In [ ]:

```
# Data from statsbomb and using pandas dataframes
import statsbomb as sb
import pandas as pd
import numpy as np

# Plotting
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

# Machine learning
from sklearn import preprocessing, model_selection, svm, metrics
from sklearn.model_selection import train_test_split, cross_val_predict, cross_v
al_score, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.utils import resample
from sklearn.feature_selection import RFE
from imblearn.over_sampling import SMOTE
from collections import Counter
```

In [2]:

```
# Get all competitions
competitions = sb.Competitions()

# Get json data
json_data = competitions.data

# Convert to dataframe
df = competitions.get_dataframe()
```

In [3]:

```
# Showing the competitions
# We will find and pick the data for World Cup 2018
df
```

Out[3]:

	competition_gender	competition_id	competition_name	country_name	match_available	m
0	female	37	FA Women's Super League	England	2019-10-20T10:18:13.183	20
1	female	37	FA Women's Super League	England	2019-06-23T15:32:29.914	23
2	male	43	FIFA World Cup	International	2019-06-23T12:38:35.142	23
3	male	11	La Liga	Spain	2019-07-29T20:44:30.861	29
4	male	11	La Liga	Spain	2019-07-30T12:42:05.563	30
5	male	11	La Liga	Spain	2019-07-24T19:44:48.866	24
6	male	11	La Liga	Spain	2019-07-29T17:46:18.935	29
7	male	11	La Liga	Spain	2019-08-27T09:48:17.842	27
8	male	11	La Liga	Spain	2019-08-01T17:44:54.870	01
9	male	11	La Liga	Spain	2019-07-11T07:44:14.533	11
10	male	11	La Liga	Spain	2019-07-06T22:42:14.468	06
11	male	11	La Liga	Spain	2019-07-04T10:05:48.149	04
12	male	11	La Liga	Spain	2019-07-02T12:37:13.627	02
13	male	11	La Liga	Spain	2019-07-02T14:37:04.106	02
14	male	11	La Liga	Spain	2019-06-28T12:41:15.620	28
15	female	49	NWSL	United States of America	2019-07-24T14:38:11.095	24
16	female	72	Women's World Cup	International	2019-07-10T13:51:28.459	10

In [4]:

```
# Get all World Cup matches
wc_matches = sb.Matches(event_id = '43', season_id = '3').get_dataframe()
```

In [5]:

```python
# Show the World Cup games
wc_matches.head()
```

Out[5]:

	away_score	away_team	competition	competition_stage	home_score	home_team	kic
0	0	782	43	{'id': 15, 'name': 'Semi-finals'}	1	771	20:00:0
1	0	768	43	{'id': 25, 'name': '3rd Place Final'}	2	782	16:00:0
2	1	772	43	{'id': 10, 'name': 'Group Stage'}	0	797	20:00:0
3	2	784	43	{'id': 10, 'name': 'Group Stage'}	0	792	16:00:0
4	0	793	43	{'id': 10, 'name': 'Group Stage'}	2	775	17:00:0

In [6]:

```python
# But we want to focus on shots and goals
# Create a list of all match ids for the matches in the 2018 World Cup
match_list = wc_matches['match_id'].tolist()

# Create an empty dataframe to add all shots
shots_df = pd.DataFrame()

# Loop through and add all shots from every match to the empty dataframe
for i in match_list:
    events = sb.Events(event_id=str(i))
    shot = events.get_dataframe(event_type='shot')
    shots_df = shots_df.append(shot)
```

In [7]:

```
# Details on our data
print(len(shots_df), "is the total number of shots from every match at the 2018
 World Cup!")
print("--------------------")

print("Columns:")
print(list(shots_df))
print("--------------------")

print("Unique values:")
print(shots_df['type'].unique())
print("--------------------")

print("Unique values in the 'play_pattern' column:")
print(shots_df['play_pattern'].unique())
```

```
1706 is the total number of shots from every match at the 2018 World
Cup!
--------------------
Columns:
['event_type', 'id', 'index', 'period', 'timestamp', 'minute', 'seco
nd', 'possession', 'possession_team', 'play_pattern', 'off_camera',
'team', 'player', 'position', 'duration', 'under_pressure', 'statsbo
mb_xg', 'key_pass_id', 'body_part', 'type', 'outcome', 'technique',
'first_time', 'follows_dribble', 'redirect', 'one_on_one', 'open_goa
l', 'deflected', 'start_location_x', 'start_location_y', 'end_locati
on_x', 'end_location_y', 'end_location_z']
--------------------
Unique values:
['Open Play' 'Free Kick' 'Penalty']
--------------------
Unique values in the 'play_pattern' column:
['Regular Play' 'From Throw In' 'From Corner' 'From Free Kick'
 'From Counter' 'From Goal Kick' 'From Kick Off' 'Other' 'From Keepe
r']
```

In [8]:

```
# Remove penalties from the data since they are misleading in our analysis of sh
ots
nopen_shots = shots_df[shots_df['type'] != 'Penalty']
```

In [9]:

```
# Create a goal column, where 1 = goal and 0 = no goal
nopen_shots['goal'] = np.where(nopen_shots['outcome'] == 'Goal', 1, 0)
```

```
/Users/SaeJin/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
```

In [10]:

```
# Show data
nopen_shots.head()
```

Out[10]:

	event_type	id	index	period	timestamp	minute	second	possession	posse
0	shot	e33fea6f-1c7e-41b0-8526-fcc5e0fbe3aa	698	1	00:14:59.800	14	59	25	
1	shot	98f8c815-5d4a-4a71-bf1a-5e0d287b2437	786	1	00:17:45.600	17	45	30	
2	shot	6af02b71-85f3-4061-b9d6-7478d099d4bb	819	1	00:18:32.760	18	32	32	
3	shot	a2299393-0163-45bc-ab77-fdbf91b8c426	909	1	00:21:00.320	21	0	36	
4	shot	8f91c4c0-704c-4a08-937c-ff2510d4e218	958	1	00:22:41.960	22	41	38	

5 rows × 34 columns

In [11]:

```
# Calculating average shot conversion rate
attempts = len(nopen_shots)
goals = sum(nopen_shots['goal'])
misses = attempts - goals
conversion_rate = goals / attempts
print('Average conversion rate: ',round(conversion_rate*100,2),"%")
```

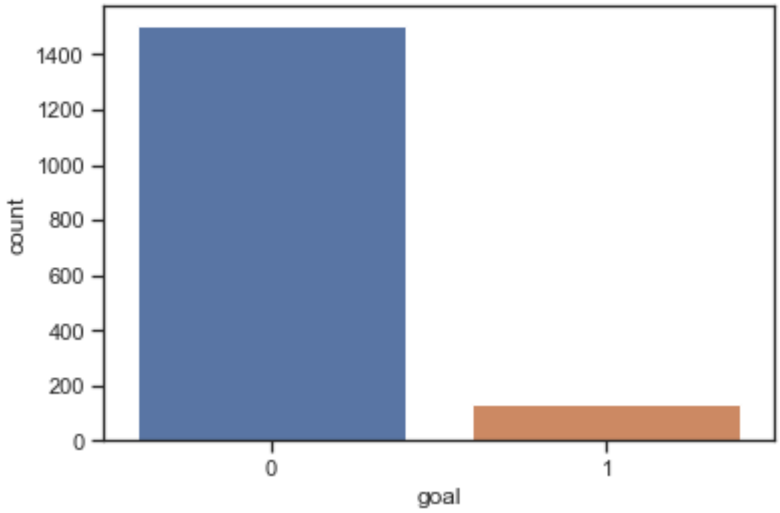
```
Average conversion rate:  8.24 %
```

In [12]:

```
# Show graph
# We see that most shots end up not being being a goal
sns.set(style="ticks", color_codes=True)
sns.countplot(x="goal", data=nopen_shots)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2655fa90>
```



In [13]:

```
# Feature engineering - Make new variables out of the raw data to improve model
# Resetting index
nopen_shots = nopen_shots.reset_index().drop('level_0', axis=1)

# Our new variables: distance
# Length of soccer field is 120 units long and width is 80 units wide
# Use distance formula and angle from the centre of the goal being scored on ,at
(120, 40), to location of shot
nopen_shots['x_distance'] = 120 - nopen_shots['start_location_x']
nopen_shots['y_distance'] = abs(40 - nopen_shots['start_location_y'])
nopen_shots['distance'] = np.sqrt((nopen_shots['x_distance']**2 + nopen_shots['y
_distance']**2))

# If we know the player's weak and strong foot, we can make look at these data i
ndividually
nopen_shots['body_part'] = np.where((nopen_shots['body_part'] == 'Right Foot')
                                    | (nopen_shots['body_part'] == 'Left Foot'), 'f
oot',
                                   np.where(nopen_shots['body_part'] == 'Head', 'he
ad', 'other'))
```

In [14]:

```
# Find model features in cols
feature_cols = ['play_pattern', 'under_pressure', 'body_part', 'technique', 'fir
st_time',
                'follows_dribble', 'redirect', 'one_on_one', 'open_goal', 'defle
cted', 'distance']


features = nopen_shots[feature_cols]
# Find model labels
labels = nopen_shots['goal']

# 0 to NA values
features = features.fillna(0)
labels = labels.fillna(0)
```

In [15]:

```
# Show features
# As you can see, we have the new variables of distance and angle and have fille
d up NA values with 0
# We have the relevant and useful information we need
features
```

Out[15]:

	play_pattern	under_pressure	body_part	technique	first_time	follows_dribble	redirect
0	Regular Play	True	foot	Normal	0	0	0
1	From Throw In	0	foot	Half Volley	True	0	0
2	Regular Play	True	foot	Normal	0	0	0
3	From Corner	0	foot	Normal	0	0	0
4	Regular Play	0	foot	Normal	True	0	0
5	From Free Kick	0	head	Normal	0	0	0
6	Regular Play	0	foot	Normal	True	0	0
7	Regular Play	0	foot	Normal	0	0	0
8	Regular Play	True	foot	Normal	True	0	0
9	From Counter	0	foot	Normal	0	0	0
10	Regular Play	0	foot	Normal	0	0	0
11	Regular Play	0	foot	Normal	True	0	0
12	Regular Play	True	head	Normal	0	0	0
13	From Free Kick	0	foot	Normal	0	0	0
14	Regular Play	True	head	Normal	0	0	0
15	Regular Play	0	foot	Normal	0	0	0
16	From Corner	True	head	Normal	0	0	0
17	Regular Play	0	foot	Normal	0	0	0
18	Regular Play	0	foot	Normal	0	0	0
19	Regular Play	0	foot	Normal	0	0	0
20	Regular Play	0	foot	Volley	0	0	0
21	From Throw In	0	head	Normal	0	0	0
22	Regular Play	0	foot	Normal	True	0	0
23	From Free Kick	0	foot	Normal	0	0	0
24	From Free Kick	0	head	Normal	0	0	0
25	Regular Play	0	foot	Normal	0	0	0
26	Regular Play	0	foot	Normal	True	0	0
27	From Throw In	0	foot	Normal	0	0	0
28	Regular Play	0	foot	Volley	True	0	0
29	From Throw In	0	foot	Normal	0	0	0
...	...	...	...	...	...	...	...

	play_pattern	under_pressure	body_part	technique	first_time	follows_dribble	redirect
1608	From Free Kick	0	foot	Normal	True	0	0
1609	From Throw In	True	foot	Normal	0	0	0
1610	From Free Kick	0	foot	Volley	True	0	0
1611	From Free Kick	0	foot	Normal	0	0	0
1612	Regular Play	0	foot	Normal	0	0	0
1613	Regular Play	0	foot	Normal	0	0	0
1614	From Corner	0	foot	Normal	0	0	0
1615	Regular Play	0	head	Normal	0	0	0
1616	Regular Play	0	foot	Normal	0	0	0
1617	From Corner	0	foot	Normal	0	0	0
1618	From Corner	0	foot	Normal	0	0	0
1619	From Corner	0	head	Normal	0	0	0
1620	From Goal Kick	0	foot	Normal	0	0	0
1621	From Corner	0	head	Diving Header	0	0	0
1622	Regular Play	True	head	Normal	0	0	0
1623	From Free Kick	True	foot	Normal	True	0	0
1624	Regular Play	0	foot	Volley	True	0	0
1625	From Throw In	0	foot	Normal	0	0	0
1626	Regular Play	0	foot	Normal	0	0	0
1627	From Free Kick	0	foot	Normal	0	0	0
1628	Regular Play	0	foot	Normal	0	0	0
1629	From Throw In	0	foot	Normal	0	0	0
1630	From Throw In	0	foot	Normal	0	0	0
1631	From Throw In	0	foot	Half Volley	True	0	0
1632	From Free Kick	0	foot	Normal	0	0	0
1633	From Corner	0	foot	Normal	0	0	0
1634	From Free Kick	0	foot	Normal	True	0	0
1635	From Goal Kick	0	foot	Normal	True	0	0
1636	From Goal Kick	0	foot	Normal	0	0	0
1637	Regular Play	0	foot	Normal	0	0	0

1638 rows × 11 columns

In [16]:

```
# Label Encoding
# Make all categorical features to discrete values
cat_cols = ['play_pattern', 'under_pressure', 'body_part', 'technique', 'first_t
ime',
                'follows_dribble', 'redirect', 'one_on_one', 'open_goal', 'defle
cted']

cat_features = features[cat_cols]
features = features.drop(cat_cols, axis=1)
features.head()
```

Out[16]:

	distance
0	20.248457
1	28.071338
2	23.021729
3	15.033296
4	18.788294

In [17]:

```
# Discrete values for each category
le = preprocessing.LabelEncoder()
cat_features = cat_features.apply(le.fit_transform)

# Merge
features = features.merge(cat_features, left_index=True, right_index=True)
features.head()
# Now we have turned all categorical features into discrete values, so we can wo
rk with a consistent dataset
```

Out[17]:

	distance	play_pattern	under_pressure	body_part	technique	first_time	follows_dribble
0	20.248457	8	1	0	4	0	0
1	28.071338	6	0	0	2	1	0
2	23.021729	8	1	0	4	0	0
3	15.033296	0	0	0	4	0	0
4	18.788294	8	0	0	4	1	0

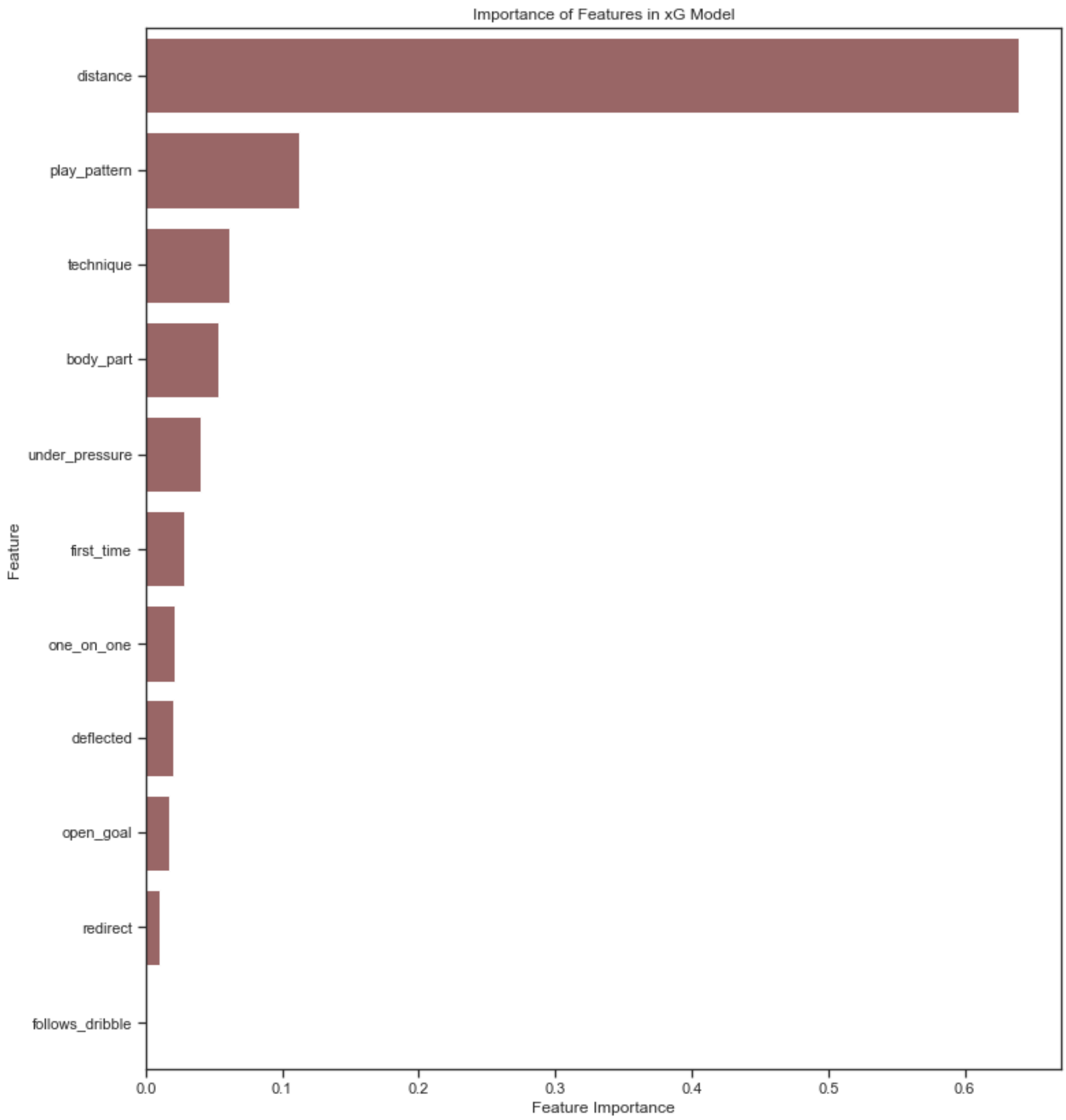
In [18]:

```
# X = model features, and y = labels
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=
0.20, shuffle=True, random_state=42)

# Using the Decision Tree Model
clf = DecisionTreeClassifier(random_state=42)

# Training
clf.fit(X_train, y_train)

# Making predictions
y_pred = clf.predict(X_test)
y_pred_prob = clf.predict_proba(X_test)
```

In [19]:

```
# Print results
print("Predicted goals from test data:", sum(y_pred))
print("xG:", "{0:.2f}".format(sum(y_pred_prob[:,1])))
print("Actual goals from test set):", sum(y_test))
print('Difference from my xG value and actual goals = ','{0:.2f}'.format(sum(y_p
red_prob[:,1])-sum(y_test)))
print('')
print(metrics.classification_report(y_test, y_pred))
# Precision means of all the goals the modelled claimed to be a goal, how many w
ere goals?
# Recall means of all the actual goals, how many did the model predict to be a g
oal?
# F1 score is the average of precision and recall, so the higher the F1 score, t
he better the model

# Find the most important feature for goal
important_features = pd.DataFrame({'feature':features.columns,'importance':np.ro
und(clf.feature_importances_,3)})
important_features = important_features.sort_values('importance',ascending=False
)

f, ax = plt.subplots(figsize=(12, 14))
g = sns.barplot(x='importance', y='feature', data=important_features,
                color="red", saturation=.2, label="Total")
g.set(xlabel='Feature Importance', ylabel='Feature', title='Importance of Featur
es in xG Model')
plt.show()
```

```
Predicted goals from test data: 24
xG: 26.50
Actual goals from test set): 27
Difference from my xG value and actual goals =  -0.50

              precision    recall  f1-score   support

           0       0.92      0.93      0.93       301
           1       0.17      0.15      0.16        27

    accuracy                           0.87       328
   macro avg       0.55      0.54      0.54       328
weighted avg       0.86      0.87      0.87       328
```



In [ ]: