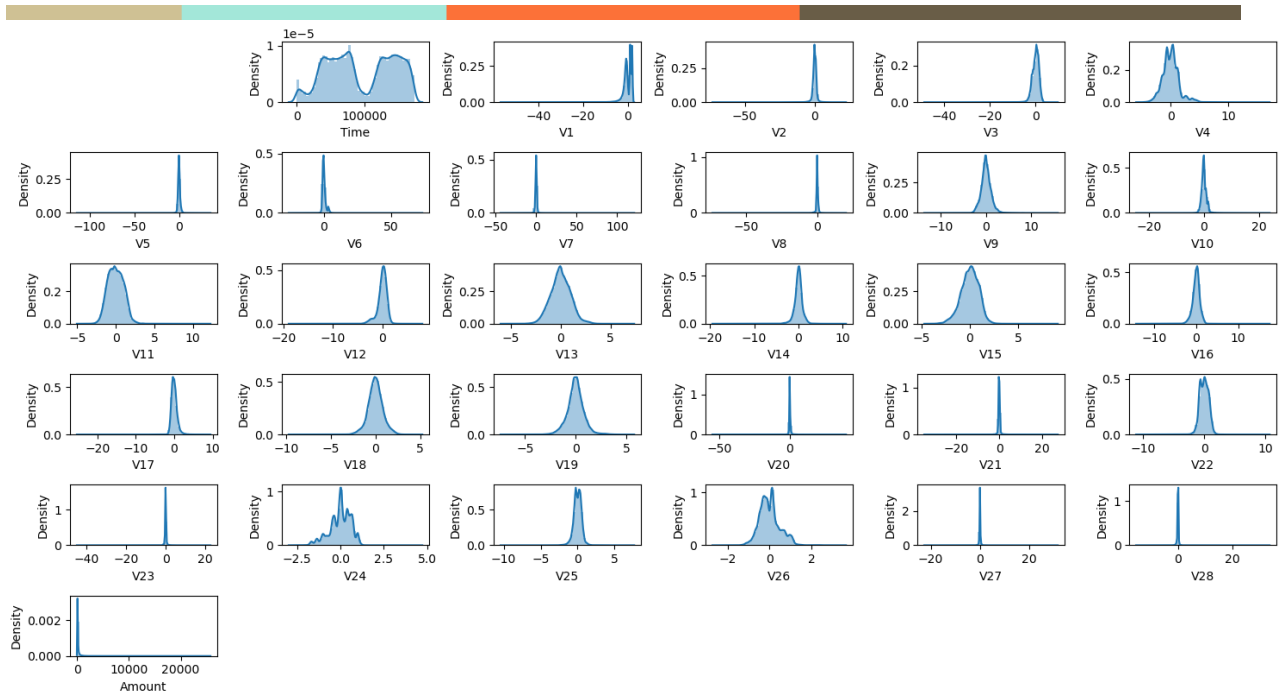
```
        plt.subplot(6,6,j+1)
```

```
        sns.distplot(df[i])
```

```
        j+=1
```

```
plt.tight_layout()
```

```
plt.show()
```



11. List down columns that are highly skewed.

```
df_n=df.select_dtypes(include=np.number)
```

```
df_n.head()
```

```
df_n.skew()
```

```
df_n.skew()
```

```
Time      -0.035568
V1        -3.280667
V2        -4.624866
V3        -2.240155
V4         0.676292
V5        -2.425901
V6         1.826581
V7         2.553907
V8        -8.521944
V9         0.554680
V10        1.187141
V11        0.356506
V12       -2.278401
V13        0.065233
V14       -1.995176
V15       -0.308423
V16       -1.100966
V17       -3.844914
V18       -0.259880
V19        0.109192
V20       -2.037155
V21        3.592991
V22       -0.213258
V23       -5.875140
V24       -0.552499
V25       -0.415793
V26        0.576693
V27       -1.170209
V28       11.192091
Amount    16.977724
dtype: float64
```

12. With the help of a standard scaler, normalize the respective column distribution.

```
df_n.std()
```

```
df_n.std()
```

```
Time      47488.145955
V1        1.958696
V2        1.651309
V3        1.516255
V4        1.415869
V5        1.380247
V6        1.332271
V7        1.237094
V8        1.194353
V9        1.098632
V10       1.088850
V11       1.020713
V12       0.999201
V13       0.995274
V14       0.958596
V15       0.915316
V16       0.876253
V17       0.849337
V18       0.838176
V19       0.814041
V20       0.770925
V21       0.734524
V22       0.725702
V23       0.624460
V24       0.605647
V25       0.521278
V26       0.482227
V27       0.403632
V28       0.330083
Amount    250.120109
dtype: float64
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled_df=pd.DataFrame(sc.fit_transform(df_n))
scaled_df.columns=df_n.columns
scaled_df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23
0	-1.998583	-0.604242	-0.044075	1.672773	0.973366	-0.245117	0.347068	0.193679	0.062837	0.331123	...	0.326118	-0.024923	0.382854	-0.176911
1	-1.998583	0.608498	0.161176	0.109797	0.316523	0.043483	-0.061820	-0.063700	0.071253	-0.232494	...	-0.089611	-0.307377	-0.880077	0.162201
2	-1.998582	-0.603500	-0.811578	1.169468	0.288231	-0.364572	1.351454	0.639776	0.207373	-1.378675	...	0.680975	0.337632	1.063358	1.456320
3	-1.998582	-0.493325	-0.112169	1.182516	-0.609727	-0.007499	0.936150	0.192071	0.316018	-1.262503	...	-0.269855	-0.147443	0.007267	-0.304777
4	-1.998541	-0.591330	0.531541	1.021412	0.284655	-0.295015	0.071999	0.479302	-0.226510	0.744325	...	0.529939	-0.012839	1.100011	-0.220123

5 rows × 30 columns

```
scaled_df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
count	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	-1.050379e-14	-7.894410e-16	2.647157e-17	-4.302564e-15	-6.662098e-16	-2.586322e-16	4.147497e-16	-8.820879e-16	-2.466016e-16	3.102531e-15
std	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00
min	-1.998583e+00	-2.879855e+01	-4.403529e+01	-3.187173e+01	-4.013919e+00	-8.240810e+01	-1.963606e+01	-3.520940e+01	-6.130252e+01	-1.222802e+02
25%	-8.552120e-01	-4.698918e-01	-3.624707e-01	-5.872142e-01	-5.993788e-01	-5.010686e-01	-5.766822e-01	-4.478860e-01	-1.746805e-01	-5.853631e-01
50%	-2.131453e-01	9.245351e-03	3.985683e-02	1.188124e-01	-1.401724e-02	-3.936882e-02	-2.058046e-01	3.241723e-02	1.871982e-02	-4.681189e-01
75%	9.372174e-01	6.716939e-01	4.867202e-01	6.774569e-01	5.250082e-01	4.433465e-01	2.991625e-01	4.611107e-01	2.740785e-01	5.435305e-01
max	1.642058e+00	1.253351e+00	1.335775e+01	6.187993e+00	1.191874e+01	2.521413e+01	5.502015e+01	9.747824e+01	1.675153e+01	1.419494e+02

8 rows × 30 columns

```
scaled_df.std()
```

```
Time      1.000002
V1        1.000002
V2        1.000002
V3        1.000002
V4        1.000002
V5        1.000002
V6        1.000002
V7        1.000002
V8        1.000002
V9        1.000002
V10       1.000002
V11       1.000002
V12       1.000002
V13       1.000002
V14       1.000002
V15       1.000002
V16       1.000002
V17       1.000002
V18       1.000002
V19       1.000002
V20       1.000002
V21       1.000002
V22       1.000002
V23       1.000002
V24       1.000002
V25       1.000002
V26       1.000002
V27       1.000002
V28       1.000002
Amount    1.000002
dtype: float64
```

13. List down columns that have high kurtosis.

```
df_n.kurtosis()
```

```
df_n.kurtosis()
```

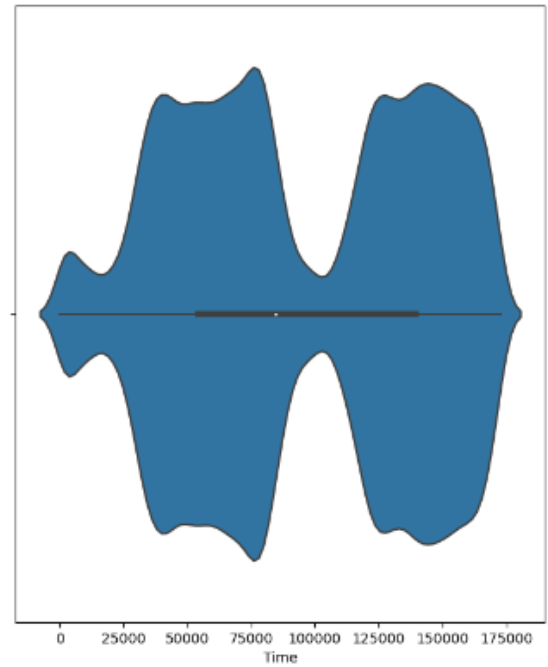
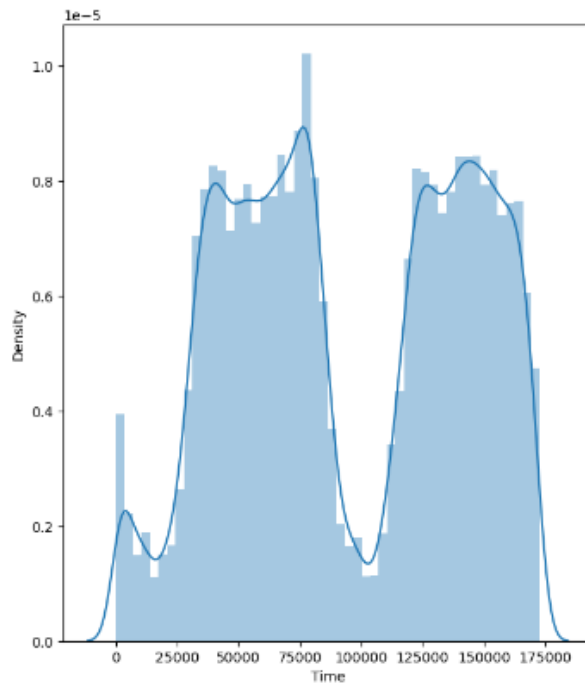
```
Time      -1.293530
V1         32.486679
V2         95.773106
V3         26.619551
V4          2.635455
V5        206.904560
V6         42.642494
V7        405.607417
V8        220.586974
V9          3.731311
V10        31.988239
V11         1.633921
V12        20.241870
V13         0.195300
V14        23.879462
V15         0.284769
V16        10.419131
V17        94.799719
V18         2.578341
V19         1.724970
V20       271.016113
V21       207.287040
V22         2.832967
V23       440.088659
V24         0.618871
V25         4.290412
V26         0.919006
V27       244.989241
V28       933.397502
Amount    845.092646
dtype: float64
```

14. What is the distribution of Time and Amount columns in the dataset ?

```
plt.figure(figsize=(15,8))
plt.subplot(1, 2, 1)
sns.distplot(df['Time'])

plt.subplot(1, 2, 2)
sns.violinplot(x=df['Time'])

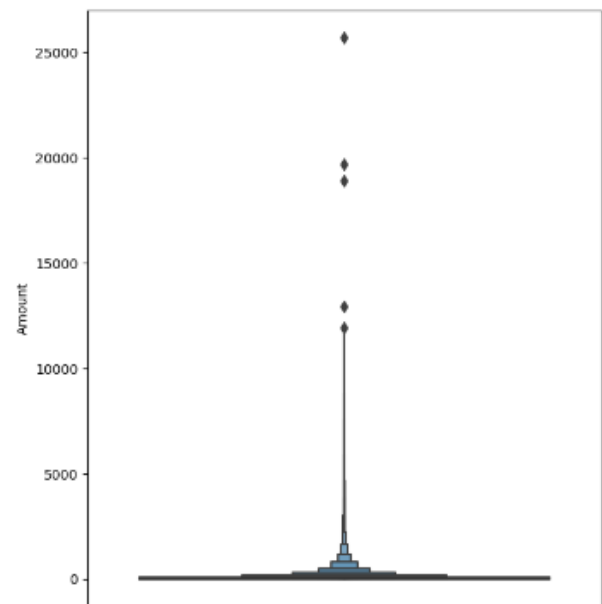
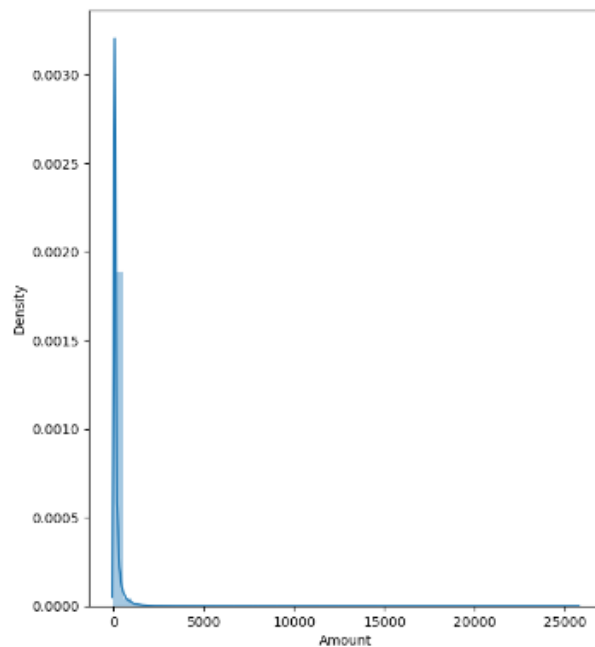
plt.show()
```



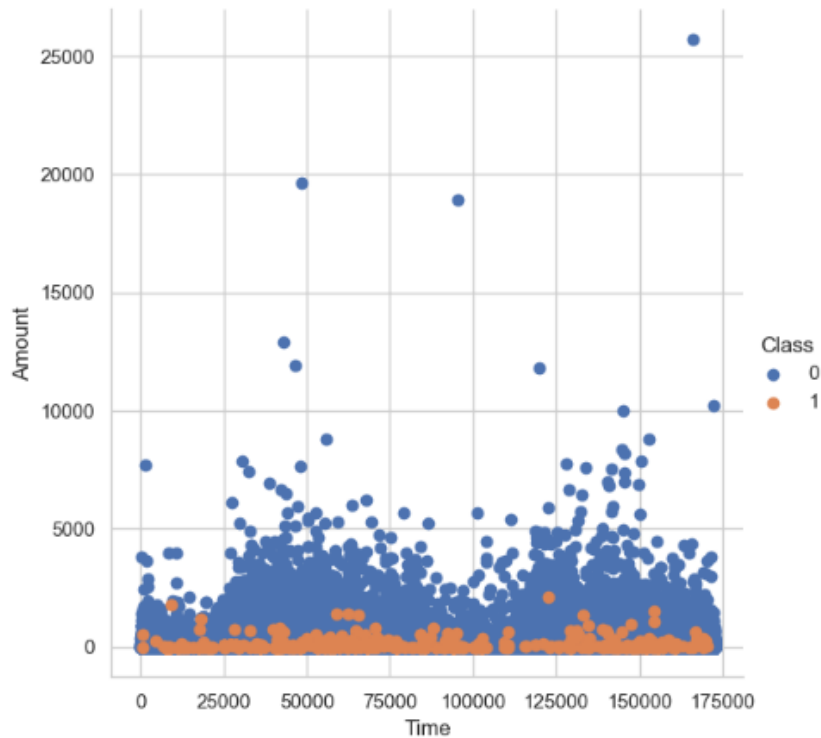
```
plt.figure(figsize=(15,8))
plt.subplot(1, 2, 1)
sns.distplot(df['Amount'])

plt.subplot(1, 2, 2)
sns.boxenplot(y=df['Amount'])

plt.show()
```



```
sns.set_style("whitegrid")
sns.FacetGrid(df, hue="Class", height = 6).map(plt.scatter, "Time", "Amount").add_legend()
plt.show()
```



INFERENCE:

From the above two plots it is clearly visible that there are frauds only on the transactions which have

transaction amount approximately less than 2500. Transactions which have transaction amount approximately above 2500 have no fraud.

As per with the time, the frauds in the transactions are evenly distributed throughout time.

15. Find the distribution of all variables with respect to the outcome 'Class' variable.


```

col = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9',
       'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18',
       'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27',
       'V28']

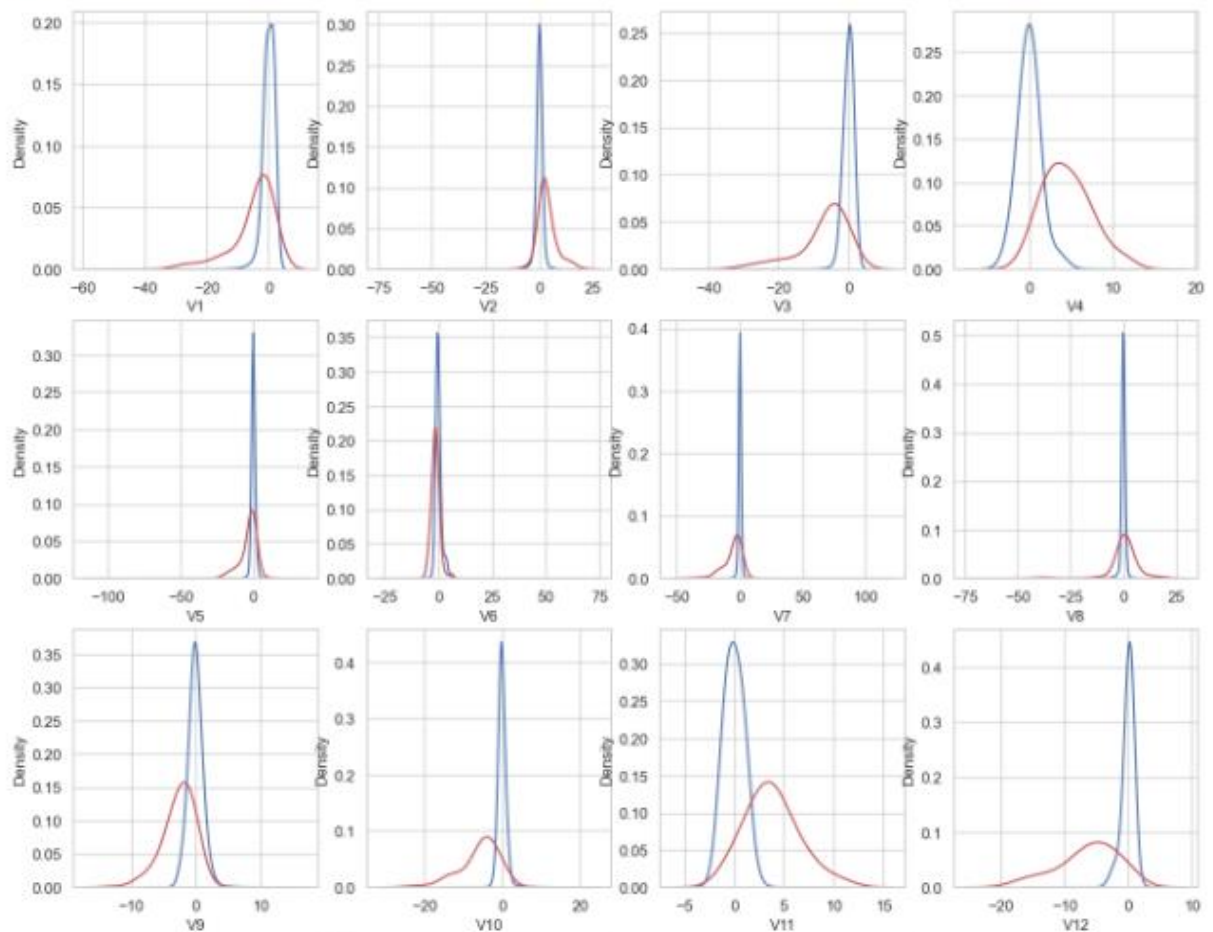
i = 0
t0 = df.loc[df['Class'] == 0]
t1 = df.loc[df['Class'] == 1]

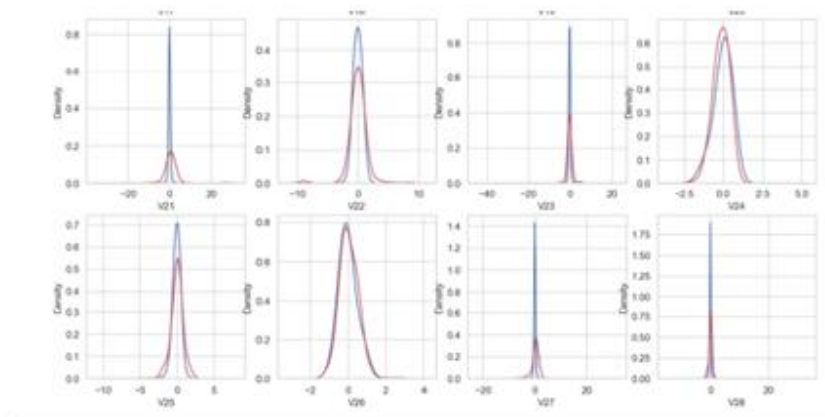
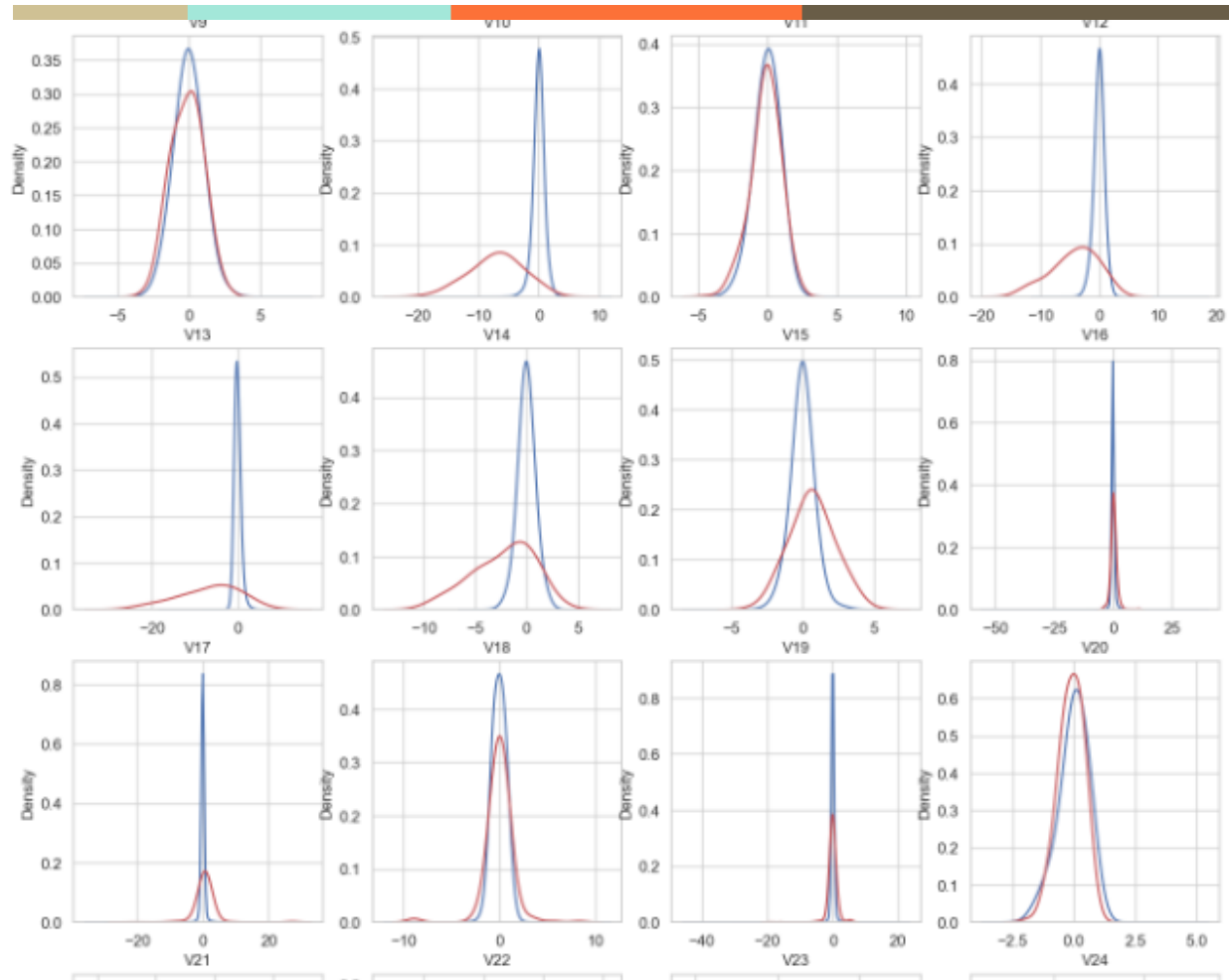
sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(8,4,figsize=(16,30))

for feature in col:
    i += 1
    plt.subplot(7,4,i)
    sns.kdeplot(t0[feature], bw=0.5, label="Class = 0", color='b')
    sns.kdeplot(t1[feature], bw=0.5, label="Class = 1", color='r')
    plt.xlabel(feature, fontsize=12)
    locs, labels = plt.xticks()
    plt.tick_params(axis='both', which='major', labelsize=12)
plt.show();

```

<Figure size 640x480 with 0 Axes>





V9, V4, V10, V11, V17-V19 have clearly separated distributions for Class values 0 and 1

V1, V2, V7, V9, V12, V14, V16, V18 have partially separated distribution for Class 0 and 1

V13, V15, V20, V22-V28 have almost similar distribution for Class 0 and 1

V5, V6, V8, V21 have quite similar distribution for Class 0 and 1

In general, with just few exceptions (Time and Amount), the features distribution for legitimate transactions (values of Class = 0) is centered around 0, sometime with a long queue at one of the extremities. In the same time, the fraudulent transactions (values of Class = 1) have a skewed (asymmetric) distribution.

16. Create a countplot for the outcome class in seaborn using percentage instead of count for each bar.

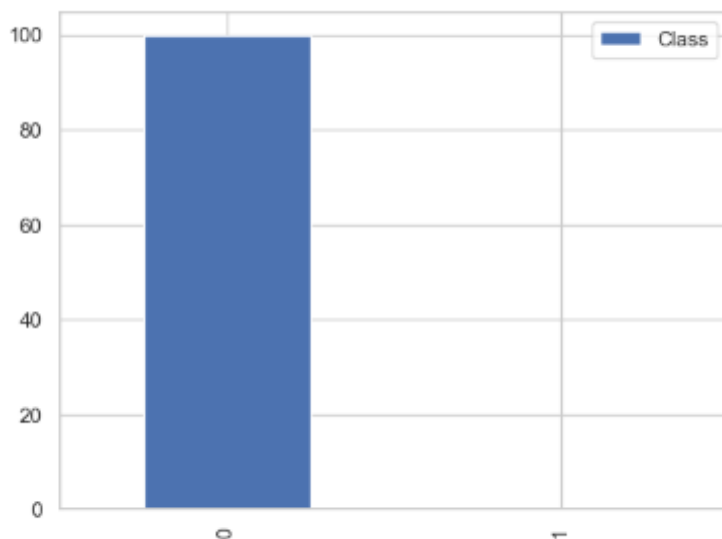
```
c_p=df['Class'].value_counts(normalize=True).mul(100).rename({0: "Not Fraud", 1: "Fraud"}).reset_index()
print(c_p)
c_p.plot.bar()
```

```

   index  Class
0  Not Fraud  99.827251
1   Fraud    0.172749

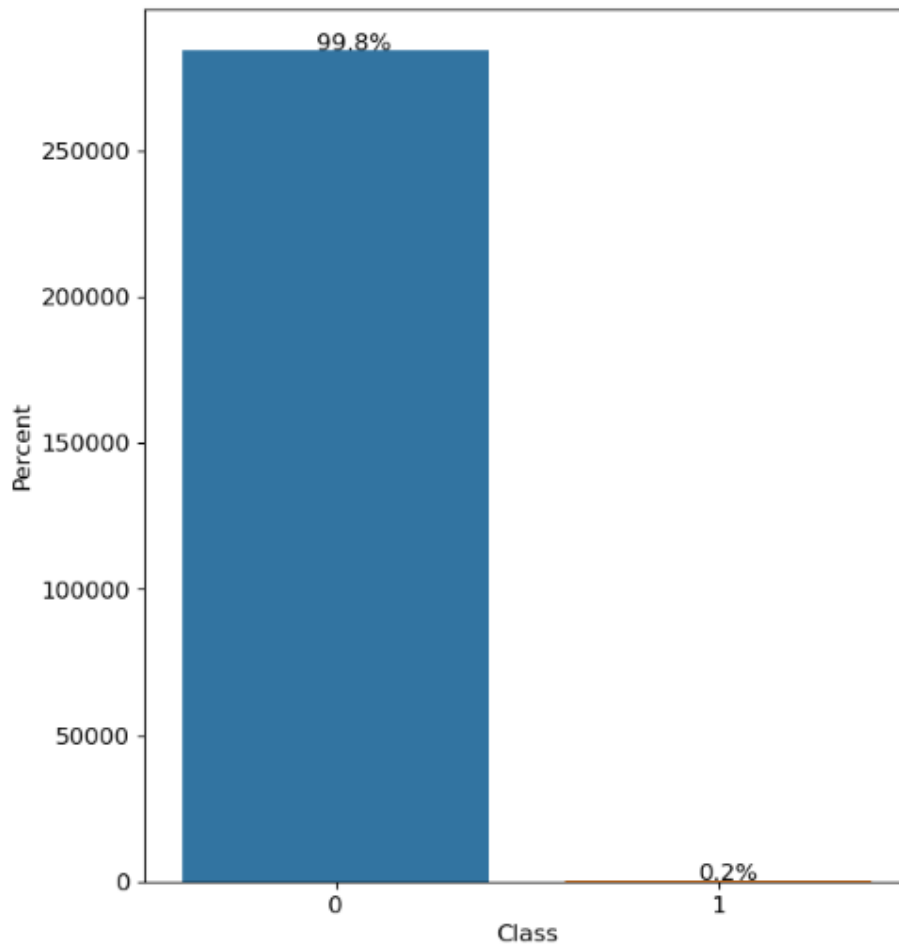
```

<AxesSubplot:>



```
def w_h(ax, feature):
    total = len(feature)
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total)
        x = p.get_x() + p.get_width() / 2 - 0.05
        y = p.get_y() + p.get_height()
        ax.annotate(percentage, (x, y), size = 12)
```

```
plt.figure(figsize=(7,8))
ax=sns.countplot(x='Class',data=df)
plt.xticks(size=12)
plt.xlabel('Class',size=12)
plt.yticks(size=12)
plt.ylabel('Percent',size=12)
w_h(ax,df.Class)
```

**INFERENCE:**

Only 492 (or 0.2%) of transaction are fraudulent.

That means the data is highly unbalanced with respect with target variable Class.

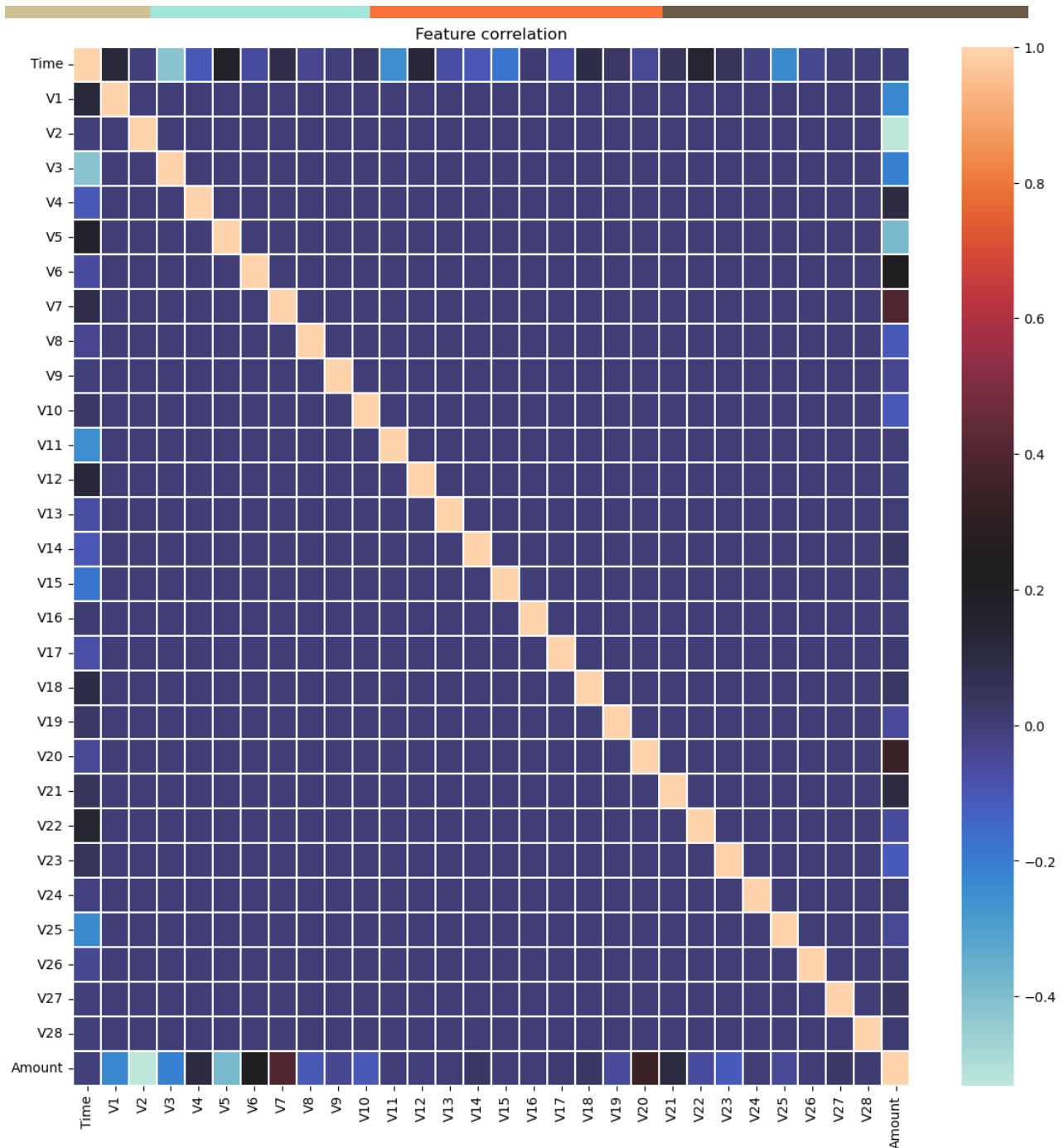
17. Plot a heatmap for correlation matrix for the given dataset. Write the Observation. Especially note down columns that are highly correlated (Positive and Negative Correlation, Consider 0.7 to 1 as high).

```
plt.figure(figsize = (14,14))
```

```
plt.title('Feature correlation')
```

```
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.1,cmap='icefire')
```

```
plt.show()
```



18. With the help of hypothesis testing check whether fraudulent transactions of higher value than normal transactions?

```
p = df[df['Class'] == 0]['Amount']
```

```
s = df[df['Class'] == 1]['Amount']
```

```
n=len(s)
```

```
x_b = s.mean()
```

```
sd = p.std()
```

```
mu = p.mean()
```

```
a=0.01
```

```
z_stat = (x_b - mu) / (sd/ n ** 0.5)
```

```
z_stat
```

```
3.008289898215099
```

```
z_critical=stats.norm.isf(a)
```

```
z_critical
```

```
2.3263478740408408
```

```
# As the z_stat is more than z_critical we reject the Null hypothesis(H0).
```

```
# So there is a 99% chance that the amount spend on fraudulent transactions are  
on average significantly lower than normal.
```

19. Perform ANOVA test for Statistical feature selection.

```
| # H0: ALL the mean class amount are same.  
| # H1: ALL the mean class amount are not same.
```

```
| t = df['Class'].nunique()  
| print('t:', t)  
  
| N = df['Class'].value_counts().sum()  
| print('N:', N)
```

```
t: 2  
N: 284807
```

```
| f_critical = round(stats.f.isf(q = 0.05, dfn = 1, dfd = 284805), 4)  
| print('Critical value for F-test:', f_critical)
```

```
Critical value for F-test: 3.8415
```

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

test = ols('Amount ~ Class',df).fit()

anova_1 = anova_lm(test)

# print the table
anova_1
```

	df	sum_sq	mean_sq	F	PR(>F)
Class	1.0	5.651107e+05	565110.729178	9.033345	0.002651
Residual	284805.0	1.781692e+10	62558.304503	NaN	NaN

```
# The above output shows that the p-value is Less than 0.05.
# Thus we reject the null hypothesis.
```

```
import statsmodels.stats.multicomp as mc
import scikit_posthocs
comp = mc.MultiComparison(data=df['Amount'],groups=df['Class'])

# tukey's range test
post_hoc = comp.tukeyhsd(alpha=0.05)

# print the summary table
post_hoc.summary()
```

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	33.9203	0.0027	11.8004	56.0402	True

INFERENCE:

The reject=True for group1 and group2 denotes that we reject the null hypothesis and conclude that the average amount is not the same.

The values in the columns lower and upper represent the lower and upper bound of the 95% confidence interval for the mean difference.

- Split the dataset randomly into train and test datasets. Use a train -testratio of 70:30 ratio.

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop(['Class'], axis=1)
```

```
Y = df['Class']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
```

```
random_state=42)

print(X.shape, Y.shape)
(284807, 30) (284807,)

print('X train shape: ', X_train.shape)

print('X test shape: ', X_test.shape)

print('y train shape: ', y_train.shape)

print('y test shape: ', y_test.shape)
X train shape: (199364, 30)
X test shape: (85443, 30)
y train shape: (199364,)
y test shape: (85443,)

print(y_test.value_counts())
0      85307
1       136
Name: Class, dtype: int64
```

21. These are just checkpoints. Please use your best analytical approach to build this report. You can mix match columns to create new ones which can be

used for better analysis. Create your own features if required. Be highly experimental and analytical here to find hidden patterns. You can use the following as checklist pointers :

- What is the shape and size of the dataset?
 - Which columns are highly skewed?
 - Which columns are highly Kurtosis driven?
 - Which columns have Wrong data type?
 - What columns seem to have outliers based on min, max and percentile values, IQR range along with the standard deviation and mean absolute deviation?
 - What columns have missing values? (Check the Missing Values section in Pandas Profiling)
 - What columns have high amount of zero and make sure that these zeroes are supposed to be there(for eg. Weight cannot be zero and any percentage of zero in column zero is erroneous)
 - What columns have high variance and standard deviation?
 - Comment on the distribution of the continuous values (Real Number: $\mathbb{R} \geq 0$)
 - Do you see any alarming trends in the extreme values (minimum 5 and maximum 5)?
 - How many Boolean columns are there in the data set and out of those how many are imbalanced?
 - Check for duplicate records across all columns (Check Warning Section)
 - Is there any imbalance in the categorical columns? (for example Gender Male and Female in which Male is 95% and Female is just 5%- How many columns are categorical?)
 - Are those categories in sync with the domain categories?
 - Check if all the categories are unique and they represent distinct information
- Based on the above questions and your observations, chart out a plan for Data Pre-processing and feature engineering

Note: Feature Engineering (Feature Selection and Feature Creation)

- The dataset contains transactions made by credit cards by European cardholders.
- This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced. The positive

class (fraud transactions) accounts for 0.172% of all transactions.

- It contains only numeric input variables, which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features and more background information about the data are not provided. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount.' Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount. This feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable, and it takes value 1 in case of fraud and 0 otherwise.
- The shape of the dataframe is(284807, 31).
- As noted earlier, the dataset is highly skewed, which can be seen from the below bar plot. Only 492 (or 0.172%) of the transactions are fraudulent. That means the data is highly unbalanced with respect to the target variable Class.
- Fraudulent transactions have an even distribution compared to non-fraudulent transactions. It means that fraudsters continue to operate at abnormal times as well, compared to the real customers who mostly transact between business hours.
- Genuine or real transactions have a larger average value, larger first quartile or Q1, smaller third quartile, and fourth quartile or Q3, Q4 respectively, besides large outliers. Fraud transactions, on the other hand, have a smaller first quartile (Q1) and mean larger fourth quartile (Q4), and smaller outliers.
- Let us now plot the number of fraud transactions by time; we see that a lot of these transactions are outliers, which means that fraud is skewed towards larger transactions.
- We can see that there is no correlation between the independent variables, and that is expected as the data providers have done PCA on the input variables except Time and Amount (PCA -> Principal component analysis). There are some small correlations between some of the independent variables and Time (inverse correlation with V3) and Amount (good correlation with V7 and V20, inverse correlation with V1 and V5).
- For some of the independent variables, we can see a good separation between the distributions of the two classes that is fraud versus nonfraud. For example, V4 and V11 have different distributions for both classes. V12, V14, and V18 are somewhat separated. V1, V2, V3, V10 have quite a distinct distribution, while V25, V26, V28 have similar distribution for both the classes.
- As a general statement, with the exception of the two independent variables, Time and Amount, the distribution for non-fraud transactions is centered on zero, sometimes with a long tail for the extreme or outlier observations. At the same time, fraud transactions have a skewed distribution.
- V3, V4, V10, V11, V17-V19 have clearly separated distributions for Class values 0 and 1
- V1, V2, V7, V9, V12, V14, V16, V18 have partially saperated distribution for Class 0



and 1

- V13, V15, V20, V22-V28 have almost similar distribution for Class 0 and 1
- V5, V6, V8, V21 have quite similar distribution for Class 0 and 1
- In general, with just few exceptions (Time and Amount), the features distribution for legitimate transactions (values of Class = 0) is centered around 0, sometime with a long queue at one of the extremities. In the same time, the fraudulent transactions (values of Class = 1) have a skewed (asymmetric) distribution.
- It is clearly visible that there are frauds only on the transactions which have transaction amount approximately less than 2500. Transactions which have transaction amount approximately above 2500 have no fraud. As per with the time, the frauds in the transactions are evenly distributed throughout time.
- V1 to V28 variables are normally distributed. Variable 'Time' and 'Amount' do not follow normal distribution

THANK YOU.