—

Classification Project

In this project we will be working with a advertising data set, indicating whether or not a particular internet user clicked on an Advertisement. We will try to create a model that will predict whether or not they will click on an ad based off the features of that user.

This data set contains the following features:

- 'Daily Time Spent on Site': consumer time on site in minutes
- 'Age': cutomer age in years
- 'Area Income': Avg. Income of geographical area of consumer
- 'Daily Internet Usage': Avg. minutes a day consumer is on the internet
- 'Ad Topic Line': Headline of the advertisement
- 'City': City of consumer
- 'Male': Whether or not consumer was male
- 'Country': Country of consumer
- 'Timestamp': Time at which consumer clicked on Ad or closed window
- 'Clicked on Ad': 0 or 1 indicated clicking on Ad

Import Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

In [3]:
```
df=pd.read_csv('advertising.csv')
```

In [4]:
```
df
```

Out[4]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
...	...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00	1
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01	1
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57	1
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54	0
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21	1

1000 rows × 10 columns

In [5]:
```
df['Male'].replace([0, 1],['Female','Male'], inplace = True)
df.rename(columns = {'Male': 'Gender'}, inplace = True)
```

Out[5]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Gender	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	Female	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	Male	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	Female	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	Male	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	Female	Iceland	2016-06-03 03:36:18	0

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Daily Time Spent on Site  1000 non-null   float64
 1   Age                       1000 non-null   int64
 2   Area Income               1000 non-null   float64
 3   Daily Internet Usage      1000 non-null   float64
 4   Ad Topic Line             1000 non-null   object
 5   City                      1000 non-null   object
 6   Gender                    1000 non-null   object
 7   Country                   1000 non-null   object
 8   Timestamp                 1000 non-null   object
 9   Clicked on Ad             1000 non-null   int64
dtypes: float64(3), int64(2), object(5)
memory usage: 78.2+ KB
```

In [7]:
```
df.describe()
```

Out[7]:

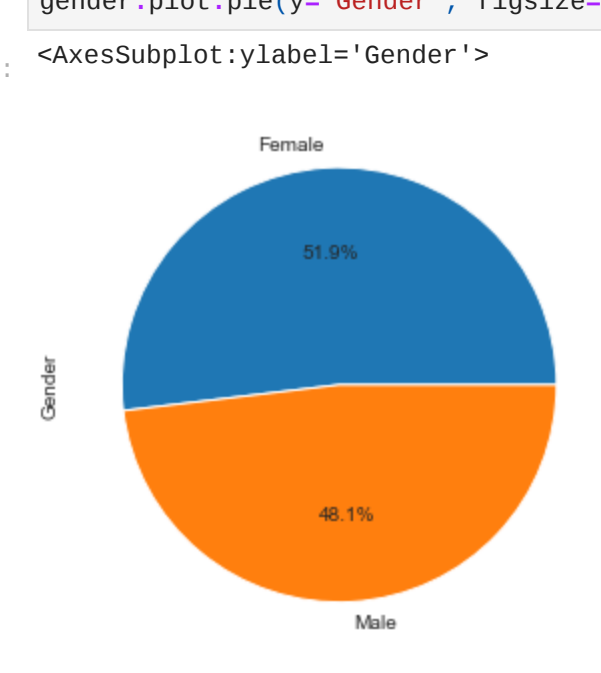
	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.50025
min	32.600000	19.000000	13996.500000	104.780000	0.00000
25%	51.360000	29.000000	47031.802500	138.830000	0.00000
50%	68.215000	35.000000	57012.300000	183.130000	0.50000
75%	78.547500	42.000000	65470.635000	218.792500	1.00000
max	91.430000	61.000000	79484.800000	269.960000	1.00000

Exploratory Data Analysis

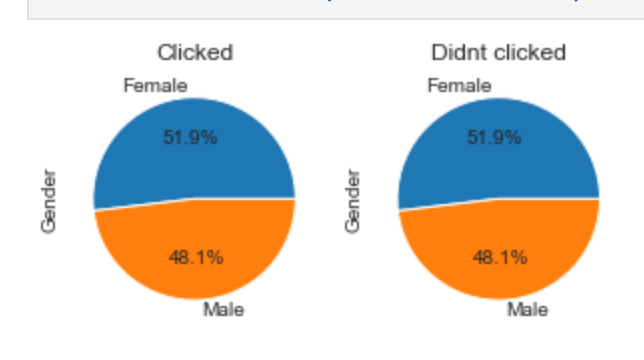
In [16]:
```
gender=df.groupby(['Gender'])['Gender'].count()
gender.plot.pie(y='Gender', figsize=(5,5), autopct='%1.1f%%')
```

Out[16]:

```
<AxesSubplot:ylabel='Gender'>
```



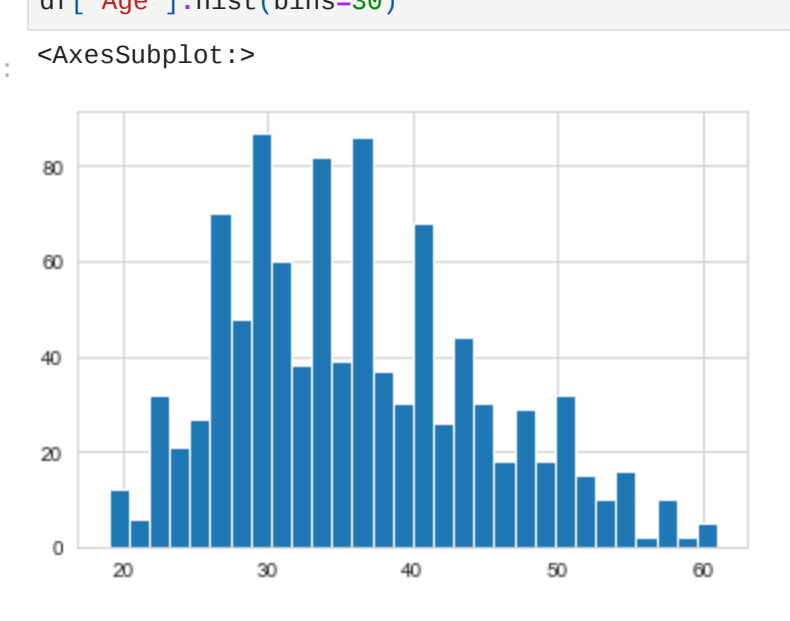
In [18]:
```
gender_click=df.groupby(['Gender'])['Gender'].count()
gender_noclick=df.groupby(['Gender'])['Gender'].count()

fig, axes = plt.subplots(1,2)
ax1=gender_click.plot.pie(y='Gender', figsize=(5,5), autopct='%1.1f%%', ax=axes[0])
ax2=gender_noclick.plot.pie(y='Gender', figsize=(5,5), autopct='%1.1f%%', ax=axes[1])
ax1.title.set_text('Clicked')
ax2.title.set_text('Didnt clicked')
```



In [11]:
```
sns.set_style('whitegrid')
df['Age'].hist(bins=30)
```

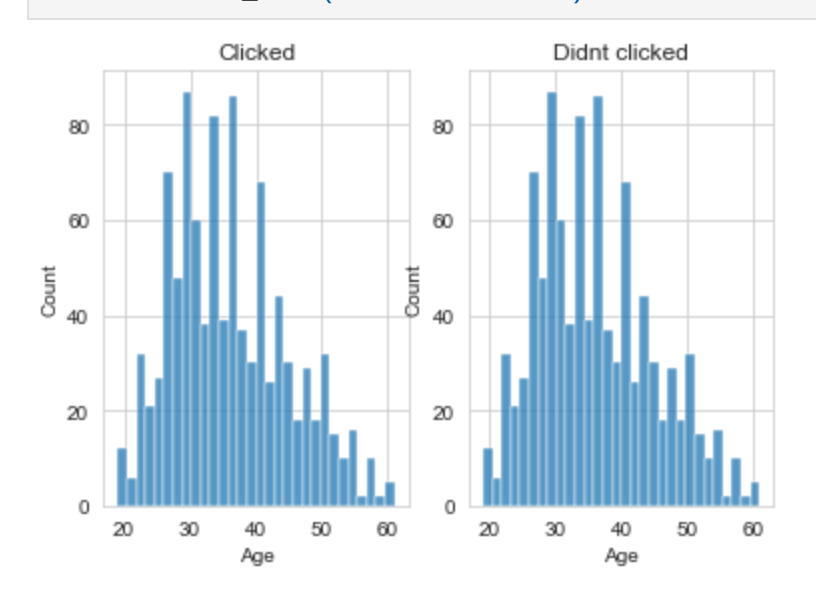
Out[11]:

```
<AxesSubplot:>
```



customer will click on ad or not based on age.

In [13]:
```
fig, axes = plt.subplots(1,2)
ax1=sns.histplot(x="Age", data=df, bins=30, ax=axes[0])
ax2=sns.histplot(x="Age", data=df, bins=30, ax=axes[1])
ax1.title.set_text('Clicked')
ax2.title.set_text('Didnt clicked')
```

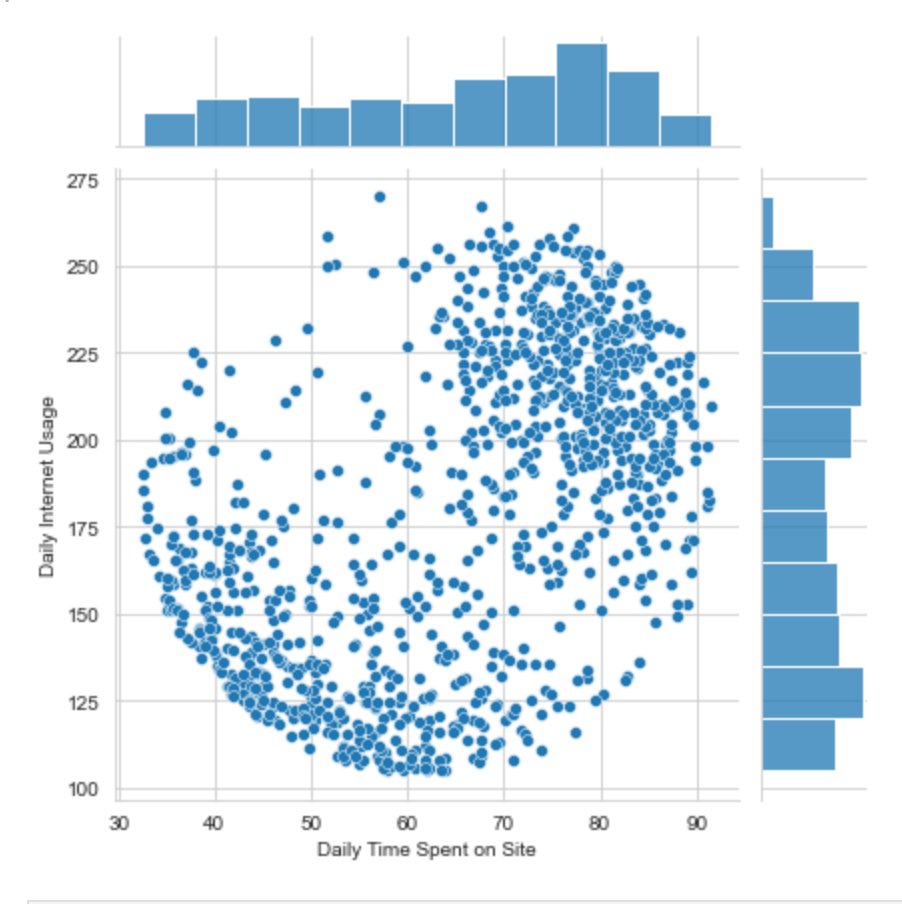


Create a jointplot showing the kde distributions of Daily Time spent on site vs. Age.

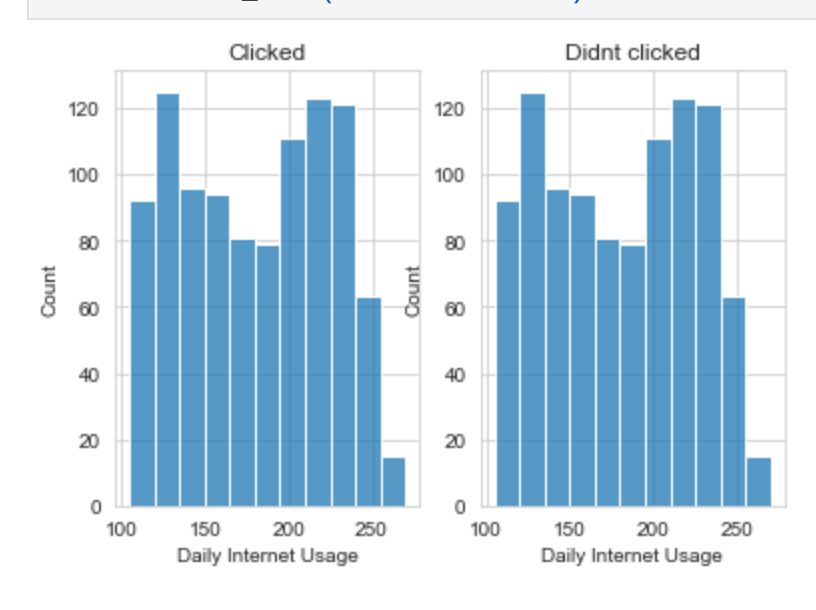
In [8]:
```
sns.jointplot(x='Daily Time Spent on Site',y='Daily Internet Usage',data=df)
```

Out[8]:

```
<seaborn.axisgrid.JointGrid at 0x275a3bca130>
```



In [20]:
```
fig, axes = plt.subplots(1,2)
ax1=sns.histplot(x="Daily Internet Usage", data=df, ax=axes[0])
ax2=sns.histplot(x="Daily Internet Usage", data=df, ax=axes[1])
ax1.title.set_text('Clicked')
ax2.title.set_text('Didnt clicked')
```



Data preprocessing

In [10]:
```
df.head()
```

Out[10]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0

In [11]:
```
df.drop("Country",axis="columns",inplace=True)
```

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
x = df[['Daily Time Spent on Site', 'Age', 'Area Income','Daily Internet Usage', 'Male']]
```

In [14]:
```
y=df['Clicked on Ad']
```

In [15]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
```

In [16]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

Logistic Regression

Now it's time to do a train test split, and train our model!

In [17]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
```

In [18]:
```
lr=LogisticRegression()
lr.fit(X_train,y_train)
params = {'C':[0.01,0.1,1,10]}
lrgrid = GridSearchCV(estimator = lr, param_grid = params, cv = KFold(5), scoring = 'accuracy')
lrgrid.fit(X_train, y_train)
```

Out[18]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=None, shuffle=False),
             estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1, 10]}, scoring='accuracy')
```

Decision Tree classifier

In [19]:
```
from sklearn import tree
```

In [20]:
```
dt = tree.DecisionTreeClassifier(max_depth = 4)
dt.fit(X_train, y_train)
params = {'max_depth' : [2,4,8]}
dtgrid = GridSearchCV(estimator = dt, param_grid = params, cv = KFold(5) , scoring = 'accuracy')
dtgrid.fit(X_train, y_train)
```

Out[20]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=None, shuffle=False),
             estimator=DecisionTreeClassifier(max_depth=4),
             param_grid={'max_depth': [2, 4, 8]}, scoring='accuracy')
```

Random Forest Classifier

In [21]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
```

In [22]:
```
rf = RandomForestClassifier(n_estimators = 10, max_depth = 10)
rf.fit(X_train, y_train)

params = {'n_estimators' : [10, 20, 50, 100], 'max_depth' : [10, 50]}
rfgrid = GridSearchCV(estimator = rf, param_grid = params, cv = KFold(5), scoring = 'accuracy')
rfgrid.fit(X_train, y_train)
```

Out[22]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=None, shuffle=False),
             estimator=RandomForestClassifier(max_depth=10, n_estimators=10),
             param_grid={'max_depth': [10, 50],
                         'n_estimators': [10, 20, 50, 100]},
             scoring='accuracy')
```

KNN Classifier

In [23]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [24]:
```
kn = KNeighborsClassifier(n_neighbors = 10, p = 2)
kn.fit(X_train, y_train)
params = {'n_neighbors' : [2, 5, 10, 50], 'weights' : ['uniform', 'distance'], 'p' :[1,2]}
kngrid = GridSearchCV(estimator = kn, param_grid = params, cv = KFold(5), scoring = 'accuracy')
kngrid.fit(X_train, y_train)
```

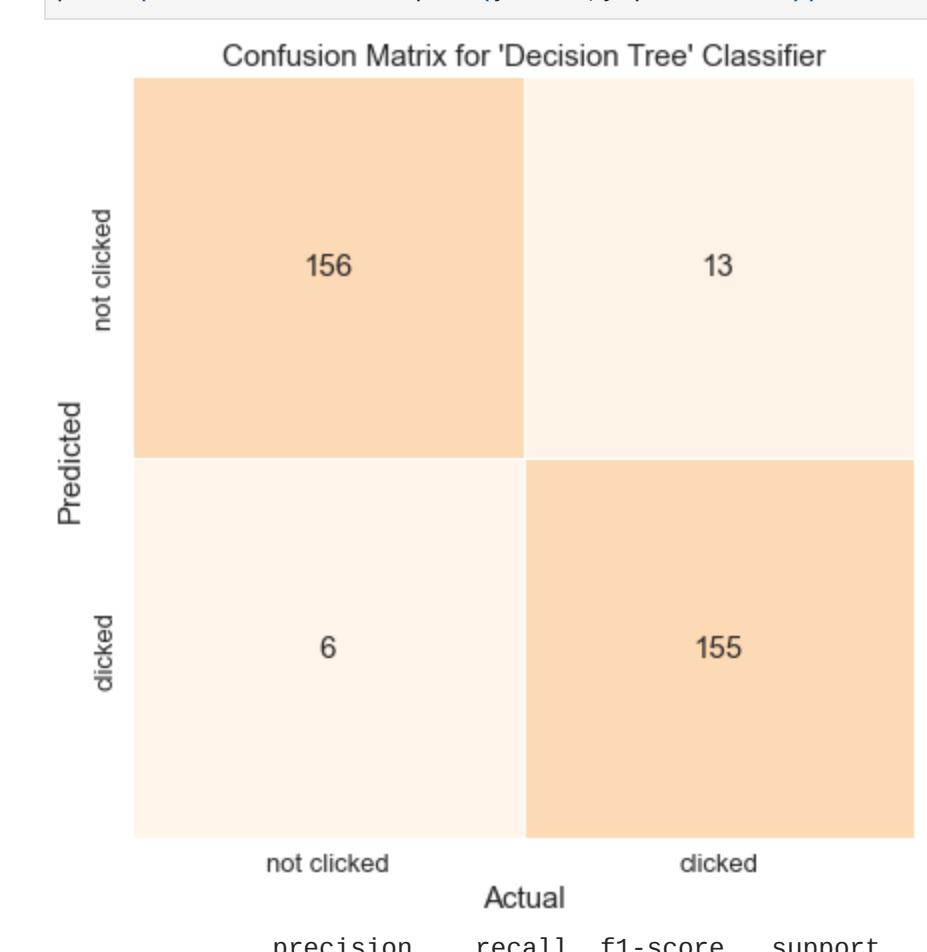
Out[24]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=None, shuffle=False),
             estimator=KNeighborsClassifier(n_neighbors=10),
             param_grid={'n_neighbors': [2, 5, 10, 50], 'p': [1, 2],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

Scores

In [26]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score

y_predict_dt = dt_best.predict(X_test)
accuracy = accuracy_score(y_test, y_predict_dt)
recall = recall_score(y_test, y_predict_dt)
performance_df['Decision_tree']['accuracy'] = accuracy
performance_df['Decision_tree']['recall'] = recall

y_predict_lr = lr_best.predict(X_test)
accuracy = accuracy_score(y_test, y_predict_lr)
recall = recall_score(y_test, y_predict_lr)
performance_df['Logistic_regression']['accuracy'] = accuracy
performance_df['Logistic_regression']['recall'] = recall

y_predict_rf = dt_best.predict(X_test)
accuracy = accuracy_score(y_test, y_predict_rf)
recall = recall_score(y_test, y_predict_rf)
performance_df['Random_forest']['accuracy'] = accuracy
performance_df['Random_forest']['recall'] = recall

y_predict_kn = dt_best.predict(X_test)
accuracy = accuracy_score(y_test, y_predict_kn)
recall = recall_score(y_test, y_predict_kn)
performance_df['K-NNeighbors']['accuracy'] = accuracy
performance_df['K-NNeighbors']['recall'] = recall
```

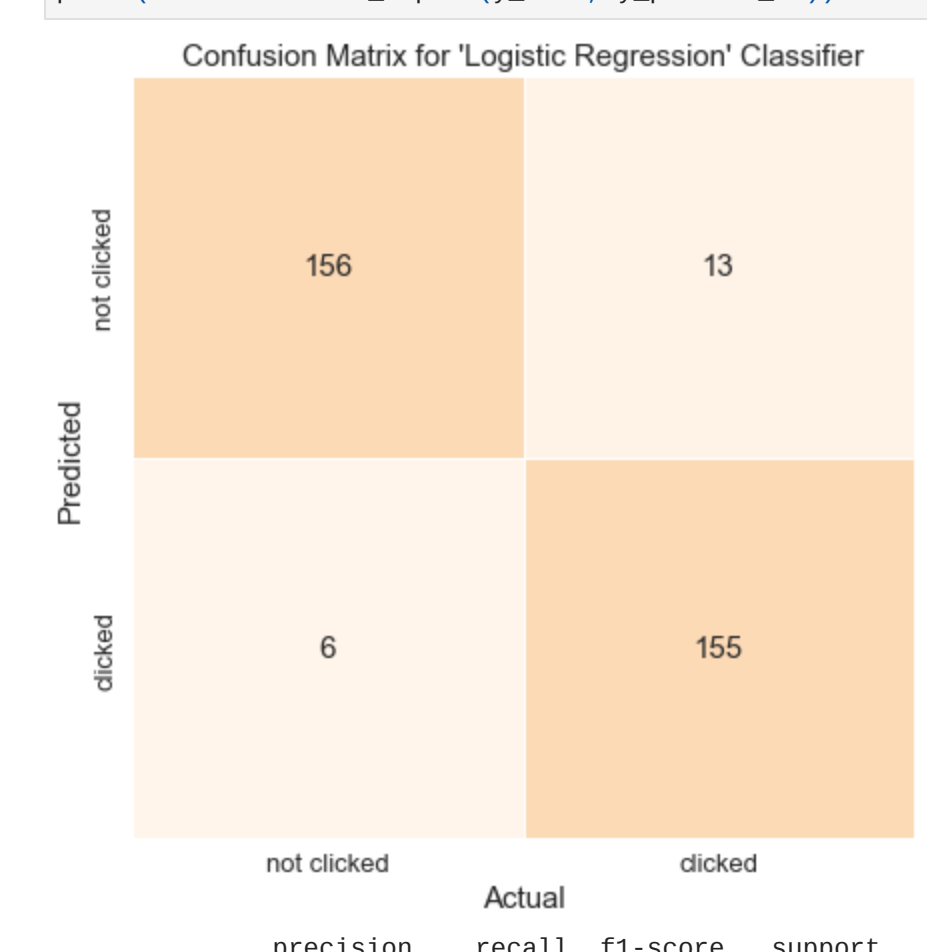
In [28]:
```
# generate confusion matrix for Decision Tree classifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
conf_mat_dt = confusion_matrix(y_test, y_predict_dt)
conf_math_dt_df = pd.DataFrame(conf_mat_dt)

conf_mat_dt.view()

fig, ax = plt.subplots(figsize = (7,7))
sns.heatmap(conf_math_dt_df.T, annot=True, annot_kws={"size": 15}, cmap="Oranges",
                                                    vmin=0, vmax=800, fmt='.0f', linewidths=1, linecolor="white", cbar=False,
                                                    xticklabels=["not clicked","clicked"],yticklabels=["not clicked","clicked"])
plt.ylabel("Predicted", fontsize=15)
plt.xlabel("Actual", fontsize=15)
ax.set_xticklabels(["not clicked","clicked"], fontsize=13)
ax.set_yticklabels(["not clicked","clicked"], fontsize=13)
plt.title("Confusion Matrix for 'Decision Tree' Classifier", fontsize=15)
plt.show()
print("")
print(classification_report(y_test,y_predict_dt))
```




```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       162
           1       0.96      0.92      0.94       168

    accuracy                           0.94       330
   macro avg       0.94      0.94      0.94       330
weighted avg       0.94      0.94      0.94       330
```

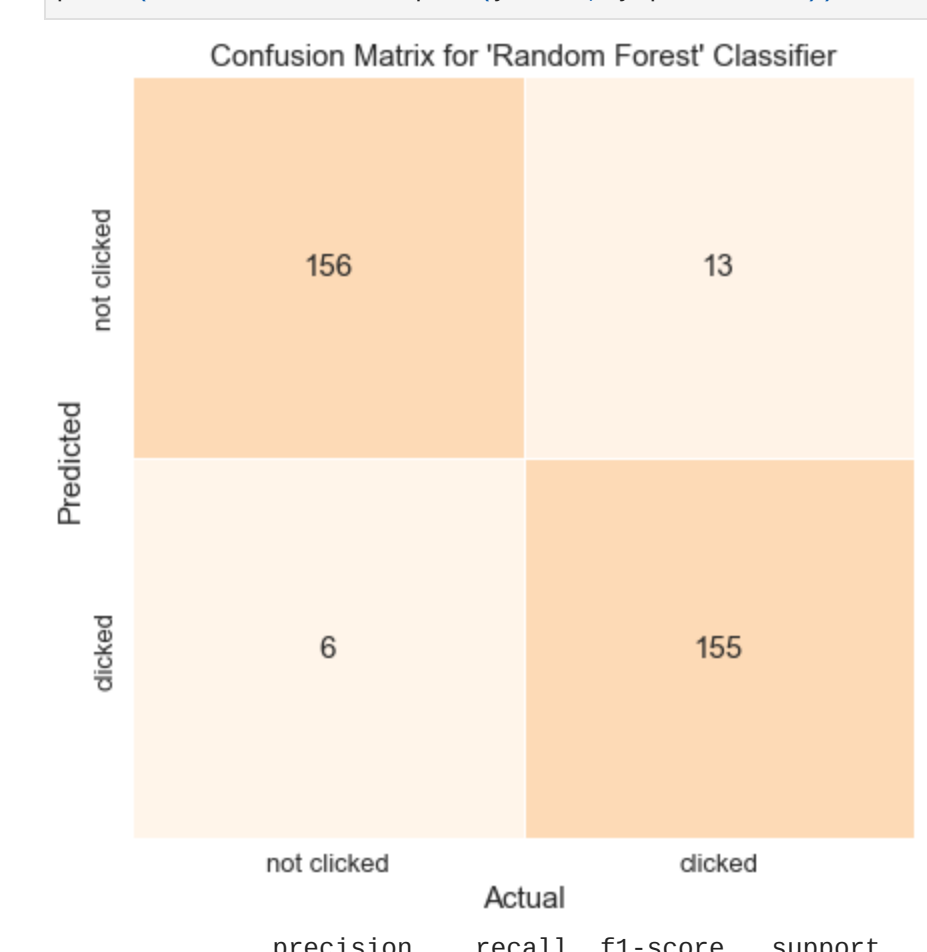
In [ ]:

In [29]:
```
# generate confusion matrix for logistic regression

conf_mat_lr = confusion_matrix(y_test, y_predict_lr)
conf_math_lr_df = pd.DataFrame(conf_mat_lr)



fig, ax = plt.subplots(figsize = (7,7))
sns.heatmap(conf_math_dt_df.T, annot=True, annot_kws={"size": 15}, cmap="Oranges",
                                                    vmin=0, vmax=800, fmt='.0f', linewidths=1, linecolor="white", cbar=False,
                                                    xticklabels=["not clicked","clicked"],yticklabels=["not clicked","clicked"])
plt.ylabel("Predicted", fontsize=15)
plt.xlabel("Actual", fontsize=15)
ax.set_xticklabels(["not clicked","clicked"], fontsize=13)
ax.set_yticklabels(["not clicked","clicked"], fontsize=13)
plt.title("Confusion Matrix for 'Logistic Regression' Classifier", fontsize=15)
plt.show()
print("")
print(classification_report(y_test, y_predict_lr))
```



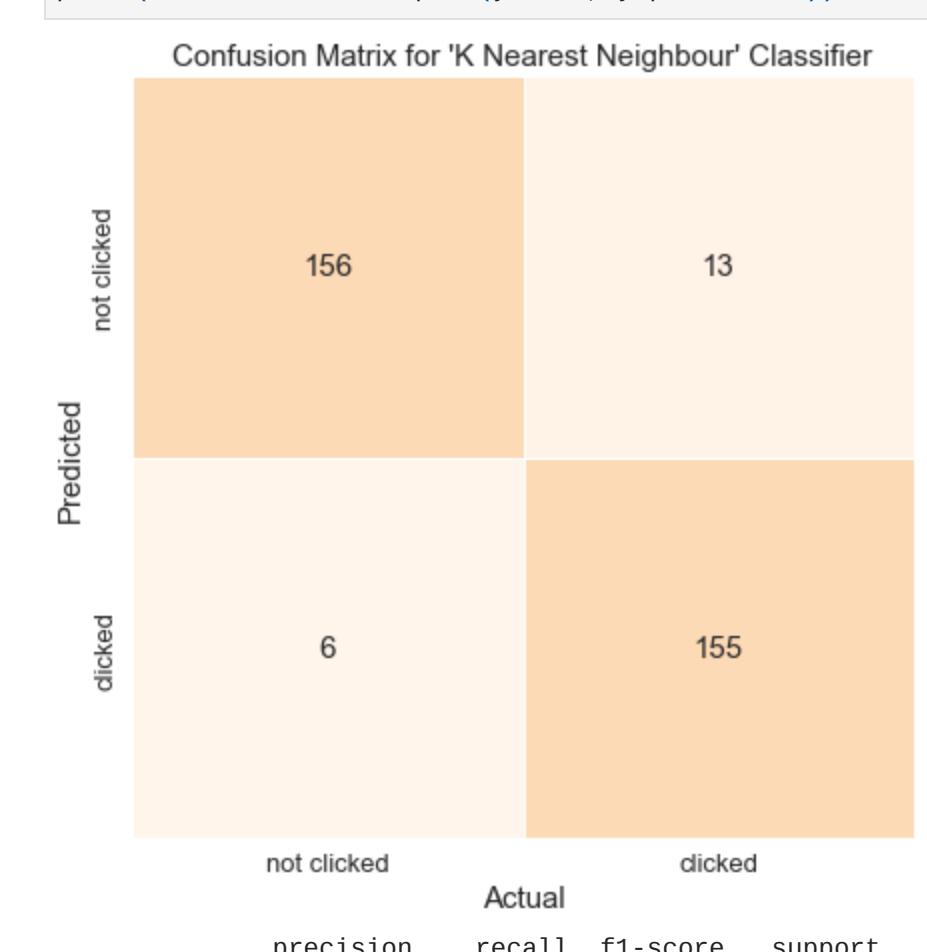

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       162
           1       0.99      0.93      0.96       168

    accuracy                           0.96       330
   macro avg       0.96      0.96      0.96       330
weighted avg       0.96      0.96      0.96       330
```

In [30]:
```
# generate confusion matrix for Random Forest Classifier

conf_mat_rf = confusion_matrix(y_test, y_predict_rf)
conf_math_rf_df = pd.DataFrame(conf_mat_rf)



fig, ax = plt.subplots(figsize = (7,7))
sns.heatmap(conf_math_dt_df.T, annot=True, annot_kws={"size": 15}, cmap="Oranges",
                                                    vmin=0, vmax=800, fmt='.0f', linewidths=1, linecolor="white", cbar=False,
                                                    xticklabels=["not clicked","clicked"],yticklabels=["not clicked","clicked"])
plt.ylabel("Predicted", fontsize=15)
plt.xlabel("Actual", fontsize=15)
ax.set_xticklabels(["not clicked","clicked"], fontsize=13)
ax.set_yticklabels(["not clicked","clicked"], fontsize=13)
plt.title("Confusion Matrix for 'Random Forest' Classifier", fontsize=15)
plt.show()
print("")
print(classification_report(y_test, y_predict_rf))
```




```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       162
           1       0.96      0.92      0.94       168

    accuracy                           0.94       330
   macro avg       0.94      0.94      0.94       330
weighted avg       0.94      0.94      0.94       330
```

In [31]:
```
# generate confusion matrix for K Nearest Neighbour
conf_mat_kn = confusion_matrix(y_test, y_predict_kn)
conf_math_kn_df = pd.DataFrame(conf_mat_kn)



fig, ax = plt.subplots(figsize = (7,7))
sns.heatmap(conf_math_dt_df.T, annot=True, annot_kws={"size": 15}, cmap="Oranges",
                                                    vmin=0, vmax=800, fmt='.0f', linewidths=1, linecolor="white", cbar=False,
                                                    xticklabels=["not clicked","clicked"],yticklabels=["not clicked","clicked"])
plt.ylabel("Predicted", fontsize=15)
plt.xlabel("Actual", fontsize=15)
ax.set_xticklabels(["not clicked","clicked"], fontsize=13)
ax.set_yticklabels(["not clicked","clicked"], fontsize=13)
plt.title("Confusion Matrix for 'K Nearest Neighbour' Classifier", fontsize=15)
plt.show()
print("")
print(classification_report(y_test, y_predict_kn))
```




```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       162
           1       0.96      0.92      0.94       168

    accuracy                           0.94       330
   macro avg       0.94      0.94      0.94       330
weighted avg       0.94      0.94      0.94       330
```

Predictions and Evaluations

In [32]:
```
predictions=lr.predict(X_test)
```

In [33]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       162
           1       0.99      0.93      0.96       168

    accuracy                           0.96       330
   macro avg       0.96      0.96      0.96       330
weighted avg       0.96      0.96      0.96       330
```

In [34]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test , predictions)
```

Out[34]:

```
array([[160,   2],
       [ 11, 157]], dtype=int64)
```

As we can see from the classification and confusion matrix our model is a good fit and have low false positive false and false negative