In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.metrics import classification_report
```

In [4]:

```
df = pd.read_csv('../Machine Learning Project/online_shoppers_intention.csv')
```

In [5]:

```
df
```

Out[5]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRela
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	
...	...	...	...	...	
12325	3	145.0	0	0.0	
12326	0	0.0	0	0.0	
12327	0	0.0	0	0.0	
12328	4	75.0	0	0.0	
12329	0	0.0	0	0.0	

12330 rows × 18 columns

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

In [7]:

```
df.describe()
```

Out[7]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRela
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000(
mean	2.315166	80.818611	0.503569	34.472398	31.731
std	3.321784	176.779107	1.270156	140.749294	44.475
min	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.000000	0.000000	0.000000	0.000000	7.000(
50%	1.000000	7.500000	0.000000	0.000000	18.000(
75%	4.000000	93.256250	0.000000	0.000000	38.000(
max	27.000000	3398.750000	24.000000	2549.375000	705.000(

In [8]:

```
df.isnull().sum() #no missing value
```

Out[8]:

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

In [9]:

```
df['Revenue'] = df['Revenue'].astype(int) #clean data type: bool to int
```

In [10]:

```
df['Weekend'] = df['Weekend'].astype(int) #clean data type: bool to int
```

In [11]:

```
month = {'Feb':2, 'Mar':3,'May':5, 'June':6, 'Jul':7, 'Aug':8, 'Sep':9, 'Oct':10, 'N
df['Month'] = df['Month'].map(month) #clean data type: str to int
```

In [12]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  int64  
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  int64  
 17  Revenue                  12330 non-null  int64  
dtypes: float64(7), int64(10), object(1)
memory usage: 1.7+ MB
```

remove outliners

In [17]:

```
print('1º Quartile: ', df['ProductRelated_Duration'].quantile(q = 0.25))
print('2º Quartile: ', df['ProductRelated_Duration'].quantile(q = 0.50))
print('3º Quartile: ', df['ProductRelated_Duration'].quantile(q = 0.75))
print('4º Quartile: ', df['ProductRelated_Duration'].quantile(q = 1.00))
#Calculate the outliers:
  # Interquartile range, IQR = Q3 - Q1
  # lower 1.5*IQR whisker = Q1 - 1.5 * IQR
  # Upper 1.5*IQR whisker = Q3 + 1.5 * IQR

print('Duration above: ', df['ProductRelated_Duration'].quantile(q = 0.75) +
                      1.5*(df['ProductRelated_Duration'].quantile(q = 0.75) - df['Pr
```

```
1º Quartile:  184.1375
2º Quartile:  598.9369047499999
3º Quartile:  1464.1572135000001
4º Quartile:  63973.52223
Duration above:  3384.1867837500004 are outliers
```

In [18]:

```
print('1º Quartile: ', df['Administrative_Duration'].quantile(q = 0.25))
print('2º Quartile: ', df['Administrative_Duration'].quantile(q = 0.50))
print('3º Quartile: ', df['Administrative_Duration'].quantile(q = 0.75))
print('4º Quartile: ', df['Administrative_Duration'].quantile(q = 1.00))
#Calculate the outliers:
  # Interquartile range, IQR = Q3 - Q1
  # lower 1.5*IQR whisker = Q1 - 1.5 * IQR
  # Upper 1.5*IQR whisker = Q3 + 1.5 * IQR

print('Duration above: ', df['Administrative_Duration'].quantile(q = 0.75) +
                      1.5*(df['Administrative_Duration'].quantile(q = 0.75) - df['Ad
```

```
1º Quartile:  0.0
2º Quartile:  7.5
3º Quartile:  93.25625
4º Quartile:  3398.75
Duration above:  233.14062499999997 are outliers
```

In [19]:

```
print('1º Quartile: ', df['Informational_Duration'].quantile(q = 0.25))
print('2º Quartile: ', df['Informational_Duration'].quantile(q = 0.50))
print('3º Quartile: ', df['Informational_Duration'].quantile(q = 0.75))
print('4º Quartile: ', df['Informational_Duration'].quantile(q = 1.00))
#Calculate the outliers:
  # Interquartile range, IQR = Q3 - Q1
  # lower 1.5*IQR whisker = Q1 - 1.5 * IQR
  # Upper 1.5*IQR whisker = Q3 + 1.5 * IQR

print('Duration above: ', df['Informational_Duration'].quantile(q = 0.75) +
                      1.5*(df['Informational_Duration'].quantile(q = 0.75) - df['Inf
```

```
1º Quartile:  0.0
2º Quartile:  0.0
3º Quartile:  0.0
4º Quartile:  2549.375
Duration above:  0.0 are outliers
```

In [20]:

```
df = df[df.ProductRelated_Duration < 3384.18]
df = df[df.Administrative_Duration < 233.14]
```

In [21]:

```
df.describe()
```

Out[21]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRela
count	10467.000000	10467.000000	10467.000000	10467.000000	10467.000(
mean	1.604662	36.259147	0.337346	20.959281	21.961
std	2.442930	56.192433	0.976302	102.611044	22.490
min	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.000000	0.000000	0.000000	0.000000	6.000(
50%	0.000000	0.000000	0.000000	0.000000	15.000(
75%	3.000000	58.033333	0.000000	0.000000	30.000(
max	19.000000	233.083333	16.000000	2252.033333	223.000(

In [22]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10467 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           10467 non-null  int64  
 1   Administrative_Duration  10467 non-null  float64
 2   Informational            10467 non-null  int64  
 3   Informational_Duration   10467 non-null  float64
 4   ProductRelated           10467 non-null  int64  
 5   ProductRelated_Duration  10467 non-null  float64
 6   BounceRates              10467 non-null  float64
 7   ExitRates                10467 non-null  float64
 8   PageValues               10467 non-null  float64
 9   SpecialDay               10467 non-null  float64
 10  Month                    10467 non-null  int64  
 11  OperatingSystems         10467 non-null  int64  
 12  Browser                  10467 non-null  int64  
 13  Region                   10467 non-null  int64  
 14  TrafficType              10467 non-null  int64  
 15  VisitorType              10467 non-null  object 
 16  Weekend                  10467 non-null  int64  
 17  Revenue                  10467 non-null  int64  
dtypes: float64(7), int64(10), object(1)
memory usage: 1.5+ MB
```

encoding

In [23]:

```
df_category = df[['Month', 'OperatingSystems','Browser','Region','TrafficType']]
```

In [24]:

```
df_category
```

Out[24]:

	Month	OperatingSystems	Browser	Region	TrafficType
0	2	1	1	1	1
1	2	2	2	1	2
2	2	4	1	9	3
3	2	3	2	2	4
4	2	3	3	1	4
...	...	...	...	...	...
12325	12	4	6	1	1
12326	11	3	2	1	8
12327	11	3	2	1	13
12328	11	2	2	3	11
12329	11	3	2	1	2

10467 rows × 5 columns

In [25]:

```
df_category = df_category.astype(str)
```

In [26]:

```
df_category.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10467 entries, 0 to 12329
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Month             10467 non-null  object
 1   OperatingSystems  10467 non-null  object
 2   Browser           10467 non-null  object
 3   Region            10467 non-null  object
 4   TrafficType       10467 non-null  object
dtypes: object(5)
memory usage: 490.6+ KB
```

In [27]:

```
df_visitor = df[['VisitorType']]
```

In [28]:

```
df_category = pd.concat([df_category,df_visitor],axis=1)
```

In [29]:

```
df_category
```

Out[29]:

	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType
0	2	1	1	1	1	Returning_Visitor
1	2	2	2	1	2	Returning_Visitor
2	2	4	1	9	3	Returning_Visitor
3	2	3	2	2	4	Returning_Visitor
4	2	3	3	1	4	Returning_Visitor
...	...	...	...	...	...	...
12325	12	4	6	1	1	Returning_Visitor
12326	11	3	2	1	8	Returning_Visitor
12327	11	3	2	1	13	Returning_Visitor
12328	11	2	2	3	11	Returning_Visitor
12329	11	3	2	1	2	New_Visitor

10467 rows × 6 columns

In [30]:

```
dummies = pd.get_dummies(df_category, drop_first=True)
```

In [31]:

```
df.drop(columns=['Month', 'OperatingSystems','Browser','Region','TrafficType','Visit
```

In [32]:

```
df
```

Out[32]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRela
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	
...	...	...	...	...	
12325	3	145.0	0	0.0	
12326	0	0.0	0	0.0	
12327	0	0.0	0	0.0	
12328	4	75.0	0	0.0	
12329	0	0.0	0	0.0	

10467 rows × 12 columns

In [33]:

```
df = pd.concat([df,dummies],axis=1)
```

In [34]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10467 entries, 0 to 12329
Data columns (total 68 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Administrative                 10467 non-null  int64  
 1   Administrative_Duration        10467 non-null  float64
 2   Informational                  10467 non-null  int64  
 3   Informational_Duration         10467 non-null  float64
 4   ProductRelated                 10467 non-null  int64  
 5   ProductRelated_Duration        10467 non-null  float64
 6   BounceRates                    10467 non-null  float64
 7   ExitRates                      10467 non-null  float64
 8   PageValues                     10467 non-null  float64
 9   SpecialDay                     10467 non-null  float64
 10  Weekend                        10467 non-null  int64  
 11  Revenue                        10467 non-null  int64  
 12  Month_11                       10467 non-null  uint8  
 13  Month_12                       10467 non-null  uint8  
 14  Month_2                        10467 non-null  uint8  
 15  Month_3                        10467 non-null  uint8  
 16  Month_5                        10467 non-null  uint8  
 17  Month_6                        10467 non-null  uint8  
 18  Month_7                        10467 non-null  uint8  
 19  Month_8                        10467 non-null  uint8  
 20  Month_9                        10467 non-null  uint8  
 21  OperatingSystems_2             10467 non-null  uint8  
 22  OperatingSystems_3             10467 non-null  uint8  
 23  OperatingSystems_4             10467 non-null  uint8  
 24  OperatingSystems_5             10467 non-null  uint8  
 25  OperatingSystems_6             10467 non-null  uint8  
 26  OperatingSystems_7             10467 non-null  uint8  
 27  OperatingSystems_8             10467 non-null  uint8  
 28  Browser_10                     10467 non-null  uint8  
 29  Browser_11                     10467 non-null  uint8  
 30  Browser_12                     10467 non-null  uint8  
 31  Browser_13                     10467 non-null  uint8  
 32  Browser_2                      10467 non-null  uint8  
 33  Browser_3                      10467 non-null  uint8  
 34  Browser_4                      10467 non-null  uint8  
 35  Browser_5                      10467 non-null  uint8  
 36  Browser_6                      10467 non-null  uint8  
 37  Browser_7                      10467 non-null  uint8  
 38  Browser_8                      10467 non-null  uint8  
 39  Region_2                       10467 non-null  uint8  
 40  Region_3                       10467 non-null  uint8  
 41  Region_4                       10467 non-null  uint8  
 42  Region_5                       10467 non-null  uint8  
 43  Region_6                       10467 non-null  uint8  
 44  Region_7                       10467 non-null  uint8  
 45  Region_8                       10467 non-null  uint8  
 46  Region_9                       10467 non-null  uint8  
 47  TrafficType_10                 10467 non-null  uint8  
 48  TrafficType_11                 10467 non-null  uint8  
 49  TrafficType_12                 10467 non-null  uint8  
 50  TrafficType_13                 10467 non-null  uint8  
 51  TrafficType_14                 10467 non-null  uint8  
```

```
 52  TrafficType_15                 10467 non-null  uint8
 53  TrafficType_16                 10467 non-null  uint8
 54  TrafficType_17                 10467 non-null  uint8
 55  TrafficType_18                 10467 non-null  uint8
 56  TrafficType_19                 10467 non-null  uint8
 57  TrafficType_2                  10467 non-null  uint8
 58  TrafficType_20                 10467 non-null  uint8
 59  TrafficType_3                  10467 non-null  uint8
 60  TrafficType_4                  10467 non-null  uint8
 61  TrafficType_5                  10467 non-null  uint8
 62  TrafficType_6                  10467 non-null  uint8
 63  TrafficType_7                  10467 non-null  uint8
 64  TrafficType_8                  10467 non-null  uint8
 65  TrafficType_9                  10467 non-null  uint8
 66  VisitorType_Other              10467 non-null  uint8
 67  VisitorType_Returning_Visitor  10467 non-null  uint8
dtypes: float64(7), int64(5), uint8(56)
memory usage: 1.6 MB
```

train_test_split

In [37]:

```
X = df.drop(columns='Revenue', axis=1)
y = df['Revenue']
```

In [38]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,train_size=0.8,random_state
print("Input Training:",X_train.shape)
print("Input Test:",X_test.shape)
print("Output Training:",y_train.shape)
print("Output Test:",y_test.shape)
```

```
Input Training: (8373, 67)
Input Test: (2094, 67)
Output Training: (8373,)
Output Test: (2094,)
```

In [39]:

```
def evaluate_model(model, x_test, y_test):
    from sklearn import metrics

    # Predict Test Data
    y_pred = model.predict(x_test)

    # Calculate accuracy, precision, recall, f1-score, and kappa score
    acc = metrics.accuracy_score(y_test, y_pred)
    prec = metrics.precision_score(y_test, y_pred)
    rec = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)
    kappa = metrics.cohen_kappa_score(y_test, y_pred)

    # Calculate area under curve (AUC)
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)

    # Display confussion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)

    return {'acc': acc, 'prec': prec, 'rec': rec, 'f1': f1, 'kappa': kappa,
            'fpr': fpr, 'tpr': tpr, 'auc': auc, 'cm': cm}
```

In [40]:

```
from sklearn import tree

# Building Decision Tree model
dtc = tree.DecisionTreeClassifier(random_state=0)
dtc.fit(X_train, y_train)
```

Out[40]:

```
DecisionTreeClassifier(random_state=0)
```

In [41]:

```
# Evaluate Model
dtc_eval = evaluate_model(dtc, X_test, y_test)

# Print result
print('Accuracy:', dtc_eval['acc'])
print('Precision:', dtc_eval['prec'])
print('Recall:', dtc_eval['rec'])
print('F1 Score:', dtc_eval['f1'])
print('Cohens Kappa Score:', dtc_eval['kappa'])
print('Area Under Curve:', dtc_eval['auc'])
print('Confusion Matrix:\n', dtc_eval['cm'])
```

```
Accuracy: 0.8801337153772684
Precision: 0.5481481481481482
Recall: 0.5342960288808665
F1 Score: 0.5411334552102376
Cohens Kappa Score: 0.4722093352533243
Area Under Curve: 0.7335761927563386
Confusion Matrix:
 [[1695  122]
 [ 129  148]]
```

In [42]:

```
from sklearn.ensemble import RandomForestClassifier

# Building Random Forest model
rf = RandomForestClassifier(random_state=0)
rf.fit(X_train, y_train)
```

Out[42]:

```
RandomForestClassifier(random_state=0)
```

In [43]:

```
# Evaluate Model
rf_eval = evaluate_model(rf, X_test, y_test)

# Print result
print('Accuracy:', rf_eval['acc'])
print('Precision:', rf_eval['prec'])
print('Recall:', rf_eval['rec'])
print('F1 Score:', rf_eval['f1'])
print('Cohens Kappa Score:', rf_eval['kappa'])
print('Area Under Curve:', rf_eval['auc'])
print('Confusion Matrix:\n', rf_eval['cm'])
```

```
Accuracy: 0.9259789875835721
Precision: 0.8112244897959183
Recall: 0.5740072202166066
F1 Score: 0.6723044397463003
Cohens Kappa Score: 0.6319558941259449
Area Under Curve: 0.9342938433447444
Confusion Matrix:
 [[1780   37]
 [ 118  159]]
```

In [44]:

```
from sklearn.naive_bayes import GaussianNB

# Building Naive Bayes model
nb = GaussianNB()
nb.fit(X_train, y_train)
```

Out[44]:

```
GaussianNB()
```

In [45]:

```
# Evaluate Model
nb_eval = evaluate_model(nb, X_test, y_test)

# Print result
print('Accuracy:', nb_eval['acc'])
print('Precision:', nb_eval['prec'])
print('Recall:', nb_eval['rec'])
print('F1 Score:', nb_eval['f1'])
print('Cohens Kappa Score:', nb_eval['kappa'])
print('Area Under Curve:', nb_eval['auc'])
print('Confusion Matrix:\n', nb_eval['cm'])
```

```
Accuracy: 0.6523400191021967
Precision: 0.25568797399783316
Recall: 0.851985559566787
F1 Score: 0.39333333333333337
Cohens Kappa Score: 0.23833925068624984
Area Under Curve: 0.8269085194184883
Confusion Matrix:
 [[1130  687]
 [  41  236]]
```

In [46]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
```

```
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
n in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic
-regression (https://scikit-learn.org/stable/modules/linear_model.html
#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[46]:

```
LogisticRegression(random_state=0)
```

In [47]:

```
# Evaluate Model
lr_eval = evaluate_model(lr, X_test, y_test)

# Print result
print('Accuracy:', lr_eval['acc'])
print('Precision:', lr_eval['prec'])
print('Recall:', lr_eval['rec'])
print('F1 Score:', lr_eval['f1'])
print('Cohens Kappa Score:', lr_eval['kappa'])
print('Area Under Curve:', lr_eval['auc'])
print('Confusion Matrix:\n', lr_eval['cm'])
```

```
Accuracy: 0.9083094555873925
Precision: 0.7814569536423841
Recall: 0.4259927797833935
F1 Score: 0.5514018691588785
Cohens Kappa Score: 0.5052191912653308
Area Under Curve: 0.8793603929196576
Confusion Matrix:
 [[1784   33]
 [ 159  118]]
```

In [48]:

```
import xgboost
xgb = xgboost.XGBClassifier()
xgb.fit(X_train, y_train)
```

```
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/xgboost/sklear
n.py:1224: UserWarning: The use of label encoder in XGBClassifier is d
eprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when con
structing XGBClassifier object; and 2) Encode your labels (y) as integ
ers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[14:07:41] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1
643227205751/work/src/learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' wa
s changed from 'error' to 'logloss'. Explicitly set eval_metric if yo
u'd like to restore the old behavior.
```

Out[48]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categoric
al=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missi
ng=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample
=1,
              tree_method='exact', validate_parameters=1, verbosity=No
ne)
```

In [49]:

```
# Evaluate Model
xgb_eval = evaluate_model(xgb, X_test, y_test)

# Print result
print('Accuracy:', xgb_eval['acc'])
print('Precision:', xgb_eval['prec'])
print('Recall:', xgb_eval['rec'])
print('F1 Score:', xgb_eval['f1'])
print('Cohens Kappa Score:', xgb_eval['kappa'])
print('Area Under Curve:', xgb_eval['auc'])
print('Confusion Matrix:\n', xgb_eval['cm'])
```

```
Accuracy: 0.9226361031518625
Precision: 0.7533039647577092
Recall: 0.6173285198555957
F1 Score: 0.6785714285714286
Cohens Kappa Score: 0.6350888214298777
Area Under Curve: 0.9332278977725413
Confusion Matrix:
 [[1761   56]
 [ 106  171]]
```

In [50]:

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100, random_state=0)
ada.fit(X_train, y_train)
```

Out[50]:

```
AdaBoostClassifier(n_estimators=100, random_state=0)
```

In [51]:

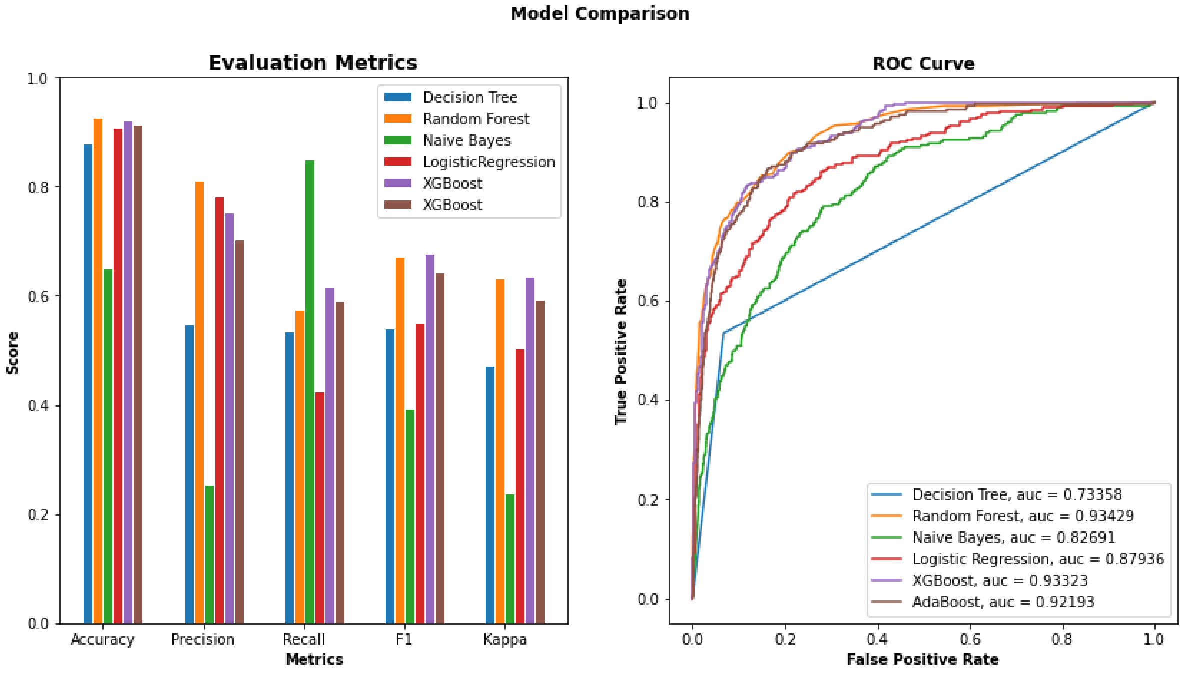
```
# Evaluate Model
ada_eval = evaluate_model(ada, X_test, y_test)

# Print result
print('Accuracy:', ada_eval['acc'])
print('Precision:', ada_eval['prec'])
print('Recall:', ada_eval['rec'])
print('F1 Score:', ada_eval['f1'])
print('Cohens Kappa Score:', ada_eval['kappa'])
print('Area Under Curve:', ada_eval['auc'])
print('Confusion Matrix:\n', ada_eval['cm'])
```

```
Accuracy: 0.9130850047755492
Precision: 0.703862660944206
Recall: 0.592057761732852
F1 Score: 0.6431372549019608
Cohens Kappa Score: 0.5940727990814372
Area Under Curve: 0.9219256957455559
Confusion Matrix:
 [[1748   69]
 [ 113  164]]
```

In [52]:

```
# Intitialize figure with two plots
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Model Comparison', fontsize=12, fontweight='bold')
fig.set_figheight(7)
fig.set_figwidth(14)
fig.set_facecolor('white')

# First plot
## set bar size
barWidth = 0.1
dtc_score = [dtc_eval['acc'], dtc_eval['prec'], dtc_eval['rec'], dtc_eval['f1'], dtc
rf_score = [rf_eval['acc'], rf_eval['prec'], rf_eval['rec'], rf_eval['f1'], rf_eval[
nb_score = [nb_eval['acc'], nb_eval['prec'], nb_eval['rec'], nb_eval['f1'], nb_eval[
lr_score = [lr_eval['acc'], lr_eval['prec'], lr_eval['rec'], lr_eval['f1'], lr_eval[
xgb_score = [xgb_eval['acc'], xgb_eval['prec'], xgb_eval['rec'], xgb_eval['f1'], xgb
ada_score = [ada_eval['acc'], ada_eval['prec'], ada_eval['rec'], ada_eval['f1'], ada

## Set position of bar on X axis
r1 = np.arange(len(dtc_score))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]
r5 = [x + barWidth for x in r4]
r6 = [x + barWidth for x in r5]

## Make the plot
ax1.bar(r1, dtc_score, width=barWidth, edgecolor='white', label='Decision Tree')
ax1.bar(r2, rf_score, width=barWidth, edgecolor='white', label='Random Forest')
ax1.bar(r3, nb_score, width=barWidth, edgecolor='white', label='Naive Bayes')
ax1.bar(r4, lr_score, width=barWidth, edgecolor='white', label='LogisticRegression')
ax1.bar(r5, xgb_score, width=barWidth, edgecolor='white', label='XGBoost')
ax1.bar(r6, ada_score, width=barWidth, edgecolor='white', label='XGBoost')

## Configure x and y axis
ax1.set_xlabel('Metrics', fontweight='bold')
labels = ['Accuracy', 'Precision', 'Recall', 'F1', 'Kappa']
ax1.set_xticks([r + (barWidth * 1.5) for r in range(len(dtc_score))], )
ax1.set_xticklabels(labels)
ax1.set_ylabel('Score', fontweight='bold')
ax1.set_ylim(0, 1)

## Create legend & title
ax1.set_title('Evaluation Metrics', fontsize=14, fontweight='bold')
ax1.legend()

# Second plot
## Comparing ROC Curve
ax2.plot(dtc_eval['fpr'], dtc_eval['tpr'], label='Decision Tree, auc = {:0.5f}'.form
ax2.plot(rf_eval['fpr'], rf_eval['tpr'], label='Random Forest, auc = {:0.5f}'.format
ax2.plot(nb_eval['fpr'], nb_eval['tpr'], label='Naive Bayes, auc = {:0.5f}'.format(r
ax2.plot(lr_eval['fpr'], lr_eval['tpr'], label='Logistic Regression, auc = {:0.5f}'.
ax2.plot(xgb_eval['fpr'], xgb_eval['tpr'], label='XGBoost, auc = {:0.5f}'.format(xgb
ax2.plot(ada_eval['fpr'], ada_eval['tpr'], label='AdaBoost, auc = {:0.5f}'.format(ad

## Configure x and y axis
ax2.set_xlabel('False Positive Rate', fontweight='bold')
ax2.set_ylabel('True Positive Rate', fontweight='bold')

## Create legend & title
```

```
ax2.set_title('ROC Curve', fontsize=12, fontweight='bold')
ax2.legend(loc=4)

plt.show()
```



In [53]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score
classifiers = {
    "Decision Tree": tree.DecisionTreeClassifier(random_state=0),
    "Random Forest": RandomForestClassifier(random_state=0),
    "Naive Bayes": GaussianNB(),
    "Logistic Regression": LogisticRegression(),
    "XGBoost": xgboost.XGBClassifier(),
    "AdaBoost": AdaBoostClassifier(n_estimators=100, random_state=0),
    }

f, axes = plt.subplots(1, 6, figsize=(20, 5), sharey='row')

for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.fit(X_train, y_train).predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    print(key, " \n Accuracy:",accuracy_score(y_test,y_pred),"\n F-score",f1_score(y
    disp = ConfusionMatrixDisplay(cf_matrix,
                                  display_labels=["Not Purchased","Purchased"])
    disp.plot(ax=axes[i], xticks_rotation=45)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()
    disp.ax_.set_xlabel('')
    if i!=0:
        disp.ax_.set_ylabel('')

f.text(0.4, 0.1, 'Predicted label', ha='left')
plt.subplots_adjust(wspace=0.40, hspace=0.1)


f.colorbar(disp.im_, ax=axes)
plt.show()
```

```
Decision Tree
 Accuracy: 0.8801337153772684
 F-score 0.5411334552102376
Random Forest
 Accuracy: 0.9259789875835721
 F-score 0.6723044397463003
Naive Bayes
 Accuracy: 0.6523400191021967
 F-score 0.39333333333333337
Logistic Regression
 Accuracy: 0.9083094555873925
 F-score 0.5514018691588785
[14:07:44] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1
643227205751/work/src/learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' wa
s changed from 'error' to 'logloss'. Explicitly set eval_metric if yo
u'd like to restore the old behavior.
```

```
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converg
e (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as sho
wn in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
```

```
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver option
s:
    https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression (https://scikit-learn.org/stable/modules/linear_model.ht
ml#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/xgboost/sklea
rn.py:1224: UserWarning: The use of label encoder in XGBClassifier is
deprecated and will be removed in a future release. To remove this wa
rning, do the following: 1) Pass option use_label_encoder=False when
 constructing XGBClassifier object; and 2) Encode your labels (y) as
 integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
XGBoost
 Accuracy: 0.9226361031518625
 F-score 0.6785714285714286
AdaBoost
 Accuracy: 0.9130850047755492
 F-score 0.6431372549019608
```

In [54]:

```
from imblearn.combine import SMOTEENN
```

In [55]:

```
sm = SMOTEENN()
X_resampled1, y_resampled1 = sm.fit_resample(X,y)
```

In [56]:

```
X_train, X_test, y_train, y_test=train_test_split(X_resampled1, y_resampled1,train_s
```

In [57]:

```
dtc = tree.DecisionTreeClassifier(random_state=0)
dtc.fit(X_train, y_train)
```

Out[57]:

```
DecisionTreeClassifier(random_state=0)
```

In [58]:

```
# Evaluate Model
dtc_eval = evaluate_model(dtc, X_test, y_test)

# Print result
print('Accuracy:', dtc_eval['acc'])
print('Precision:', dtc_eval['prec'])
print('Recall:', dtc_eval['rec'])
print('F1 Score:', dtc_eval['f1'])
print('Cohens Kappa Score:', dtc_eval['kappa'])
print('Area Under Curve:', dtc_eval['auc'])
print('Confusion Matrix:\n', dtc_eval['cm'])
```

```
Accuracy: 0.9500683994528044
Precision: 0.9523809523809523
Recall: 0.9559748427672956
F1 Score: 0.9541745134965474
Cohens Kappa Score: 0.8993289603507442
Area Under Curve: 0.9495016642622085
Confusion Matrix:
 [[1258   76]
 [  70 1520]]
```

In [59]:

```
rf = RandomForestClassifier(random_state=0)
rf.fit(X_train, y_train)
```

Out[59]:

```
RandomForestClassifier(random_state=0)
```

In [60]:

```
# Evaluate Model
rf_eval = evaluate_model(rf, X_test, y_test)

# Print result
print('Accuracy:', rf_eval['acc'])
print('Precision:', rf_eval['prec'])
print('Recall:', rf_eval['rec'])
print('F1 Score:', rf_eval['f1'])
print('Cohens Kappa Score:', rf_eval['kappa'])
print('Area Under Curve:', rf_eval['auc'])
print('Confusion Matrix:\n', rf_eval['cm'])
```

```
Accuracy: 0.9678522571819426
Precision: 0.9645962732919254
Recall: 0.9767295597484277
F1 Score: 0.9706250000000001
Cohens Kappa Score: 0.9351295728109511
Area Under Curve: 0.9956410945470661
Confusion Matrix:
 [[1277   57]
 [  37 1553]]
```

In [61]:

```
nb = GaussianNB()
nb.fit(X_train, y_train)
```

Out[61]:

```
GaussianNB()
```

In [62]:

```
# Evaluate Model
nb_eval = evaluate_model(nb, X_test, y_test)

# Print result
print('Accuracy:', nb_eval['acc'])
print('Precision:', nb_eval['prec'])
print('Recall:', nb_eval['rec'])
print('F1 Score:', nb_eval['f1'])
print('Cohens Kappa Score:', nb_eval['kappa'])
print('Area Under Curve:', nb_eval['auc'])
print('Confusion Matrix:\n', nb_eval['cm'])
```

```
Accuracy: 0.8406292749658003
Precision: 0.7983014861995754
Recall: 0.9459119496855346
F1 Score: 0.865860679332182
Cohens Kappa Score: 0.672994699120872
Area Under Curve: 0.9492235014568189
Confusion Matrix:
 [[ 954  380]
 [  86 1504]]
```

In [63]:

```
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
```

```
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
n in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic
-regression (https://scikit-learn.org/stable/modules/linear_model.html
#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[63]:

```
LogisticRegression(random_state=0)
```

In [64]:

```
# Evaluate Model
lr_eval = evaluate_model(lr, X_test, y_test)

# Print result
print('Accuracy:', lr_eval['acc'])
print('Precision:', lr_eval['prec'])
print('Recall:', lr_eval['rec'])
print('F1 Score:', lr_eval['f1'])
print('Cohens Kappa Score:', lr_eval['kappa'])
print('Area Under Curve:', lr_eval['auc'])
print('Confusion Matrix:\n', lr_eval['cm'])
```

```
Accuracy: 0.9261285909712722
Precision: 0.966078697421981
Recall: 0.8955974842767296
F1 Score: 0.9295039164490861
Cohens Kappa Score: 0.8521509353399117
Area Under Curve: 0.982524303885793
Confusion Matrix:
 [[1284   50]
 [ 166 1424]]
```

In [65]:

```
xgb = xgboost.XGBClassifier()
xgb.fit(X_train, y_train)
```

```
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/xgboost/sklear
n.py:1224: UserWarning: The use of label encoder in XGBClassifier is d
eprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when con
structing XGBClassifier object; and 2) Encode your labels (y) as integ
ers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[14:07:53] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1
643227205751/work/src/learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' wa
s changed from 'error' to 'logloss'. Explicitly set eval_metric if yo
u'd like to restore the old behavior.
```

Out[65]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categoric
al=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missi
ng=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample
=1,
              tree_method='exact', validate_parameters=1, verbosity=No
ne)
```

In [66]:

```
# Evaluate Model
xgb_eval = evaluate_model(xgb, X_test, y_test)

# Print result
print('Accuracy:', xgb_eval['acc'])
print('Precision:', xgb_eval['prec'])
print('Recall:', xgb_eval['rec'])
print('F1 Score:', xgb_eval['f1'])
print('Cohens Kappa Score:', xgb_eval['kappa'])
print('Area Under Curve:', xgb_eval['auc'])
print('Confusion Matrix:\n', xgb_eval['cm'])
```

```
Accuracy: 0.9702462380300958
Precision: 0.9693941286695815
Recall: 0.9761006289308176
F1 Score: 0.9727358194923221
Cohens Kappa Score: 0.9399929799797325
Area Under Curve: 0.9956144569224822
Confusion Matrix:
 [[1285   49]
 [  38 1552]]
```

In [67]:

```
ada = AdaBoostClassifier(n_estimators=100, random_state=0)
ada.fit(X_train, y_train)
```

Out[67]:

```
AdaBoostClassifier(n_estimators=100, random_state=0)
```

In [68]:

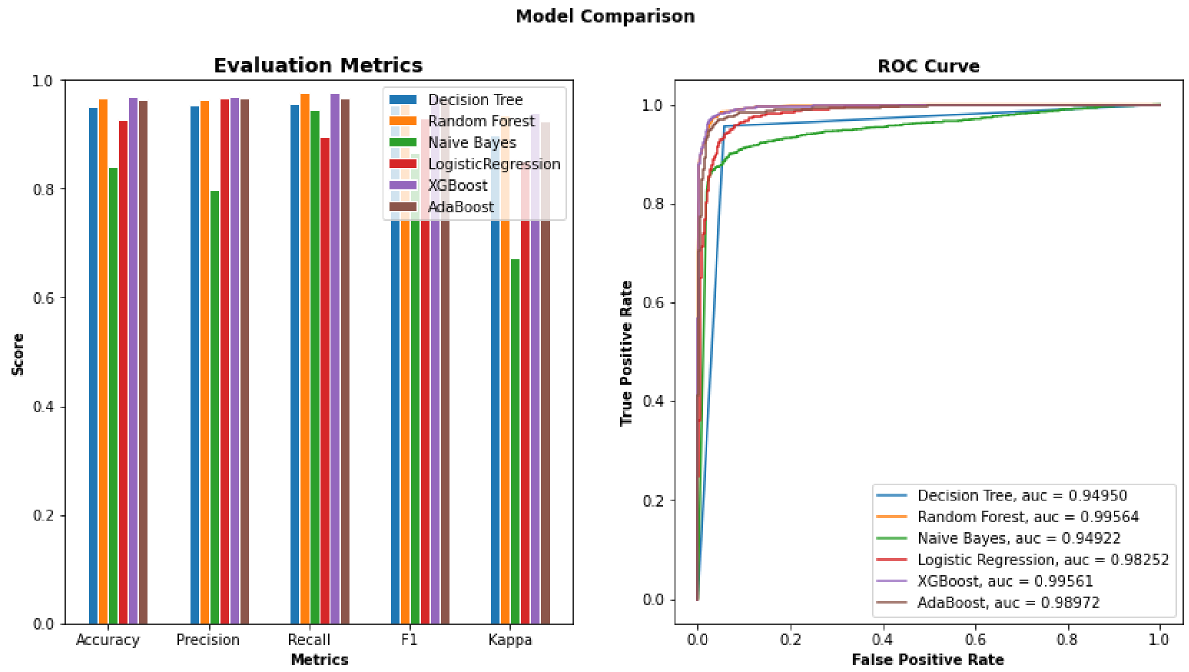
```
# Evaluate Model
ada_eval = evaluate_model(ada, X_test, y_test)

# Print result
print('Accuracy:', ada_eval['acc'])
print('Precision:', ada_eval['prec'])
print('Recall:', ada_eval['rec'])
print('F1 Score:', ada_eval['f1'])
print('Cohens Kappa Score:', ada_eval['kappa'])
print('Area Under Curve:', ada_eval['auc'])
print('Confusion Matrix:\n', ada_eval['cm'])
```

```
Accuracy: 0.9630642954856361
Precision: 0.966624685138539
Recall: 0.9654088050314465
F1 Score: 0.9660163624921335
Cohens Kappa Score: 0.9255669593780464
Area Under Curve: 0.9897207056848935
Confusion Matrix:
 [[1281   53]
 [  55 1535]]
```

In [69]:

```
# Intitialize figure with two plots
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Model Comparison', fontsize=12, fontweight='bold')
fig.set_figheight(7)
fig.set_figwidth(14)
fig.set_facecolor('white')

# First plot
## set bar size
barWidth = 0.1
dtc_score = [dtc_eval['acc'], dtc_eval['prec'], dtc_eval['rec'], dtc_eval['f1'], dtc
rf_score = [rf_eval['acc'], rf_eval['prec'], rf_eval['rec'], rf_eval['f1'], rf_eval[
nb_score = [nb_eval['acc'], nb_eval['prec'], nb_eval['rec'], nb_eval['f1'], nb_eval[
lr_score = [lr_eval['acc'], lr_eval['prec'], lr_eval['rec'], lr_eval['f1'], lr_eval[
xgb_score = [xgb_eval['acc'], xgb_eval['prec'], xgb_eval['rec'], xgb_eval['f1'], xgb
ada_score = [ada_eval['acc'], ada_eval['prec'], ada_eval['rec'], ada_eval['f1'], ada

## Set position of bar on X axis
r1 = np.arange(len(dtc_score))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]
r5 = [x + barWidth for x in r4]
r6 = [x + barWidth for x in r5]

## Make the plot
ax1.bar(r1, dtc_score, width=barWidth, edgecolor='white', label='Decision Tree')
ax1.bar(r2, rf_score, width=barWidth, edgecolor='white', label='Random Forest')
ax1.bar(r3, nb_score, width=barWidth, edgecolor='white', label='Naive Bayes')
ax1.bar(r4, lr_score, width=barWidth, edgecolor='white', label='LogisticRegression')
ax1.bar(r5, xgb_score, width=barWidth, edgecolor='white', label='XGBoost')
ax1.bar(r6, ada_score, width=barWidth, edgecolor='white', label='AdaBoost')

## Configure x and y axis
ax1.set_xlabel('Metrics', fontweight='bold')
labels = ['Accuracy', 'Precision', 'Recall', 'F1', 'Kappa']
ax1.set_xticks([r + (barWidth * 1.5) for r in range(len(dtc_score))], )
ax1.set_xticklabels(labels)
ax1.set_ylabel('Score', fontweight='bold')
ax1.set_ylim(0, 1)

## Create legend & title
ax1.set_title('Evaluation Metrics', fontsize=14, fontweight='bold')
ax1.legend()

# Second plot
## Comparing ROC Curve
ax2.plot(dtc_eval['fpr'], dtc_eval['tpr'], label='Decision Tree, auc = {:0.5f}'.form
ax2.plot(rf_eval['fpr'], rf_eval['tpr'], label='Random Forest, auc = {:0.5f}'.format
ax2.plot(nb_eval['fpr'], nb_eval['tpr'], label='Naive Bayes, auc = {:0.5f}'.format(r
ax2.plot(lr_eval['fpr'], lr_eval['tpr'], label='Logistic Regression, auc = {:0.5f}'.
ax2.plot(xgb_eval['fpr'], xgb_eval['tpr'], label='XGBoost, auc = {:0.5f}'.format(xgb
ax2.plot(ada_eval['fpr'], ada_eval['tpr'], label='AdaBoost, auc = {:0.5f}'.format(ac

## Configure x and y axis
ax2.set_xlabel('False Positive Rate', fontweight='bold')
ax2.set_ylabel('True Positive Rate', fontweight='bold')

## Create legend & title
```

```
ax2.set_title('ROC Curve', fontsize=12, fontweight='bold')
ax2.legend(loc=4)

plt.show()
```



In [70]:

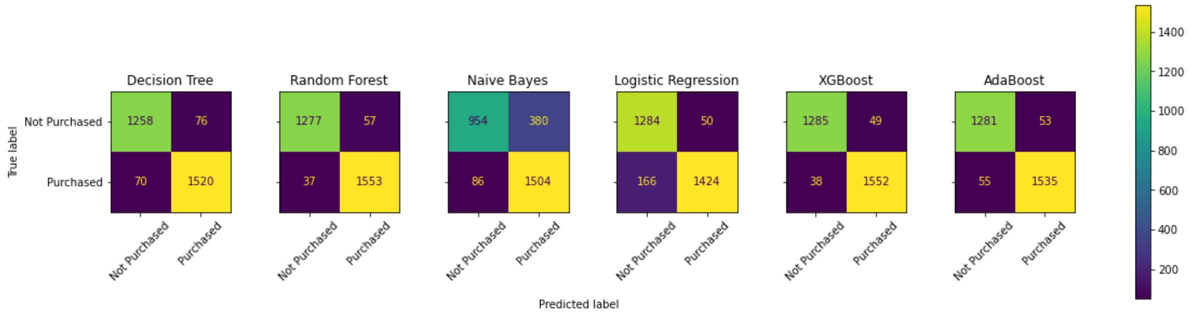
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score
classifiers = {
    "Decision Tree": tree.DecisionTreeClassifier(random_state=0),
    "Random Forest": RandomForestClassifier(random_state=0),
    "Naive Bayes": GaussianNB(),
    "Logistic Regression": LogisticRegression(),
    "XGBoost": xgboost.XGBClassifier(),
    "AdaBoost": AdaBoostClassifier(n_estimators=100, random_state=0),
    }

f, axes = plt.subplots(1, 6, figsize=(20, 5), sharey='row')

for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.fit(X_train, y_train).predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    print(key, " \n Accuracy:",accuracy_score(y_test,y_pred),"\n F-score",f1_score(y
    disp = ConfusionMatrixDisplay(cf_matrix,
                                  display_labels=["Not Purchased","Purchased"])
    disp.plot(ax=axes[i], xticks_rotation=45)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()
    disp.ax_.set_xlabel('')
    if i!=0:
        disp.ax_.set_ylabel('')

f.text(0.4, 0.1, 'Predicted label', ha='left')
plt.subplots_adjust(wspace=0.40, hspace=0.1)


f.colorbar(disp.im_, ax=axes)
plt.show()
```

```
Decision Tree
 Accuracy: 0.9500683994528044
 F-score 0.9541745134965474
Random Forest
 Accuracy: 0.9678522571819426
 F-score 0.9706250000000001
Naive Bayes
 Accuracy: 0.8406292749658003
 F-score 0.865860679332182
Logistic Regression
 Accuracy: 0.9261285909712722
 F-score 0.9295039164490861
```

```
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converg
e (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as sho
wn in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver option
s:
    https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression (https://scikit-learn.org/stable/modules/linear_model.ht
ml#logistic-regression)
```

```
  n_iter_i = _check_optimize_result(
/Users/ingrid/opt/anaconda3/lib/python3.9/site-packages/xgboost/sklea
rn.py:1224: UserWarning: The use of label encoder in XGBClassifier is
deprecated and will be removed in a future release. To remove this wa
rning, do the following: 1) Pass option use_label_encoder=False when
 constructing XGBClassifier object; and 2) Encode your labels (y) as
 integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[14:07:57] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1
643227205751/work/src/learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' wa
s changed from 'error' to 'logloss'. Explicitly set eval_metric if yo
u'd like to restore the old behavior.
XGBoost
 Accuracy: 0.9702462380300958
 F-score 0.9727358194923221
AdaBoost
 Accuracy: 0.9630642954856361
 F-score 0.9660163624921335
```



hypertuning

In [71]:

```
## Hyper Parameter Optimization

params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

In [72]:

```
classifier_smote_hpo = xgboost.XGBClassifier()
#classifier_smote_hpo.fit(X_train, y_train)
```

In [73]:

```
from sklearn.model_selection import RandomizedSearchCV
```

In [74]:

```
def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now() - start_time).total_seconds(), 3600
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, r
```

In [75]:

```
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

In [76]:

```
random_search=RandomizedSearchCV(classifier_smote_hpo,param_distributions=params,
                                  n_iter=5,scoring='roc_auc',n_jobs=-1,cv=20,verbose=

from datetime import datetime

start_time = timer(None)
random_search.fit(X_resampled1, y_resampled1)
timer(start_time) # timing ends here for "start_time" variable
```

```
Fitting 20 folds for each of 5 candidates, totalling 100 fits
```

In [77]:

```
random_search.best_estimator_
```

Out[77]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4,
              enable_categorical=False, gamma=0.1, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=nan, monotone_constraints
='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, predict
or='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_wei
ght=1,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

In [78]:

```
random_search.best_params_
```

Out[78]:

```
{'min_child_weight': 1,
 'max_depth': 10,
 'learning_rate': 0.3,
 'gamma': 0.1,
 'colsample_bytree': 0.4}
```

In [79]:

```
classifier=xgboost.XGBClassifier(colsample_bytree = 0.5, gamma = 0.1, learning_rate
                                 max_depth = 12, min_child_weight = 3)
```

In [80]:

```
classifier = classifier.fit(X_train, y_train)
```

```
[14:13:40] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1
643227205751/work/src/learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' wa
s changed from 'error' to 'logloss'. Explicitly set eval_metric if yo
u'd like to restore the old behavior.
```

In [81]:

```
y_pred_new = classifier.predict(X_test)
```

In [82]:

```
result = confusion_matrix(y_test, y_pred_new)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred_new)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test, y_pred_new)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[1286   48]
 [  38 1552]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      1334
           1       0.97      0.98      0.97      1590

    accuracy                           0.97      2924
   macro avg       0.97      0.97      0.97      2924
weighted avg       0.97      0.97      0.97      2924

Accuracy: 0.9705882352941176
```

prediction

In [83]:

```
print(X_test[0:5])
```

```
       Administrative  Administrative_Duration  Informational  \
6039                0                      0.0              0
13089               0                      0.0              0
847                 0                      0.0              0
6370                0                      0.0              0
2336                0                      0.0              0

       Informational_Duration  ProductRelated  ProductRelated_Duration
\
6039                      0.0              40               694.017857
13089                     0.0              15               844.445154
847                       0.0               3                38.000000
6370                      0.0              21               607.589286
2336                      0.0              10               146.766667

       BounceRates  ExitRates  PageValues  SpecialDay  ...  TrafficTyp
e_20  \
6039      0.005000   0.017500    0.000000         0.0  ...
0
13089     0.000000   0.001505  110.799663         0.0  ...
0
847       0.100000   0.122222    0.000000         0.0  ...
0
6370      0.028571   0.051323    0.000000         0.0  ...
0
2336      0.000000   0.002273    0.000000         0.0  ...
0

       TrafficType_3  TrafficType_4  TrafficType_5  TrafficType_6  \
6039               0              0              0              0
13089              0              0              0              0
847                0              0              0              0
6370               0              0              0              0
2336               0              0              0              0

       TrafficType_7  TrafficType_8  TrafficType_9  VisitorType_Other
\
6039               0              0              0                  0
13089              0              0              0                  0
847                0              0              0                  0
6370               0              0              0                  0
2336               0              0              0                  0

       VisitorType_Returning_Visitor
6039                               0
13089                              0
847                                1
6370                               1
2336                               1

[5 rows x 67 columns]
```

In [84]:

```
print(y_pred_new[0:5])
```

```
[0 1 0 0 0]
```

In [85]:

```
print(y_test[0:5])
```

```
6039     0
13089    1
847      0
6370     0
2336     0
Name: Revenue, dtype: int64
```