Data Science and Business Analytics

Practice Project II

Stock Price Prediction using Machine Learning in Python

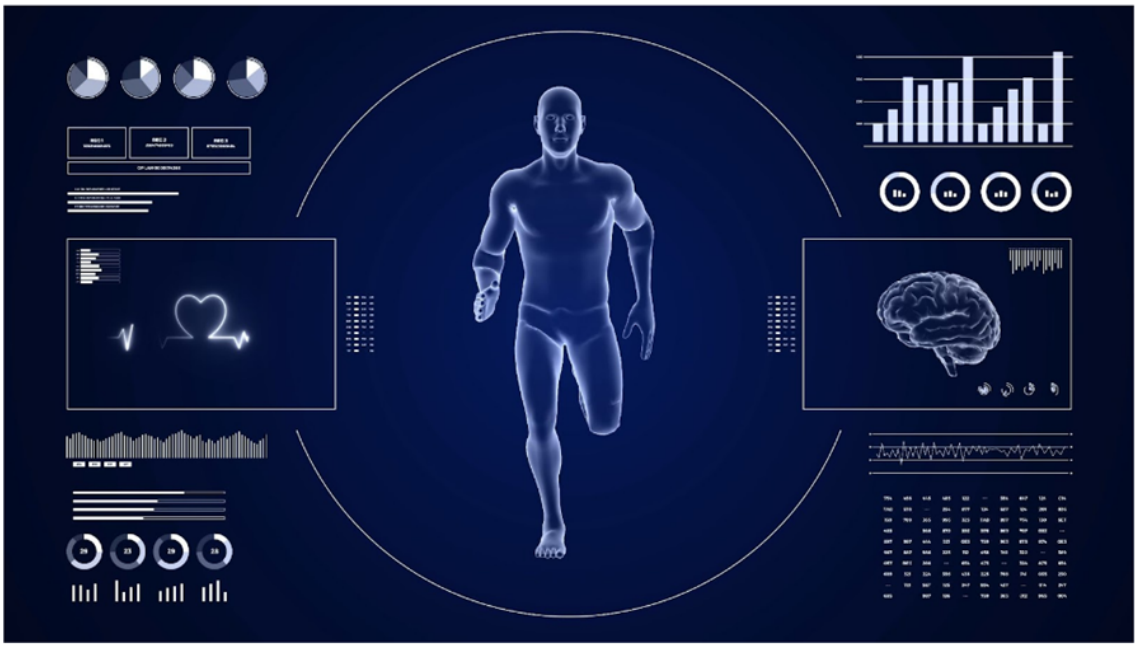
By

Hayford Osumanu

December 2022

Artificial Intelligence and

Machine Learning is the Oxgyen to Business Survival



Stock Price Prediction using Machine Learning in Python

Importing Libraries

In [ ]:

```
!pip install yahoo_fin
!pip install datetime
!pip install feedparser
```

```
!pip install pandas
!pip install requests
!pip install requests_html
!pip install pandas yfinance plotly
```

In [13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, LSTM
import math
from sklearn.preprocessing import MinMaxScaler
```

In [14]:
```
import yahoo_fin.stock_info as si
# OR
from yahoo_fin.stock_info import*
import pandas as pd
```

In [15]:
```
# Pandas and NumPy for managing datasets
import pandas as pd
import numpy as np

# Matplotlib for additional customization
from matplotlib import pyplot as plt
# Seaborn for plotting and styling
import seaborn as sns
# # Command to tell Python to actually display the graphs
%matplotlib inline
pd.set_option('display.float_format', lambda x: '%.2f' % x) # To supress numerical disp
```

In [16]:
```
# First, we just need to load the stock_info module from yahoo_fin.
# import stock_info module from yahoo_fin
from yahoo_fin import stock_info as si
```

In [17]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn import metrics

import warnings
warnings.filterwarnings('ignore')
```

Importing Dataset

In [18]:
```
# Importing the prices of the stock
import pandas as pd
import yfinance as yf
import datetime
from datetime import date, timedelta
import plotly.graph_objects as go
import plotly.express as px

today = date.today()

d1 = today.strftime("%Y-%m-%d")
end_date = d1
d2 = date.today() - timedelta(days=10000)
d2 = d2.strftime("%Y-%m-%d")
start_date = d2

data = yf.download('TSLA',
                   start=start_date,
                   end=end_date,
                   progress=True)
data["Date"] = data.index
data = data[["Date", "Open", "High", "Low",
             "Close", "Adj Close", "Volume"]]
data.reset_index(drop=True, inplace=True)
```

```
[**********100%**********]  1 of 1 completed
```

In [19]:
```
# Creating  a well readable header label
for header in data.columns:
    header_replace = header.replace(
        " ", "_"
    )  # creates new header with "_" instead of " ".
    data.rename(
        {header: header_replace}, axis=1, inplace=True
    )  # sets new header as header made in line above
```

Part I: Data Overview

In [20]:
```
# Observing the current 50 prices of the stock prices
print(data.tail(50))
```

```
           Date   Open   High    Low  Close  Adj_Close     Volume
3100 2022-10-20 208.28 215.55 202.00 207.28     207.28  117798100
3101 2022-10-21 206.42 214.66 203.80 214.44     214.44   75713800
3102 2022-10-24 205.82 213.50 198.59 211.25     211.25  100446800
3103 2022-10-25 210.10 224.35 210.00 222.42     222.42   96507900
3104 2022-10-26 219.40 230.60 218.20 224.64     224.64   85012500
3105 2022-10-27 229.77 233.81 222.85 225.09     225.09   61638800
3106 2022-10-28 225.40 228.86 216.35 228.52     228.52   69152400
3107 2022-10-31 226.19 229.85 221.94 227.54     227.54   61554300
3108 2022-11-01 234.05 237.40 227.28 227.82     227.82   62688800
3109 2022-11-02 226.04 227.87 214.82 214.98     214.98   63070300
3110 2022-11-03 211.36 221.20 210.14 215.31     215.31   56538800
3111 2022-11-04 222.60 223.80 203.08 207.47     207.47   98622200
3112 2022-11-07 208.65 208.90 196.66 197.08     197.08   93916500
3113 2022-11-08 194.02 195.20 186.75 191.30     191.30  128803400
```

```
3114 2022-11-09 190.78 195.89 177.12 177.59     177.59  127062700
3115 2022-11-10 189.90 191.00 180.03 190.72     190.72  132703000
3116 2022-11-11 186.00 196.52 182.59 195.97     195.97  114403600
3117 2022-11-14 192.77 195.73 186.34 190.95     190.95   92226600
3118 2022-11-15 195.88 200.82 192.06 194.42     194.42   91293800
3119 2022-11-16 191.51 192.57 185.66 186.92     186.92   66567600
3120 2022-11-17 183.96 186.16 180.90 183.17     183.17   64336000
3121 2022-11-18 185.05 185.19 176.55 180.19     180.19   76048900
3122 2022-11-21 175.85 176.77 167.54 167.87     167.87   92882700
3123 2022-11-22 168.63 170.92 166.19 169.91     169.91   78452300
3124 2022-11-23 173.57 183.62 172.50 183.20     183.20  109536700
3125 2022-11-25 185.06 185.20 180.63 182.86     182.86   50672700
3126 2022-11-28 179.96 188.50 179.00 182.92     182.92   92905200
3127 2022-11-29 184.99 186.38 178.75 180.83     180.83   83357100
3128 2022-11-30 182.43 194.76 180.63 194.70     194.70  109186400
3129 2022-12-01 197.08 198.92 191.80 194.70     194.70   80046200
3130 2022-12-02 191.78 196.25 191.11 194.86     194.86   73533400
3131 2022-12-05 189.44 191.27 180.55 182.45     182.45   93122700
3132 2022-12-06 181.22 183.65 175.33 179.82     179.82   92150800
3133 2022-12-07 175.03 179.38 172.22 174.04     174.04   84213300
3134 2022-12-08 172.20 175.20 169.06 173.44     173.44   97624500
3135 2022-12-09 173.84 182.50 173.36 179.05     179.05  104746600
3136 2022-12-12 176.10 177.37 167.52 167.82     167.82  109794500
3137 2022-12-13 174.87 175.05 156.91 160.95     160.95  175862700
3138 2022-12-14 159.25 161.62 155.31 156.80     156.80  140682300
3139 2022-12-15 153.44 160.93 153.28 157.67     157.67  122334500
3140 2022-12-16 159.64 160.99 150.04 150.23     150.23  138459600
3141 2022-12-19 154.00 155.25 145.82 149.87     149.87  139390600
3142 2022-12-20 146.05 148.47 137.66 137.80     137.80  159563300
3143 2022-12-21 139.34 141.26 135.89 137.57     137.57  145417400
3144 2022-12-22 136.00 136.63 122.26 125.35     125.35  210090300
3145 2022-12-23 126.37 128.62 121.02 123.15     123.15  166396100
3146 2022-12-27 117.50 119.67 108.76 109.10     109.10  208643400
3147 2022-12-28 110.35 116.27 108.24 112.71     112.71  221070500
3148 2022-12-29 120.39 123.57 117.50 121.82     121.82  221923300
3149 2022-12-30 119.95 124.48 119.75 123.18     123.18  157304500
```

Columns/Variable/Features of the Dataset

In [21]:
```
# Extracting the columns/variables of the dataset
data.columns
```

Out[21]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj_Close', 'Volume'], dtype='object')
```

Observing the Dimention of the Dataset

In [22]:
```
# checking the shape of the data
print(f"There are {data.shape[0]} rows and {data.shape[1]} columns.")
```

```
There are 3150 rows and 7 columns.
```

Data Types of the Dataset

In [23]:
```
# Checking the data types of the variables/columns for the dataset
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype         
---  ------     --------------  -----         
 0   Date       3150 non-null   datetime64[ns]
 1   Open       3150 non-null   float64       
 2   High       3150 non-null   float64       
 3   Low        3150 non-null   float64       
 4   Close      3150 non-null   float64       
 5   Adj_Close  3150 non-null   float64       
 6   Volume     3150 non-null   int64         
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 172.4 KB
```

Checking the Missing Values of the Dataset

In [24]:
```
# Checking for missing values in the dataset
data.isnull().sum()
```

Out[24]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj_Close    0
Volume       0
dtype: int64
```

In [25]:
```
# Checking the total number of missing values in the dataset
data.isnull().sum().sum()
```

Out[25]:
```
0
```

Checking the Duplicates in the Dataset

In [26]:
```
# checking for duplicate values
print("There are about: ", data.duplicated().sum(), "dupplicates in the dataset")
```

```
There are about:  0 dupplicates in the dataset
```

Removing Duplicates from the Dataset

In [27]:
```
# dropping duplicate entries from the data
data.drop_duplicates(inplace=True)

# resetting the index of data frame since some rows will be removed
data.reset_index(drop=True, inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype         
---  ------     --------------  -----         
 0   Date       3150 non-null   datetime64[ns]
 1   Open       3150 non-null   float64       
 2   High       3150 non-null   float64       
 3   Low        3150 non-null   float64       
 4   Close      3150 non-null   float64       
 5   Adj_Close  3150 non-null   float64       
 6   Volume     3150 non-null   int64         
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 172.4 KB
```

Statistical Summary of the Dataset

In [28]:
```
# let's view the statistical summary of minimum numerical columns in the data
data.describe(include=np.number).T.style.highlight_min(color="green", axis=0)
```

Out[28]:

	count	mean	std	min	25%	
Open	3150.000000	58.860577	95.658595	1.076000	8.976167	16.22
High	3150.000000	60.176703	97.854579	1.108667	9.117500	16.49
Low	3150.000000	57.402961	93.175266	0.998667	8.765667	15.94
Close	3150.000000	58.807496	95.526397	1.053333	8.957666	16.22
Adj_Close	3150.000000	58.807496	95.526397	1.053333	8.957666	16.22
Volume	3150.000000	93595945.587302	81698177.972931	1777500.000000	42346575.000000	75966000.00

In [29]:
```
# let's view the statistical summary of maximum numerical columns in the data
data.describe(include=np.number).T.style.highlight_max(color="indigo", axis=0)
```

Out[29]:

	count	mean	std	min	25%	
Open	3150.000000	58.860577	95.658595	1.076000	8.976167	16.22
High	3150.000000	60.176703	97.854579	1.108667	9.117500	16.49
Low	3150.000000	57.402961	93.175266	0.998667	8.765667	15.94
Close	3150.000000	58.807496	95.526397	1.053333	8.957666	16.22
Adj_Close	3150.000000	58.807496	95.526397	1.053333	8.957666	16.22
Volume	3150.000000	93595945.587302	81698177.972931	1777500.000000	42346575.000000	75966000.00

In [30]:
```
# Extracting the Quantiles of the dataset
data.quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T.style.highlight_max(
    color="purple", axis=0)
```

Out[30]:

	0.25	0.5	0.6	0.75	0.9

	0.25	0.5	0.6	0.75	0.9	
Open	8.976167	16.229000	18.891600	24.622500	235.468671	
High	9.117500	16.491000	19.212666	25.086666	239.795665	
Low	8.765667	15.945000	18.567999	24.158668	231.216997	
Close	8.957666	16.222334	18.943734	24.448000	236.094336	
Adj_Close	8.957666	16.222334	18.943734	24.448000	236.094336	
Volume	42346575.000000	75966000.000000	89365920.000000	117297750.000000	191132850.000000	258

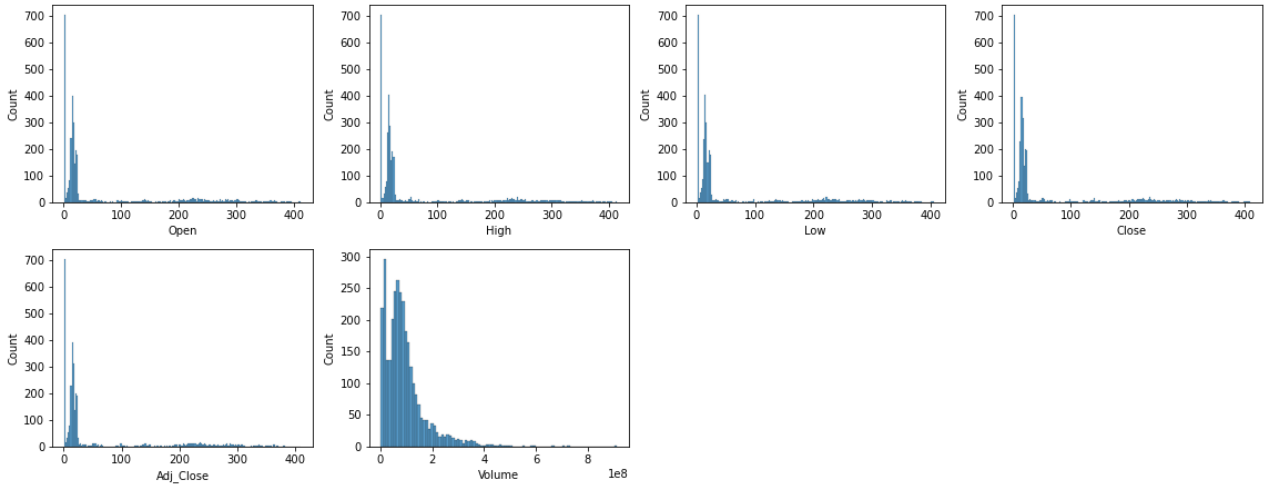
In [31]:
```
# Extracting the Quantiles of the dataset
data.quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T.style.highlight_min(
    color="red", axis=0
)
```

Out[31]:

	0.25	0.5	0.6	0.75	0.9	
Open	8.976167	16.229000	18.891600	24.622500	235.468671	
High	9.117500	16.491000	19.212666	25.086666	239.795665	
Low	8.765667	15.945000	18.567999	24.158668	231.216997	
Close	8.957666	16.222334	18.943734	24.448000	236.094336	
Adj_Close	8.957666	16.222334	18.943734	24.448000	236.094336	
Volume	42346575.000000	75966000.000000	89365920.000000	117297750.000000	191132850.000000	258

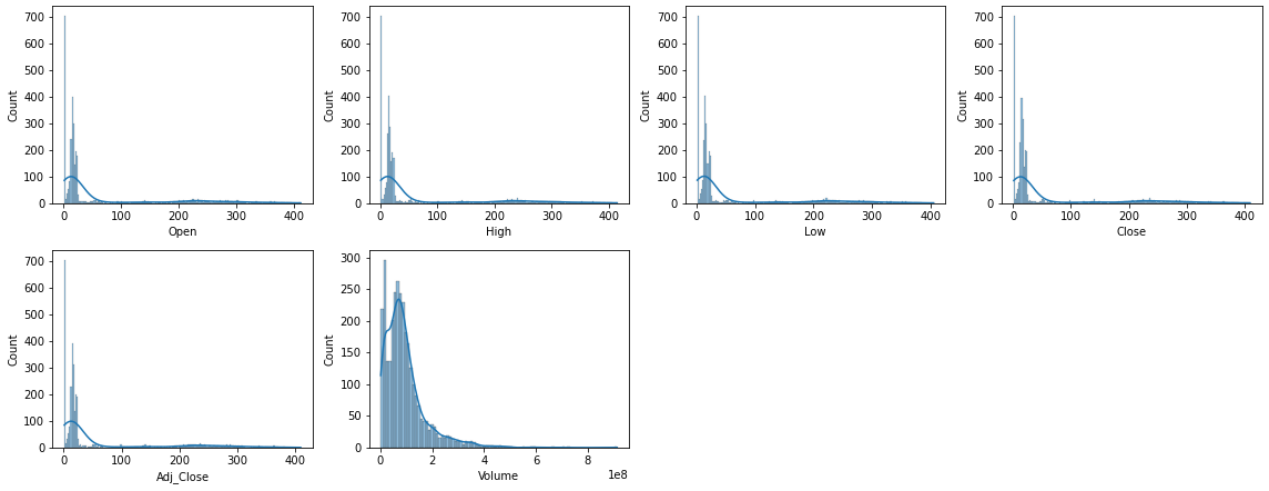
Graphical Univariate Analysis

In [32]:
```
# Checking the histogram plot of numerical variables of the entire dataset
cols = 4
rows = 5
num_cols = data.select_dtypes(exclude="datetime64").columns
fig = plt.figure(figsize=(cols * 4, rows * 3))
for i, col in enumerate(num_cols):

    ax = fig.add_subplot(rows, cols, i + 1)

    sns.histplot(x=data[col], ax=ax)

fig.tight_layout()
plt.show()
```



In [33]:

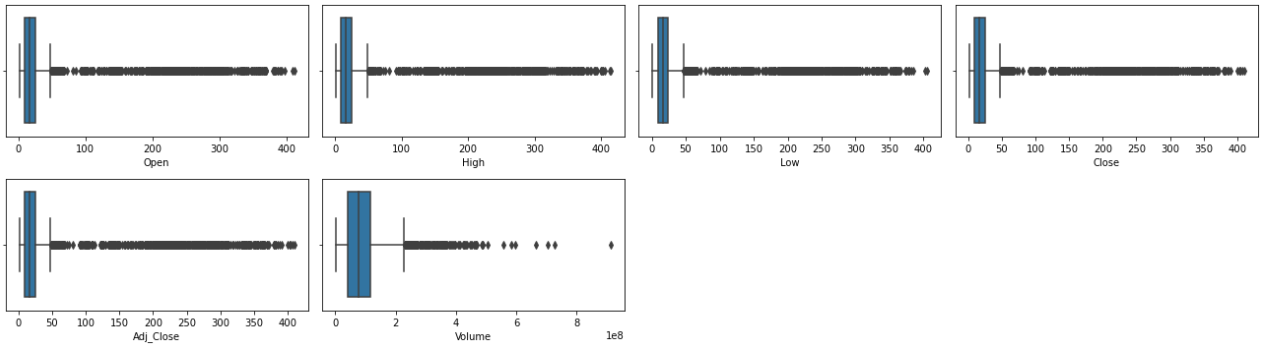
```
# Checking the histogram plot of numerical variables of the entire dataset
cols = 4
rows = 5
num_cols = data.select_dtypes(exclude="datetime64").columns
fig = plt.figure(figsize=(cols * 4, rows * 3))
for i, col in enumerate(num_cols):

    ax = fig.add_subplot(rows, cols, i + 1)

    sns.histplot(x=data[col], kde=True, ax=ax)

fig.tight_layout()
plt.show()
```



In [34]:

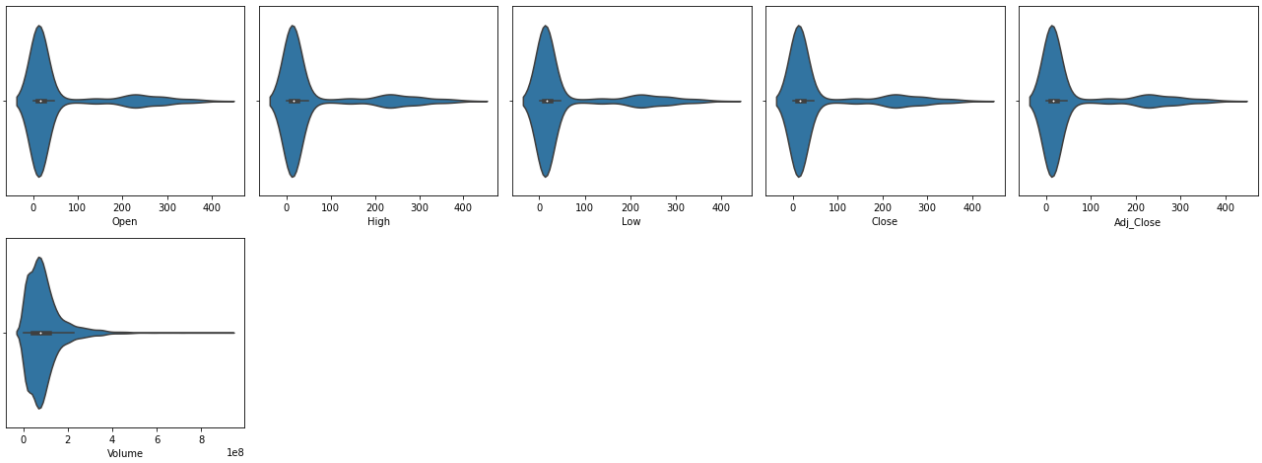
```
# Checking the boxplot of the numerical variable of the dataset
cols = 4
rows = 5
num_cols = data.select_dtypes(exclude="datetime64").columns
fig = plt.figure(figsize=(18, 12))
for i, col in enumerate(num_cols):

    ax = fig.add_subplot(rows, cols, i + 1)

    sns.boxplot(x=data[col], ax=ax)
```

```
fig.tight_layout()
plt.show()
```



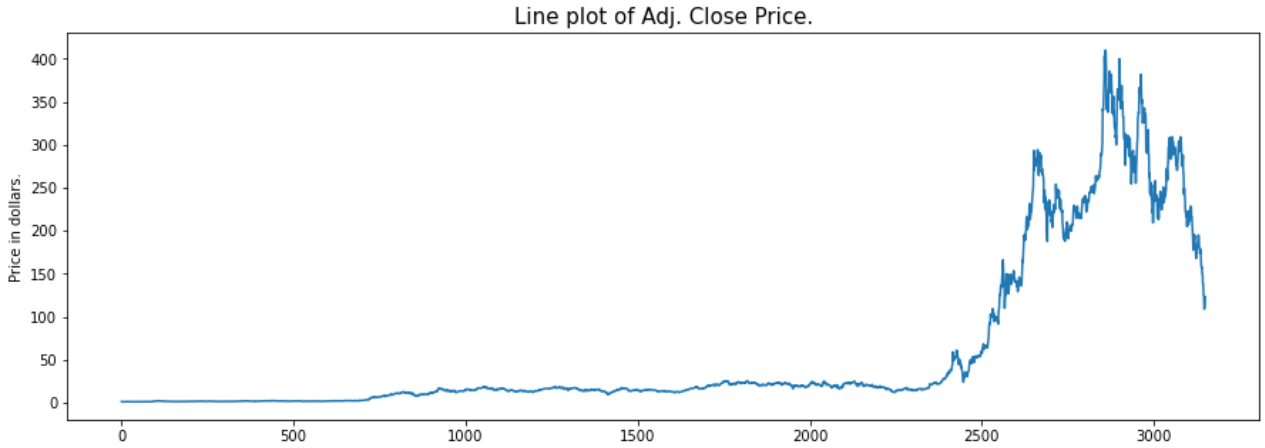
In [35]:

```
# Checking the voilin plot of the numerical variables
cols = 5
rows = 5
num_cols = data.select_dtypes(exclude="datetime64").columns
fig = plt.figure(figsize=(18, 16))
for i, col in enumerate(num_cols):

    ax = fig.add_subplot(rows, cols, i + 1)

    sns.violinplot(x=data[col], ax=ax)

fig.tight_layout()
plt.show()
```



In [36]:

```
plt.figure(figsize=(15,5))
plt.plot(data['Adj_Close'])
plt.title('Line plot of Adj. Close Price.', fontsize=15)
plt.ylabel('Price in dollars.')
plt.show()
```



Runing the codes for histogram and boxplots

In [37]:
```
data.columns
```

Out[37]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj_Close', 'Volume'], dtype='object')
```

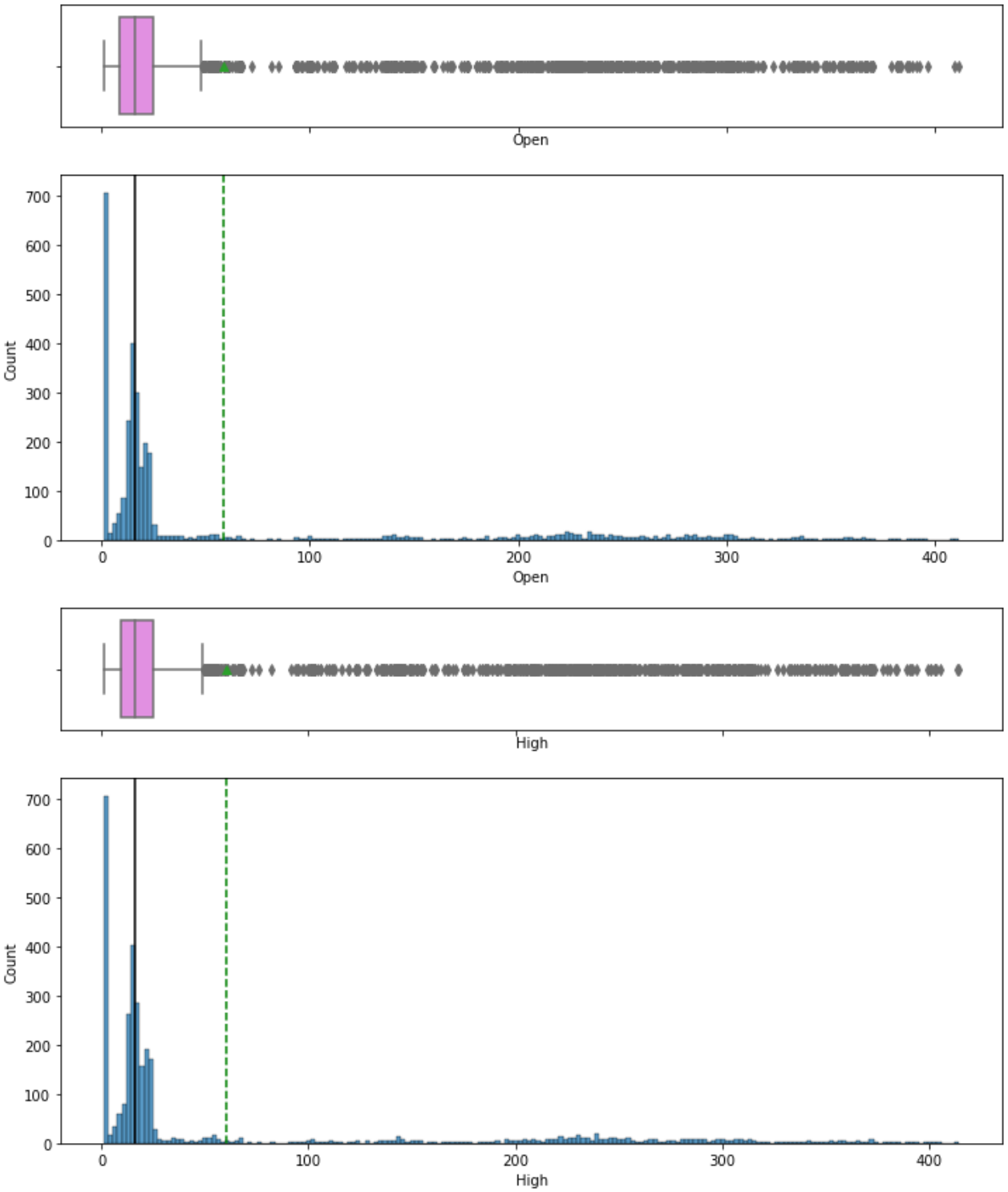
In [38]:
```
h_data=data.copy()
hist_data=h_data.drop('Date',axis=1)
```

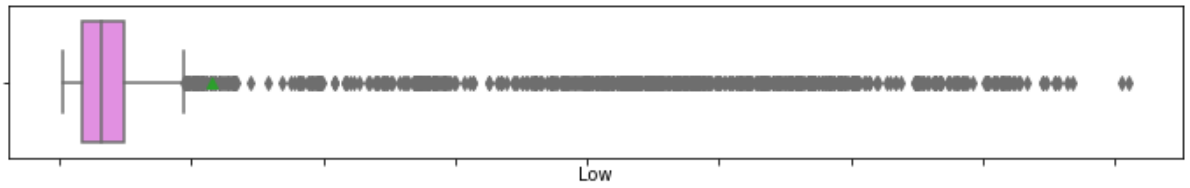
In [39]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(hist_data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=hist_data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=hist_data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        hist_data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
```

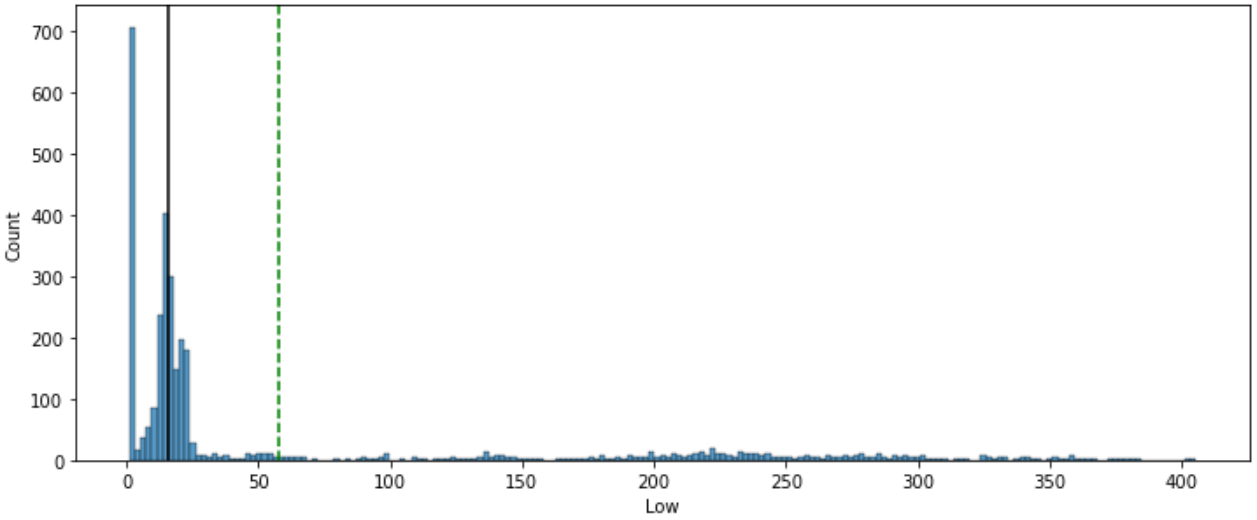
```
        hist_data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

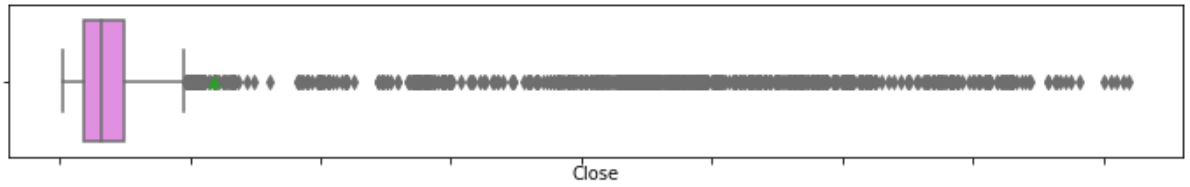
In [40]:

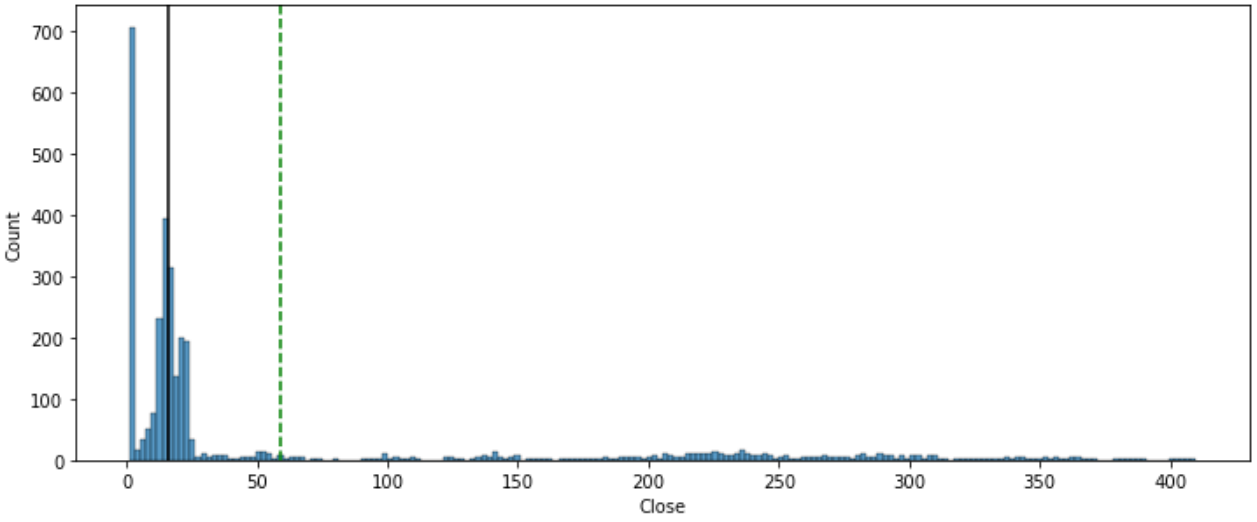
```
for feature in hist_data.columns:
    histogram_boxplot(
        hist_data, feature, figsize=(12, 7), kde=False, bins=None
    )  ## Please change the dataframe name as you define while reading the data
```

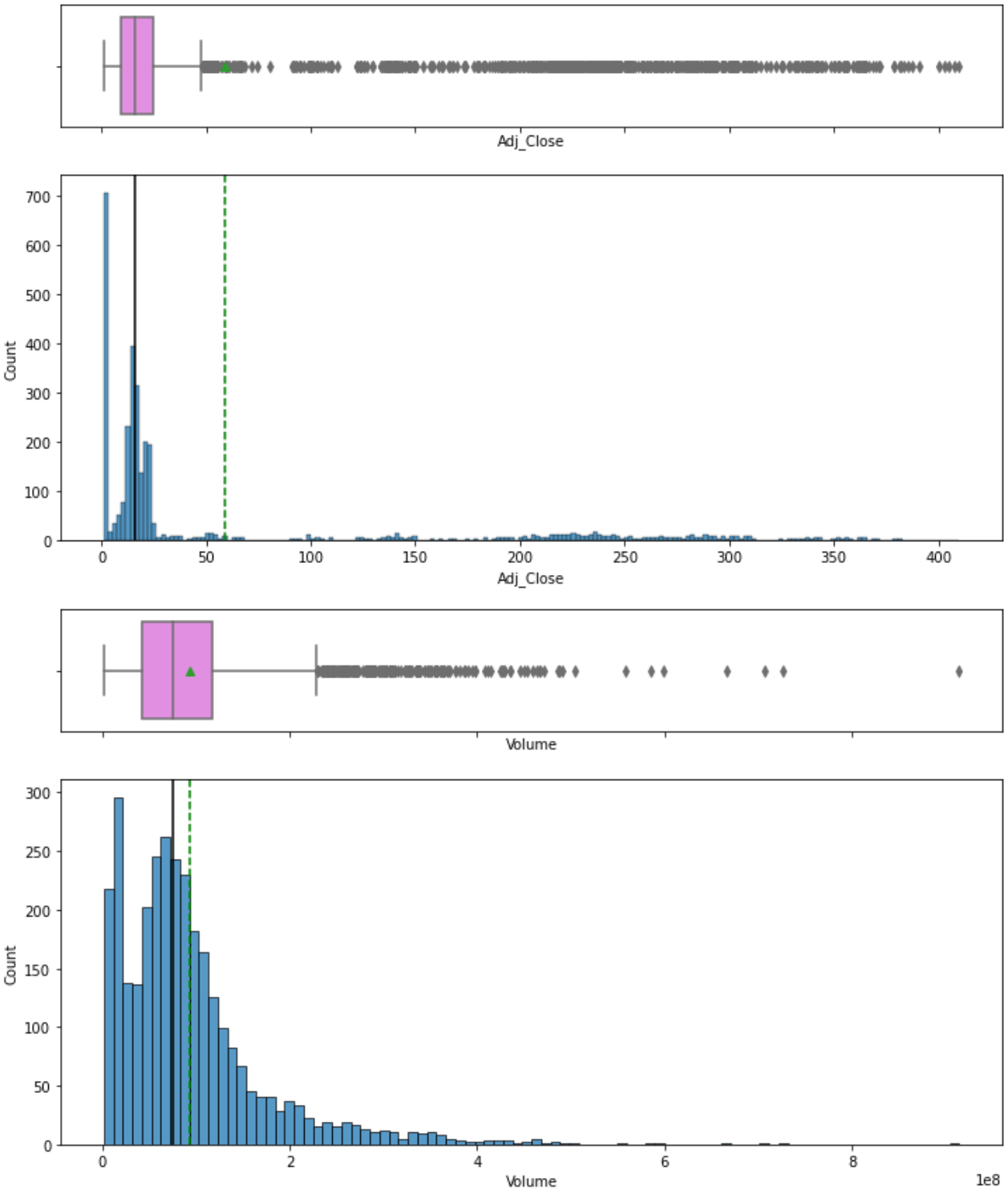



Low


Count
Low


Close


Count
Close

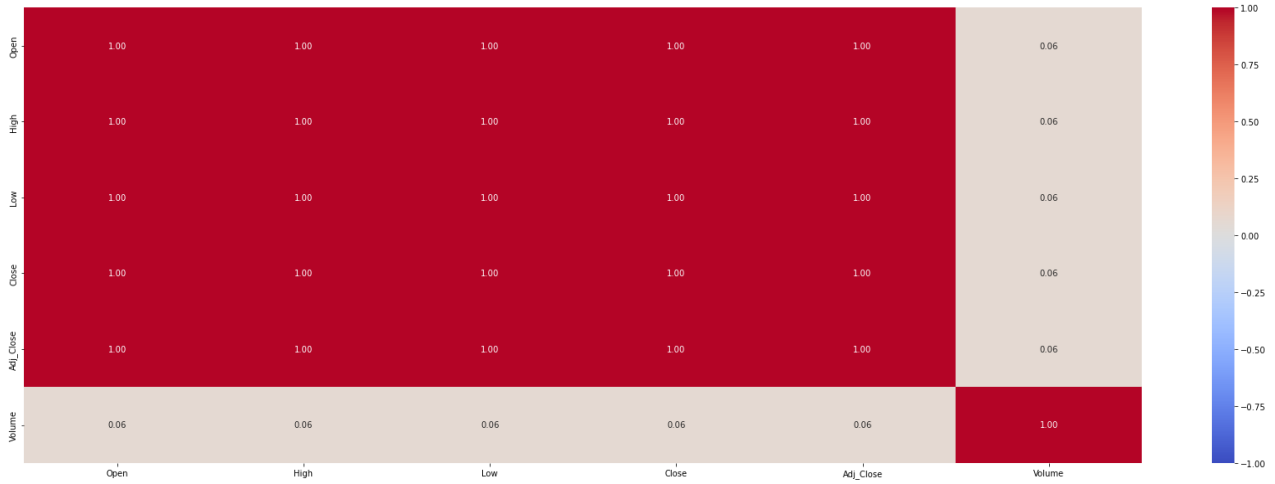


Part II: Multivariate Data Analysis

Correlation and Pairplot Analysis

In [41]:

```
# Displaying the correlation between numerical variables of the dataset
plt.figure(figsize=(30, 10))
sns.heatmap(hist_data.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="coolwarm")
plt.show()
```

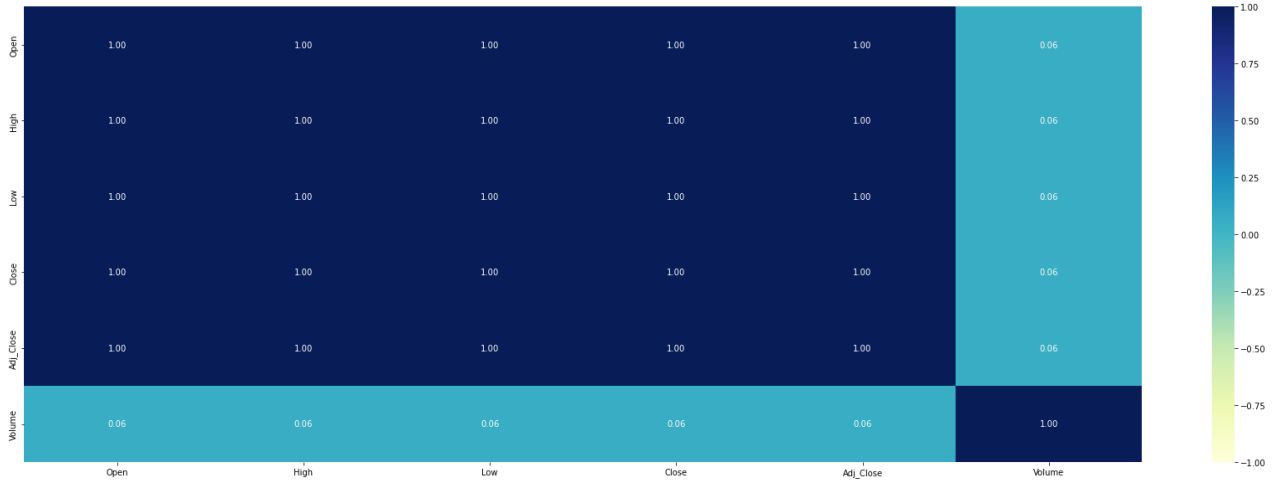


In [42]:

```
# creates heatmap showing correlation of numeric columns in data
plt.figure(figsize=(30, 10))
sns.heatmap(hist_data.corr(), vmin=-1, vmax=1, cmap="YlGnBu", annot=True, fmt=".2f")
```

Out[42]:

```
<AxesSubplot:>
```



Part III: Data Preparation for Model Building

Feature Engineering

Feature Engineering helps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

In [43]:

```
df=data.copy()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       3150 non-null   datetime64[ns]
```

```
 1   Open       3150 non-null   float64
 2   High       3150 non-null   float64
 3   Low        3150 non-null   float64
 4   Close      3150 non-null   float64
 5   Adj_Close  3150 non-null   float64
 6   Volume     3150 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 172.4 KB
```

A quarter is defined as a group of three months. Every company prepares its quarterly results and publishes them publicly so, that people can analyze the company's performance. These quarterly results affect the stock prices heavily which is why we have added this feature because this can be a helpful feature for the learning model.

In [44]:
```
df['Day'] = df['Date'].dt.day
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year
df['Quarter_End'] = np.where(df['Month']%3==0,1,0)
df.head()
```

Out[44]:

	Date	Open	High	Low	Close	Adj_Close	Volume	Day	Month	Year	Quarter_End
0	2010-06-29	1.27	1.67	1.17	1.59	1.59	281494500	29	6	2010	1
1	2010-06-30	1.72	2.03	1.55	1.59	1.59	257806500	30	6	2010	1
2	2010-07-01	1.67	1.73	1.35	1.46	1.46	123282000	1	7	2010	0
3	2010-07-02	1.53	1.54	1.25	1.28	1.28	77097000	2	7	2010	0
4	2010-07-06	1.33	1.33	1.06	1.07	1.07	103003500	6	7	2010	0

In [45]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         3150 non-null   datetime64[ns]
 1   Open         3150 non-null   float64
 2   High         3150 non-null   float64
 3   Low          3150 non-null   float64
 4   Close        3150 non-null   float64
 5   Adj_Close    3150 non-null   float64
 6   Volume       3150 non-null   int64
 7   Day          3150 non-null   int64
 8   Month        3150 non-null   int64
 9   Year         3150 non-null   int64
 10  Quarter_End  3150 non-null   int32
dtypes: datetime64[ns](1), float64(5), int32(1), int64(4)
memory usage: 258.5 KB
```

In [46]:
```
df.tail()
```

Out[46]:

	Date	Open	High	Low	Close	Adj_Close	Volume	Day	Month	Year	Quarter_End

	Date	Open	High	Low	Close	Adj_Close	Volume	Day	Month	Year	Quarter_End
3145	2022-12-23	126.37	128.62	121.02	123.15	123.15	166396100	23	12	2022	1
3146	2022-12-27	117.50	119.67	108.76	109.10	109.10	208643400	27	12	2022	1
3147	2022-12-28	110.35	116.27	108.24	112.71	112.71	221070500	28	12	2022	1
3148	2022-12-29	120.39	123.57	117.50	121.82	121.82	221923300	29	12	2022	1
3149	2022-12-30	119.95	124.48	119.75	123.18	123.18	157304500	30	12	2022	1

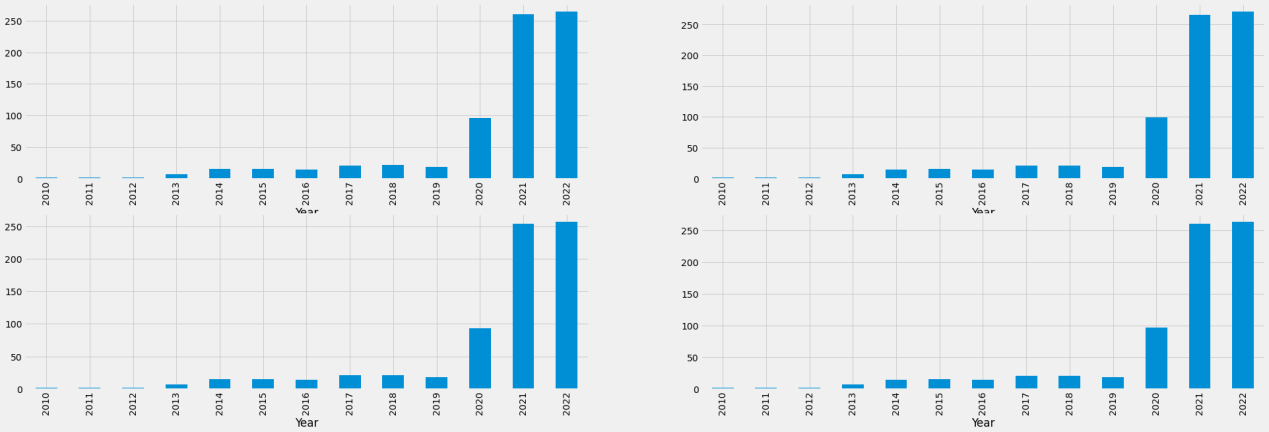
In [47]:

```
plt.style.use("fivethirtyeight")
sns.pairplot(df, hue="Quarter_End", vars=hist_data.columns, diag_kind="kde");
```

In [48]:

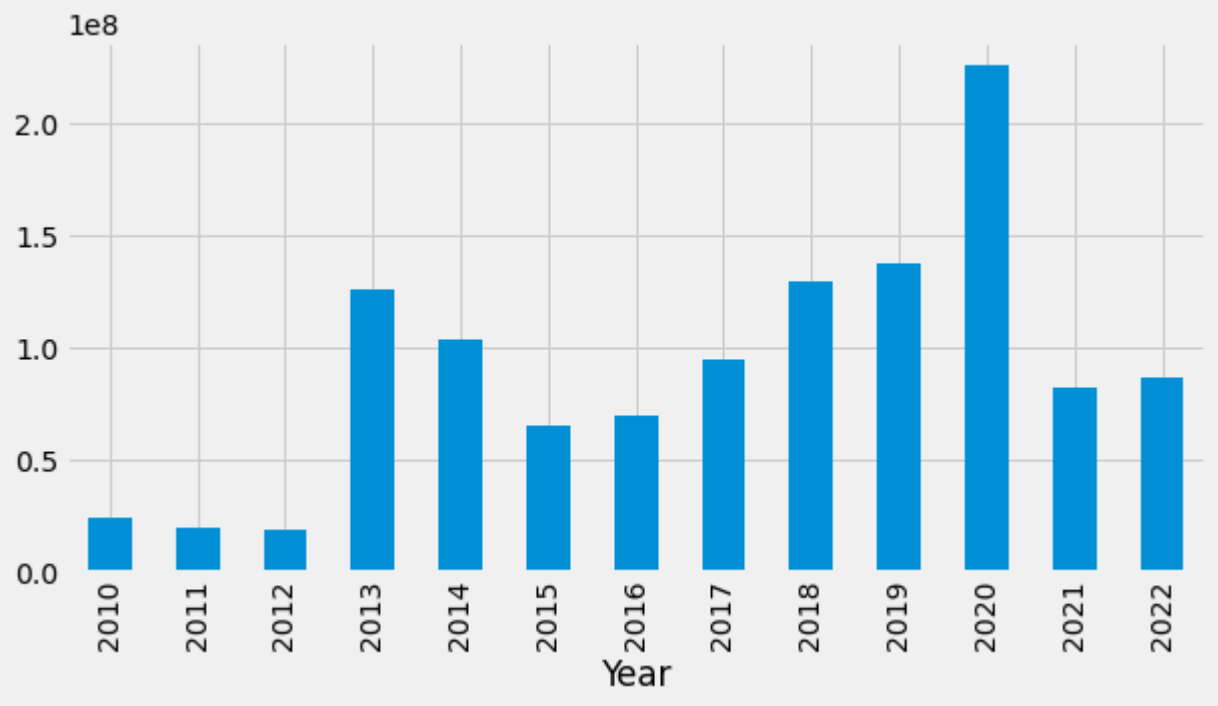
```
data_grouped = df.groupby('Year').mean()
plt.subplots(figsize=(30,10))
```

```
for i, col in enumerate(['Open', 'High', 'Low', 'Adj_Close']):
  plt.subplot(2,2,i+1)
  data_grouped[col].plot.bar()
plt.show()
```



In [49]:

```
data_grouped = df.groupby('Year').mean()
plt.subplots(figsize=(20,10))

for i, col in enumerate(['Volume']):
  plt.subplot(2,2,i+1)
  data_grouped[col].plot.bar()
plt.show()
```



In [50]:

```
df.groupby('Quarter_End').mean()
```

Out[50]:

Quarter_End	Open	High	Low	Close	Adj_Close	Volume	Day	Month	Year

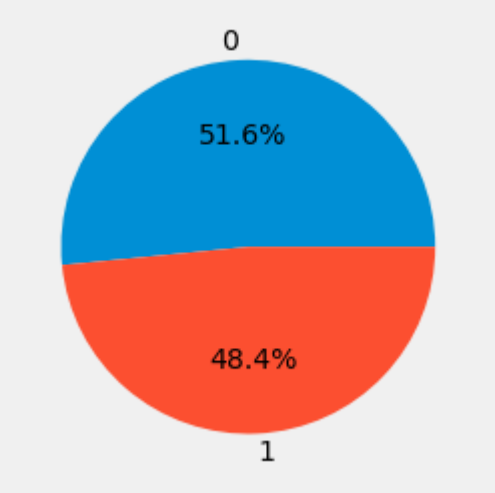
Quarter_End	Open	High	Low	Close	Adj_Close	Volume	Day	Month	Year
0	58.41	59.72	56.96	58.34	58.34	95091796.27	15.70	6.21	2016.23
1	59.75	61.07	58.27	59.73	59.73	90650769.27	15.80	7.56	2016.25

Prices are higher in the months which are quarter end as compared to that of the non-quarter end months. The volume of trades is lower in the months which are quarter end.

In [51]:
```
df['open-close']  = df['Open'] - df['Close']
df['low-high']  = df['Low'] - df['High']
df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
```

Above we have added some more columns which will help in the training of our model. We have added the target feature which is a signal whether to buy or not we will train our model to predict this only. But before proceeding let's check whether the target is balanced or not using a pie chart.

In [52]:
```
plt.pie(df['target'].value_counts().values,
        labels=[0, 1], autopct='%1.1f%%')
plt.show()
```



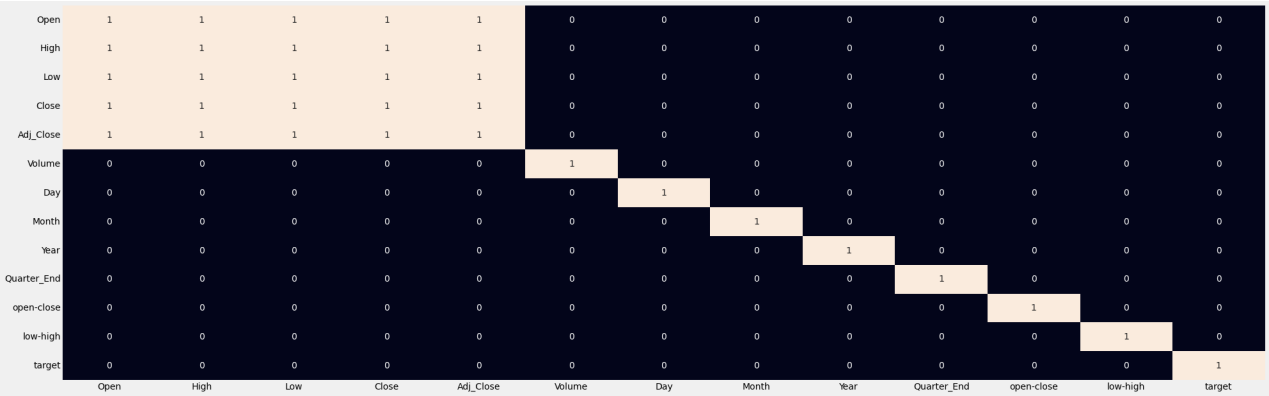
In [53]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         3150 non-null   datetime64[ns]
 1   Open         3150 non-null   float64
 2   High         3150 non-null   float64
 3   Low          3150 non-null   float64
 4   Close        3150 non-null   float64
 5   Adj_Close    3150 non-null   float64
 6   Volume       3150 non-null   int64
 7   Day          3150 non-null   int64
 8   Month        3150 non-null   int64
```

```
 9   Year          3150 non-null   int64
 10  Quarter_End   3150 non-null   int32
 11  open-close    3150 non-null   float64
 12  low-high      3150 non-null   float64
 13  target        3150 non-null   int32
dtypes: datetime64[ns](1), float64(7), int32(2), int64(4)
memory usage: 320.0 KB
```

hen we add features to our dataset we have to ensure that there are no highly correlated features as they do not help in the learning process of the algorithm.

In [54]:
```
plt.figure(figsize=(30, 10))

# As our concern is with the highly
# correlated features only so, we will visualize
# our heatmap as per that criteria only.
sb.heatmap(df.corr() > 0.9, annot=True, cbar=False)
plt.show()
```



From the above heatmap, we can say that there is a high correlation between OHLC that is pretty obvious, and the added features are not highly correlated with each other or previously provided features which means that we are good to go and build our model.

Data Splitting and Normalization

In [55]:
```
features = df[['open-close', 'low-high', 'Quarter_End']]
target = df['target']

scaler = StandardScaler()
features = scaler.fit_transform(features)

X_train, X_valid, Y_train, Y_valid = train_test_split(
    features, target, test_size=0.3, random_state=2023)
print(X_train.shape, X_valid.shape)
```

```
(2205, 3) (945, 3)
```

In [56]:
```
X_train, X_test, Y_train, Y_test = train_test_split(features, target, test_size=0.3, ra
print(X_train.shape, X_test.shape)
```

```
(2205, 3) (945, 3)
```

In [57]:

```
print("*" * 60)
print("Shape of Training set : ", X_train.shape)

print("*" * 60)
print("Shape of test set : ", X_test.shape)

print("*" * 60)
print("Percentage of classes in training set:")
print("*" * 60)
print(Y_train.value_counts(normalize=True))
print("*" * 60)
print("Percentage of classes in test set:")

print(Y_test.value_counts(normalize=True))
print("*" * 60)
```

```
************************************************************
Shape of Training set :  (2205, 3)
************************************************************
Shape of test set :  (945, 3)
************************************************************
Percentage of classes in training set:
************************************************************
1   0.52
0   0.48
Name: target, dtype: float64
************************************************************
Percentage of classes in test set:
1   0.50
0   0.50
Name: target, dtype: float64
************************************************************
```

In [58]:

```
models = [LogisticRegression(), SVC(
  kernel='poly', probability=True), XGBClassifier()]

for i in range(3):
  models[i].fit(X_train, Y_train)

  print(f'{models[i]} : ')
  print('Training Accuracy : ', metrics.roc_auc_score(
    Y_train, models[i].predict_proba(X_train)[:,1]))
  print('Validation Accuracy : ', metrics.roc_auc_score(
    Y_valid, models[i].predict_proba(X_valid)[:,1]))
  print()
```

```
LogisticRegression() :
Training Accuracy :  0.509303214916792
Validation Accuracy :  0.5070237050043899

SVC(kernel='poly', probability=True) :
Training Accuracy :  0.5158677176464382
Validation Accuracy :  0.5121615810503306

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
```

```
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...) :
Training Accuracy :  0.9634827386132868
Validation Accuracy :  0.832454623640501
```

Part II:Building Models for Decision Trees, Bagging Boosting

Importing all the Relevant Libraries for Data Analysis

In [ ]:
```
# installing the needed upgrade optimum performance
!pip install nb_black
!pip install ipython --upgrade
!pip install xgboost
```

In [60]:
```
# Other important packages for data data analysis
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from statsmodels.formula.api import ols
from statsmodels.graphics.gofplots import ProbPlot
import statsmodels.api as sm
from scipy import stats;
```

In [61]:
```
#format numeric data for easier readability
pd.set_option(
    "display.float_format", lambda x: "%.2f" % x
)  # to display numbers rounded off to 2 decimal places
```

In [62]:
```
# this will help in making the Python code more structured automatically (good coding p
%load_ext nb_black

# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns



# split the data into train and test
from sklearn.model_selection import train_test_split
```

```
# to build linear regression_model
from sklearn.linear_model import LinearRegression

# to check model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# to build linear regression_model using statsmodels
import statsmodels.api as sm

# to compute VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

# # Command to tell Python to actually display the graphs
%matplotlib inline
pd.set_option('display.float_format', lambda x: '%.2f' % x) # To supress numerical disp

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)

# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200);
```

In [63]:

```
# this will help in making the Python code more structured automatically (good coding p
%load_ext nb_black
import warnings

warnings.filterwarnings("ignore")

# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd
import xgboost as xgb

# Library to split data
from sklearn.model_selection import train_test_split

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)


# Libraries different ensemble classifiers
from sklearn.ensemble import (
    BaggingClassifier,
    RandomForestClassifier,
    AdaBoostClassifier,
    GradientBoostingClassifier,
    StackingClassifier,
)

from xgboost import XGBClassifier
```

```
from sklearn.tree import DecisionTreeClassifier

# Libraries to get different metric scores
from sklearn import metrics
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
)

# To tune different models
from sklearn.model_selection import GridSearchCV
```

```
The nb_black extension is already loaded. To reload it, use:
  %reload_ext nb_black
```

In [64]:

```
# Libraries to help with reading and manipulating data

import pandas as pd
import numpy as np

# Library to split data
from sklearn.model_selection import train_test_split

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)

# To build Logistic Regression model for prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant
from sklearn.linear_model import LogisticRegression

# To get diferent metric scores
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    plot_confusion_matrix,
    precision_recall_curve,
    roc_curve,
    make_scorer,
)

# Libraries to build decision tree classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
# To tune different models
from sklearn.model_selection import GridSearchCV
```

In [65]:
```
# Import all the Required Statistical Distributions
from scipy import stats
from scipy.stats import ttest_1samp
from numpy import sqrt, abs
from scipy.stats import norm
from scipy.stats import ttest_ind
from scipy.stats import ttest_rel
from statsmodels.stats.proportion import proportions_ztest
from scipy.stats import chi2
from scipy.stats import f
from scipy.stats import chi2_contingency
from scipy.stats import f_oneway
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scipy.stats import levene
```

In [66]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.impute import SimpleImputer

from sklearn.tree import DecisionTreeClassifier
```

First, let's create functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

- The model_performance_classification_sklearn function will be used to check the model performance of models.
- The confusion_matrix_sklearn function will be used to plot the confusion matrix.

In [67]:
```
# defining a function to compute different metrics to check performance of a classifica


def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)
```

```
    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    return df_perf
```

In [68]:

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    Y_pred = model.predict(predictors)
    cm = confusion_matrix(target, Y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(15, 6))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

In [69]:

```
## Function to create confusion matrix
def make_confusion_matrix(model, y_actual, labels=[1, 0]):
    """
    model : classifier to predict values of X
    y_actual : ground truth

    """
    y_predict = model.predict(X_test)
    cm = metrics.confusion_matrix(y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(
        cm,
        index=[i for i in ["Actual - No", "Actual - Yes"]],
        columns=[i for i in ["Predicted - No", "Predicted - Yes"]],
    )
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cm.flatten() / np.sum(cm)
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)
```

```
    plt.figure(figsize=(15, 7))
    sns.heatmap(df_cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

In [70]:

```
##  Function to calculate different metric scores of the model - Accuracy, Recall and P
def get_metrics_score(model, flag=True):
    """
    model : classifier to predict values of X

    """
    # defining an empty list to store train and test results
    score_list = []

    # Predicting on train and tests
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    # Accuracy of the model
    train_acc = model.score(X_train, Y_train)
    test_acc = model.score(X_test, Y_test)

    # Recall of the model
    train_recall = metrics.recall_score(Y_train, pred_train)
    test_recall = metrics.recall_score(Y_test, pred_test)

    # Precision of the model
    train_precision = metrics.precision_score(Y_train, pred_train)
    test_precision = metrics.precision_score(Y_test, pred_test)

    score_list.extend(
        (
            train_acc,
            test_acc,
            train_recall,
            test_recall,
            train_precision,
            test_precision,
        )
    )

    # If the flag is set to True then only the following print statements will be dispa
    if flag == True:
        print("Accuracy on training set : ", model.score(X_train, Y_train))
        print("Accuracy on test set : ", model.score(X_test, Y_test))
        print("Recall on training set : ", metrics.recall_score(Y_train, pred_train))
        print("Recall on test set : ", metrics.recall_score(Y_test, pred_test))
        print(
            "Precision on training set : ", metrics.precision_score(Y_train, pred_train
        )
        print("Precision on test set : ", metrics.precision_score(Y_test, pred_test))

    return score_list  # returning the list with train and test scores
```

Building the model

- We are going to build 2 ensemble models here - Bagging Classifier and Random Forest Classifier.
- First, let's build these models with default parameters and then use hyperparameter tuning to optimize the model performance.
- We will calculate all three metrics - Accuracy, Precision and Recall but the metric of interest here is recall.
- `Recall` - It gives the ratio of True positives to Actual positives, so high Recall implies low false negatives, i.e. low chances of predicting a defaulter as non defaulter

Decision Tree Model

- ***The decision tree model would be considered overfit if the metric of interest is 'accuracy' since the difference between the training data accuracy and testing data accuracy is very large which indicates the fact that the model is not able to generalize to new data points and is overfitting the training dataset.***

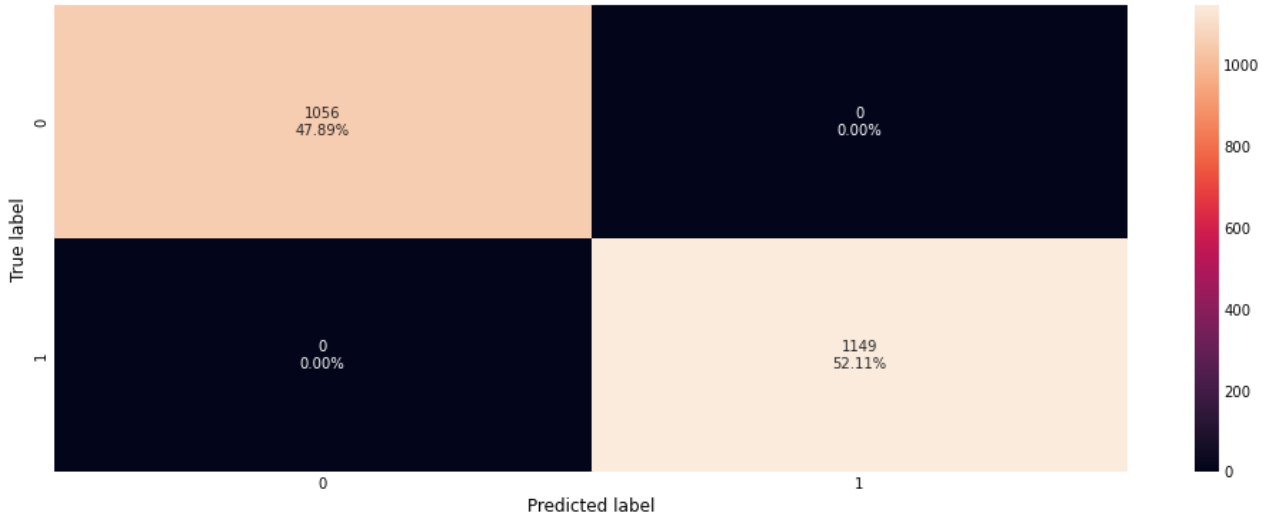
- ***When classification problems exhibit a significant imbalance in the distribution of the target classes, it is good to use stratified sampling to ensure that relative class frequencies are approximately preserved in train and test sets. This is done using the stratify parameter in the train_test_split function.***

In [71]:
```
model = DecisionTreeClassifier(
    criterion="gini", random_state=1
)  ## Complete the code to define decision tree classifier with random state = 1
model.fit(
    X_train, Y_train
)  ## Complete the code to fit decision tree classifier on the train data
```

Out[71]:
```
DecisionTreeClassifier(random_state=1)
```

Checking model performance on training set

In [72]:
```
# confusion_matrix_sklearn(model, X_train, Y_train) ## Complete the code to create conf
confusion_matrix_sklearn(model, X_train, Y_train)
```



In [73]:
```
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, Y_train
)  ## Complete the code to check performance on train data
decision_tree_perf_train
```

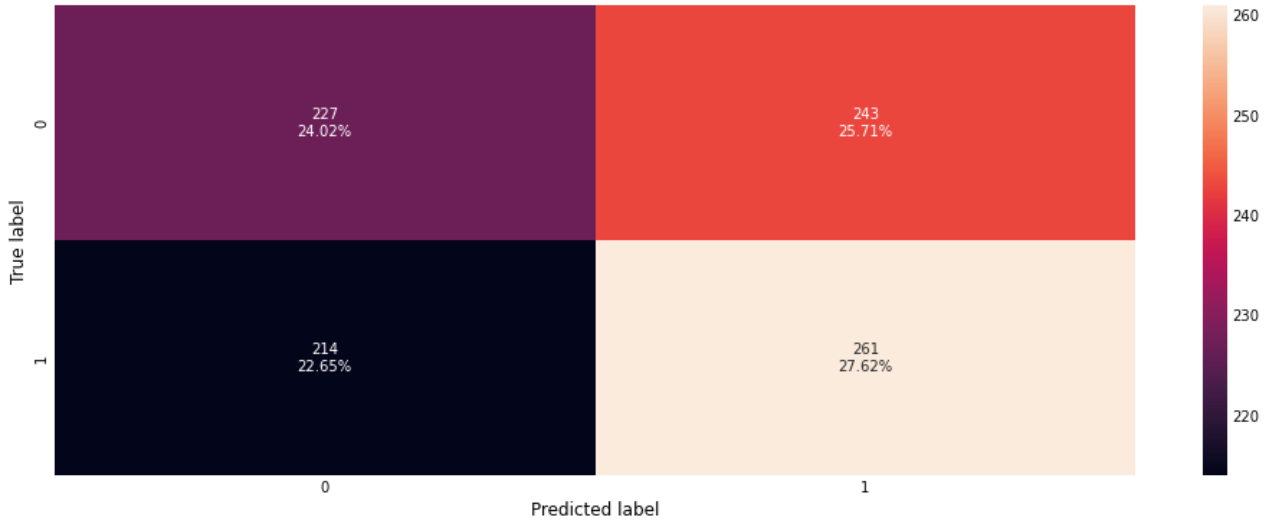
Out[73]:

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

Observation

The decision tree model is highly overfitting the train dataset.

In [74]:
```
confusion_matrix_sklearn(
    model, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [75]:
```
decision_tree_perf_test = model_performance_classification_sklearn(
    model, X_test, Y_test
```

```
)  ## Complete the code to check performance for test data
decision_tree_perf_test
```

Out[75]:

	Accuracy	Recall	Precision	F1
0	0.52	0.55	0.52	0.53

- The decision tree is overfitting the training data.

Method2: Weighted Decision Tree Model

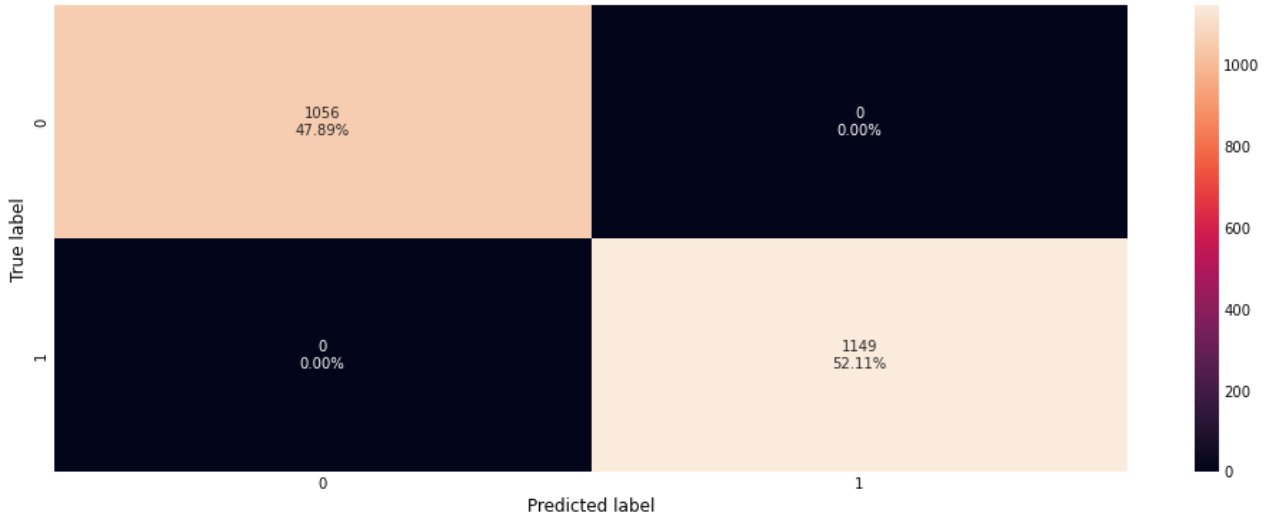
- We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split.
- If the frequency of class A is 10% and the frequency of class B is 90%, then class B will become the dominant class and the decision tree will become biased toward the dominant classes.
- In this case, we can pass a dictionary {0:0.17,1:0.83} to the model to specify the weight of each class and the decision tree will give more weightage to class 1.
- class_weight is a hyperparameter for the decision tree classifier.

In [76]:
```
dtree = DecisionTreeClassifier(
    criterion="gini", class_weight={0: 0.17, 1: 0.83}, random_state=1
)
```

In [77]:
```
dtree.fit(X_train, Y_train)
```

Out[77]:
```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
```

In [78]:
```
confusion_matrix_sklearn(dtree, X_train, Y_train)
```



In [79]:

```
# Training Performance Mearsures
dtree_model_train_perf = model_performance_classification_sklearn(
    dtree, X_train, Y_train
)
print("Training performance \n")
dtree_model_train_perf
```

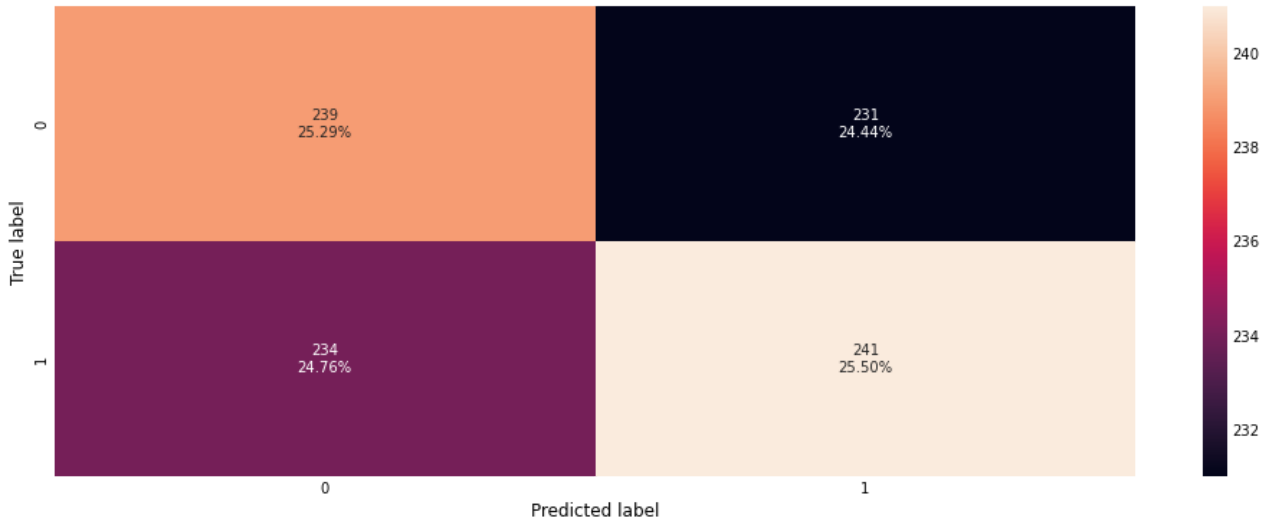
```
Training performance
```

Out[79]:

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

In [80]:

```
confusion_matrix_sklearn(dtree, X_test, Y_test)
```



In [81]:

```
# Test Performance Measures
dtree_model_test_perf = model_performance_classification_sklearn(dtree, X_test, Y_test)
print("Testing performance \n")
dtree_model_test_perf
```

```
Testing performance
```

Out[81]:

	Accuracy	Recall	Precision	F1
0	0.51	0.51	0.51	0.51

- Decision tree is working well on the training data but is not able to generalize well on the test data concerning the recall.

Hyperparameter Tuning - Decision Tree

In [82]:

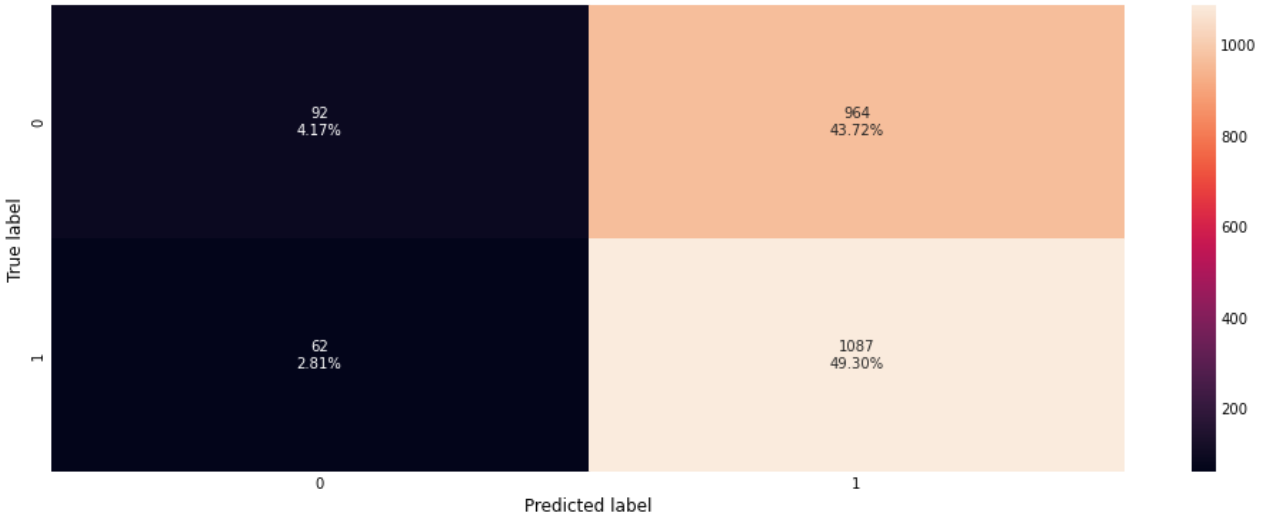
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight="balanced", random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(10, 30, 5),
    "min_samples_leaf": [3, 5, 7],
    "max_leaf_nodes": [2, 3, 5],
    "min_impurity_decrease": [0.0001, 0.001],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    dtree_estimator, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with n_jobs = -1

grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on the train data

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_estimator.fit(X_train, Y_train)
```

Out[82]:

```
DecisionTreeClassifier(class_weight='balanced', max_depth=10, max_leaf_nodes=
5,
                       min_impurity_decrease=0.0001, min_samples_leaf=5,
                       random_state=1)
```

In [83]:

```
confusion_matrix_sklearn(
    dtree_estimator, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```

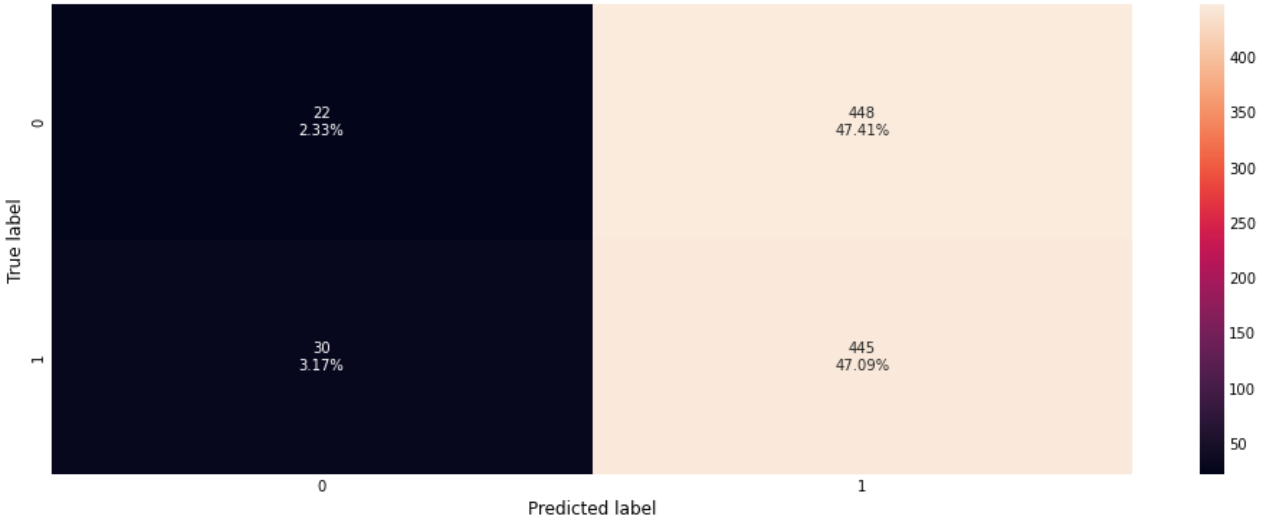


In [84]:
```
# Tunned Training Performance Measures
dtree_estimator_model_train_perf = model_performance_classification_sklearn(
    dtree_estimator, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
dtree_estimator_model_train_perf
```

Out[84]:

	Accuracy	Recall	Precision	F1
0	0.53	0.95	0.53	0.68

In [85]:
```
confusion_matrix_sklearn(
    dtree_estimator, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [86]:
```
# Tunned Test Performance Measures
dtree_estimator_model_test_perf = model_performance_classification_sklearn(
    dtree_estimator, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
dtree_estimator_model_test_perf
```

Out[86]:

	Accuracy	Recall	Precision	F1
0	0.49	0.94	0.50	0.65

Hyperparameter Tuning Weighted Decision Tree

In [87]:

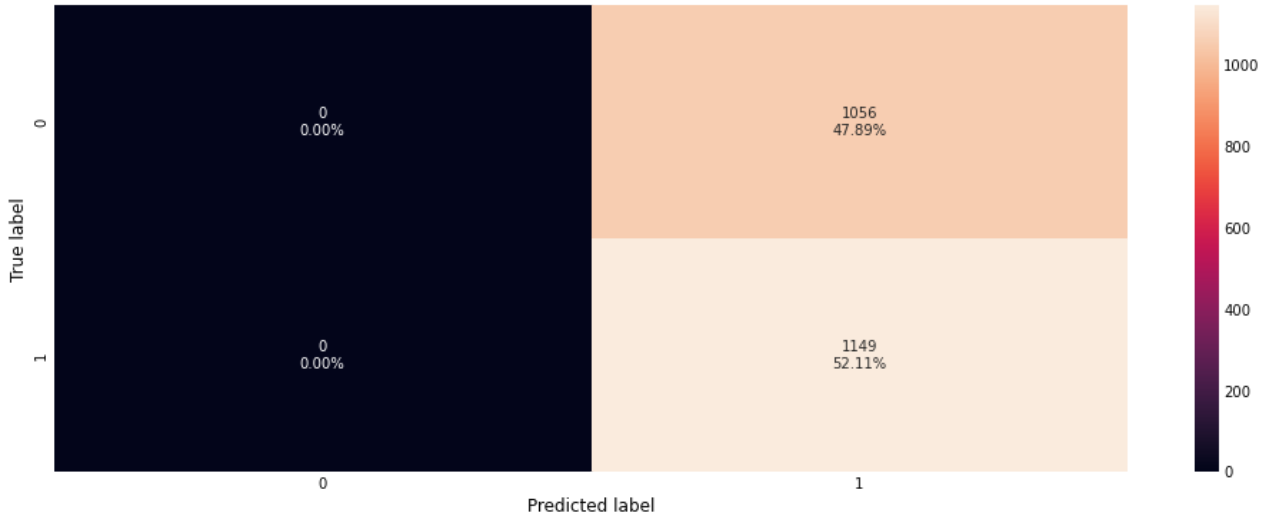
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(
    class_weight={0: 0.17, 1: 0.83}, random_state=1
)

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(2, 30),
    "min_samples_leaf": [1, 2, 5, 7, 10],
    "max_leaf_nodes": [2, 3, 5, 10, 15],
    "min_impurity_decrease": [0.0001, 0.001, 0.01, 0.1],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer)
grid_obj = grid_obj.fit(X_train, Y_train)

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_estimator.fit(X_train, Y_train)
```

Out[87]:

```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83}, max_depth=2,
                       max_leaf_nodes=2, min_impurity_decrease=0.0001,
                       min_samples_leaf=10, random_state=1)
```

In [88]:

```
confusion_matrix_sklearn(dtree_estimator, X_train, Y_train)
```



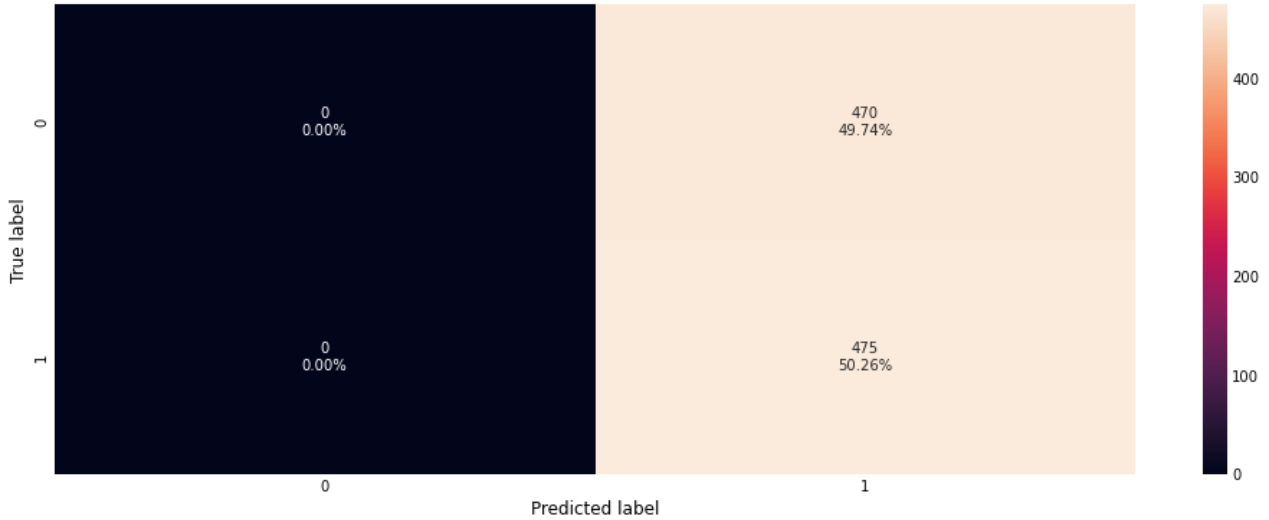
In [89]:
```
# Training Performance Measures
dtree_estimator_model_train_perf = model_performance_classification_sklearn(
    dtree_estimator, X_train, Y_train
)

print("Training performance \n")
dtree_estimator_model_train_perf
```

```
Training performance
```

Out[89]:

	Accuracy	Recall	Precision	F1
0	0.52	1.00	0.52	0.69

In [90]:
```
confusion_matrix_sklearn(dtree_estimator, X_test, Y_test)
```



In [91]:
```
# Testing Performance Measures
dtree_estimator_model_test_perf = model_performance_classification_sklearn(
    dtree_estimator, X_test, Y_test
```

```
)
print("Testing performance \n")
dtree_estimator_model_test_perf
```

```
Testing performance
```

Out[91]:

	Accuracy	Recall	Precision	F1
0	0.50	1.00	0.50	0.67

- Overfitting in decision tree has reduced but the recall has also reduced.
- The decision tree model has a high recall but, the precision is quite lower.
- The performance of the model after hyperparameter tuning can be generalized.
- The coefficients of F1 score of for both train and test dataset are 0.812 and 0.809 respectively.
- The decision tree is overfitting the training data as there is a huge difference between training and test scores for all the metrics.
- The test recall is very low i.e. only 58%.

Bagging Classifier

- ***Bagging refers to bootstrap sampling and aggregation. This means that in bagging at the beginning samples are chosen randomly with replacement to train the individual models and then model predictions undergo aggregation to combine them for the final prediction to consider all the possible outcomes.***
- ***Bagging makes the model more robust since the final prediction is made on the basis of a number of outputs that have been given by a large number of independent models. It prevents overfitting the model to the original data since the individual models do not have access to the original data and are only built on samples that have been randomly chosen from the original data with replacement. Bagging follows the parallel model building i.e the output of individual models is independent of each other.***

Some of the important hyperparameters available for bagging classifier are:

- base_estimator: The base estimator to fit on random subsets of the dataset. If None(default), then the base estimator is a decision tree.
- n_estimators: The number of trees in the forest, default = 100.
- max_features: The number of features to consider when looking for the best split.
- bootstrap: Whether bootstrap samples are used when building trees. If False, the entire dataset is used to build each tree, default=True.
- bootstrap_features: If it is true, then features are drawn with replacement. Default value is False.

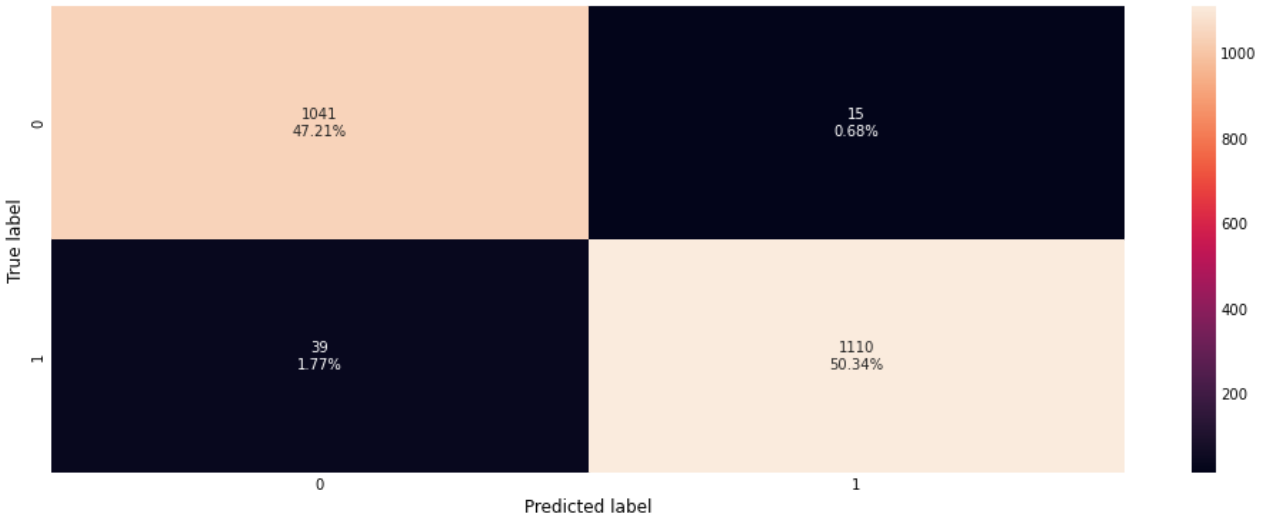
- max_samples: If bootstrap is True, then the number of samples to draw from X to train each base estimator. If None (default), then draw N samples, where N is the number of observations in the train data.
- oob_score: Whether to use out-of-bag samples to estimate the generalization accuracy, default=False.

In [92]:
```
bagging_classifier = BaggingClassifier(
    random_state=1
)  ## Complete the code to define bagging classifier with random state = 1
bagging_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit bagging classifier on the train data
```

Out[92]:
```
BaggingClassifier
BaggingClassifier(random_state=1)
```

Checking model performance on training set

In [93]:
```
confusion_matrix_sklearn(
    bagging_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



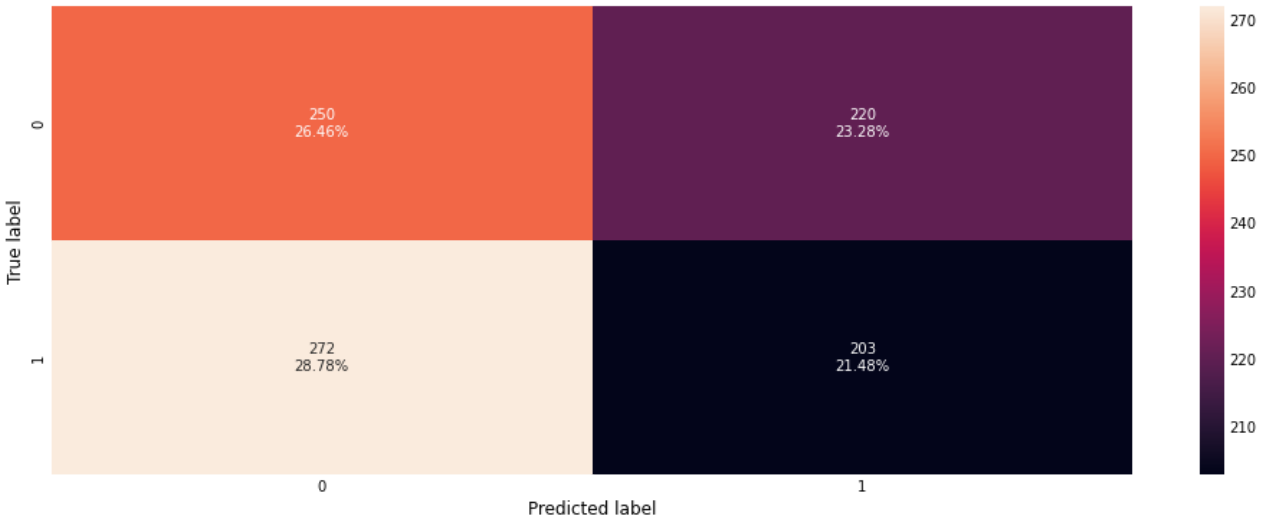
In [94]:
```
bagging_classifier_model_train_perf = model_performance_classification_sklearn(
    bagging_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
bagging_classifier_model_train_perf
```

Out[94]:

	Accuracy	Recall	Precision	F1
0	0.98	0.97	0.99	0.98

Checking model performance on test set

In [95]:
```
confusion_matrix_sklearn(
    bagging_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [96]:
```
bagging_classifier_model_test_perf = model_performance_classification_sklearn(bagging_c
bagging_classifier_model_test_perf
```

Out[96]:

	Accuracy	Recall	Precision	F1
0	0.48	0.43	0.48	0.45

- The overfitting has decrease slightly in the training data
- The test model performance is lower than in hyperparamenter tuned Decision tree

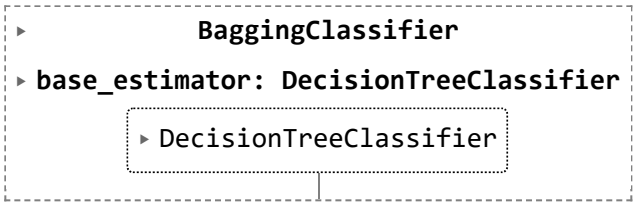
---

Method2: Bagging Classifier - Weighted Decision Tree

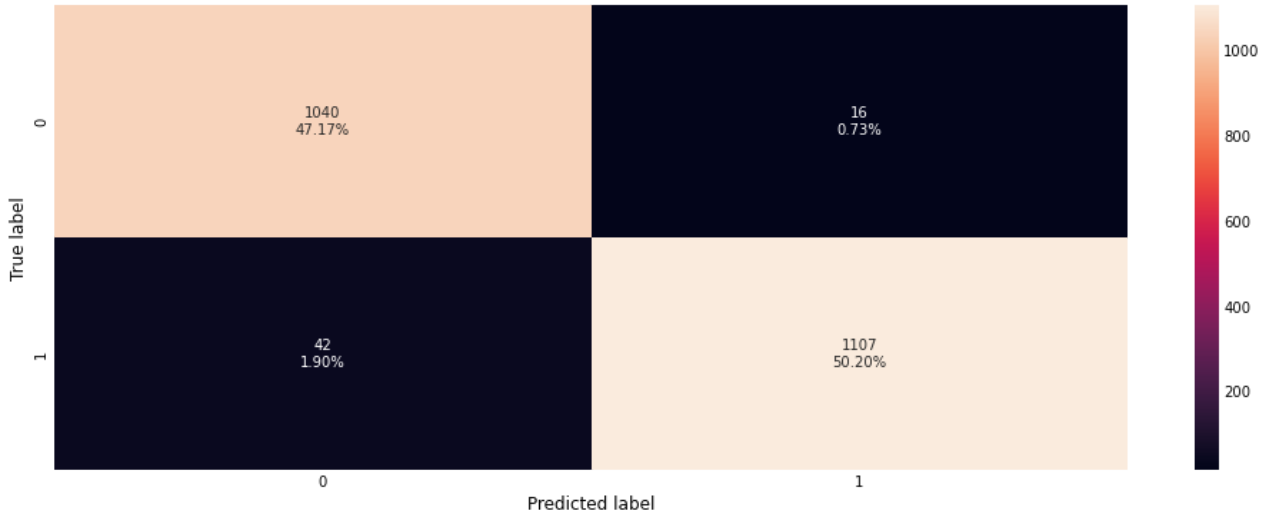
Bagging Classifier with weighted decision tree

In [97]:
```
bagging_wt = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(
        criterion="gini", class_weight={0: 0.17, 1: 0.83}, random_state=1
    ),
    random_state=1,
)
bagging_wt.fit(X_train, Y_train)
```

Out[97]:



In [98]:
```
confusion_matrix_sklearn(bagging_wt, X_train, Y_train)
```



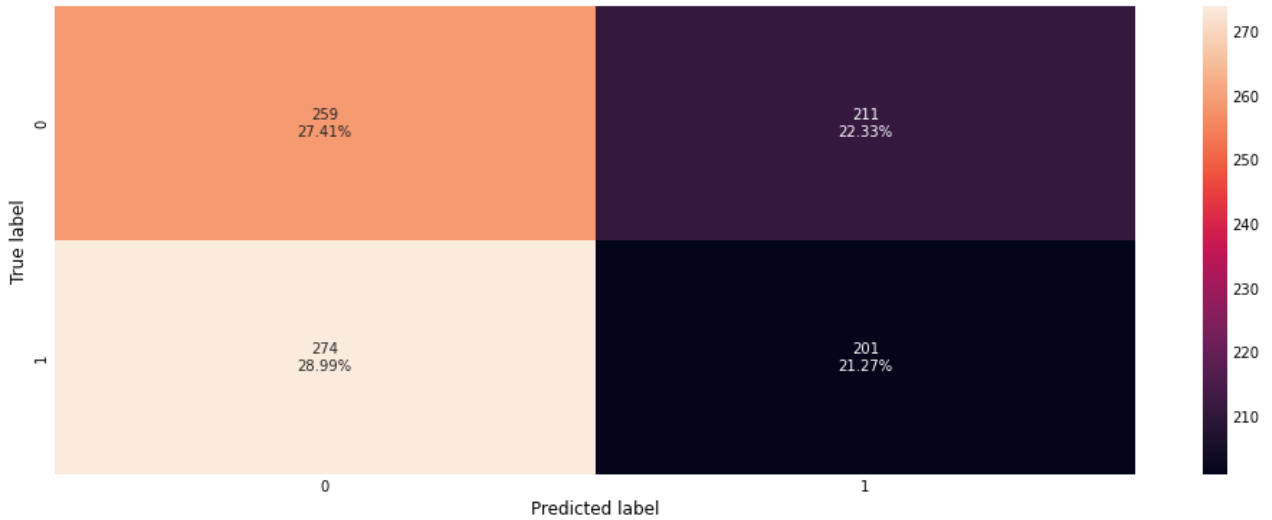
In [99]:
```
# Training Performance Measures
bagging_wt_model_train_perf = model_performance_classification_sklearn(
    bagging_wt, X_train, Y_train
)
print("Training performance \n")
bagging_wt_model_train_perf
```

```
Training performance
```

Out[99]:

	Accuracy	Recall	Precision	F1
0	0.97	0.96	0.99	0.97

In [100…
```
confusion_matrix_sklearn(bagging_wt, X_test, Y_test)
```



In [101…
```
# Testing Performance Measures
```

```
bagging_wt_model_test_perf = model_performance_classification_sklearn(
    bagging_wt, X_test, Y_test
)
print("Testing performance \n")
bagging_wt_model_test_perf
```

```
Testing performance
```

Out[101…

	Accuracy	Recall	Precision	F1
0	0.49	0.42	0.49	0.45

- Bagging classifier with a weighted decision tree is giving very good accuracy and prediction but is not able to generalize well on test data in terms of recall.

Hyperparameter Tuning - Bagging Classifier

Bagging Classifier

Some of the important hyperparameters available for bagging classifier are:

- `base_estimator:` The base estimator to fit on random subsets of the dataset. If None(default), then the base estimator is a decision tree.
- `n_estimators:` The number of trees in the forest, default = 100. max_features: The number of features to consider when looking for the best split.
- `bootstrap:` Whether bootstrap samples are used when building trees. If False, the entire dataset is used to build each tree, default=True.
- `bootstrap_features:` If it is true, then features are drawn with replacement. Default value is False.
- `max_samples:` If bootstrap is True, then the number of samples to draw from X to train each base estimator. If None (default), then draw N samples, where N is the number of observations in the train data.
- `oob_score:` Whether to use out-of-bag samples to estimate the generalization accuracy, default=False.

In [102…

```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_samples": [0.7, 0.8, 0.9],
    "max_features": [0.7, 0.8, 0.9],
    "n_estimators": np.arange(90, 120, 10),
}
```

```
# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    bagging_estimator_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
bagging_estimator_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
bagging_estimator_tuned.fit(X_train, Y_train)
```

Out[102…

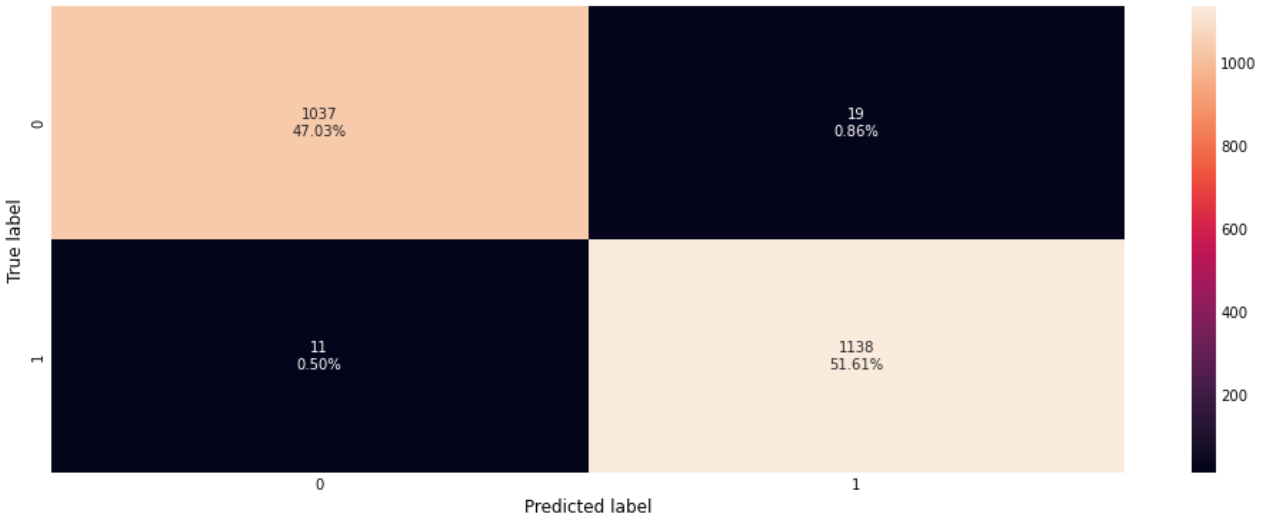
BaggingClassifier

```
BaggingClassifier(max_features=0.7, max_samples=0.7, n_estimators=110,
                  random_state=1)
```

Checking model performance on training set

In [103…

```
confusion_matrix_sklearn(
    bagging_estimator_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [104…

```
# Training Performance Mearsures
bagging_estimator_tuned_model_train_perf = model_performance_classification_sklearn(
    bagging_estimator_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
bagging_estimator_tuned_model_train_perf
```

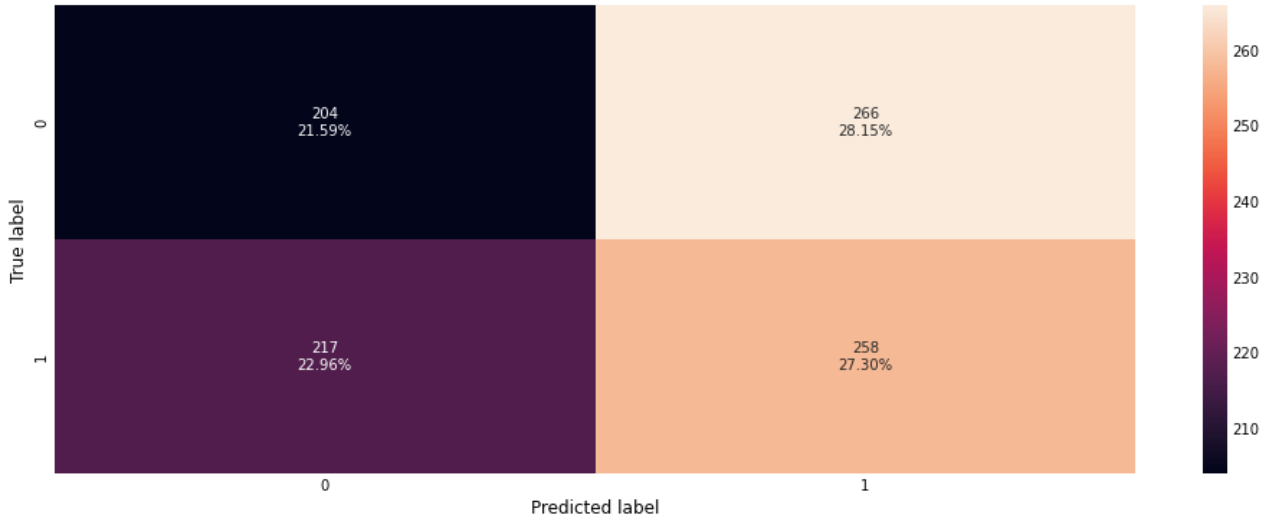
Out[104…

	Accuracy	Recall	Precision	F1
0	0.99	0.99	0.98	0.99

Checking model performance on test set

In [105…

```
confusion_matrix_sklearn(
    bagging_estimator_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [106…

```
# Testing Performce Measures
bagging_estimator_tuned_model_test_perf = model_performance_classification_sklearn(
    bagging_estimator_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
bagging_estimator_tuned_model_test_perf
```

Out[106…

	Accuracy	Recall	Precision	F1
0	0.49	0.54	0.49	0.52

The model performance has increased but the training data is still overfitting

Method 2: Logistic Regression as the base estimator for Bagging Classifier

Let's try using logistic regression as the base estimator for bagging classifier:

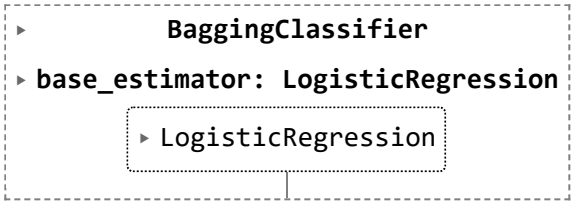
- Now, let's try and change the `base_estimator` of the bagging classifier, which is a decision tree by default.
- We will pass the logistic regression as the base estimator for bagging classifier.

In [107…

```
bagging_lr = BaggingClassifier(
    base_estimator=LogisticRegression(
        solver="liblinear", random_state=1, max_iter=1000
    ),
    random_state=1,
```

```
)
bagging_lr.fit(X_train, Y_train)
```

Out[107…



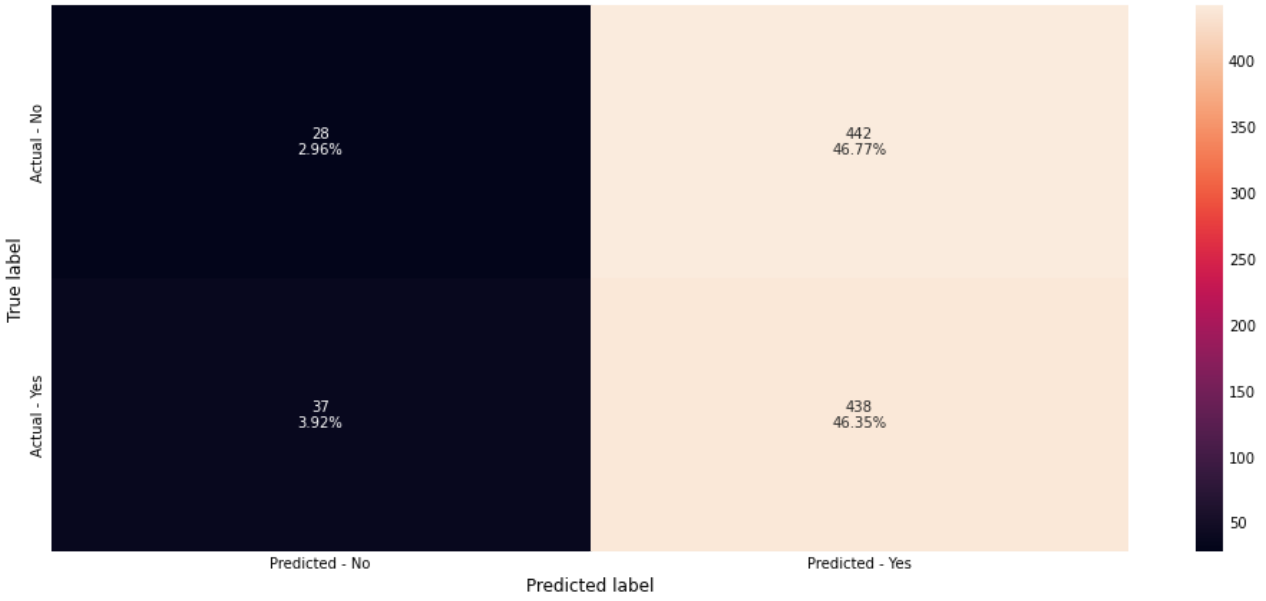
In [108…

```
# Using above defined function to get accuracy, recall and precision on train and test
bagging_lr_score = get_metrics_score(bagging_lr)
```

```
Accuracy on training set :  0.5287981859410431
Accuracy on test set :  0.4931216931216931
Recall on training set :  0.9286335944299391
Recall on test set :  0.9221052631578948
Precision on training set :  0.5271739130434783
Precision on test set :  0.49772727272727274
```

In [109…

```
make_confusion_matrix(bagging_lr, Y_test)
```



Insights

- Bagging classifier with logistic regression as base_estimator is not overfitting the data but the test recall is very low.
- Ensemble models are less interpretable than decision tree but bagging classifier is even less interpretable than random forest. It does not even have a feature importance attribute.

Tuning Bagging Classifier- Weighte Model

In [110…

```
# grid search for bagging classifier
```

```
bagging_estimator_weighted = DecisionTreeClassifier(
    class_weight={0: 0.13, 1: 0.87}, random_state=1
)
param_grid = {
    "base_estimator": [bagging_estimator_weighted],
    "n_estimators": [5, 7, 15, 51, 101],
    "max_features": [0.7, 0.8, 0.9, 1],
}

grid = GridSearchCV(
    BaggingClassifier(random_state=1, bootstrap=True),
    param_grid=param_grid,
    scoring="recall",
    cv=5,
)
grid.fit(X_train, Y_train)

## getting the best estimator
bagging_estimator = grid.best_estimator_
bagging_estimator.fit(X_train, Y_train)
```

Out[110…

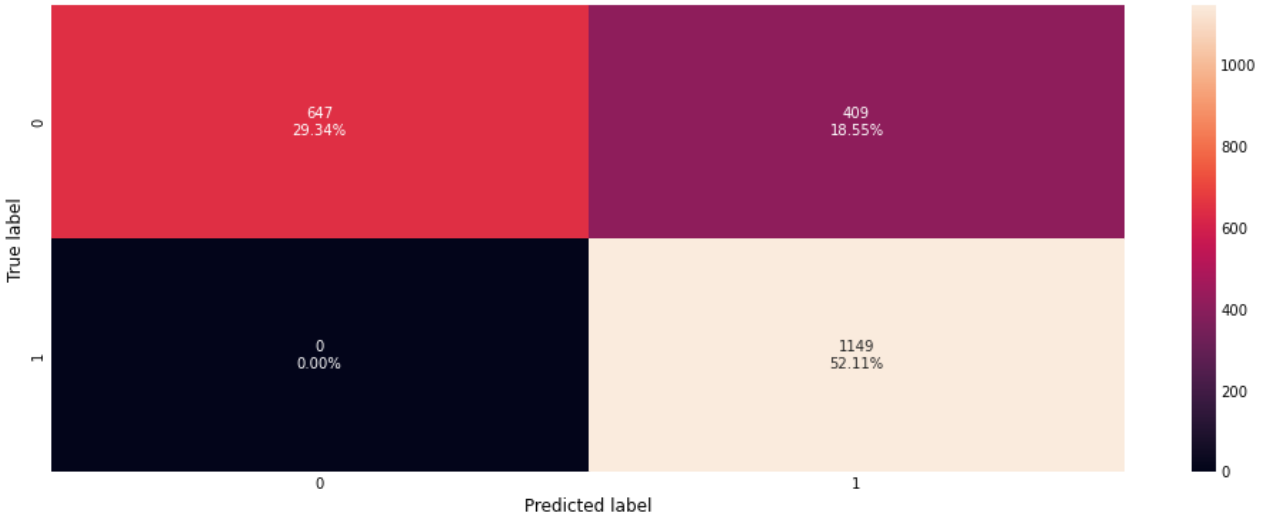
BaggingClassifier

base_estimator: DecisionTreeClassifier

DecisionTreeClassifier

In [111…

```
confusion_matrix_sklearn(bagging_estimator, X_train, Y_train)
```



In [112…

```
# Training Performance Measures
bagging_estimator_model_train_perf = model_performance_classification_sklearn(
    bagging_estimator, X_train, Y_train
)
print("Training performance: \n ")
bagging_estimator_model_train_perf
```

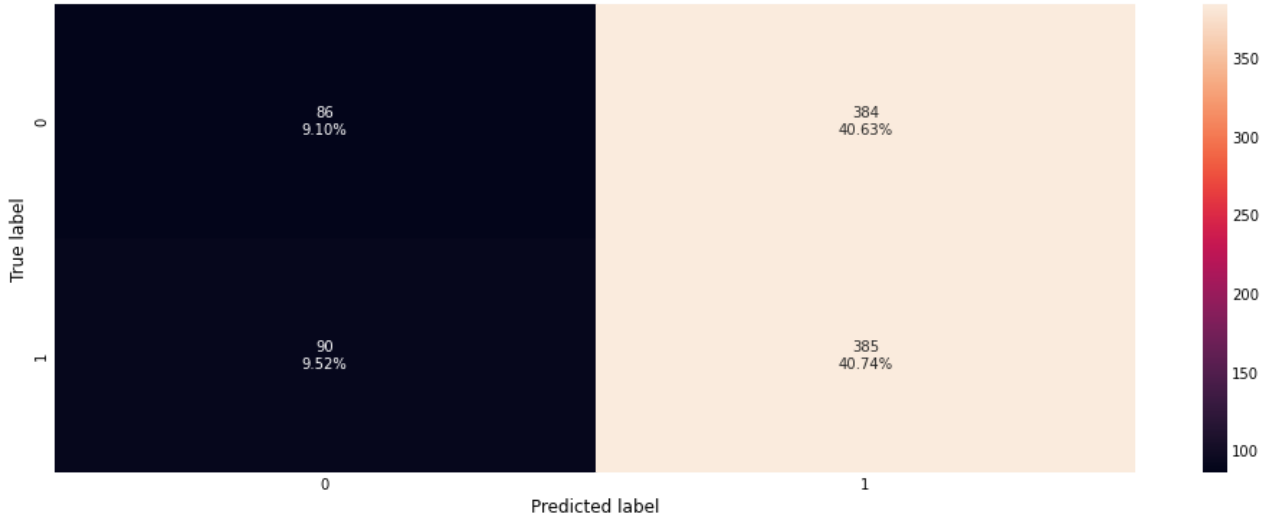
Training performance:

Out[112...

	Accuracy	Recall	Precision	F1
0	0.81	1.00	0.74	0.85

In [113...

```
confusion_matrix_sklearn(bagging_estimator, X_test, Y_test)
```



In [114...

```
# Testing Performance Mearsures
bagging_estimator_model_test_perf = model_performance_classification_sklearn(
    bagging_estimator, X_test, Y_test
)
print("Testing performance \n")
bagging_estimator_model_test_perf
```

```
Testing performance
```

Out[114...

	Accuracy	Recall	Precision	F1
0	0.50	0.81	0.50	0.62

- Recall has improved but the accuracy and precision of the model has dropped drastically which is an indication that overall the model is making many mistakes.

Random Forest

- ***Random forest randomly picks a subset of independent variables for each node's split, where m is the size of the subset and M is the total number of independent variables, where m is generally less than M. This is done to make the individual trees even more independent/different from each other and incorporate more diversity in our final prediction thereby, making the entire model more robust.***

- ***In Random Forest, to get different n-models with the same algorithm, we use Bootstrap aggregation. This means that at the beginning samples are chosen randomly with replacement to train the individual models and then model predictions undergo aggregation to combine them for the final prediction to consider all the possible outcomes.***
- ***The problem of overfitting in a decision tree can be overcome by random forest since the individual trees in a random forest do not have access to the original dataset and are only built on observations that have been sampled with replacement from the original dataset.***
- ***Since the random forest uses multiple tree models to reach a final prediction, it is more robust than a single decision tree model and prevents instabilities due to changes in data. Random forest is less interpretable and has higher computational complexity than decision trees as it utilizes multiple tree models to reach a prediction.***
- ***Random forest prevents overfitting since the individual trees in a random forest do not have access to the original dataset and are only built on observations that have been sampled with replacement from the original dataset. Moreover, aggregation of results from different trees in a random forest reduces the chances of overfitting and so there is no need to prune a random forest.***
- ***In a classification setting, for a new test data point, the final prediction by a random forest is done by taking the mode of the individual predictions while in a regression setting, for a new test data point, the final prediction by a random forest is done by taking the average of individual predictions.***

Random Forest Classifier

Now, let's see if we can get a better model by tuning the random forest classifier. Some of the important hyperparameters available for random forest classifier are:

- n_estimators: The number of trees in the forest, default = 100.
- max_features: The number of features to consider when looking for the best split.
- class_weight: Weights associated with classes in the form {class_label: weight}.If not given, all classes are supposed to have weight one.
- For example: If the frequency of class 0 is 80% and the frequency of class 1 is 20% in the data, then class 0 will become the dominant class and the model will become biased toward the dominant classes. In this case, we can pass a dictionary {0:0.2,1:0.8} to the model to specify the weight of each class and the random forest will give more weightage to class 1.
- bootstrap: Whether bootstrap samples are used when building trees. If False, the entire dataset is used to build each tree, default=True.
- max_samples: If bootstrap is True, then the number of samples to draw from X to train each base estimator. If None (default), then draw N samples, where N is the number of observations in the train data.

- oob_score: Whether to use out-of-bag samples to estimate the generalization accuracy, default=False.
- Note: A lot of hyperparameters of Decision Trees are also available to tune Random Forest like max_depth, min_sample_split etc.

In [115…

```
# Fitting the model
rf_estimator = RandomForestClassifier(
    random_state=1
)  ## Complete the code to define random forest with random state = 1 and class_weight
rf_estimator.fit(
    X_train, Y_train
)  ## Complete the code to fit random forest on the train data
```

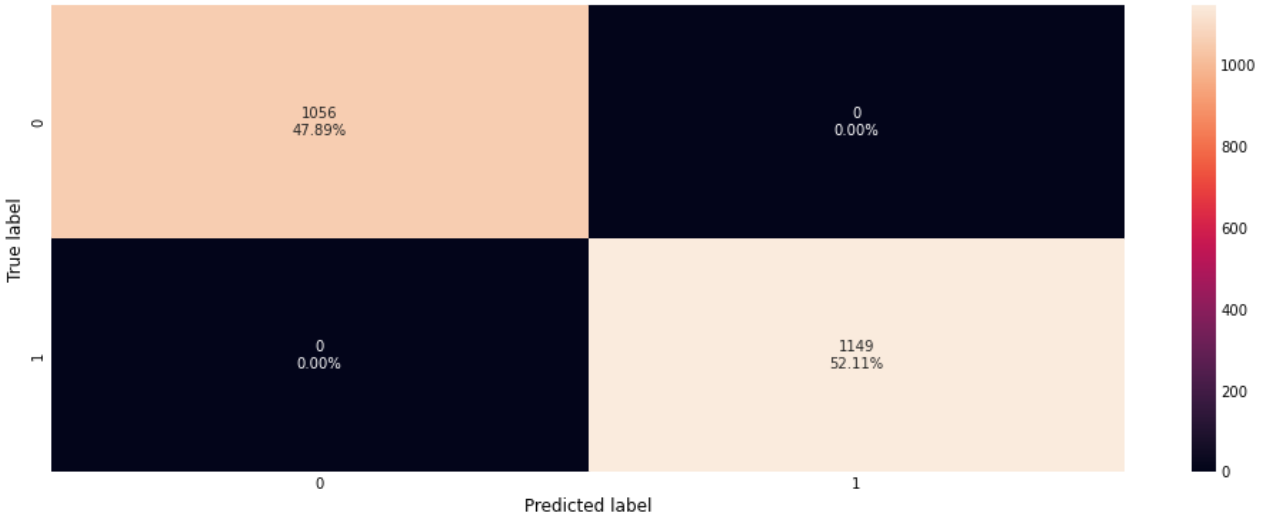
Out[115…

```
RandomForestClassifier
RandomForestClassifier(random_state=1)
```

Checking model performance on training set

In [116…

```
confusion_matrix_sklearn(
    rf_estimator, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [117…

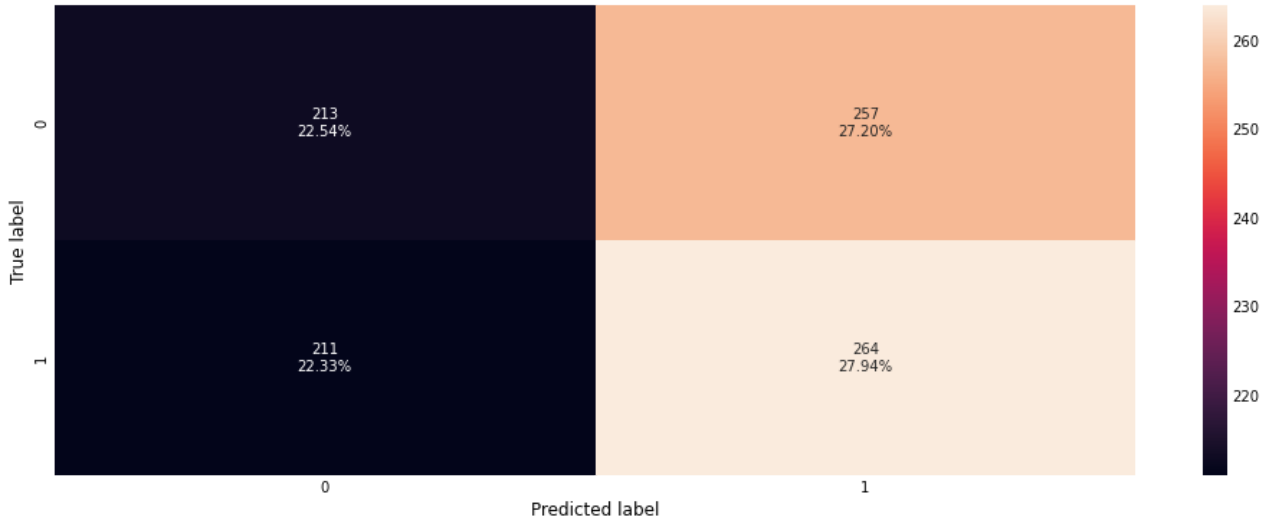
```
# Training Performance Measures
rf_estimator_model_train_perf = model_performance_classification_sklearn(
    rf_estimator, X_train, Y_train
)  ## Complete the code to check performance on train data
rf_estimator_model_train_perf
```

Out[117…

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

Checking model performance on test set

In [118…]
```
confusion_matrix_sklearn(
    rf_estimator, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [119…]
```
# Testing Prformance Measures
rf_estimator_model_test_perf = model_performance_classification_sklearn(
    rf_estimator, X_test, Y_test
)  ## Complete the code to check performance for test data
rf_estimator_model_test_perf
```

Out[119…]

	Accuracy	Recall	Precision	F1
0	0.50	0.56	0.51	0.53

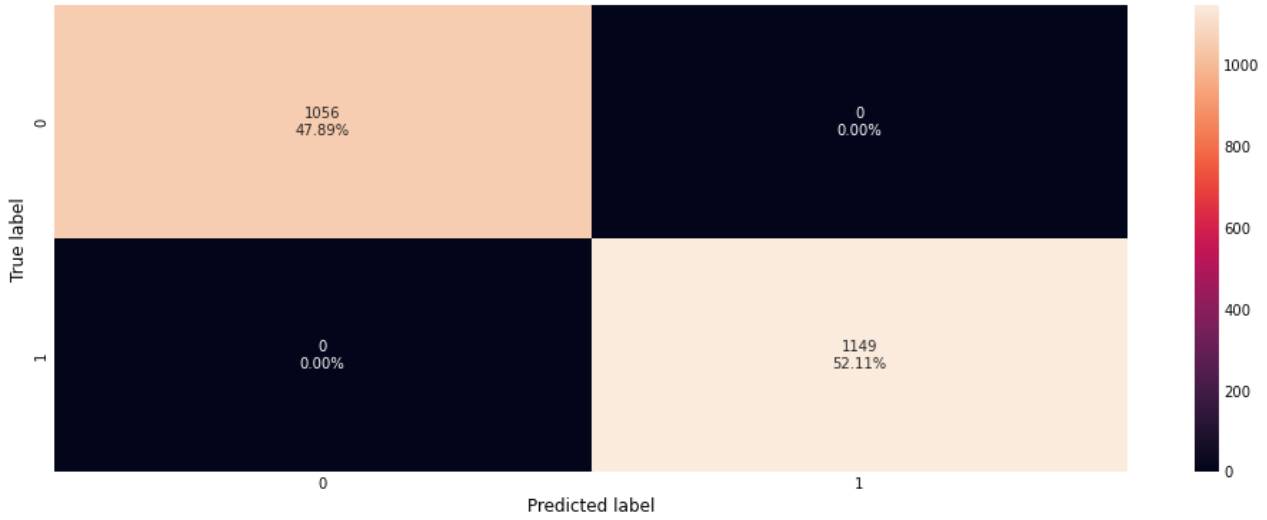
Random Forest -Weighted Class

Random forest with class weights

In [120…]
```
rf_wt = RandomForestClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
rf_wt.fit(X_train, Y_train)
```

Out[120…]
```
RandomForestClassifier
RandomForestClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
```

In [121…]
```
confusion_matrix_sklearn(rf_wt, X_train, Y_train)
```



In [122…

```
# Training Performance Measures
rf_wt_model_train_perf = model_performance_classification_sklearn(
    rf_wt, X_train, Y_train
)
print("Training performance \n")
rf_wt_model_train_perf
```

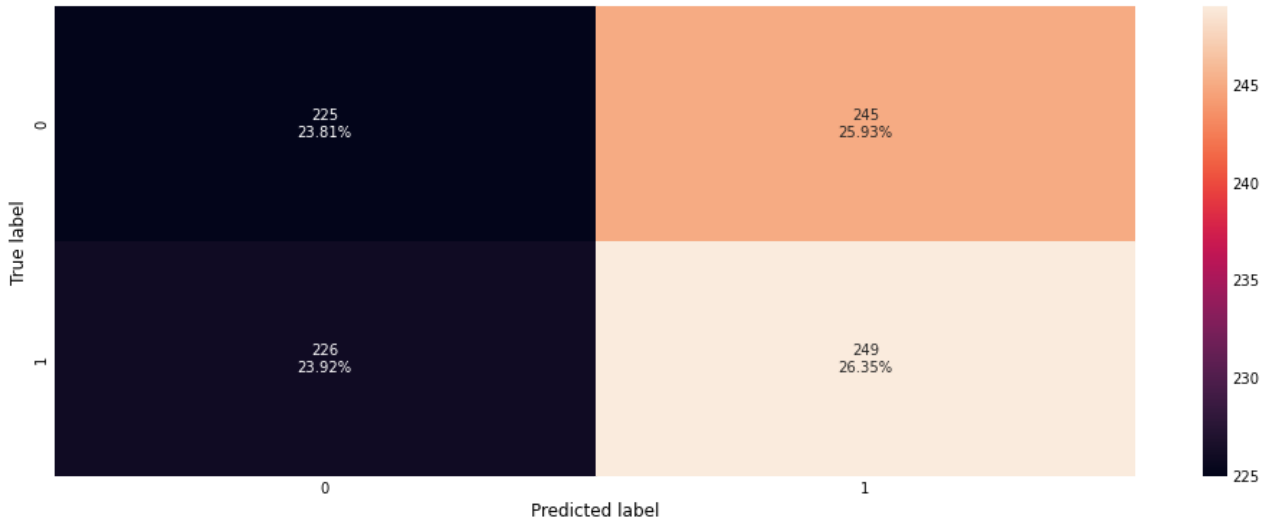
Training performance

Out[122…

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

In [123…

```
confusion_matrix_sklearn(rf_wt, X_test, Y_test)
```



In [124…

```
# Testing Performance Measures
rf_wt_model_test_perf = model_performance_classification_sklearn(rf_wt, X_test, Y_test)
print("Testing performance \n")
```

```
rf_wt_model_test_perf
```

```
Testing performance
```

Out[124…

	Accuracy	Recall	Precision	F1
0	0.50	0.52	0.50	0.51

- There is not much improvement in metrics of weighted random forest as compared to the unweighted random forest.

- Random forest is overfitting the training data as there is a huge difference between training and test scores for all the metrics.
- The test recall is even lower than the decision tree but has a higher test precision.

Hyperparameter Tuning - Random Forest

- ***The class_weight is the hyperparameter of Random Forest which is useful in dealing with imbalanced data by giving more importance to the minority class. By giving more class_weight to a certain class than the other class, we tell the model that it is more important to correctly predict a certain class than the other class.***

Let's try using class_weights for random forest:

- The model performance is not very good. This may be due to the fact that the classes are imbalanced with 70% non-defaulters and 30% defaulters.

- We should make the model aware that the class of interest here is 'defaulters'.

- We can do so by passing the parameter `class_weights` available for random forest. This parameter is not available for the bagging classifier.

- class_weight specifies the weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.

- We can choose class_weights={0:0.3,1:0.7} because that is the original imbalance in our data.

In [125…

```
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(random_state=1, oob_score=True, bootstrap=True)

parameters = {
    "max_depth": list(np.arange(5, 15, 5)),
    "max_features": ["sqrt", "log2"],
    "min_samples_split": [3, 5, 7],
    "n_estimators": np.arange(10, 40, 10),
}

# Type of scoring used to compare parameter combinations
```

```
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    rf_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5 and n_jobs = -1
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on the train data

# Set the clf to the best combination of parameters
rf_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rf_tuned.fit(X_train, Y_train)
```

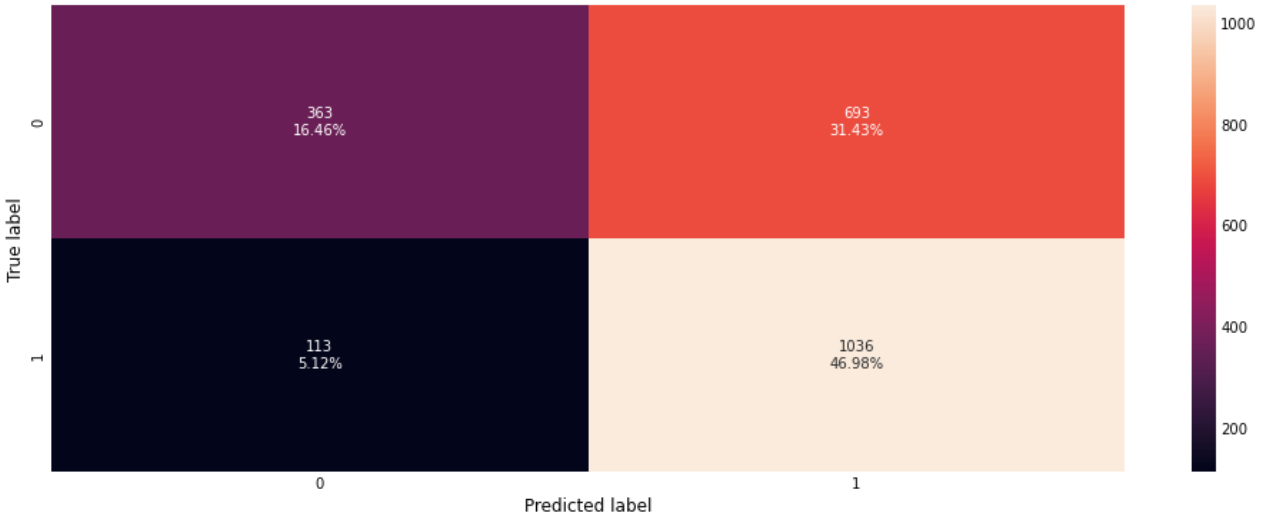
Out[125…

```
RandomForestClassifier(max_depth=5, min_samples_split=3, n_estimators=30,
                       oob_score=True, random_state=1)
```

Checking model performance on training set

In [126…

```
confusion_matrix_sklearn(
    rf_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [127…

```
# Training Performance Measures
rf_tuned_model_train_perf = model_performance_classification_sklearn(
    rf_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
rf_tuned_model_train_perf
```

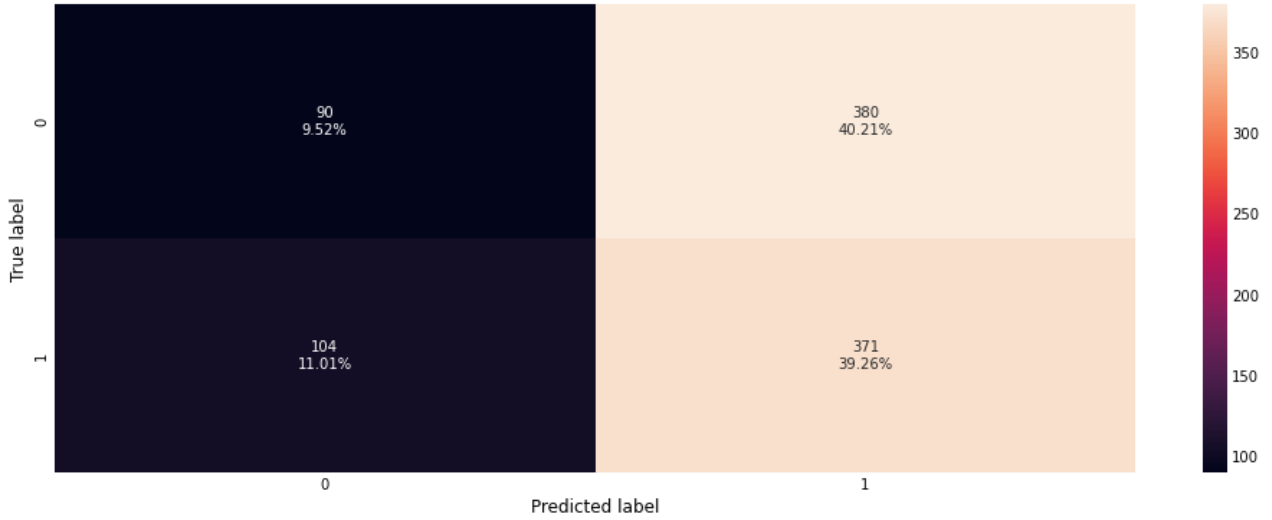
Out[127…

	Accuracy	Recall	Precision	F1
0	0.63	0.90	0.60	0.72

Checking model performance on test set

In [128…

```
confusion_matrix_sklearn(
    rf_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [129…

```
# Test Performance Measures
rf_tuned_model_test_perf = model_performance_classification_sklearn(
    rf_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
rf_tuned_model_test_perf
```

Out[129…

	Accuracy	Recall	Precision	F1
0	0.49	0.78	0.49	0.61

Method 2

Class_Weights for Random Forest - Hyperparameter Tuning

Let's try using class_weights for random forest:

- The model performance is not very good. This may be due to the fact that the classes are imbalanced with 70% non-defaulters and 30% defaulters.
- We should make the model aware that the class of interest here is 'defaulters'.
- We can do so by passing the parameter `class_weights` available for random forest. This parameter is not available for the bagging classifier.

- class_weight specifies the weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.
- We can choose class_weights={0:0.3,1:0.7} because that is the original imbalance in our data.

In [130…

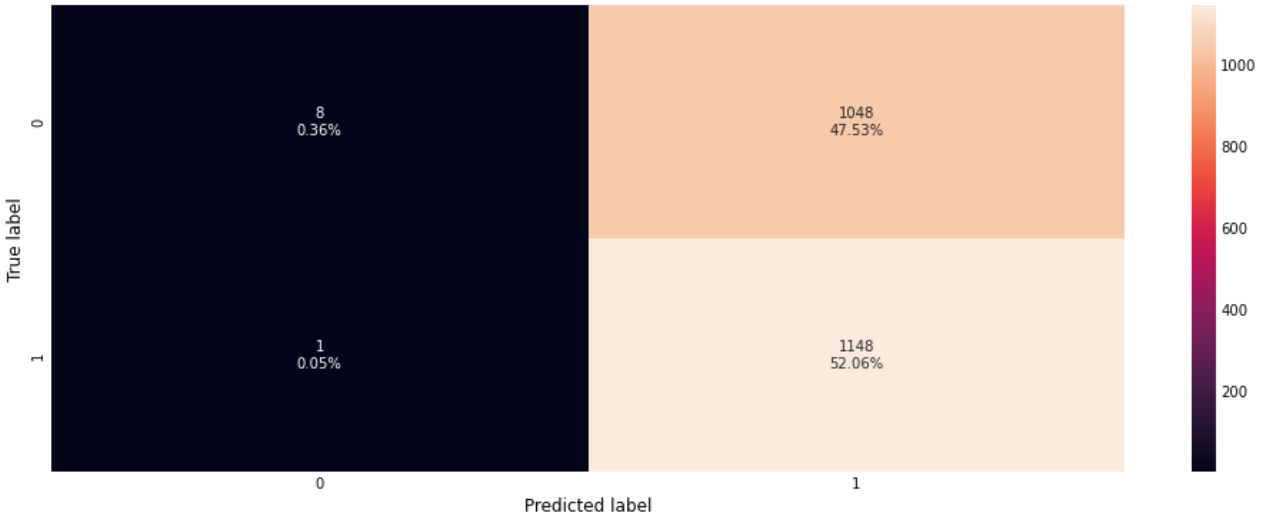
```
# Choose the type of classifier.
rf_estimator_weighted = RandomForestClassifier(random_state=1)

# Grid of parameters to choose from
## add from article
parameters = {
    "class_weight": [{0: 0.3, 1: 0.7}],
    "n_estimators": [100, 150, 200, 250],
    "min_samples_leaf": np.arange(5, 10),
    "max_features": np.arange(0.2, 0.7, 0.1),
    "max_samples": np.arange(0.3, 0.7, 0.1),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(rf_estimator_weighted, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, Y_train)

# Set the clf to the best combination of parameters
rf_estimator_weighted = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rf_estimator_weighted.fit(X_train, Y_train)
```

Out[130…

```
RandomForestClassifier(class_weight={0: 0.3, 1: 0.7}, max_features=0.2,
                       max_samples=0.3, min_samples_leaf=9, random_state=1)
```

In [131…

```
confusion_matrix_sklearn(rf_estimator_weighted, X_train, Y_train)
```



In [132…

```
# Training Performance Measures
rf_wt_model_train_perf = model_performance_classification_sklearn(
    rf_estimator_weighted, X_train, Y_train
)
print("Training performance \n")
rf_wt_model_train_perf
```

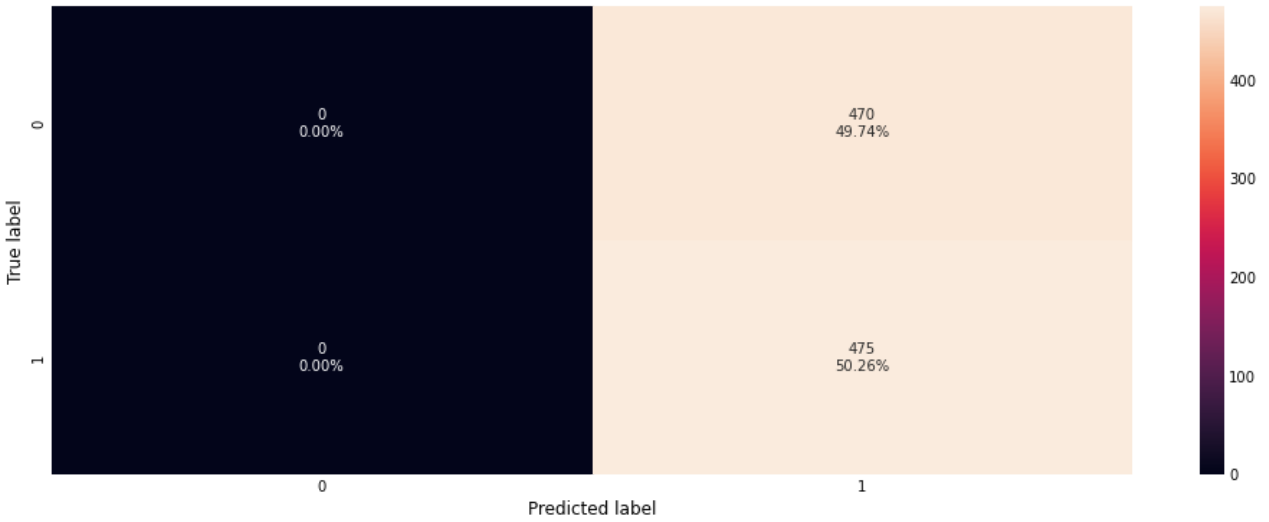
```
Training performance
```

Out[132…

	Accuracy	Recall	Precision	F1
0	0.52	1.00	0.52	0.69

In [133…

```
confusion_matrix_sklearn(rf_estimator_weighted, X_test, Y_test)
```



In [134…

```
# Testing Performance Measures
rf_wt_model_test_perf = model_performance_classification_sklearn(
    rf_estimator_weighted, X_test, Y_test
)
print("Testing performance \n")
rf_wt_model_test_perf
```

```
Testing performance
```

Out[134…

	Accuracy	Recall	Precision	F1
0	0.50	1.00	0.50	0.67

- Random forest after tuning has given same performance as un-tuned random forest.

Summary Performance Measures of Bagging Models Train vs Test Models

In [135…

```
# Performance comparison of Bagging models

bagging_comp_df = pd.concat(
    [
        decision_tree_perf_train.T,
        decision_tree_perf_test.T,
        rf_estimator_model_train_perf.T,
        rf_estimator_model_test_perf.T,
        bagging_classifier_model_train_perf.T,
        bagging_classifier_model_test_perf.T,
    ],
    axis=1,
)
bagging_comp_df.columns = [
    "Decision Tree (train)",
    "Decision Tree (test)",
    "Random Forest (train)",
    "Random Forest (test)",
    "Bagging Classifier (train)",
    "Bagging Classifier (test)",
]
print("Training performance comparison:")
bagging_comp_df
```

```
Training performance comparison:
```

Out[135…

	Decision Tree (train)	Decision Tree (test)	Random Forest (train)	Random Forest (test)	Bagging Classifier (train)	Bagging Classifier (test)
Accuracy	1.00	0.52	1.00	0.50	0.98	0.48
Recall	1.00	0.55	1.00	0.56	0.97	0.43
Precision	1.00	0.52	1.00	0.51	0.99	0.48
F1	1.00	0.53	1.00	0.53	0.98	0.45

Summary Performance Measures of Bagging Models Tuning Train vs Tuning Test Models

In [136…

```
bagging_tuned_comp_df = pd.concat(
    [
        dtree_estimator_model_train_perf.T,
        dtree_estimator_model_test_perf.T,
        rf_tuned_model_train_perf.T,
        rf_tuned_model_test_perf.T,
        bagging_estimator_tuned_model_train_perf.T,
        bagging_estimator_tuned_model_test_perf.T,
    ],
    axis=1,
)
bagging_tuned_comp_df.columns = [
    "Decision Tree Tuned(train)",
    "Decision Tree Tuned(test)",
    "Random Forest Tuned(train)",
```

```
    "Random Forest Tuned(test)",
    "Bagging Classifier Tuned (train)",
    "Bagging Classifier Tuned (test)",
    ]
print("Bagging tuned model performance comparison:")
bagging_tuned_comp_df
```

```
Bagging tuned model performance comparison:
```

Out[136…

	Decision Tree Tuned(train)	Decision Tree Tuned(test)	Random Forest Tuned(train)	Random Forest Tuned(test)	Bagging Classifier Tuned (train)	Bagging Classifier Tuned (test)
Accuracy	0.52	0.50	0.63	0.49	0.99	0.49
Recall	1.00	1.00	0.90	0.78	0.99	0.54
Precision	0.52	0.50	0.60	0.49	0.98	0.49
F1	0.69	0.67	0.72	0.61	0.99	0.52

Boosting Decision Tree Models

AdaBoost Classifier

In [137…

```
ab_classifier = AdaBoostClassifier(
    random_state=1
)  ## Complete the code to define AdaBoost Classifier with random state = 1
ab_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit AdaBoost Classifier on the train data
```

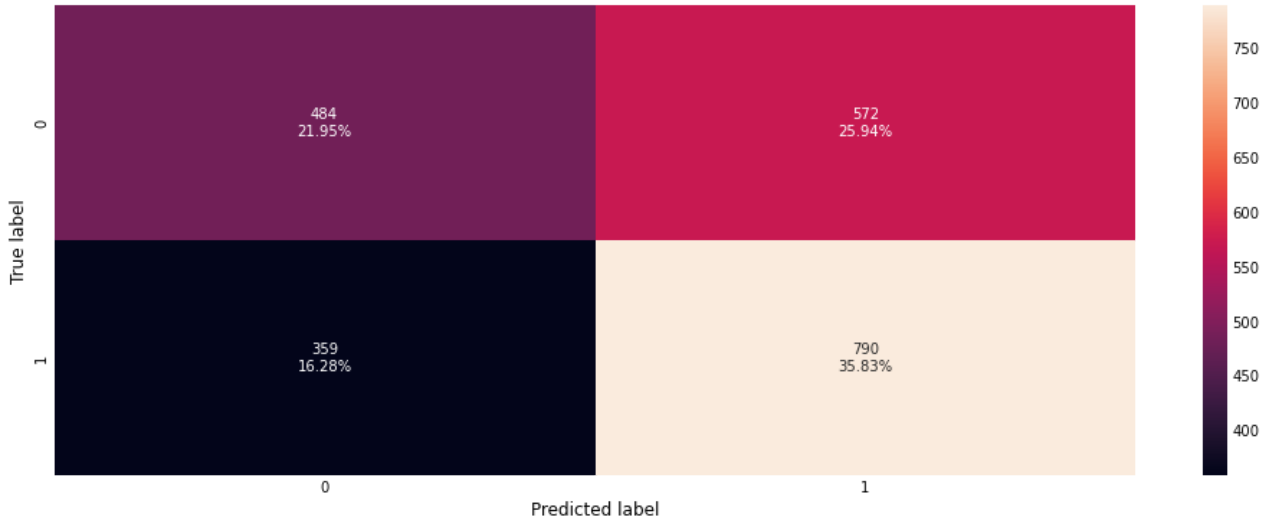
Out[137…

```
▾        AdaBoostClassifier

AdaBoostClassifier(random_state=1)
```

Checking model performance on training set

In [138…

```
confusion_matrix_sklearn(
    ab_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [139…

```
ab_classifier_model_train_perf = model_performance_classification_sklearn(
    ab_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
ab_classifier_model_train_perf
```

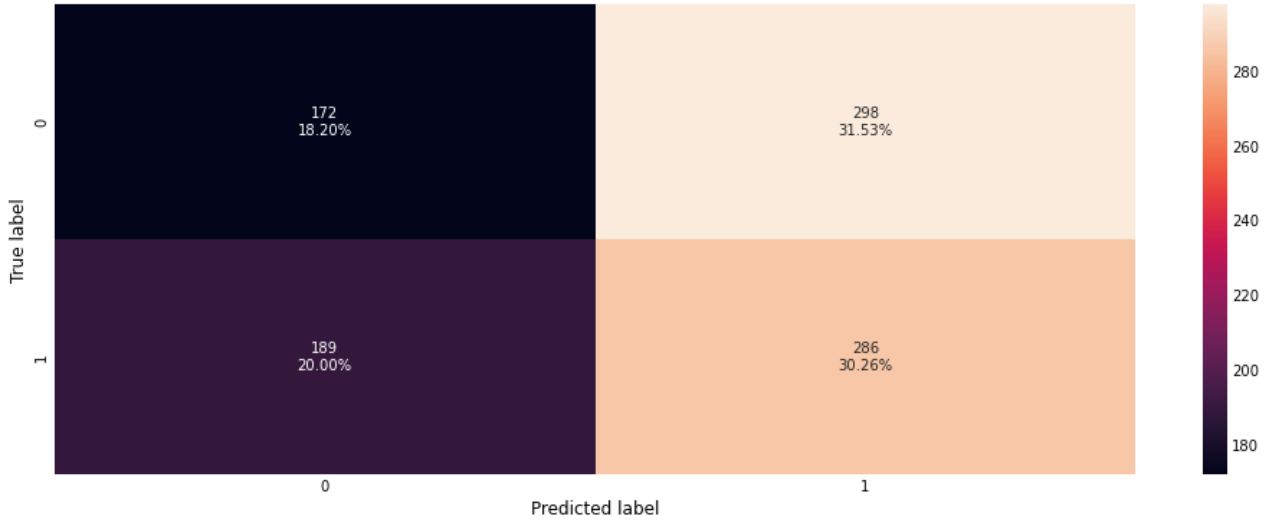
Out[139…

	Accuracy	Recall	Precision	F1
0	0.58	0.69	0.58	0.63

Checking model performance on test set

In [140…

```
confusion_matrix_sklearn(
    ab_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [141…

```
ab_classifier_model_test_perf = model_performance_classification_sklearn(
    ab_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
ab_classifier_model_test_perf
```

Out[141…

	Accuracy	Recall	Precision	F1
0	0.48	0.60	0.49	0.54

- Overfitting is further reduced
- Model performance is similar to that of tuned Random Forest

---

Hyperparameter Tuning - AdaBoost Classifier

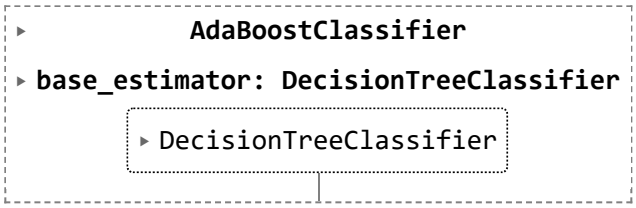
- An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
- Some important hyperparamters are:
  - base_estimator: The base estimator from which the boosted ensemble is built. By default the base estimator is a decision tree with max_depth=1
  - n_estimators: The maximum number of estimators at which boosting is terminated. Default value is 50.
  - learning_rate: Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.

In [142…

```
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    # Let's try different max_depth for base_estimator
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, class_weight="balanced", random_state=1),
        DecisionTreeClassifier(max_depth=2, class_weight="balanced", random_state=1),
        DecisionTreeClassifier(max_depth=3, class_weight="balanced", random_state=1),
    ],
    "n_estimators": np.arange(60, 100, 10),
    "learning_rate": np.arange(0.1, 0.4, 0.1),
}

# Type of scoring used to compare parameter  combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    abc_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
```

```
abc_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
abc_tuned.fit(X_train, Y_train)
```

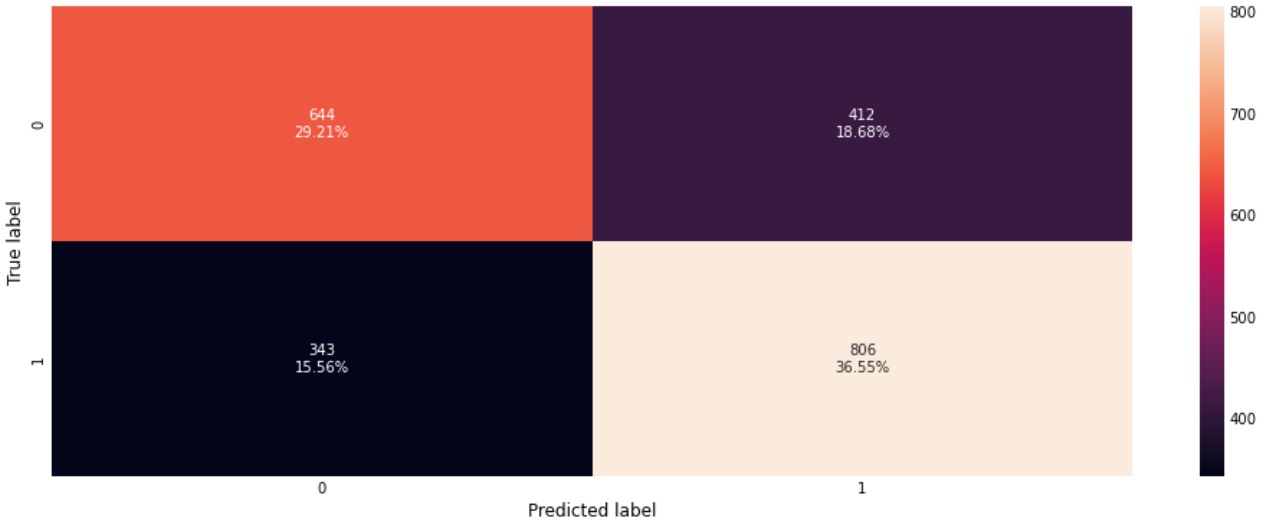
Out[142…



Checking model performance on training set

In [143…

```
confusion_matrix_sklearn(
    abc_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [144…

```
abc_tuned_model_train_perf = model_performance_classification_sklearn(
    abc_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
abc_tuned_model_train_perf
```

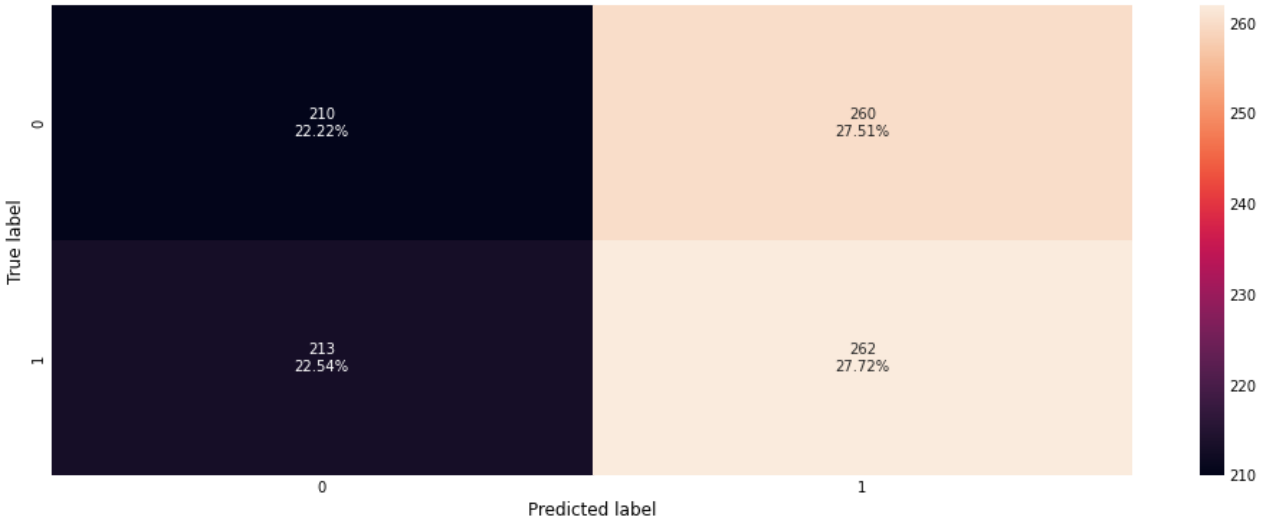
Out[144…

	Accuracy	Recall	Precision	F1
0	0.66	0.70	0.66	0.68

Checking model performance on test set

In [145…

```
confusion_matrix_sklearn(
    abc_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [146…

```
abc_tuned_model_test_perf = model_performance_classification_sklearn(
    abc_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
abc_tuned_model_test_perf
```

Out[146…

	Accuracy	Recall	Precision	F1
0	0.50	0.55	0.50	0.53

- Tuning reduces over fitting
- Precision is improved but F1 is reduced

Gradient Boosting Classifier

- Most of the hyperparameters available are same as random forest classifier.
- init: An estimator object that is used to compute the initial predictions. If 'zero', the initial raw predictions are set to zero. By default, a DummyEstimator predicting the classes priors is used.
- There is no class_weights parameter in gradient boosting.

In [147…

```
gb_classifier = GradientBoostingClassifier(
    random_state=1
)  ## Complete the code to define Gradient Boosting Classifier with random state = 1
gb_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit Gradient Boosting Classifier on the train data
```

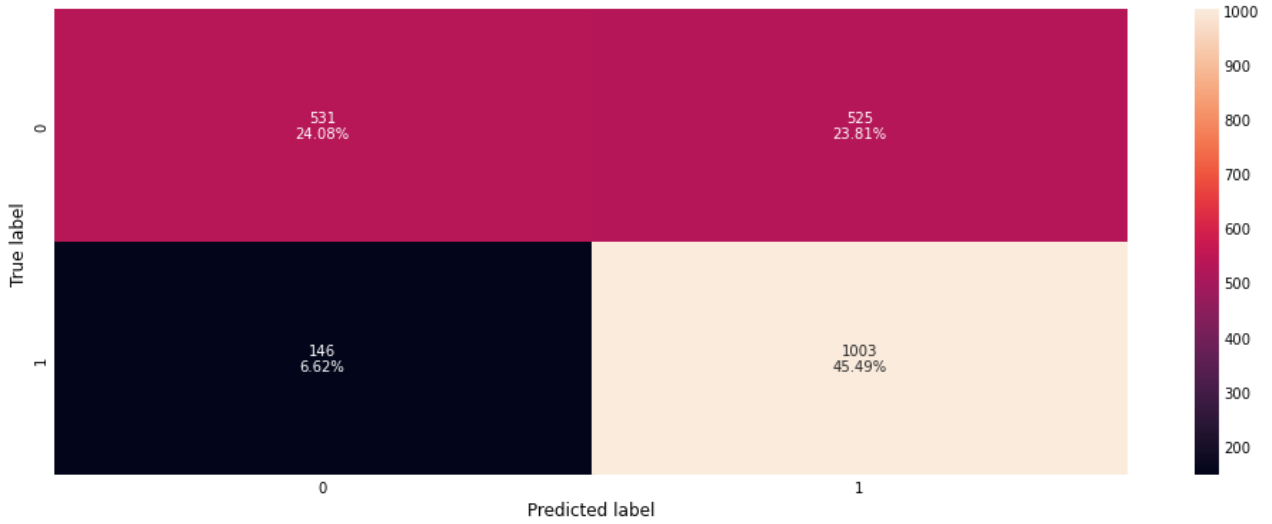
Out[147…

```
GradientBoostingClassifier(random_state=1)
```

Checking model performance on training set

In [148…

```
confusion_matrix_sklearn(
    gb_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [149…

```
gb_classifier_model_train_perf = model_performance_classification_sklearn(
    gb_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
gb_classifier_model_train_perf
```

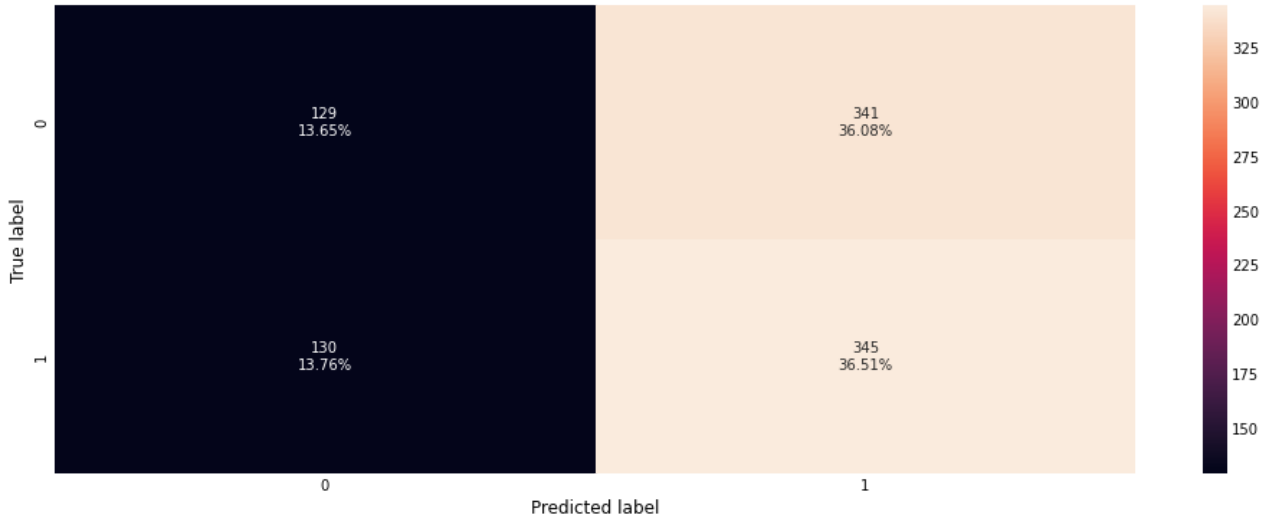
Out[149…

	Accuracy	Recall	Precision	F1
0	0.70	0.87	0.66	0.75

Checking model performance on test set

In [150…

```
confusion_matrix_sklearn(
    gb_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [151…
```
gb_classifier_model_test_perf = model_performance_classification_sklearn(
    gb_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
gb_classifier_model_test_perf
```

Out[151…

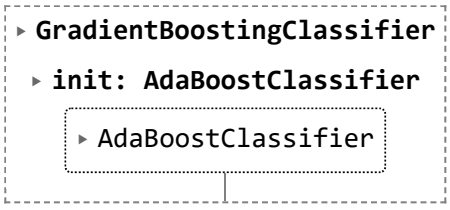
	Accuracy	Recall	Precision	F1
0	0.50	0.73	0.50	0.59

- No significant increase in the model performance

Hyperparameter Tuning - Gradient Boosting Classifier

In [152…
```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1))

# Grid of parameters to choose from
parameters = {
    "n_estimators": [200, 250, 300],
    "subsample": [0.8, 0.9, 1],
    "max_features": [0.7, 0.8, 0.9, 1],
    "learning_rate": np.arange(0.1, 0.4, 0.1),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    gbc_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
gbc_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
gbc_tuned.fit(X_train, Y_train)
```

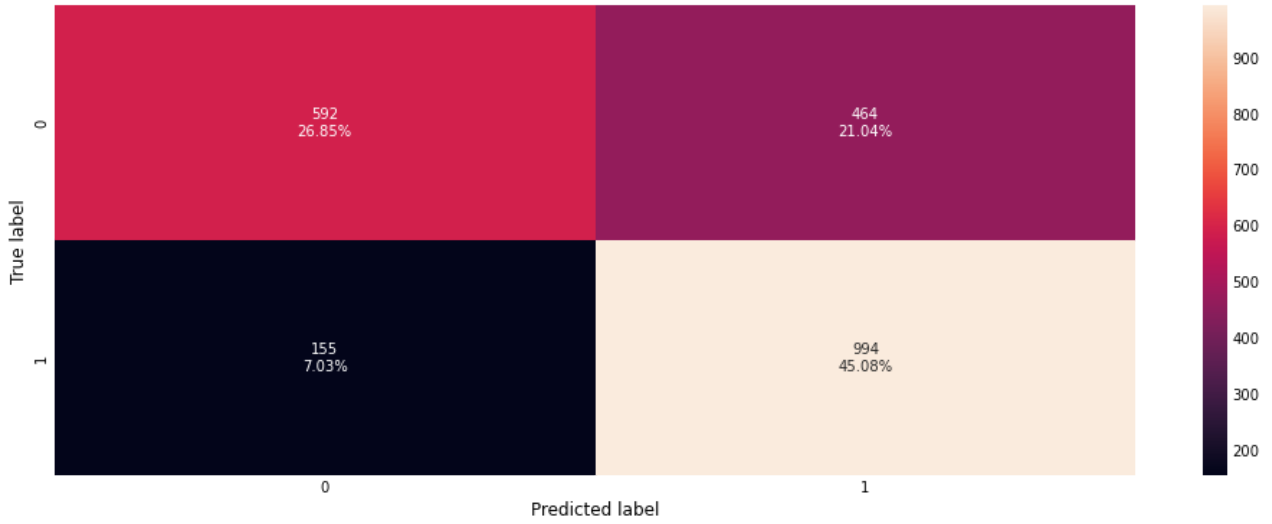
Out[152…



Checking model performance on training set

In [153…
```
confusion_matrix_sklearn(
    gbc_tuned, X_train, Y_train
```

```
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [154…

```
gbc_tuned_model_train_perf = model_performance_classification_sklearn(
    gbc_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
gbc_tuned_model_train_perf
```

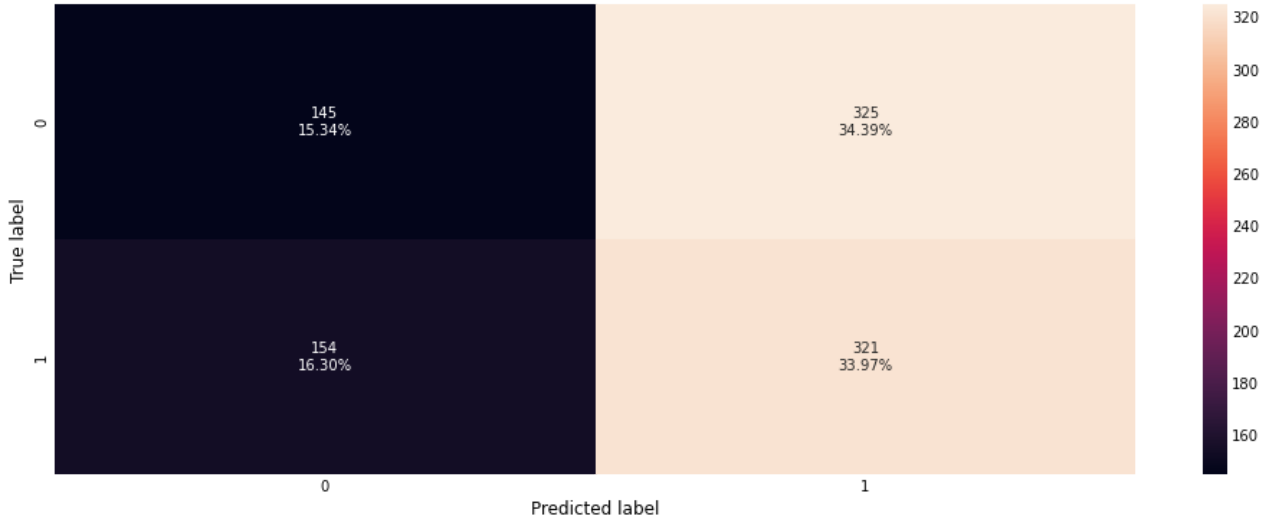
Out[154…

	Accuracy	Recall	Precision	F1
0	0.72	0.87	0.68	0.76

Checking model performance on test set

In [155…

```
confusion_matrix_sklearn(
    gbc_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [156…

```
gbc_tuned_model_test_perf = model_performance_classification_sklearn(
    gbc_tuned, X_test, Y_test
```

```
)  ## Complete the code to check performance for test data on tuned estimator
gbc_tuned_model_test_perf
```

Out[156…

	Accuracy	Recall	Precision	F1
0	0.49	0.68	0.50	0.57

Performance of Gradient Boster remains the same after hyperparameter tuning

XGBoost Classifier

XGBoost has many hyper parameters which can be tuned to increase the model performance.
Some of the important parameters are:

- scale_pos_weight:Control the balance of positive and negative weights, useful for unbalanced classes. It has range from 0 to ∞.
- subsample: Corresponds to the fraction of observations (the rows) to subsample at each step. By default it is set to 1 meaning that we use all rows.
- colsample_bytree: Corresponds to the fraction of features (the columns) to use.
- colsample_bylevel: The subsample ratio of columns for each level. Columns are subsampled from the set of columns chosen for the current tree.
- colsample_bynode: The subsample ratio of columns for each node (split). Columns are subsampled from the set of columns chosen for the current level.
- max_depth: is the maximum number of nodes allowed from the root to the farthest leaf of a tree.
- learning_rate/eta: Makes the model more robust by shrinking the weights on each step.
- gamma: A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.

In [157…

```
xgb_classifier = XGBClassifier(
    random_state=1
)  ## Complete the code to define XGBoost Classifier with random state = 1 and eval_met
xgb_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit XGBoost Classifier on the train data
```

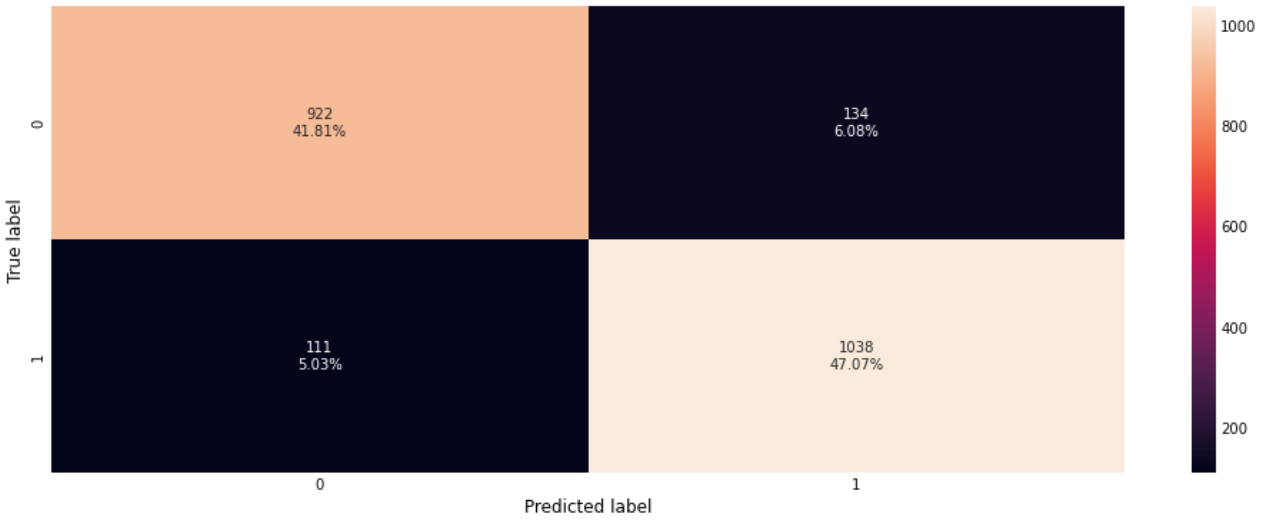
Out[157…

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=1,
              reg_alpha=0, reg_lambda=1, ...)
```

Checking model performance on training set

In [158…

```
confusion_matrix_sklearn(
    xgb_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [159…

```
xgb_classifier_model_train_perf = model_performance_classification_sklearn(
    xgb_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
xgb_classifier_model_train_perf
```

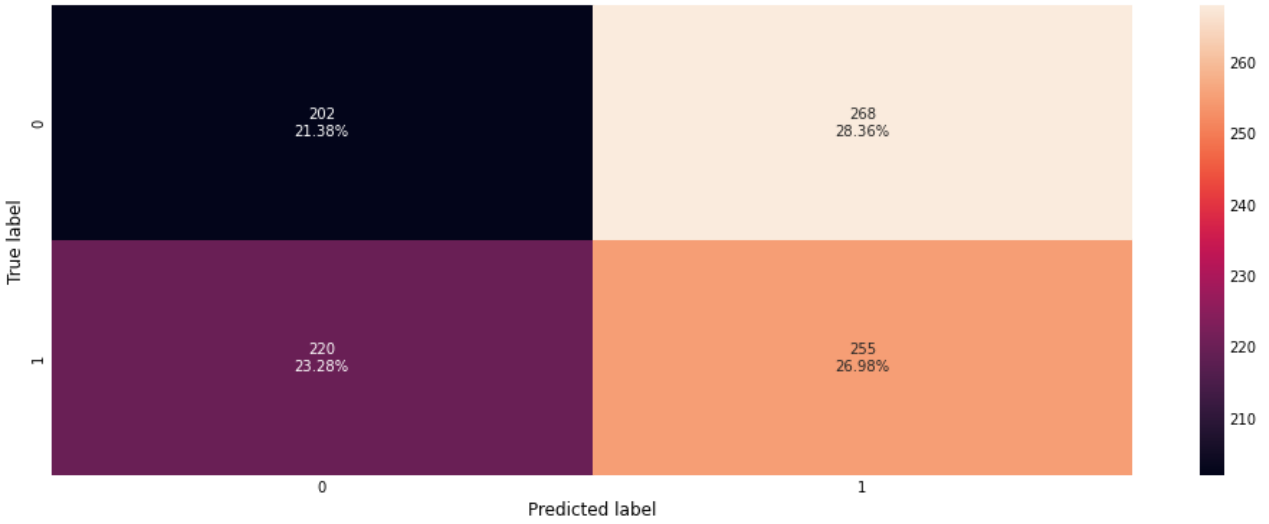
Out[159…

	Accuracy	Recall	Precision	F1
0	0.89	0.90	0.89	0.89

Checking model performance on test set

In [160…

```
confusion_matrix_sklearn(
    xgb_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [161…

```
xgb_classifier_model_test_perf = model_performance_classification_sklearn(
    xgb_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
xgb_classifier_model_test_perf
```

Out[161…

	Accuracy	Recall	Precision	F1
0	0.48	0.54	0.49	0.51

- xgb_classifier model is slightly overfitting
- Peformance is only slightly lower than hyperparameter tuned Gradient Boosting Classifier

Hyperparameter Tuning - XGBoost Classifier

In [162…

```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(random_state=1, eval_metric="logloss")

# Grid of parameters to choose from
parameters = {
    "n_estimators": np.arange(150, 250, 50),
    "scale_pos_weight": [1, 2],
    "subsample": [0.7, 0.9, 1],
    "learning_rate": np.arange(0.1, 0.4, 0.1),
    "gamma": [1, 3, 5],
    "colsample_bytree": [0.7, 0.8, 0.9],
    "colsample_bylevel": [0.8, 0.9, 1],
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    xgb_tuned, parameters, scoring=scorer, cv=5
```

```
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
xgb_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb_tuned.fit(X_train, Y_train)
```

Out[162…

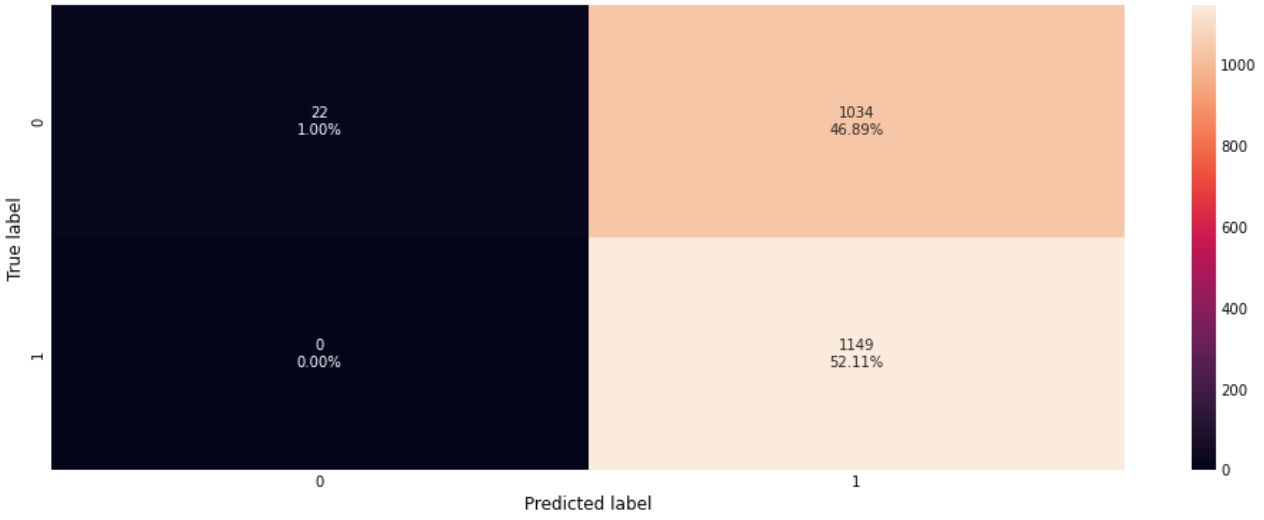
XGBClassifier

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric='logloss', gamma=5, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_bin=256,
              max_cat_to_onehot=4, max_delta_step=0, max_depth=6, max_leaves
=0,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=150, n_jobs=0, num_parallel_tree=1, predictor='au
to',
```

Checking model performance on training set

In [163…

```
confusion_matrix_sklearn(
    xgb_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [164…

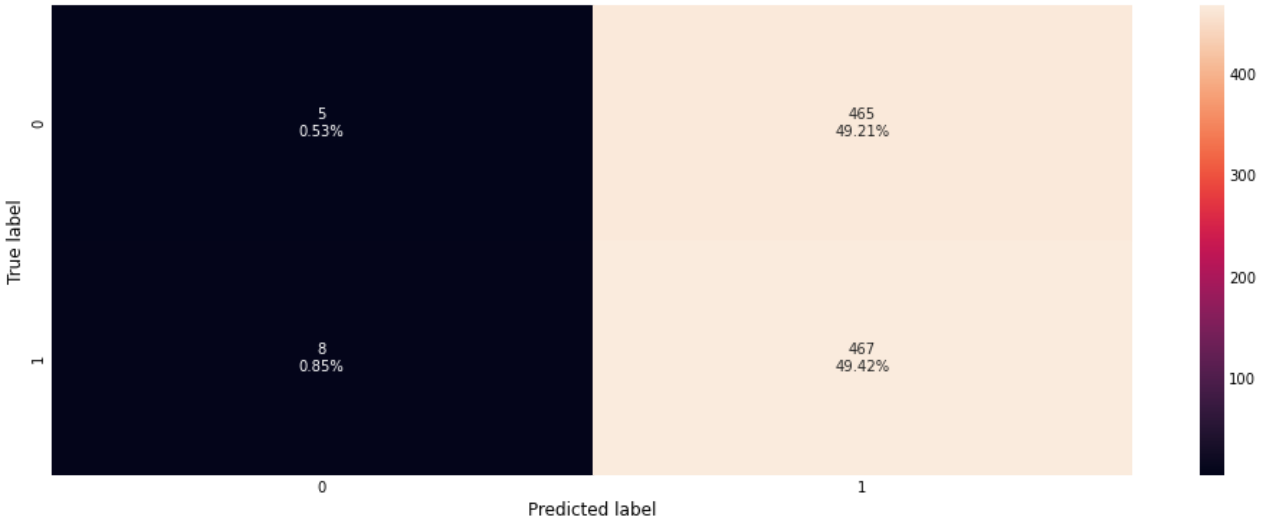
```
xgb_tuned_model_train_perf = model_performance_classification_sklearn(
    xgb_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
xgb_tuned_model_train_perf
```

Out[164…

	Accuracy	Recall	Precision	F1
0	0.53	1.00	0.53	0.69

Checking model performance on test set

In [165…
```
confusion_matrix_sklearn(
    xgb_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [166…
```
xgb_tuned_model_test_perf = model_performance_classification_sklearn(
    xgb_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
xgb_tuned_model_test_perf
```

Out[166…

	Accuracy	Recall	Precision	F1
0	0.50	0.98	0.50	0.66

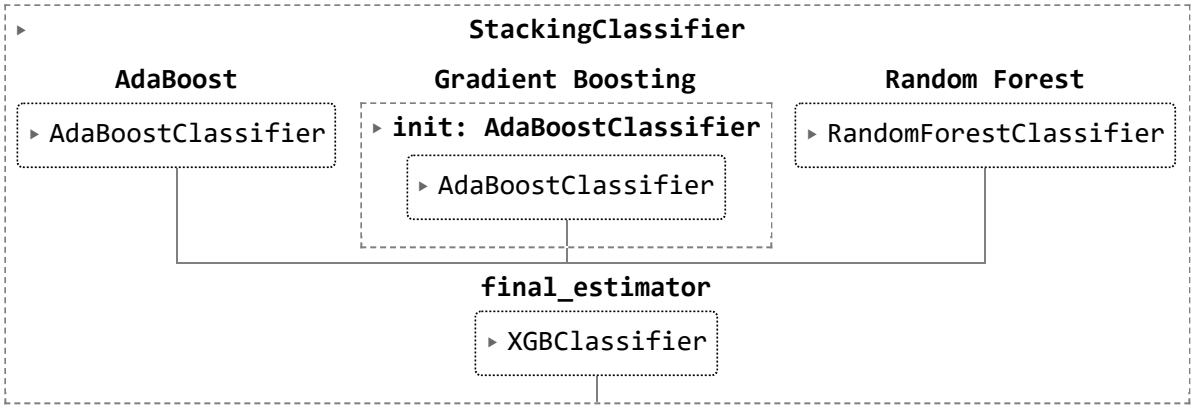
- Hypeparameter tuning of XG model reduces overfitting
- The model performance for hypertuned XGBoost is lower than XGBoost

Stacking Classifier

In [167…
```
estimators = [
    ("AdaBoost", ab_classifier),
    ("Gradient Boosting", gbc_tuned),
    ("Random Forest", rf_tuned),
]

final_estimator = xgb_tuned

stacking_classifier = StackingClassifier(
    estimators=estimators, final_estimator=final_estimator, cv=5
```

```
)  ## Complete the code to define Stacking Classifier

stacking_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit Stacking Classifier on the train data
```

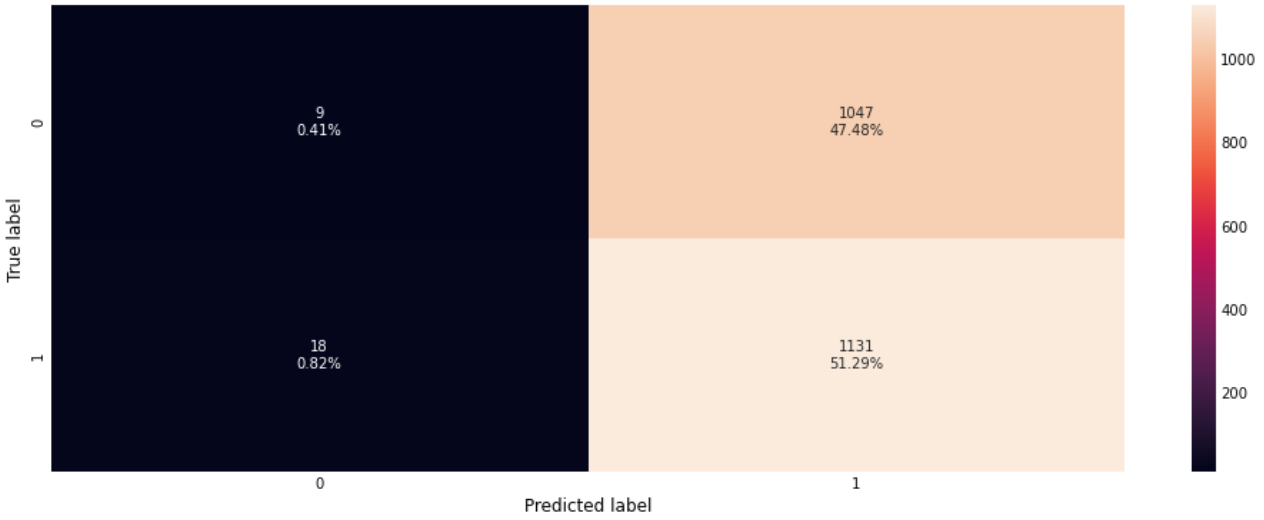
Out[167…



Checking model performance on training set

In [168…

```
confusion_matrix_sklearn(
    stacking_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [169…

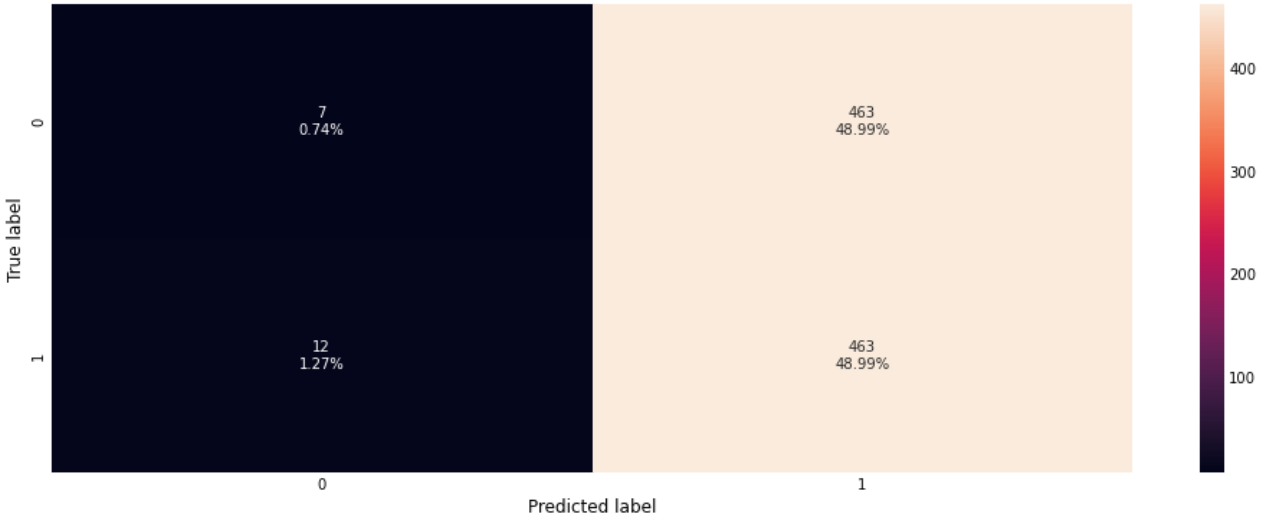
```
stacking_classifier_model_train_perf = model_performance_classification_sklearn(
    stacking_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
stacking_classifier_model_train_perf
```

Out[169…

	Accuracy	Recall	Precision	F1
0	0.52	0.98	0.52	0.68

Checking model performance on test set

In [170…
```
confusion_matrix_sklearn(
    stacking_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [171…
```
stacking_classifier_model_test_perf = model_performance_classification_sklearn(
    stacking_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
stacking_classifier_model_test_perf
```

Out[171…

	Accuracy	Recall	Precision	F1
0	0.50	0.97	0.50	0.66

- Model performance is similar to hypertuned XG Boost

Summary Performance Comparison Measures of Boosting Train vs Test

In [172…
```
boosting_comp_df = pd.concat(
    [
        ab_classifier_model_train_perf.T,
        ab_classifier_model_test_perf.T,
        gb_classifier_model_train_perf.T,
        gb_classifier_model_test_perf.T,
        xgb_classifier_model_train_perf.T,
        xgb_classifier_model_test_perf.T,
    ],
    axis=1,
)

boosting_comp_df.columns = [
    "Adaboost Classifier (train)",
    "Adabosst Classifier (test)",
    "Gradient Boost Classifier (train)",
```

```
    "Gradient Boost Classifier (test)",
    "XGBoost Classifier (train)",
    "XGBoost Classifier (test)",
]
print("Boosting performance comparison:")
boosting_comp_df
```

```
Boosting performance comparison:
```

Out[172…

	Adaboost Classifier (train)	Adabosst Classifier (test)	Gradient Boost Classifier (train)	Gradient Boost Classifier (test)	XGBoost Classifier (train)	XGBoost Classifier (test)
Accuracy	0.58	0.48	0.70	0.50	0.89	0.48
Recall	0.69	0.60	0.87	0.73	0.90	0.54
Precision	0.58	0.49	0.66	0.50	0.89	0.49
F1	0.63	0.54	0.75	0.59	0.89	0.51

Summary Performance Comparison Measures of Boosting Tuned Training vs Tuned Testing

In [173…

```
boosting_tuned_comp_df = pd.concat(
    [
        abc_tuned_model_train_perf.T,
        abc_tuned_model_test_perf.T,
        gbc_tuned_model_train_perf.T,
        gbc_tuned_model_test_perf.T,
        xgb_tuned_model_train_perf.T,
        xgb_tuned_model_test_perf.T,
        stacking_classifier_model_train_perf.T,
        stacking_classifier_model_test_perf.T,
    ],
    axis=1,
)
boosting_tuned_comp_df.columns = [
    "Adaboost Classifier tuned (train)",
    "Adabosst Classifier tuned (test)",
    "Gradient Boost Classifier tuned (train)",
    "Gradient Boost Classifier tuned (test)",
    "XGBoost Classifier tuned (train)",
    "XGBoost Classifier tuned (test)",
    "Stacking Classifier (train)",
    "Stacking Classifier (test)",
]
print("Boosting Tuned performance comparison:")
boosting_tuned_comp_df
```

```
Boosting Tuned performance comparison:
```

Out[173…

	Adaboost Classifier tuned (train)	Adabosst Classifier tuned (test)	Gradient Boost Classifier tuned (train)	Gradient Boost Classifier tuned (test)	XGBoost Classifier tuned (train)	XGBoost Classifier tuned (test)	Stacking Classifier (train)	Stacking Classifier (test)
Accuracy	0.66	0.50	0.72	0.49	0.53	0.50	0.52	0.50
Recall	0.70	0.55	0.87	0.68	1.00	0.98	0.98	0.97
Precision	0.66	0.50	0.68	0.50	0.53	0.50	0.52	0.50
F1	0.68	0.53	0.76	0.57	0.69	0.66	0.68	0.66

Summary Performance Measures of all Bagging and Boosting: All Training Models

In [174…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        decision_tree_perf_train.T,
        dtree_estimator_model_train_perf.T,
        rf_estimator_model_train_perf.T,
        rf_tuned_model_train_perf.T,
        bagging_classifier_model_train_perf.T,
        bagging_estimator_tuned_model_train_perf.T,
        ab_classifier_model_train_perf.T,
        abc_tuned_model_train_perf.T,
        gb_classifier_model_train_perf.T,
        gbc_tuned_model_train_perf.T,
        xgb_classifier_model_train_perf.T,
        xgb_tuned_model_train_perf.T,
        stacking_classifier_model_train_perf.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Tuned",
    "Random Forest ",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Classifier Tuned",
    "Adaboost Classifier",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "XGBoost Classifier",
    "XGBoost Classifier Tuned",
    "Stacking Classifier",
]
print("Training performance comparison:")
models_train_comp_df
```

Training performance comparison:

Out[174…

	Decision Tree	Decision Tree Tuned	Random Forest	Random Forest Tuned	Bagging Classifier	Bagging Classifier Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier
Accuracy	1.00	0.52	1.00	0.63	0.98	0.99	0.58	0.66	0.70
Recall	1.00	1.00	1.00	0.90	0.97	0.99	0.69	0.70	0.87
Precision	1.00	0.52	1.00	0.60	0.99	0.98	0.58	0.66	0.66
F1	1.00	0.69	1.00	0.72	0.98	0.99	0.63	0.68	0.75

Summary Performance Measures of all Bagging and Boosting: All Testing Models

In [175…

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [
        decision_tree_perf_test.T,
        dtree_estimator_model_test_perf.T,
        rf_estimator_model_test_perf.T,
        rf_tuned_model_test_perf.T,
        bagging_classifier_model_test_perf.T,
        bagging_estimator_tuned_model_test_perf.T,
        ab_classifier_model_test_perf.T,
        abc_tuned_model_test_perf.T,
        gb_classifier_model_test_perf.T,
        gbc_tuned_model_test_perf.T,
        xgb_classifier_model_test_perf.T,
        xgb_tuned_model_test_perf.T,
        stacking_classifier_model_test_perf.T,
    ],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Tuned",
    "Random Forest",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Classifier Tuned",
    "Adaboost Classifier",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "XGBoost Classifier",
    "XGBoost Classifier Tuned",
    "Stacking Classifier",
]
print("Testing performance comparison:")
models_test_comp_df
```

Testing performance comparison:

Out[175...

	Decision Tree	Decision Tree Tuned	Random Forest	Random Forest Tuned	Bagging Classifier	Bagging Classifier Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier
Accuracy	0.52	0.50	0.50	0.49	0.48	0.49	0.48	0.50	0.50
Recall	0.55	1.00	0.56	0.78	0.43	0.54	0.60	0.55	0.73
Precision	0.52	0.50	0.51	0.49	0.48	0.49	0.49	0.50	0.50
F1	0.53	0.67	0.53	0.61	0.45	0.52	0.54	0.53	0.59

Observations:

- Accuracy appears to be similar in all Bagging and Boosting models and lower in Decision Tree models.
- Recall seems to be the highest in Tuned Decision Tree Classifier (1.00), followed by Tuned AdaBoost Classifier
- Precision looks similar in all Bagging and Boosting models and slightly lower in Tuned Decision Tree model.
- F1 coefficient appears to be higher in Boosting models than Bagging models.
- Interestingly, there is virtually no overfitting in AdaBoost and Gradient Boost models (both default and tuned). AdaBoost is very slightly overfitting than Gradient Boost models.

Important features of the final selected model

Selected Model - Tunned Random Forest

Using Decision Tree for Prediction Diabetes Outcome

In [176...

```
# Text report showing the rules of a decision tree -
feature_names = list(X_train.columns)
print(tree.export_text(model, feature_names=feature_names, show_weights=True))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7576/4207447750.py in <module>
      1 # Text report showing the rules of a decision tree -
----> 2 feature_names = list(X_train.columns)
      3 print(tree.export_text(model,feature_names=feature_names,show_weights=True))

AttributeError: 'numpy.ndarray' object has no attribute 'columns'
```

General Models Observations

In [ ]: