In [23]:
```
#import Libraries

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

In [2]:
```
# Read csv file

with open("C:/vinay/Python/Credit card Fraud Detection/creditcard_2023.csv"
    df=pd.read_csv(creditcard, index_col=None)
```

In [3]:
```
# Set pandas display options
pd.set_option('display.max_columns', None)

# Display sample of csv file
df.head()
```

Out[3]:

	id	V1	V2	V3	V4	V5	V6	V7	V8	
0	0	-0.260648	-0.469648	2.496266	-0.083724	0.129681	0.732898	0.519014	-0.130006	0.7
1	1	0.985100	-0.356045	0.558056	-0.429654	0.277140	0.428605	0.406466	-0.133118	0.3
2	2	-0.260272	-0.949385	1.728538	-0.457986	0.074062	1.419481	0.743511	-0.095576	-0.2
3	3	-0.152152	-0.508959	1.746840	-1.090178	0.249486	1.143312	0.518269	-0.065130	-0.2
4	4	-0.206820	-0.165280	1.527053	-0.448293	0.106125	0.530549	0.658849	-0.212660	1.0

Data cleaning

In [4]:
```
# check for null values

Null_values=df.isnull().any().any()
print(f'Null values in the data frame : {Null_values}')
```

```
Null values in the data frame : False
```

In [5]:
```
# check for duplicate values

duplicate_values=df[df.duplicated()]
print(f'Duplicate values in the data frame : {len(duplicate_values)}')
```

```
Duplicate values in the data frame : 0
```

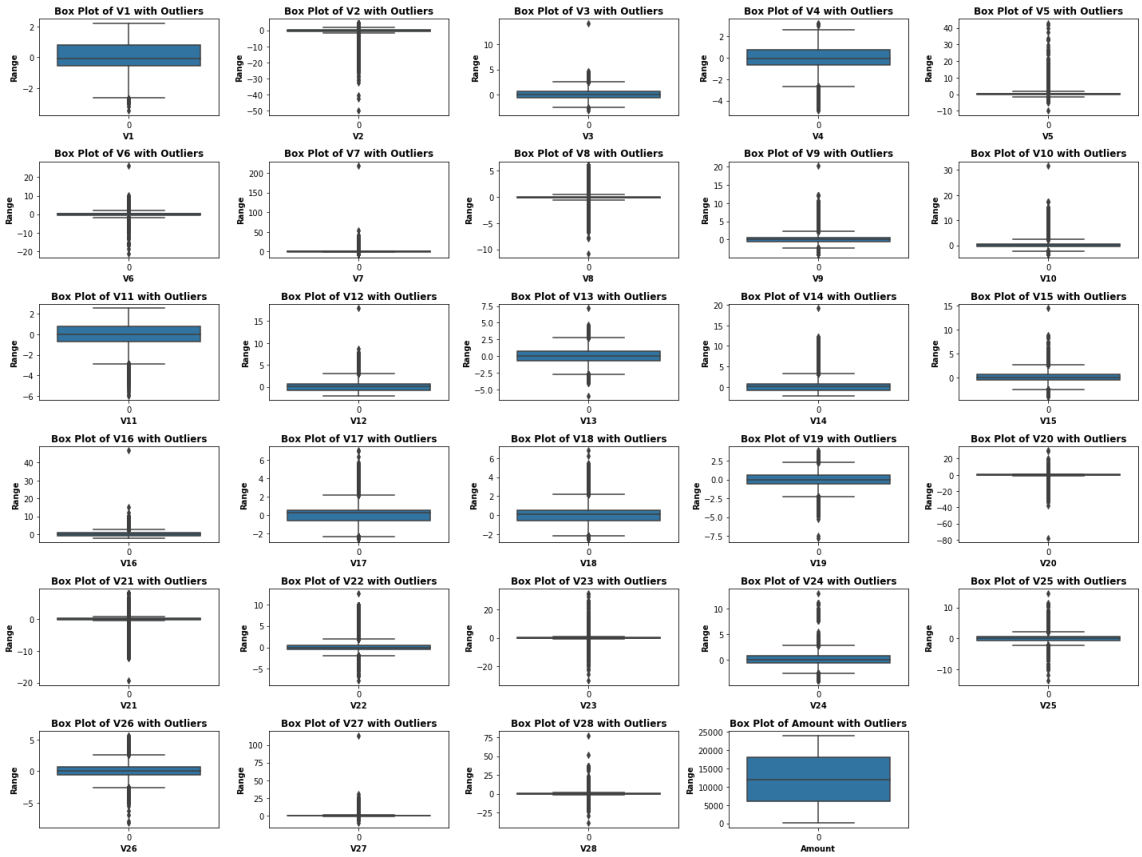
In [6]:
```
#check for data types

data_types=df.info()
print(data_types)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568630 entries, 0 to 568629
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      568630 non-null  int64  
 1   V1      568630 non-null  float64
 2   V2      568630 non-null  float64
 3   V3      568630 non-null  float64
 4   V4      568630 non-null  float64
 5   V5      568630 non-null  float64
 6   V6      568630 non-null  float64
 7   V7      568630 non-null  float64
 8   V8      568630 non-null  float64
 9   V9      568630 non-null  float64
 10  V10     568630 non-null  float64
 11  V11     568630 non-null  float64
 12  V12     568630 non-null  float64
 13  V13     568630 non-null  float64
 14  V14     568630 non-null  float64
 15  V15     568630 non-null  float64
 16  V16     568630 non-null  float64
 17  V17     568630 non-null  float64
 18  V18     568630 non-null  float64
 19  V19     568630 non-null  float64
 20  V20     568630 non-null  float64
 21  V21     568630 non-null  float64
 22  V22     568630 non-null  float64
 23  V23     568630 non-null  float64
 24  V24     568630 non-null  float64
 25  V25     568630 non-null  float64
 26  V26     568630 non-null  float64
 27  V27     568630 non-null  float64
 28  V28     568630 non-null  float64
 29  Amount  568630 non-null  float64
 30  Class   568630 non-null  int64  
dtypes: float64(29), int64(2)
memory usage: 134.5 MB
None
```

In [7]:
```
#Extract columns for outliers
df_columns=df.iloc[:,1:30].columns
print(df_columns)
```

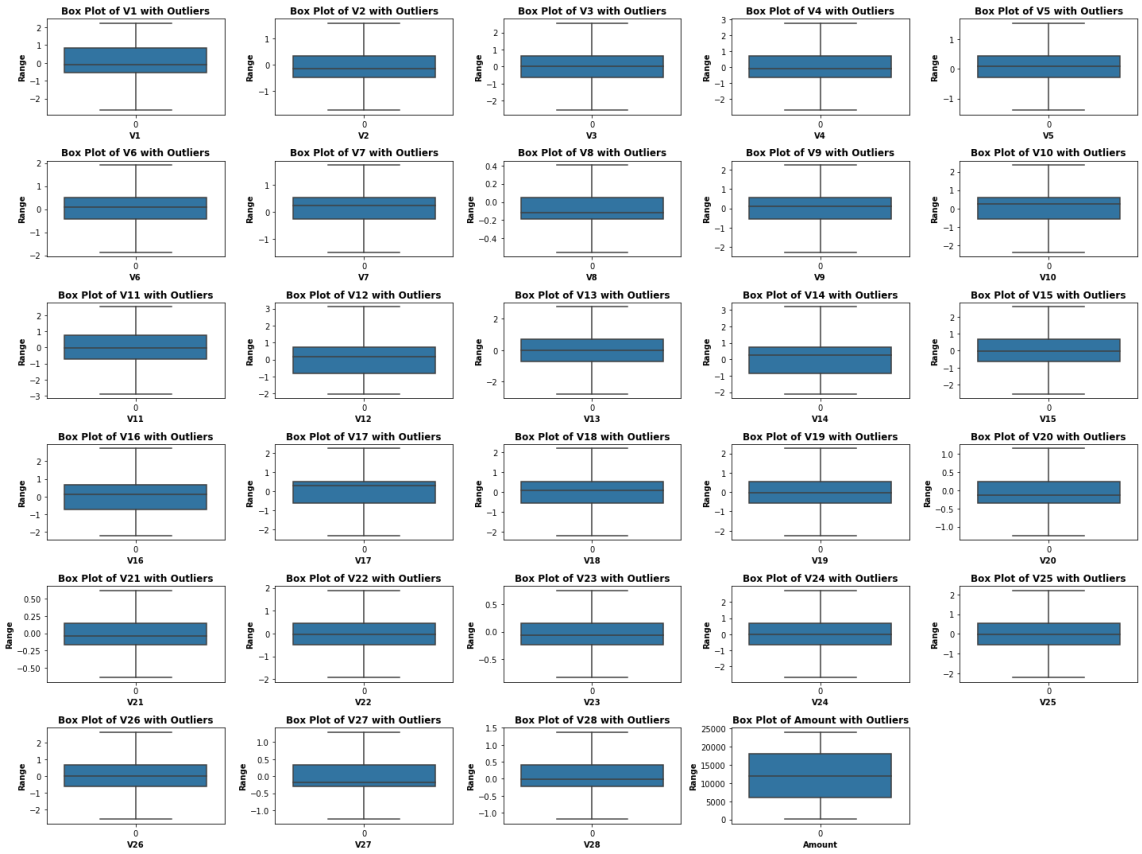
```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V2
1',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'],
      dtype='object')
```

In [8]:
```
num_rows = 6
num_cols = 5

# Create a grid of subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, 15))

# Flatten the axes array for easier iteration
axes = axes.flatten()

# Iterate through each column and create a box plot in the corresponding su
for i, column in enumerate(df_columns):
    sns.boxplot(data=df[column], ax=axes[i])
    axes[i].set_xlabel(column,fontweight='bold')
    axes[i].set_ylabel('Range',fontweight='bold')
    axes[i].set_title(f"Box Plot of {column} with Outliers",fontweight='bol

# remove excess subplots
fig.delaxes(axes[-1])

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plots
plt.show()
```



In [10]:
```
for column in df_columns:

    #calculate quartiles and IQR
    Q1=df[column].quantile(0.25)
    Q3=df[column].quantile(0.75)

    IQR=Q3-Q1

    #calculate upper and lower limits
    upper_limit=Q3+1.5*IQR
    lower_limit=Q1-1.5*IQR

    #resetting the outliers to upper and lower limit
    df.loc[df[column]>=upper_limit,column]=upper_limit
    df.loc[df[column]<=lower_limit,column]=lower_limit
```

In [11]:
```
num_rows = 6
num_cols = 5

# Create a grid of subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, 15))

# Flatten the axes array for easier iteration
axes = axes.flatten()

# Iterate through each column and create a box plot in the corresponding su
for i, column in enumerate(df_columns):
    sns.boxplot(data=df[column], ax=axes[i])
    axes[i].set_xlabel(column,fontweight='bold')
    axes[i].set_ylabel('Range',fontweight='bold')
    axes[i].set_title(f"Box Plot of {column} with Outliers",fontweight='bol

# remove excess subplots
fig.delaxes(axes[-1])

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plots
plt.show()
```



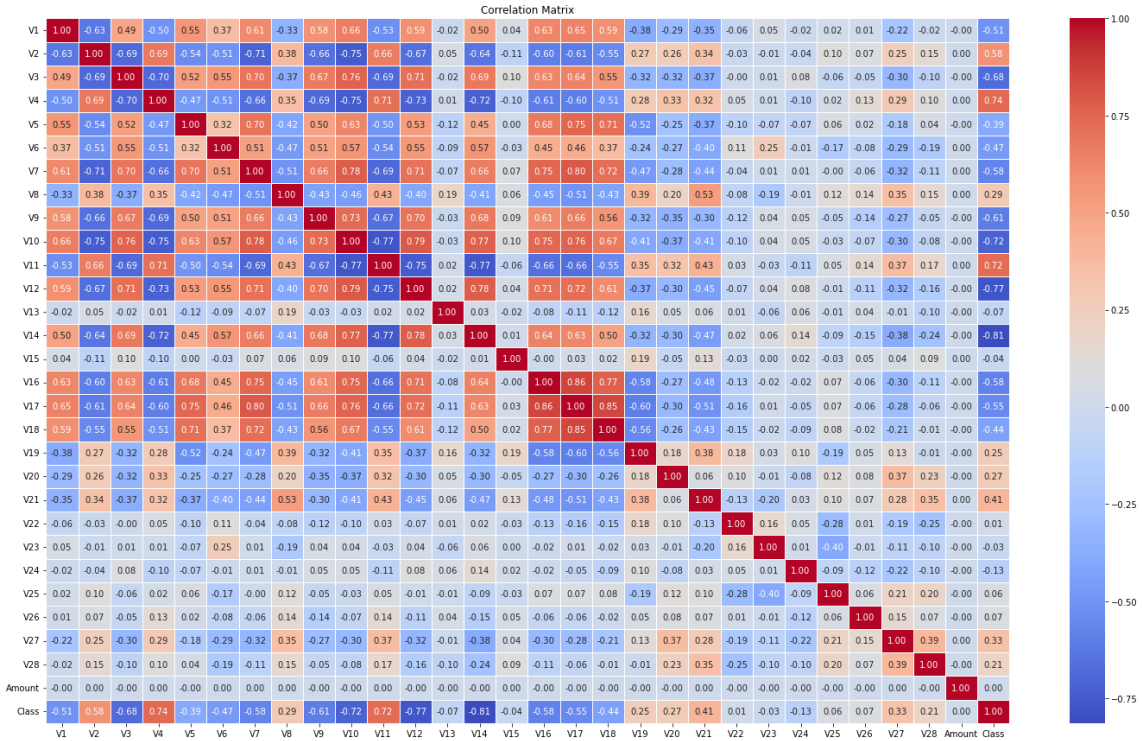
Exploratory Data Analysis

In [19]:
```
df.describe()
```

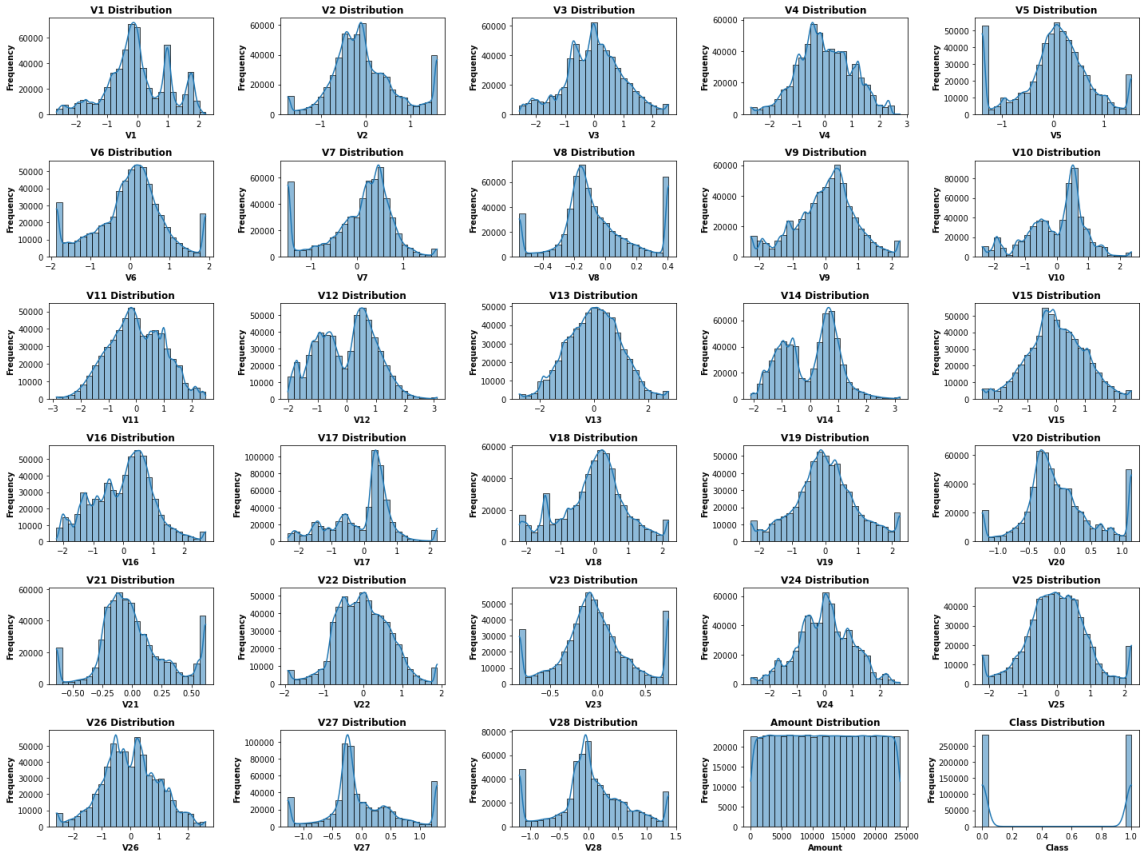
Out[19]:

	id	V1	V2	V3	V4	
count	568630.000000	568630.000000	568630.000000	568630.000000	568630.000000	568630.0
mean	284314.500000	0.000010	-0.034770	-0.002767	0.001832	0.0
std	164149.486121	0.999974	0.733655	0.991889	0.994372	0.7
min	0.000000	-2.662202	-1.732027	-2.566054	-2.700558	-1.3
25%	142157.250000	-0.565286	-0.486678	-0.649299	-0.656020	-0.2
50%	284314.500000	-0.093638	-0.135894	0.000353	-0.073762	0.0
75%	426471.750000	0.832658	0.343555	0.628538	0.707005	0.4
max	568629.000000	2.229046	1.588905	2.545293	2.751542	1.5

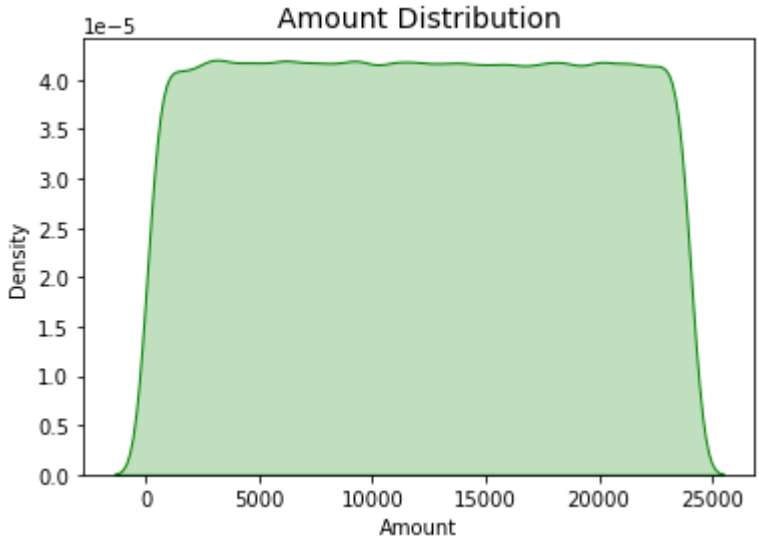
In [12]:
```
#create a correlation Matrix
correlation_matrix=df.iloc[:,1:].corr()

# set plot size
plt.figure(figsize=(25,15))

#plot heat map using seaborn
sns.heatmap(data=correlation_matrix,annot=True,cmap='coolwarm',fmt=".2f",li

#set title
plt.title("Correlation Matrix")

#show plot
plt.show()
```



In [14]:
```
num_rows = 6
num_cols = 5

# Create a grid of subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, 15))

# Flatten the axes array for easier iteration
axes = axes.flatten()

# Iterate through each column and create a box plot in the corresponding su
for i, column in enumerate(df.iloc[:,1:]):
    sns.histplot(data=df[column],bins=25, kde=True , ax=axes[i])
    axes[i].set_xlabel(column,fontweight='bold')
    axes[i].set_ylabel('Frequency',fontweight='bold')
    axes[i].set_title(f"{column} Distribution",fontweight='bold')


# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plots
plt.show()
```



In [15]:
```
# Observing the Amount Disribution
sns.kdeplot(data= df['Amount'],color = 'Green', fill=True)
plt.title('Amount Distribution',size=14)
plt.show()
```



Data Pre-Processing

In [16]:
```
x=df.iloc[:,1:30]
y=df['Class']

#import train_test split library
from sklearn.model_selection import train_test_split

# Split into train-validation
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_sta
```

In [17]:
```
#checking if this is a blanaced dataset
y.value_counts()
```

Out[17]:
```
0    284315
1    284315
Name: Class, dtype: int64
```

In [18]:
```
from sklearn.preprocessing import StandardScaler

#intilize standard scalar
std_scalar=StandardScaler()

#Transfrom X_train dataset into scalar
x_train=std_scalar.fit_transform(x_train)

#Transform x_test dataset into scalar
x_test=std_scalar.transform(x_test)
```

Machine Learning

In [32]:
```
#import Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score #To perform cross valid
from sklearn.metrics import accuracy_score #To calculate accuracy of the mo

#intitlize logistic regression
LR=LogisticRegression(max_iter=1000)

#fit the data
LR.fit(x_train,y_train)

#predict the data
y_pred=LR.predict(x_test)

#calculate accuracy score
accuracy=accuracy_score(y_test,y_pred)

print(f"The accuracy of logistic regression is : {accuracy}")

# calculate cross validation score
cv_score=cross_val_score(LR,x,y,cv=5)
print(f"The cross validation score of Logistic Regerssion is: {cv_score}")

#calculate mean and standard deviation of cross validation scores
cv_score_mean=np.mean(cv_score)
cv_score_std=np.std(cv_score)

print(f'Mean of cross validation scores: {cv_score_mean}')
print(f'standard deviation of cross validation scores: {cv_score_std}')

#store the results
DF_ML=[]
results=[accuracy,cv_score_mean,cv_score_std]
DF_ML.append(results)
```

```
The accuracy of logistic regression is : 0.9674568700209275
The cross validation score of Logistic Regerssion is: [0.95541917 0.959191
39 0.95485641 0.95508503 0.9588045 ]
Mean of cross validation scores: 0.9566712976803897
standard deviation of cross validation scores: 0.0019120302911903248
```

In [33]:
```
#import Random forest Classiffier
from sklearn.ensemble import RandomForestClassifier

#intitlize logistic regression
RF=RandomForestClassifier()

#fit the data
RF.fit(x_train,y_train)

#predict the data
y_pred=RF.predict(x_test)

#calculate accuracy score
accuracy=accuracy_score(y_test,y_pred)

print(f"The accuracy of Random Forest Classifier is : {accuracy}")

# calculate cross validation score
cv_score=cross_val_score(RF,x,y,cv=5)
print(f"The cross validation score of Random Forest Classifier is: {cv_scor

#calculate mean and standard deviation of cross validation scores
cv_score_mean=np.mean(cv_score)
cv_score_std=np.std(cv_score)

print(f'Mean of cross validation scores: {cv_score_mean}')
print(f'standard deviation of cross validation scores: {cv_score_std}')

#store the results
results=[accuracy,cv_score_mean,cv_score_std]
DF_ML.append(results)
```

```
The accuracy of Random Forest Classifier is : 0.9998681040395336
The cross validation score of Random Forest Classifier is: [0.99956035 0.9
9995603 0.99978897 0.99978017 0.9998681 ]
Mean of cross validation scores: 0.9997907250760599
standard deviation of cross validation scores: 0.0001315320116864274
```

In [35]:
```
#import Decision Tree
from sklearn.tree import DecisionTreeClassifier

#intitlize DecisionTreeClassifier
DT=DecisionTreeClassifier()

#fit the data
DT.fit(x_train,y_train)

#predict the data
y_pred=DT.predict(x_test)

#calculate accuracy score
accuracy=accuracy_score(y_test,y_pred)

print(f"The accuracy of Decision Tree Classifier is : {accuracy}")

# calculate cross validation score
cv_score=cross_val_score(DT,x,y,cv=5)
print(f"The cross validation score of Decision Tree Classifier is: {cv_scor

#calculate mean and standard deviation of cross validation scores
cv_score_mean=np.mean(cv_score)
cv_score_std=np.std(cv_score)

print(f'Mean of cross validation scores: {cv_score_mean}')
print(f'standard deviation of cross validation scores: {cv_score_std}')

#store the reuslts
results=[accuracy,cv_score_mean,cv_score_std]
DF_ML.append(results)
```

```
The accuracy of Decision Tree Classifier is : 0.998003974464942
The cross validation score of Decision Tree Classifier is: [0.99426692 0.9
9753794 0.99694001 0.99732691 0.99750277]
Mean of cross validation scores: 0.996714911277984
standard deviation of cross validation scores: 0.0012422530164340664
```

In [46]:
```
#prepare a dataset for model comparision
Algo_names=["LogisticRegression","RandomForestClassifier","DecisionTreeClas
names=['Accuracy','cv_score_mean','cv_score_std']
model_comparision=pd.DataFrame(data=DF_ML,columns=names)
model_comparision.insert(0,'Model_Name',Algo_names)
```

In [47]:
```
model_comparision
```

Out[47]:

	Model_Name	Accuracy	cv_score_mean	cv_score_std
0	LogisticRegression	0.967457	0.956671	0.001912
1	RandomForestClassifier	0.999868	0.999791	0.000132
2	DecisionTreeClassifier	0.998004	0.996715	0.001242

In [62]:
```
fig,axes=plt.subplots(nrows=1,ncols=3,figsize=(20,5))
axes=axes.flatten()
sns.barplot(x='Model_Name',y='Accuracy',data=model_comparision,ax=axes[0])
sns.barplot(x='Model_Name',y='cv_score_mean',data=model_comparision,ax=axes
sns.barplot(x='Model_Name',y='cv_score_std',data=model_comparision,ax=axes[
fig.suptitle('Metrics : Model Comparision',fontsize=20,fontweight='bold')
plt.tight_layout()
plt.show()
```