Modeling NHL Expected Goals

In [1]:

```
# Standard Packages
import pandas as pd
import numpy as np
import warnings
import re

# Viz Packages
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Modeling Packages
## Modeling Prep
from sklearn.model_selection import train_test_split, cross_val_sco
GridSearchCV, RandomizedSearchCV

## SKLearn Data Prep Modules
from sklearn.preprocessing import StandardScaler, OneHotEncoder, La
PolynomialFeatures, PowerTransformer, Normalizer, MaxAbsScaler

from sklearn.impute import SimpleImputer

## SKLearn Classification Models
from sklearn.linear_model import LogisticRegression, Ridge, Lasso,
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassif
ExtraTreesClassifier, VotingClassifier, StackingRegressor, AdaBoost
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler


## SKLearn Pipeline Setup
from imblearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

## SKLearn Model Optimization
from sklearn.feature_selection import RFE, f_regression

## Boosting
from xgboost import XGBRegressor
from xgboost import XGBClassifier

## SKLearn Metrics
### Classification Scoring/Evaluation
from sklearn.metrics import classification_report, accuracy_score,
ConfusionMatrixDisplay, log_loss, confusion_matrix, RocCurveDisplay
```

In [3]:

```
# Notebook Config
## Suppress Python Warnings (Future, Deprecation)
warnings.filterwarnings("ignore", category= FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning)

## Suppress Pandas Warnings (SettingWithCopy)
pd.options.mode.chained_assignment = None

## Pandas Display Config
pd.options.display.max_columns = None
pd.options.display.width = None

## Display SKLearn estimators as diagrams
from sklearn import set_config
set_config(display= 'diagram')
```

EDA

In [4]:

```
# Use 2021-22 data - most recent full season
s21_shots_df = pd.read_csv('project-data/shots_2021.csv')
s21_shots_df
```

Out[4]:

	shotID	arenaAdjustedShotDistance	arenaAdjustedXCord	arenaAdjustedXCordABS	aren
0	0	42.520583	61.0	61.0	
1	1	30.610456	-65.0	65.0	
2	2	85.381497	-8.0	8.0	
3	3	29.274562	-60.0	60.0	
4	4	26.305893	63.0	63.0	
...	...	...	...	...	
121466	73	30.463092	61.0	61.0	
121467	74	16.278821	73.0	73.0	
121468	75	41.194660	-48.0	48.0	
121469	76	17.000000	72.0	72.0	
121470	77	28.017851	61.0	61.0	

121471 rows × 124 columns

The dataset has 121,471 rows of shot instances and 124 columns

In [5]:
```
# Check the numeric column stats
s21_shots_df.describe()
```

Out[5]:

	shotID	arenaAdjustedShotDistance	arenaAdjustedXCord	arenaAdjustedXCordABS
count	121471.000000	121471.000000	121471.000000	121471.000000
mean	60657.050086	34.290366	-0.733278	60.360037
std	35065.714971	18.798219	63.143212	18.553677
min	0.000000	1.000000	-99.000000	0.000000
25%	30289.500000	18.000000	-64.000000	46.000000
50%	60657.000000	33.000000	-5.000000	63.000000
75%	91024.500000	49.000000	62.000000	76.000000
max	121392.000000	98.412398	99.000000	99.000000

Lots of numeric features of differing units/magnitude - data will require scaling

In [109]:
```
# Check data types and non-null counts
s21_shots_df.info(verbose=True, show_counts=True) # data is rather
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 121471 entries, 0 to 121470
Data columns (total 124 columns):
 #    Column                                                  Non-N
ull Count   Dtype
---   ------                                                  -----
---------   -----
 0    shotID                                                  12147
1 non-null  int64
 1    arenaAdjustedShotDistance                               12147
1 non-null  float64
 2    arenaAdjustedXCord                                      12147
1 non-null  float64
 3    arenaAdjustedXCordABS                                   12147
1 non-null  float64
 4    arenaAdjustedYCord                                      12147
1 non-null  float64
 5    arenaAdjustedYCordAbs                                   12147
1 non-null  float64
```

In [6]:
```
# Check event types to make sure all rows are shots. Valid event va
s21_shots_df['event'].value_counts() # all events are shots
```

Out[6]:
```
SHOT    80180
MISS    32594
GOAL     8697
Name: event, dtype: int64
```

In [7]:
```
# check if postseason included
s21_shots_df['isPlayoffGame'].value_counts() # playoff games are in
```

Out[7]:
```
0    113158
1      8313
Name: isPlayoffGame, dtype: int64
```

In [8]:
```
# Check period column in case shootout attempts are included
s21_shots_df['period'].value_counts()
```

Out[8]:
```
2    41584
1    39317
3    38793
4     1716
5       47
6       14
Name: period, dtype: int64
```

In [9]:
```
# Check how many extra periods are attributable to the post season
len(s21_shots_df.loc[(s21_shots_df['period'] >= 5) & (s21_shots_df[
```

Out[9]:
```
61
```

All instances of a 5th and 6th period are attributable to the post season, so shootouts have already been scrubbed

In [10]:
```
# Summarize missing data
pd.set_option('display.max_rows', 125)
s21_shots_df.isna().sum()
```

Out[10]:
```
shotID                                                  0
arenaAdjustedShotDistance                               0
arenaAdjustedXCord                                      0
arenaAdjustedXCordABS                                   0
arenaAdjustedYCord                                      0
arenaAdjustedYCordAbs                                   0
averageRestDifference                                   0
awayEmptyNet                                            0
awayPenalty1Length                                      0
awayPenalty1TimeLeft                                    0
awaySkatersOnIce                                        0
awayTeamCode                                            0
awayTeamGoals                                           0
defendingTeamAverageTimeOnIce                           0
defendingTeamAverageTimeOnIceOfDefencemen               0
defendingTeamAverageTimeOnIceOfDefencemenSinceFaceoff   0
defendingTeamAverageTimeOnIceOfForwards                 0
defendingTeamAverageTimeOnIceOfForwardsSinceFaceoff     0
defendingTeamAverageTimeOnIceSinceFaceoff               0
```

Features with missing values and counts

- goalieNameForShot 800
  - these probably are not missing, but indicative of empty net situations
- playerPositionThatDidEvent 4
- shooterLeftRight 3
- shooterName 3
- shooterPlayerId 3
- shotType 5

Inspect shotType as other columns will be dropped

In [11]:
```
# Check shotType values
print(s21_shots_df['shotType'].value_counts())
print(s21_shots_df['shotType'].isna().sum())
```

```
WRIST    68003
SLAP     16413
SNAP     15529
BACK      8985
TIP       8404
DEFL      3129
WRAP      1003
Name: shotType, dtype: int64
5
```

In [13]:
```
# Drop the 5 rows missing shot type data
s21_shots_df = s21_shots_df.dropna(subset=['shotType'])
s21_shots_df
```

Out[13]:

	shotID	arenaAdjustedShotDistance	arenaAdjustedXCord	arenaAdjustedXCordABS	aren
0	0	42.520583	61.0	61.0	
1	1	30.610456	-65.0	65.0	
2	2	85.381497	-8.0	8.0	
3	3	29.274562	-60.0	60.0	
4	4	26.305893	63.0	63.0	
...	...	...	...	...	
121466	73	30.463092	61.0	61.0	
121467	74	16.278821	73.0	73.0	
121468	75	41.194660	-48.0	48.0	
121469	76	17.000000	72.0	72.0	
121470	77	28.017851	61.0	61.0	

121466 rows × 124 columns

In [16]:
```
# Check values of the target variable goal
s21_shots_df['goal'].value_counts() # binary with 1 representing a
```

Out[16]:
```
0    112773
1      8693
Name: goal, dtype: int64
```

Our outcomes are very imbalanced. This will have to be addressed with oversampling or subsetting of data.

Visualize Shot Types & Goals

In [27]:
```
# Get normalized target variable ('goal') distribution
s21_shots_df['goal'].value_counts(normalize=True) # very imbalanced
```

Out[27]:
```
0    0.928433
1    0.071567
Name: goal, dtype: float64
```

A model-less prediction (guess) will correctly predict a shot resulting in a goal scored only ~7.16% of the time

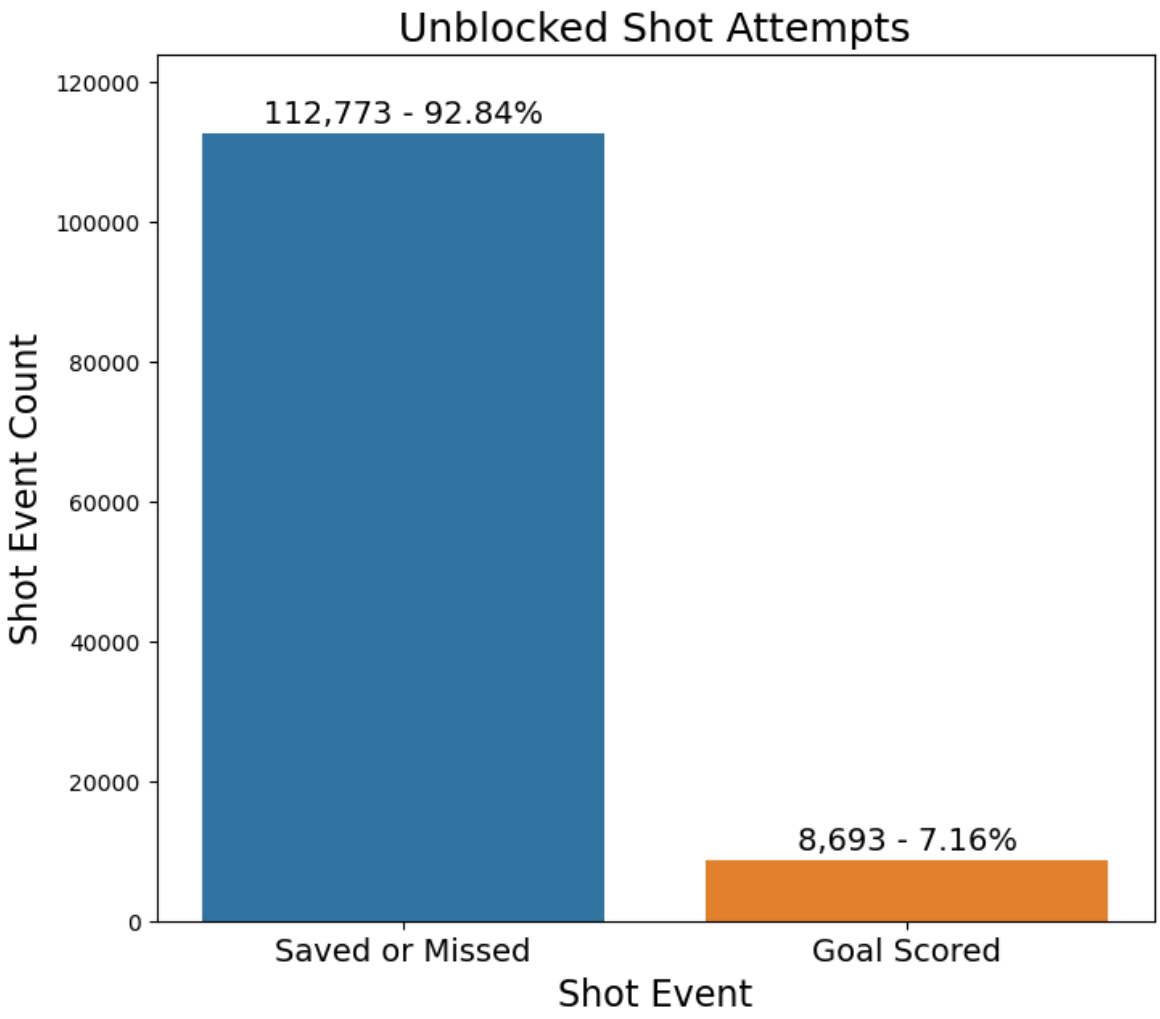
In [40]:
```
# Create goals DF
goals_df = s21_shots_df['goal'].value_counts().rename_axis('Shot Ev
goals_df['Shot Event Count % of Unblocked Shot Attempts'] = goals_d
goals_df['Shot Event'][0] = 'Saved or Missed'
goals_df['Shot Event'][1] = 'Goal Scored'
goals_df
```

Out[40]:

	Shot Event	Shot Event Count	Shot Event Count % of Unblocked Shot Attempts
0	Saved or Missed	112773	0.928433
1	Goal Scored	8693	0.071567

goals_df['Shot Event Count'].astype(str)

In [75]:
```
event_labels = goals_df["Shot Event Count"].map('{:,d}'.format) + '

fig, ax = plt.subplots(figsize=(8, 7))
ax = sns.barplot(x='Shot Event', y='Shot Event Count', data=goals_d
ax.bar_label(ax.containers[0], labels=event_labels, padding=2, size
ax.margins(y=0.1)
ax.margins(x=0.05)
plt.title("Unblocked Shot Attempts", size = 18)
plt.xlabel("Shot Event", size = 16)
plt.xticks(rotation=0, size=14)
plt.ylabel("Shot Event Count", size = 16)
plt.show()
```



In [78]:
```
# Random choice accuracy score
baseline_pred = np.ones(s21_shots_df.shape[0])
accuracy_score(s21_shots_df['goal'],baseline_pred)
```

Out[78]: 0.07156735218085719

In [79]:
```
# Random choice log loss as that is the industry standard metric fo
baseline_probs = np.repeat(s21_shots_df['goal'].value_counts(normal
log_loss(s21_shots_df['goal'], baseline_probs)
```

Out[79]: 0.2576744610404474

In [80]:
```
# Create shots taken by shotType DF
shot_types_df = s21_shots_df['shotType'].value_counts().rename_axis
shot_types_df['Shot Type % of Shots Taken'] = shot_types_df['Shots '
shot_types_df
```

Out[80]:

	Shot Type	Shots Taken	Shot Type % of Shots Taken
0	WRIST	68003	0.559852
1	SLAP	16413	0.135124
2	SNAP	15529	0.127846
3	BACK	8985	0.073971
4	TIP	8404	0.069188
5	DEFL	3129	0.025760
6	WRAP	1003	0.008257

In [81]:
```
# Create df for goal shot types
goals_scored = s21_shots_df['goal'] == 1
goals_by_type = goals_scored.groupby(s21_shots_df['shotType']).sum(
goals_by_type['Shot Type % of Goals Scored'] = goals_by_type['Goals
goals_by_type
```

Out[81]:

	Shot Type	Goals Scored	Shot Type % of Goals Scored
0	BACK	834	0.095939
1	DEFL	280	0.032210
2	SLAP	829	0.095364
3	SNAP	1356	0.155988
4	TIP	804	0.092488
5	WRAP	43	0.004947
6	WRIST	4547	0.523065

In [82]:

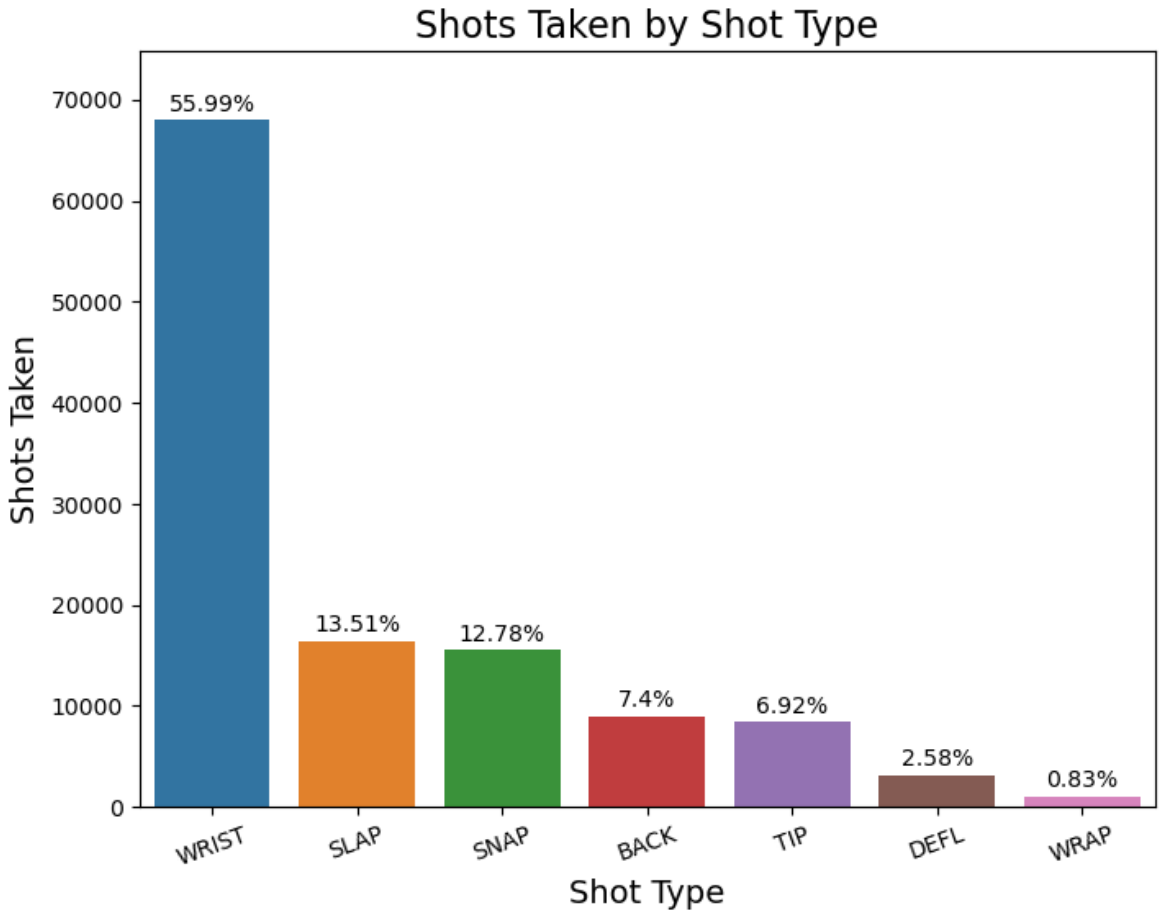
```
# Merge the goals scored df into shots taken
shot_types_df = shot_types_df.merge(goals_by_type, on='Shot Type', 
shot_types_df
```

Out[82]:

	Shot Type	Shots Taken	Shot Type % of Shots Taken	Goals Scored	Shot Type % of Goals Scored
0	WRIST	68003	0.559852	4547	0.523065
1	SLAP	16413	0.135124	829	0.095364
2	SNAP	15529	0.127846	1356	0.155988
3	BACK	8985	0.073971	834	0.095939
4	TIP	8404	0.069188	804	0.092488
5	DEFL	3129	0.025760	280	0.032210
6	WRAP	1003	0.008257	43	0.004947

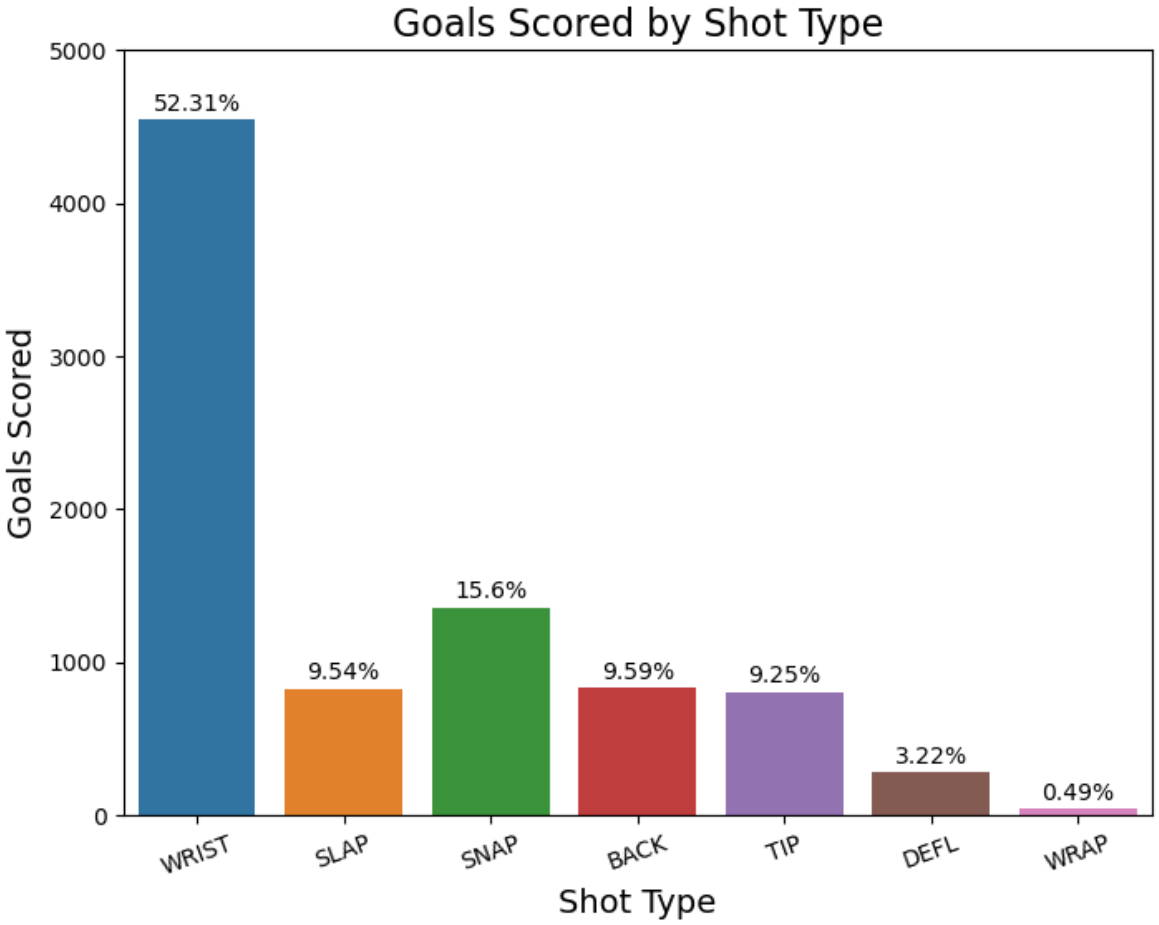
In [83]:

```
shots_labels = np.round((shot_types_df['Shot Type % of Shots Taken'

fig, ax = plt.subplots(figsize=(8, 6))
ax = sns.barplot(x='Shot Type', y='Shots Taken', data=shot_types_df
ax.bar_label(ax.containers[0], labels=shots_labels, padding=2)
ax.margins(y=0.1)
# ax.margins(x=0.05)
plt.title("Shots Taken by Shot Type", size = 16)
plt.xlabel("Shot Type", size = 14)
plt.xticks(rotation=20)
plt.ylabel("Shots Taken", size = 14)
plt.show()
```



In [84]:

```
goals_labels = np.round((shot_types_df['Shot Type % of Goals Scored

fig, ax = plt.subplots(figsize=(8, 6))
ax = sns.barplot(x='Shot Type', y='Goals Scored', data=shot_types_d
ax.bar_label(ax.containers[0], labels=goals_labels, padding=2)
ax.margins(y=0.1)
# ax.margins(x=0.05)
plt.title("Goals Scored by Shot Type", size = 16)
plt.xlabel("Shot Type", size = 14)
plt.xticks(rotation=20)
plt.ylabel("Goals Scored", size = 14)
plt.show()
```



Feature Engineering

In [85]:
```
# Features from dataset to be included model
cols_to_keep = (pd.read_csv('project-data/feature-list.csv', header
cols_to_keep = cols_to_keep[0].to_list()
cols_to_keep
```

Out[85]:
```
['homeSkatersOnIce',
 'awaySkatersOnIce',
 'isHomeTeam',
 'shotType',
 'shotRush',
 'arenaAdjustedShotDistance',
 'arenaAdjustedXCordABS',
 'arenaAdjustedYCordAbs',
 'shotAngleAdjusted',
 'shotAnglePlusRebound',
 'shotAnglePlusReboundSpeed',
 'shotOnEmptyNet',
 'timeSinceLastEvent',
 'distanceFromLastEvent',
 'lastEventxCord_adjusted',
 'lastEventyCord_adjusted',
 'speedFromLastEvent',
 'offWing',
 'goal']
```

In [86]:
```
# New df with unnecessary columns dropped and index reset
shots_df = s21_shots_df[cols_to_keep].reset_index(drop=True)
shots_df
```

Out[86]:

	homeSkatersOnIce	awaySkatersOnIce	isHomeTeam	shotType	shotRush	arenaAdjuste
0	5	5	1.0	WRIST	0	
1	5	5	0.0	WRIST	0	
2	5	5	1.0	WRIST	0	
3	5	5	0.0	WRIST	0	
4	5	5	1.0	WRIST	0	
...	...	...	...	...	...	
121461	5	5	1.0	SNAP	0	
121462	5	5	1.0	TIP	0	
121463	5	5	0.0	SNAP	0	
121464	6	5	1.0	TIP	0	
121465	6	5	1.0	WRIST	0	

121466 rows × 19 columns

Add Game Strength State Variable

Takes the number of players on the ice for each team (homeSkatersOnIce & awaySkatersOnIce) to derive even strength, powerplay, and shorthanded game states

In [87]:
```
# add column for game_strength_state
shots_df['game_strength_state'] = ''
```

In [88]:

```
# Add even strength game state values
shots_df.loc[(shots_df['homeSkatersOnIce'] == 3) & (shots_df['awayS
             'game_strength_state'] = 'EV3'
shots_df.loc[(shots_df['homeSkatersOnIce'] == 4) & (shots_df['awayS
             'game_strength_state'] = 'EV4'
shots_df.loc[(shots_df['homeSkatersOnIce'] == 5) & (shots_df['awayS
             'game_strength_state'] = 'EV5'
shots_df
```

Out[88]:

	homeSkatersOnIce	awaySkatersOnIce	isHomeTeam	shotType	shotRush	arenaAdjuste
0	5	5	1.0	WRIST	0	
1	5	5	0.0	WRIST	0	
2	5	5	1.0	WRIST	0	
3	5	5	0.0	WRIST	0	
4	5	5	1.0	WRIST	0	
...	...	...	...	...	...	
121461	5	5	1.0	SNAP	0	
121462	5	5	1.0	TIP	0	
121463	5	5	0.0	SNAP	0	
121464	6	5	1.0	TIP	0	
121465	6	5	1.0	WRIST	0	

121466 rows × 20 columns

In [89]:

```
# Add Home team game_strength_state
shots_df.loc[(shots_df['isHomeTeam'] == 1) & (shots_df['homeSkaters
       (shots_df['awaySkatersOnIce'] == 4), 'game_strength_state'] =

shots_df.loc[(shots_df['isHomeTeam'] == 1) & (shots_df['homeSkaters
       (shots_df['awaySkatersOnIce'] == 3), 'game_strength_state'] =

shots_df.loc[(shots_df['isHomeTeam'] == 1) & (shots_df['homeSkaters
       (shots_df['awaySkatersOnIce'] == 3), 'game_strength_state'] =

shots_df.loc[(shots_df['isHomeTeam'] == 1) & (shots_df['homeSkaters
       (shots_df['awaySkatersOnIce'] == 5), 'game_strength_state'] =

shots_df.loc[(shots_df['isHomeTeam'] == 1) & (shots_df['homeSkaters
       (shots_df['awaySkatersOnIce'] == 4), 'game_strength_state'] =
```

In [90]:
```
# Add Away team game_strength_state
shots_df.loc[(shots_df['isHomeTeam'] == 0) & (shots_df['homeSkaters(
       (shots_df['awaySkatersOnIce'] == 5), 'game_strength_state'] 

shots_df.loc[(shots_df['isHomeTeam'] == 0) & (shots_df['homeSkaters(
       (shots_df['awaySkatersOnIce'] == 5), 'game_strength_state'] 

shots_df.loc[(shots_df['isHomeTeam'] == 0) & (shots_df['homeSkaters(
       (shots_df['awaySkatersOnIce'] == 4), 'game_strength_state'] 

shots_df.loc[(shots_df['isHomeTeam'] == 0) & (shots_df['homeSkaters(
       (shots_df['awaySkatersOnIce'] == 6), 'game_strength_state'] 

shots_df.loc[(shots_df['isHomeTeam'] == 0) & (shots_df['homeSkaters(
       (shots_df['awaySkatersOnIce'] == 6), 'game_strength_state'] 
```

In [91]:
```
# Add shorthandeed game_strength_state
# Binning all man down strength states into one shorthanded, 'SH', 
shots_df.loc[(shots_df['isHomeTeam'] == 1) & (shots_df['homeSkaters(
             'game_strength_state'] = 'SH'
shots_df.loc[(shots_df['isHomeTeam'] == 0) & (shots_df['awaySkaters(
             'game_strength_state'] = 'SH'
```

In [92]:
```
shots_df
```

Out[92]:

	homeSkatersOnIce	awaySkatersOnIce	isHomeTeam	shotType	shotRush	arenaAdjuste
0	5	5	1.0	WRIST	0	
1	5	5	0.0	WRIST	0	
2	5	5	1.0	WRIST	0	
3	5	5	0.0	WRIST	0	
4	5	5	1.0	WRIST	0	
...	...	...	...	...	...	
121461	5	5	1.0	SNAP	0	
121462	5	5	1.0	TIP	0	
121463	5	5	0.0	SNAP	0	
121464	6	5	1.0	TIP	0	
121465	6	5	1.0	WRIST	0	

121466 rows × 20 columns

In [93]:

```
# Check game_strength_state value counts to ensure proper encoding
print(shots_df['game_strength_state'].value_counts())
# Also check normalized count formatted as a percentage
print(shots_df['game_strength_state'].value_counts(normalize=True).
```

```
EV5        95454
PP_5v4     16058
SH          3900
PP_6v5      2137
EV4         1495
EV3         1322
PP_5v3       535
PP_6v4       294
PP_4v3       263
               8
Name: game_strength_state, dtype: int64
EV5        78.60%
PP_5v4     13.20%
SH          3.20%
PP_6v5      1.80%
EV4         1.20%
EV3         1.10%
PP_5v3      0.40%
PP_6v4      0.20%
PP_4v3      0.20%
            0.00%
Name: game_strength_state, dtype: object
```

8 situations are unencoded. These must be strength state situations such as 6 on 3, which are very rare. These unencoded situations are so infrequent, they make up less than .00% of the data

In [95]:

```
# Given the rarity of 6v3 states, best to drop those 8 rows
shots_df = shots_df.loc[shots_df['game_strength_state'] != '']
```

In [97]:

```
# Drop homeSkatersOnIce & awaySkatersOnIce now that we have derived
shots_df = shots_df.drop(['homeSkatersOnIce', 'awaySkatersOnIce'],
```

```
---------------------------------------------------------------------
------
KeyError                                  Traceback (most recent call
last)
Cell In[97], line 2
      1 # Drop homeSkatersOnIce & awaySkatersOnIce now that we have d
erived strength states
----> 2 shots_df = shots_df.drop(['homeSkatersOnIce', 'awaySkatersOnI
ce'], axis=1)

File ~/miniforge3/envs/learn-env/lib/python3.8/site-packages/pandas/u
til/_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.de
corate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_arg
s)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~/miniforge3/envs/learn-env/lib/python3.8/site-packages/pandas/c
ore/frame.py:5399, in DataFrame.drop(self, labels, axis, index, colum
ns, level, inplace, errors)
   5251 @deprecate_nonkeyword_arguments(version=None, allowed_args=
["self", "labels"])
   5252 def drop(  # type: ignore[override]
   5253     self,
   (...)
   5260     errors: IgnoreRaise = "raise",
   5261 ) -> DataFrame | None:
   5262     """
   5263     Drop specified labels from rows or columns.
   5264
   (...)
   5397             weight  1.0     0.8
   5398     """
-> 5399     return super().drop(
   5400         labels=labels,
   5401         axis=axis,
   5402         index=index,
   5403         columns=columns,
   5404         level=level,
   5405         inplace=inplace,
   5406         errors=errors,
   5407     )

File ~/miniforge3/envs/learn-env/lib/python3.8/site-packages/pandas/u
til/_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.de
corate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_arg
s)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
```

```
    330     )
--> 331 return func(*args, **kwargs)

File ~/miniforge3/envs/learn-env/lib/python3.8/site-packages/pandas/core/generic.py:4505, in NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   4503 for axis, labels in axes.items():
   4504     if labels is not None:
-> 4505         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4507 if inplace:
   4508     self._update_inplace(obj)

File ~/miniforge3/envs/learn-env/lib/python3.8/site-packages/pandas/core/generic.py:4546, in NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)
   4544         new_axis = axis.drop(labels, level=level, errors=errors)
   4545     else:
-> 4546         new_axis = axis.drop(labels, errors=errors)
   4547     indexer = axis.get_indexer(new_axis)
   4549 # Case for non-unique axis
   4550 else:

File ~/miniforge3/envs/learn-env/lib/python3.8/site-packages/pandas/core/indexes/base.py:6934, in Index.drop(self, labels, errors)
   6932 if mask.any():
   6933     if errors != "ignore":
-> 6934         raise KeyError(f"{list(labels[mask])} not found in axis")
   6935     indexer = indexer[~mask]
   6936 return self.delete(indexer)

KeyError: "['homeSkatersOnIce', 'awaySkatersOnIce'] not found in axis"
```

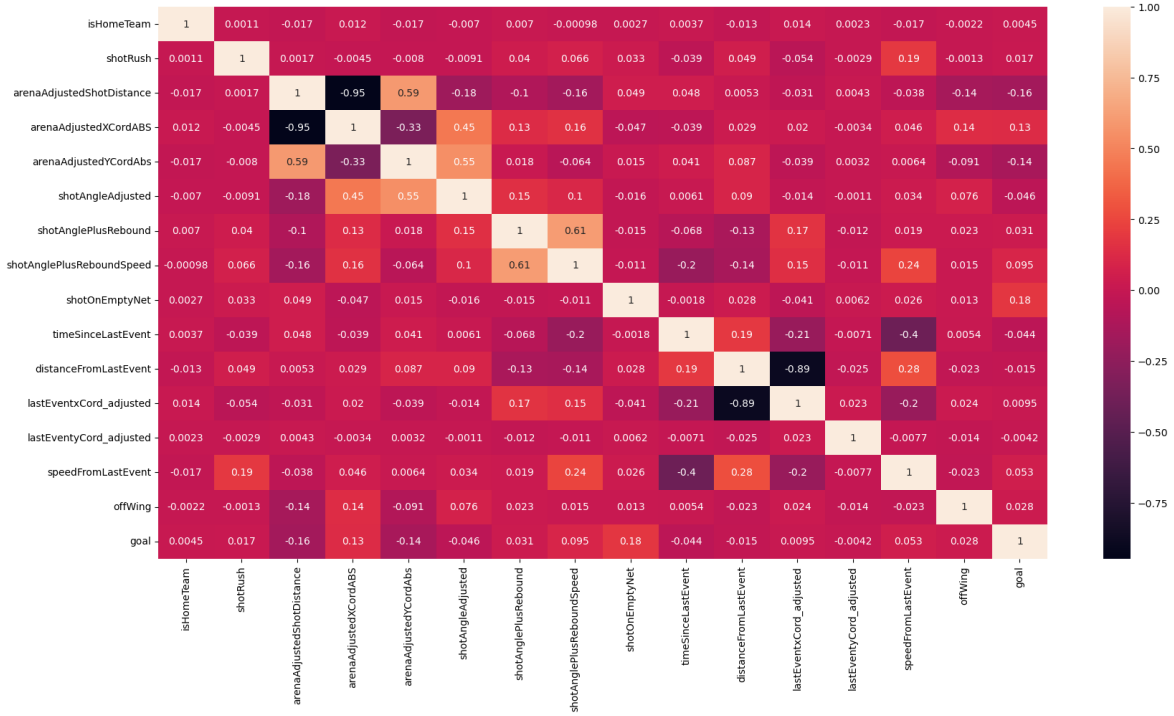
In [98]:

```
# check dtypes and non-null counts of our new df
shots_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 121458 entries, 0 to 121465
Data columns (total 18 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   isHomeTeam                  121458 non-null  float64
 1   shotType                    121458 non-null  object 
 2   shotRush                    121458 non-null  int64  
 3   arenaAdjustedShotDistance   121458 non-null  float64
 4   arenaAdjustedXCordABS       121458 non-null  float64
 5   arenaAdjustedYCordAbs       121458 non-null  float64
 6   shotAngleAdjusted           121458 non-null  float64
 7   shotAnglePlusRebound        121458 non-null  float64
 8   shotAnglePlusReboundSpeed   121458 non-null  float64
 9   shotOnEmptyNet              121458 non-null  int64  
 10  timeSinceLastEvent          121458 non-null  int64  
 11  distanceFromLastEvent       121458 non-null  float64
 12  lastEventxCord_adjusted     121458 non-null  int64  
 13  lastEventyCord_adjusted     121458 non-null  int64  
 14  speedFromLastEvent          121458 non-null  float64
 15  offWing                     121458 non-null  int64  
 16  goal                        121458 non-null  int64  
 17  game_strength_state         121458 non-null  object 
dtypes: float64(9), int64(7), object(2)
memory usage: 17.6+ MB
```

In [99]:

```
# Check correlation of selected features
plt.figure(figsize=(20,10))
cor = shots_df.corr()
sns.heatmap(cor, annot=True)
plt.show()
```



In [100]:

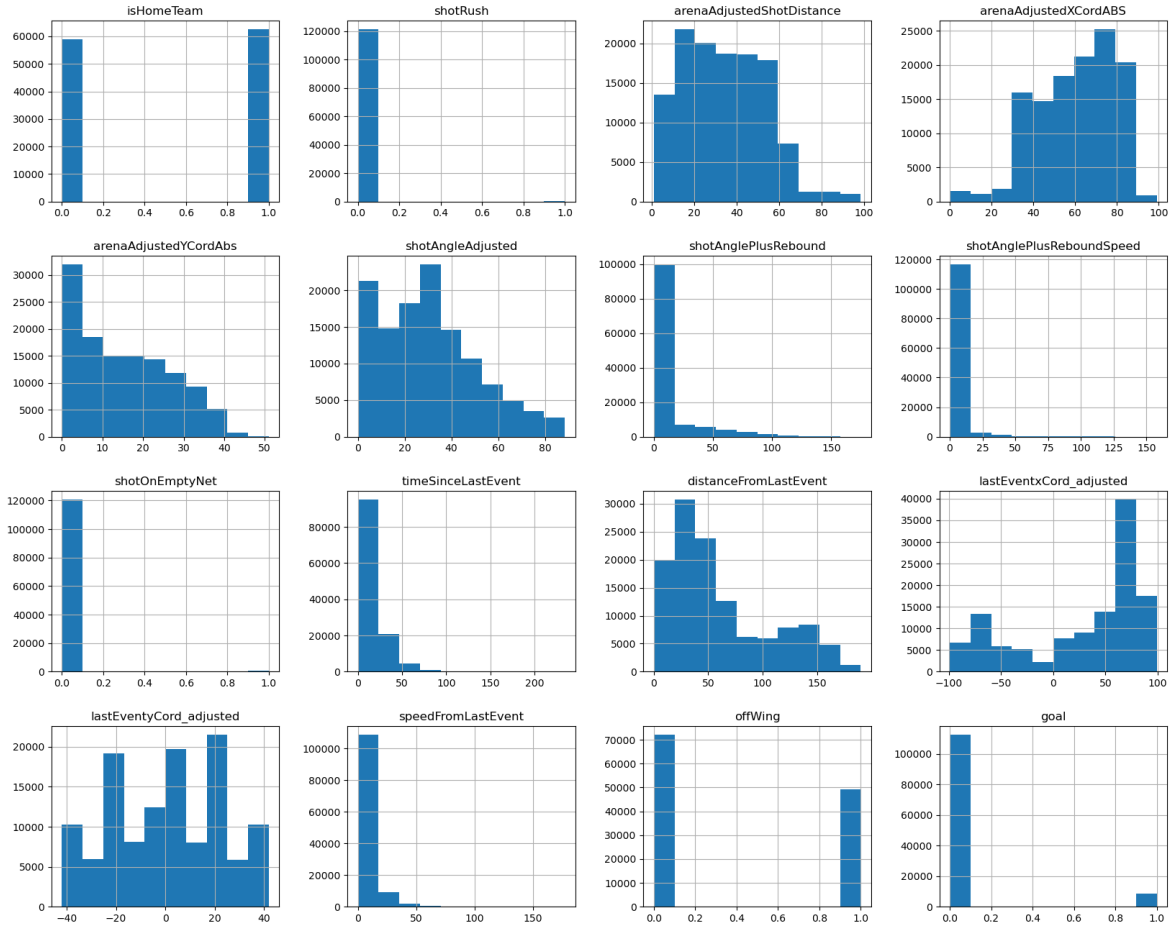
```
# Table of heatmap values
shots_df.corr().sort_values('goal')
```

Out[100]:

	isHomeTeam	shotRush	arenaAdjustedShotDistance	arenaAdjustec
arenaAdjustedShotDistance	-0.016761	0.001695	1.000000	
arenaAdjustedYCordAbs	-0.017398	-0.007993	0.590378	
shotAngleAdjusted	-0.007015	-0.009066	-0.182928	
timeSinceLastEvent	0.003695	-0.039049	0.047815	
distanceFromLastEvent	-0.013339	0.049403	0.005268	
lastEventyCord_adjusted	0.002280	-0.002862	0.004298	
isHomeTeam	1.000000	0.001061	-0.016761	
lastEventxCord_adjusted	0.014344	-0.054039	-0.030524	
shotRush	0.001061	1.000000	0.001695	
offWing	-0.002152	-0.001291	-0.143048	
shotAnglePlusRebound	0.006989	0.039548	-0.100762	
speedFromLastEvent	-0.016954	0.188586	-0.038165	
shotAnglePlusReboundSpeed	-0.000976	0.065869	-0.157389	
arenaAdjustedXCordABS	0.012293	-0.004502	-0.947619	
shotOnEmptyNet	0.002700	0.032686	0.048856	
goal	0.004467	0.016706	-0.159274	

In [101]:

```
# Visualize distributions of numerical features
shots_df.hist(bins=10, figsize=(20, 16))
plt.show()
```



Baseline

Modeling

Build Pipeline

In [102]:

```
# Separate target variable 'goal' from feature set
# and perform a stratified test_train_split due to high imbalance
X = shots_df.drop(['goal'], axis = 1)
y = shots_df['goal']
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify =
```

In [108]:
```
# X_train.shape
y_train.value_counts()
```

Out[108]:
```
0    84574
1     6519
Name: goal, dtype: int64
```

In [103]:
```
# Assign training sets of numeric and categorical columns to respec
num_features = X_train.select_dtypes(['int', 'float']).columns
cat_features = X_train.select_dtypes(['object']).columns
```

In [109]:
```
# Establish pipelines for each feature type
numeric_pipeline = Pipeline([('ss', StandardScaler())])

nominal_pipeline = Pipeline([
    ('onehotenc', OneHotEncoder(handle_unknown = 'ignore')),
    ('onehotnorm', MaxAbsScaler())])

# declare scoring metric list
scoring = ['neg_log_loss', 'accuracy']
```

In [110]:
```
# Instantiate the column transformer
ct = ColumnTransformer(transformers=
    [("numpipe", numeric_pipeline, num_features),
     ("nominalpipe", nominal_pipeline, cat_features)])
```

In [111]:
```
ct
```

Out[111]:
```
ColumnTransformer(transformers=[('numpipe',
                                 Pipeline(steps=[('ss', StandardScale
r())]),
                                 Index(['isHomeTeam', 'shotRush', 'ar
enaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEvent',
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')),
                                ('nominalpipe',
                                 Pipeline(steps=[('onehotenc',
                                                  OneHotEncoder(handl
e_unknown='ignore')),
                                                 ('onehotnorm',
                                                  MaxAbsScaler())]),
                                 Index(['shotType', 'game_strength_st
ate'], dtype='object'))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [112]:
```
num_features
```

Out[112]:
```
Index(['isHomeTeam', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEvent',
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')
```

In [122]:
```
# get feature names after encoding
feature_names = list(nominal_pipeline.named_steps['onehotenc'].fit(
```

In [124]:

```
import re
# removes the OHE strings at front end of feature names
def clean_ohe_names(feature_list):
    extracted_names = []
    for feature in feature_list:
        match = re.search(r'_([^_]*)$', feature)
        if match:
            extracted_names.append(match.group(1))
    return extracted_names
```

In [125]:

```
cleaned_features = clean_ohe_names(feature_names)
num_features_names = list(num_features)
# put into a dataframe
feature_names_df = pd.DataFrame(cleaned_features)
# add the numerical cols at end of dataframe
feature_names_df = feature_names_df.append(num_features_names)
feature_names_df = feature_names_df.reset_index().drop(columns = 'i
```

Basic Logistic Regression Model

In [115]:

```
# build baseline log reg pipepline
steps = [('preprocess', ct),
         ('logreg', LogisticRegression(random_state = 42, max_iter=

base_log_reg_pipeline = Pipeline(steps)

base_log_reg_pipeline.fit(X_train, y_train)

# Predict using the pipeline
base_log_y_pred = base_log_reg_pipeline.predict(X_test)
```

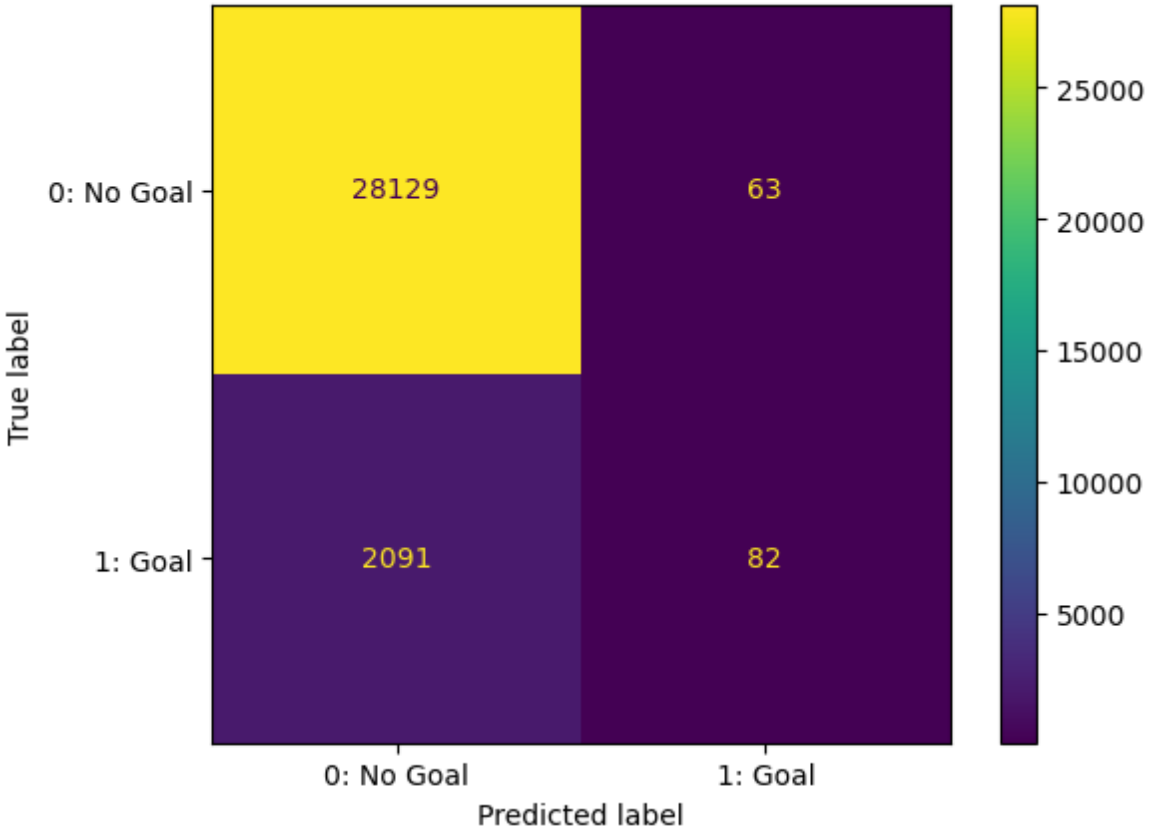
In [117]:

```
# Evaluate the accuracy of the predictions
accuracy = accuracy_score(y_test, base_log_y_pred)
print(f'Test accuracy: {accuracy:.3f}')

# Calculate the F1 score for the test set
f1 = f1_score(y_test, base_log_y_pred,average='macro')
print(f'Test F1 score: {f1:.3f}')

# Calculate the AUC-ROC score for the test set
auc_roc = roc_auc_score(y_test, base_log_reg_pipeline.predict_proba
print(f'Test AUC-ROC score: {auc_roc:.3f}')

# Calculate the Log Loss score for the test set
log_loss_score = log_loss(y_test, base_log_reg_pipeline.predict_pro
print(f'Test log loss score: {log_loss_score:.3f}')
```

```
Test accuracy: 0.929
Test F1 score: 0.517
Test AUC-ROC score: 0.756
Test log loss score: 0.228
```

In [119]:

```
# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
cm = confusion_matrix(y_test, base_log_y_pred)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()

# Calculate and print the classification report
report = classification_report(y_test, base_log_y_pred)
print(f'Classification report:\n{report}')
# Calculate the Log Loss score for the test set
log_loss_score = log_loss(y_test, base_log_reg_pipeline.predict_pro
print(f'Test log loss score: {log_loss_score:.3f}')# Calculate the
# Calculate the AUC-ROC score for the test set
auc_roc = roc_auc_score(y_test, base_log_reg_pipeline.predict_proba
print(f'Test AUC-ROC score: {auc_roc:.3f}')
```



Classification report:

	precision	recall	f1-score	support
0	0.93	1.00	0.96	28192
1	0.57	0.04	0.07	2173
accuracy			0.93	30365
macro avg	0.75	0.52	0.52	30365
weighted avg	0.90	0.93	0.90	30365

Test log loss score: 0.228
Test AUC-ROC score: 0.756

Basic Random Forrest Model

In [120]:
```
# build baseline trees pipeplin
steps = [('preprocess', ct),
         ('rf', RandomForestClassifier(random_state = 42))]

rf_pipeline = Pipeline(steps)

rf_pipeline.fit(X_train, y_train)
```

Out[120]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('numpipe',
                                                  Pipeline(steps=[('s
s',
                                                                  St
andardScaler())]),
                                                  Index(['isHomeTea
m', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEvent',
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')),
                                                 ('nominalpipe',
                                                  Pipeline(steps=[('o
nehotenc',
                                                                  On
eHotEncoder(handle_unknown='ignore')),
                                                                 ('o
nehotnorm',
                                                                  Ma
xAbsScaler())]),
                                                  Index(['shotType',
'game_strength_state'], dtype='object'))])),
                ('rf', RandomForestClassifier(random_state=42))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [126]:

```
pd.DataFrame(zip(feature_names_df[0].values,
                 rf_pipeline[1].feature_importances_)).sort_values(
```

Out[126]:

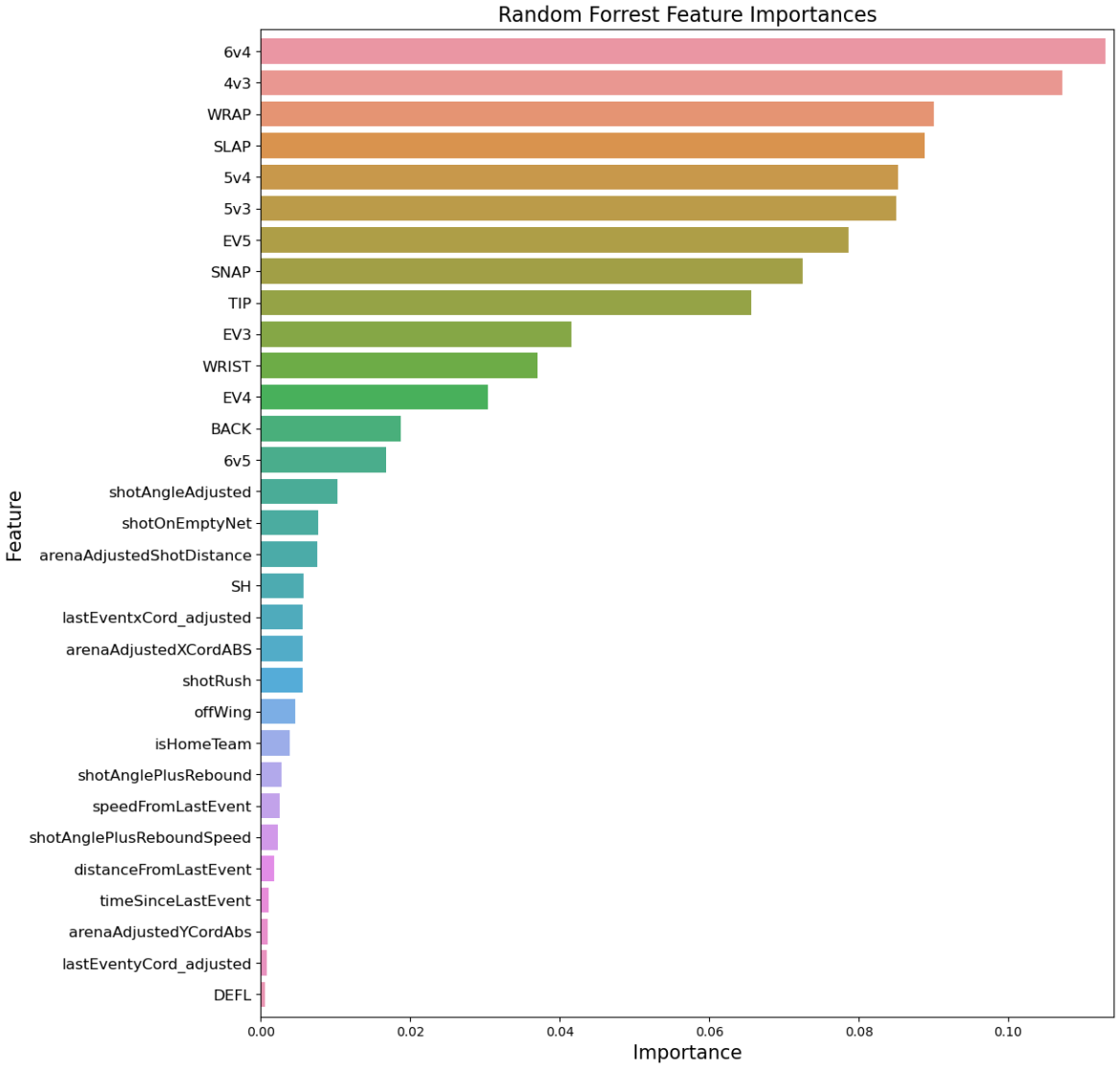
	0	1
13	6v4	0.113028
10	4v3	0.107194
5	WRAP	0.090000
2	SLAP	0.088823
12	5v4	0.085278

In [154]:

```
# create df for viz
feat_importances = pd.DataFrame(columns=['Feature','Importance'])
feat_importances['Feature'] = feature_names_df[0].values
feat_importances['Importance'] = rf_pipeline[1].feature_importances_
feat_importances.sort_values('Importance',inplace=True, ascending=F
```

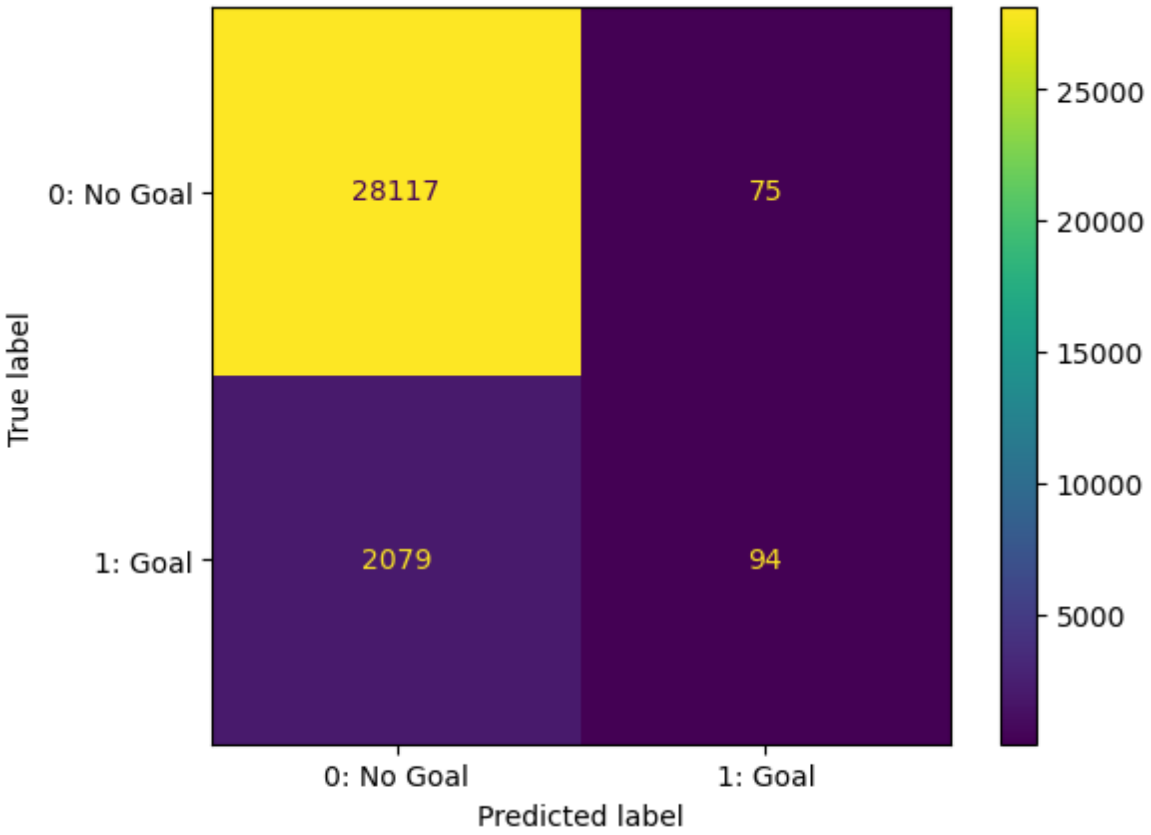
In [181]:

```
fig, ax = plt.subplots(figsize=(12, 14))
ax = sns.barplot(x='Importance', y='Feature', data=feat_importances
# ax.bar_label(ax.containers[0], padding=2)
ax.margins(y=0.01)
ax.margins(x=0.01)
plt.title("Random Forrest Feature Importances", size = 16)
plt.xlabel("Importance", size = 15)
plt.xticks(rotation=0)
plt.ylabel("Feature", size = 15)
plt.yticks(rotation=0, size=12)
plt.show()
```



In [179]:

```
# Predict using the pipeline
y_pred_rf = rf_pipeline.predict(X_test)

# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
cm = confusion_matrix(y_test, y_pred_rf)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()

# Calculate and print the classification report
report = classification_report(y_test, y_pred_rf)
print(f'Classification report:\n{report}')
# Calculate the AUC-ROC score for the test set
auc_roc = roc_auc_score(y_test, rf_pipeline.predict_proba(X_test)[:
print(f'Test AUC-ROC score: {auc_roc:.3f}')
# Calculate the Log-Loss score for the test set
log_loss_score = log_loss(y_test, rf_pipeline.predict_proba(X_test)
print(f'Test log loss score: {log_loss_score:.3f}')
```



```
Classification report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     28192
           1       0.56      0.04      0.08      2173

    accuracy                           0.93     30365
   macro avg       0.74      0.52      0.52     30365
weighted avg       0.90      0.93      0.90     30365

Test AUC-ROC score: 0.737
Test log loss score: 0.287
```

Adding SMOTE to Handle Data Imbalance

In [184]:

```
# RandomForestClassifier with smote
steps = [('preprocess', ct),
    ('smote', SMOTE(sampling_strategy='minority')),
#           ('random', RandomOverSampler()),
         ('rf_clf', RandomForestClassifier())]

rf_clf_smote = Pipeline(steps)

rf_clf_smote.fit(X_train,y_train)
print('test: {}'.format(rf_clf_smote.score(X_test,y_test)))
print('train: {}'.format(rf_clf_smote.score(X_train,y_train)))
```

```
test: 0.9130577968055327
train: 0.9999780444161461
```

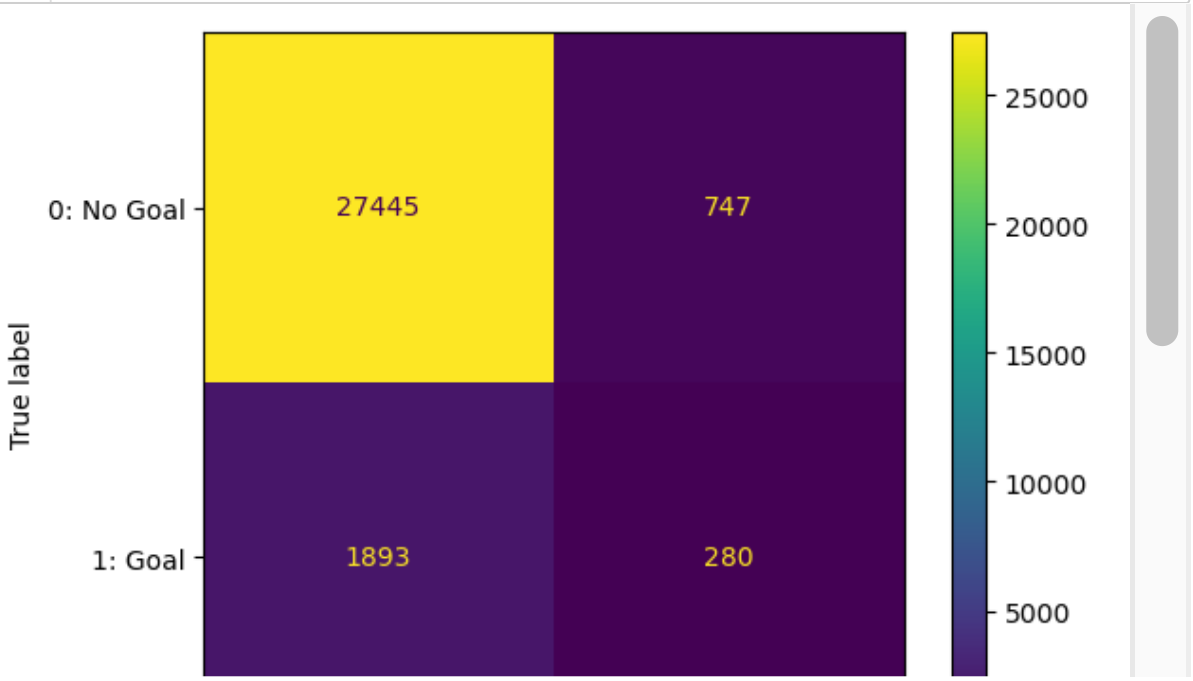
In [186]:
```
1 rf_clf_smote
```

Out[186]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('numpipe',
                                                  Pipeline(steps=[('s
s',
                                                                   St
andardScaler())]),
                                                  Index(['isHomeTea
m', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEv...
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')),
                                                 ('nominalpipe',
                                                  Pipeline(steps=[('o
nehotenc',
                                                                   On
eHotEncoder(handle_unknown='ignore')),
                                                                  ('o
nehotnorm',
                                                                   Ma
xAbsScaler())]),
                                                  Index(['shotType',
'game_strength_state'], dtype='object'))])),
                ('smote', SMOTE(sampling_strategy='minority')),
                ('rf_clf', RandomForestClassifier())])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [185]:

```
# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
y_pred_clf_smote = rf_clf_smote.predict(X_test)
cm = confusion_matrix(y_test, y_pred_clf_smote)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()

# Calculate and print the classification report
report = classification_report(y_test, y_pred_clf_smote)
print(f'Classification report:\n{report}')
# Calculate the AUC-ROC score for the test set
auc_roc = roc_auc_score(y_test, rf_clf_smote.predict_proba(X_test)[
print(f'Test AUC-ROC score: {auc_roc:.3f}')
# Calculate the Log Loss score for the test set
log_loss_score = log_loss(y_test, rf_clf_smote.predict_proba(X_test
print(f'Test log loss score: {log_loss_score:.3f}')
```



Cross Validated Logistic Regression with SMOTE

In [194]:

```
# Logistic Regression CV
steps = [('preprocess', ct),
        ('smote', SMOTE(sampling_strategy='minority')),
         ('logisticregression', LogisticRegression(max_iter = 10000

log_cv_pipeline = Pipeline(steps=steps)

#paramters to test with the grid search
log_params = {'logisticregression__solver' : ['saga','lbfgs'],
                'logisticregression__penalty': [None, 'l2'],
                'logisticregression__C': [.05, 0.1, 1]}

log_cv = GridSearchCV(log_cv_pipeline, param_grid=log_params, cv=5,

# log_cv = GridSearchCV(log_cv_pipeline, param_grid=log_params, cv=
#                       refit = 'neg_log_loss', verbose=2, error_sc
```

In [195]:
```
1 log_cv.fit(X_train, y_train)
```

Out[195]:
```
GridSearchCV(cv=5, error_score='raise',
             estimator=Pipeline(steps=[('preprocess',
                                        ColumnTransformer(transformer
s=[('numpipe',

Pipeline(steps=[('ss',

StandardScaler())]),

Index(['isHomeTeam', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnE
m...

OneHotEncoder(handle_unknown='ignore')),

('onehotnorm',

MaxAbsScaler())]),

Index(['shotType', 'game_strength_state'], dtype='object'))])),
                                       ('smote',
                                        SMOTE(sampling_strategy='mino
rity')),
                                       ('logisticregression',
                                        LogisticRegression(max_iter=1
0000))]),
             param_grid={'logisticregression__C': [0.05, 0.1, 1],
                         'logisticregression__penalty': [None, 'l2'],
                         'logisticregression__solver': ['saga', 'lbfg
s']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [196]:
```
# print best params
log_cv.best_params_

## cache for comp
# {'logisticregression__C': 1,
#  'logisticregression__penalty': None,
#  'logisticregression__solver': 'lbfgs'}
```

Out[196]:
```
{'logisticregression__C': 0.05,
 'logisticregression__penalty': 'l2',
 'logisticregression__solver': 'lbfgs'}
```

In [197]:
```
# assign params to best_model
best_model = log_cv.best_estimator_.get_params()['logisticregressio
```

Add best params to base Log Red Model for comp

In [229]:
```
# Log Best Params w/o smote
steps = [('preprocess', ct),
#         ('smote', SMOTE(sampling_strategy='minority')),
        ('best_model', best_model)]

log_best_pipeline = Pipeline(steps=steps)
log_best_pipeline.fit(X_train, y_train)
```

Out[229]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('numpipe',
                                                  Pipeline(steps=[('s
s',
                                                                  St
andardScaler())]),
                                                  Index(['isHomeTea
m', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEvent',
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')),
                                                 ('nominalpipe',
                                                  Pipeline(steps=[('o
nehotenc',
                                                                  On
eHotEncoder(handle_unknown='ignore')),
                                                                 ('o
nehotnorm',
                                                                  Ma
xAbsScaler())]),
                                                  Index(['shotType',
'game_strength_state'], dtype='object'))])),
                ('best_model', LogisticRegression(C=0.05, max_iter=10
000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

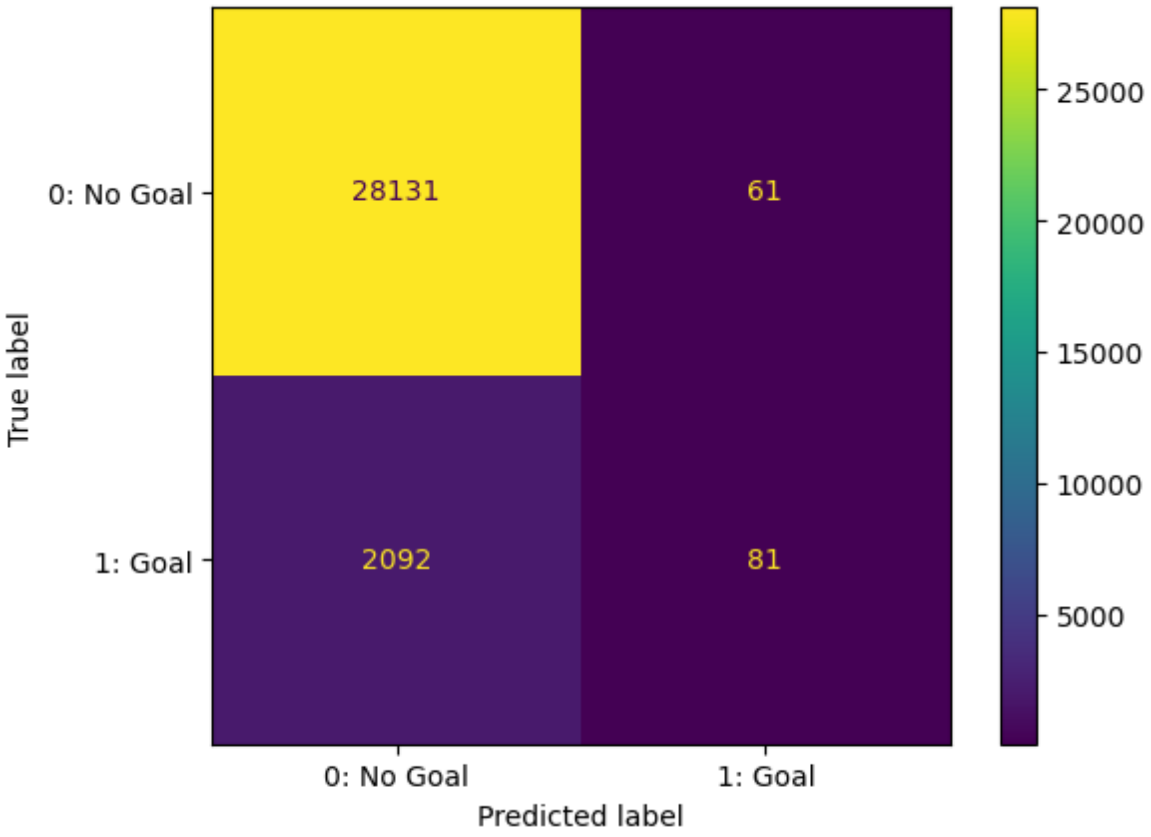
In [230]:
```
1 log_best_pipeline
```

Out[230]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('numpipe',
                                                  Pipeline(steps=[('s
s',
                                                                   St
andardScaler())]),
                                                  Index(['isHomeTea
m', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEvent',
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')),
                                                 ('nominalpipe',
                                                  Pipeline(steps=[('o
nehotenc',
                                                                   On
eHotEncoder(handle_unknown='ignore')),
                                                                  ('o
nehotnorm',
                                                                   Ma
xAbsScaler())]),
                                                  Index(['shotType',
'game_strength_state'], dtype='object'))])),
                ('best_model', LogisticRegression(C=0.05, max_iter=10
000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [232]:

```
# Predict using the pipeline
y_pred_log_cv = log_best_pipeline.predict(X_test)

# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
cm = confusion_matrix(y_test, y_pred_log_cv)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()

# Calculate and print the classification report
report = classification_report(y_test, y_pred_log_cv)
print(f'Classification report:\n{report}')
# Calculate the AUC-ROC score for the test set
log_best_auc_roc = roc_auc_score(y_test, log_best_pipeline.predict_
print(f'Test AUC-ROC score: {log_best_auc_roc:.3f}')
log_best_log_loss = log_loss(y_test, log_best_pipeline.predict_prob
print(f'Test log loss score: {log_best_log_loss:.3f}')
```



Classification report:

	precision	recall	f1-score	support
0	0.93	1.00	0.96	28192
1	0.57	0.04	0.07	2173
accuracy			0.93	30365
macro avg	0.75	0.52	0.52	30365
weighted avg	0.90	0.93	0.90	30365

Test AUC-ROC score: 0.755
Test log loss score: 0.228

Visualize CM for best Log Reg Model with SMOTE

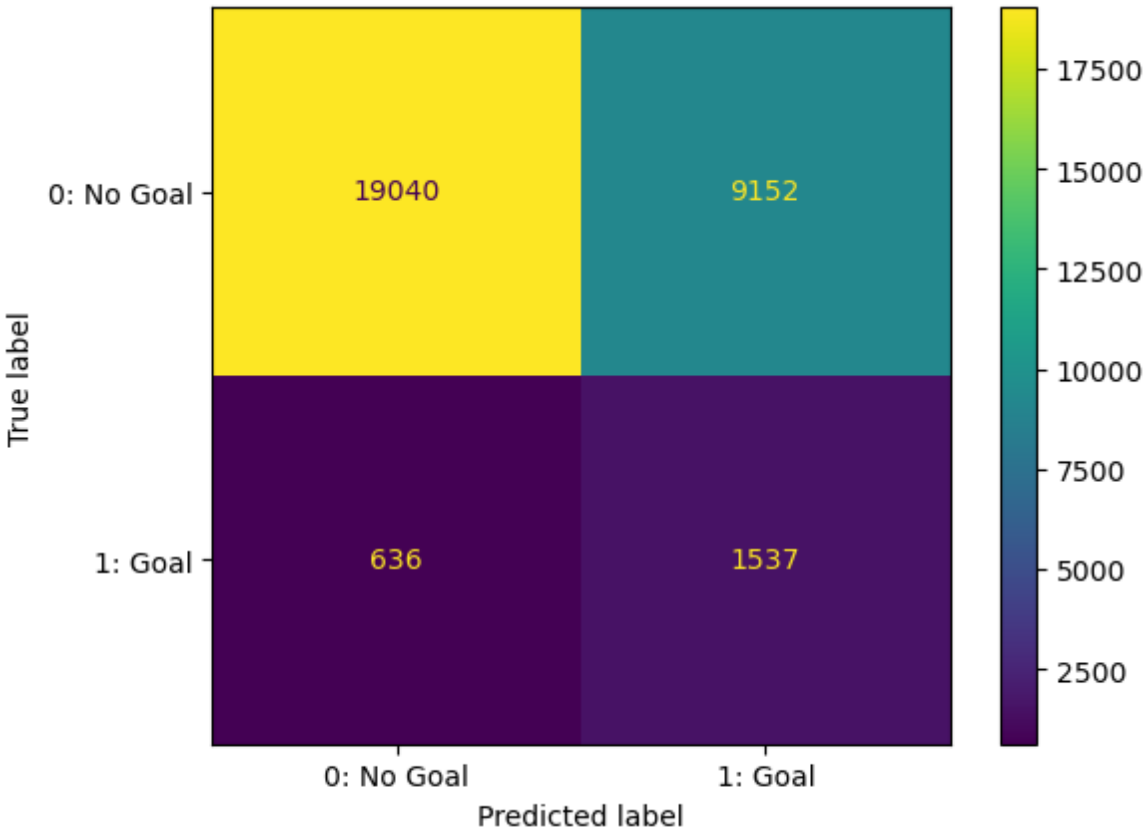
In [227]:
```
# Log Best Params w/ smote
steps = [('preprocess', ct),
        ('smote', SMOTE(sampling_strategy='minority')),
        ('best_model', best_model)]

log_best_pipeline_smote = Pipeline(steps=steps)
log_best_pipeline_smote.fit(X_train, y_train)
```

Out[227]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('numpipe',
                                                  Pipeline(steps=[('s
s',
                                                                  St
andardScaler())]),
                                                  Index(['isHomeTea
m', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEv...
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')),
                                                 ('nominalpipe',
                                                  Pipeline(steps=[('o
nehotenc',
                                                                  On
eHotEncoder(handle_unknown='ignore')),
                                                                 ('o
nehotnorm',
                                                                  Ma
xAbsScaler())]),
                                                  Index(['shotType',
'game_strength_state'], dtype='object'))])),
                ('smote', SMOTE(sampling_strategy='minority')),
                ('best_model', LogisticRegression(C=0.05, max_iter=10
000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [228]:

```
# with smote in steps
# Predict using the pipeline
y_pred_log_cv_smote = log_best_pipeline_smote.predict(X_test)

# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
cm = confusion_matrix(y_test, y_pred_log_cv_smote)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()

# Calculate and print the classification report
report = classification_report(y_test, y_pred_log_cv_smote)
print(f'Classification report:\n{report}')
# Calculate the AUC-ROC score for the test set
log_smote_best_auc_roc = roc_auc_score(y_test, log_best_pipeline_sm
print(f'Test AUC-ROC score: {log_smote_best_auc_roc:.3f}')
# Calculate the AUC-ROC score for the test set
log_smote_best_log_loss = log_loss(y_test, log_best_pipeline_smote.
print(f'Test log loss score: {log_smote_best_log_loss:.3f}')
```



Classification report:

	precision	recall	f1-score	support
0	0.97	0.68	0.80	28192
1	0.14	0.71	0.24	2173
accuracy			0.68	30365
macro avg	0.56	0.69	0.52	30365
weighted avg	0.91	0.68	0.76	30365

Test AUC-ROC score: 0.755
Test log loss score: 0.583

XGBoost

In [225]:

```
# XGBoost CV with smote
steps = [('preprocess', ct),
        ('smote', SMOTE(sampling_strategy='minority')),
#          ('random', RandomOverSampler()),
         ('gradient_booster', XGBClassifier())]

gb_pipeline = Pipeline(steps)

gb_params = {'gradient_booster__n_estimators': [50, 100, 250, 500],
             'gradient_booster__learning_rate': [.001, .01, .1, 1],
             'gradient_booster__max_depth': [3, 4, 5, 6]
            }

gb_cv = GridSearchCV(gb_pipeline, param_grid=gb_params, cv=2, verbo

gb_cv.fit(X_train,y_train)
# print('test: {}'.format(gb_cv.score(X_test,y_test)))
# print('train: {}'.format(gb_cv.score(X_train,y_train)))
```

```
11438.29s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.34s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.34s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.34s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.36s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.37s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.38s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.38s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
11438.39s - pydevd: Sending message related to process being replaced
timed-out after 5 seconds
```

Out[225]:
```
GridSearchCV(cv=2, error_score='raise',
             estimator=Pipeline(steps=[('preprocess',
                                        ColumnTransformer(transformer
s=[('numpipe',

Pipeline(steps=[('ss',

StandardScaler())]),

Index(['isHomeTeam', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnE
m...
                                                    max_depth=None,
                                                    max_leaves=Non
e,
                                                    min_child_weigh
t=None,
                                                    missing=nan,
                                                    monotone_constr
aints=None,
                                                    n_estimators=10
0,
                                                    n_jobs=None,
                                                    num_parallel_tr
ee=None,
                                                    predictor=None,
                                                    random_state=No
ne, ...))]),
             n_jobs=-1,
             param_grid={'gradient_booster__learning_rate': [0.001,
0.01, 0.1,
                                                             1],
                         'gradient_booster__max_depth': [3, 4, 5, 6],
                         'gradient_booster__n_estimators': [50, 100,
250, 500]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [233]:
```
# get best parameters
print("Best parameters:", gb_cv.best_params_)
```

```
Best parameters: {'gradient_booster__learning_rate': 0.1, 'gradient_b
ooster__max_depth': 5, 'gradient_booster__n_estimators': 500}
```

In [234]:
```
# assign to cross validated model params to best_boost
best_boost = gb_cv.best_estimator_.get_params()['gradient_booster']
```

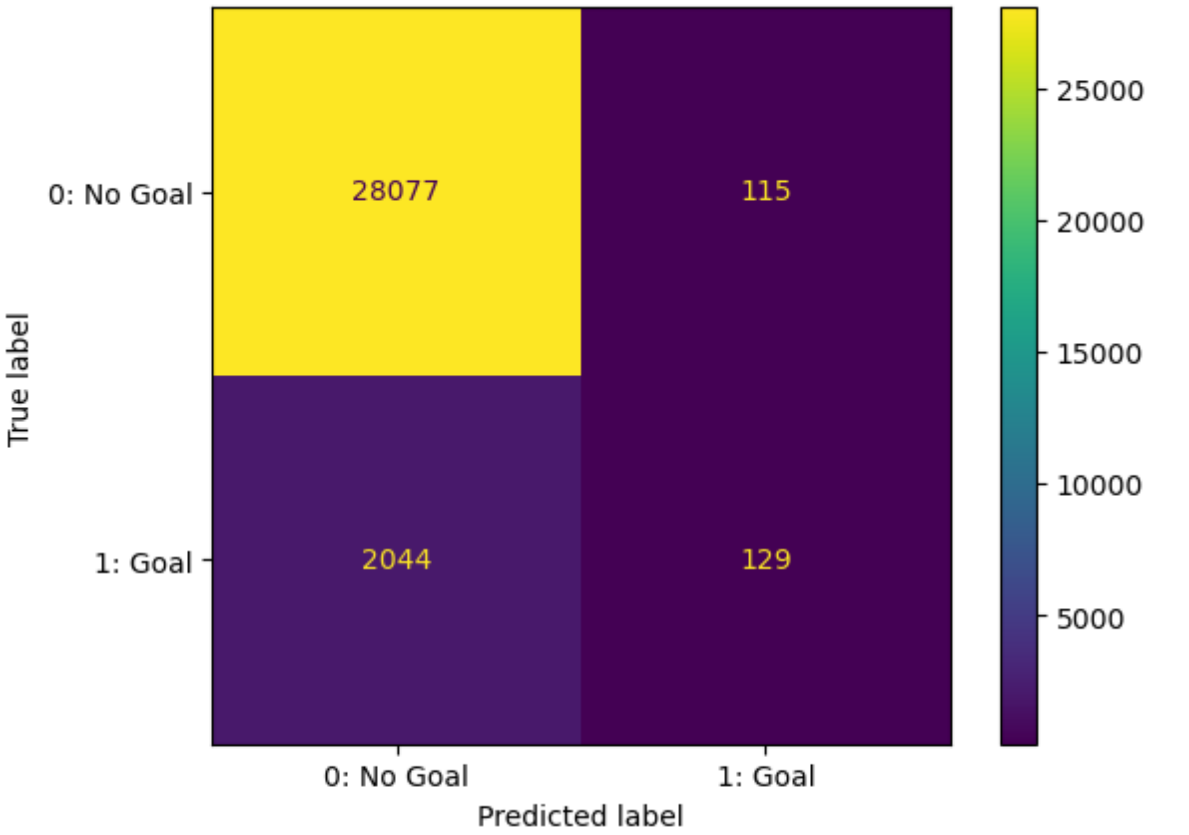
In [235]:
```
# XGBoost Best Params
steps = [('preprocess', ct),
         ('smote', SMOTE(sampling_strategy='minority')),
         ('best_boost', best_boost)]

gb_best_pipeline = Pipeline(steps=steps)
gb_best_pipeline.fit(X_train, y_train)
```

Out[235]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('numpipe',
                                                  Pipeline(steps=[('s
s',
                                                                  St
andardScaler())]),
                                                  Index(['isHomeTea
m', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEv...
                                 feature_types=None, gamma=0, gpu_id=-
1,
                                 grow_policy='depthwise', importance_ty
pe=None,
                                 interaction_constraints='', learning_r
ate=0.1,
                                 max_bin=256, max_cat_threshold=64,
                                 max_cat_to_onehot=4, max_delta_step=0,
                                 max_depth=5, max_leaves=0, min_child_w
eight=1,
                                 missing=nan, monotone_constraints
='()',
                                 n_estimators=500, n_jobs=0, num_parall
el_tree=1,
                                 predictor='auto', random_state=0,
...))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [236]:

```
# Predict using the pipeline
y_pred_gb = gb_best_pipeline.predict(X_test)

# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
cm = confusion_matrix(y_test, y_pred_gb)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()

# Calculate and print the classification report
report = classification_report(y_test, y_pred_gb)
print(f'Classification report:\n{report}')
# Calculate the AUC-ROC score for the test set
auc_roc = roc_auc_score(y_test, gb_best_pipeline.predict_proba(X_te
print(f'Test AUC-ROC score: {auc_roc:.3f}')
# Calculate the Log Loss score for the test set
log_loss_score = log_loss(y_test, gb_best_pipeline.predict_proba(X_
print(f'Test log loss score: {log_loss_score:.3f}')# Calculate the
```



```
Classification report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     28192
           1       0.53      0.06      0.11      2173

    accuracy                           0.93     30365
   macro avg       0.73      0.53      0.53     30365
weighted avg       0.90      0.93      0.90     30365

Test AUC-ROC score: 0.739
Test log loss score: 0.237
```

Create subset

In [237]:
```
# check goal values
shots_df['goal'].value_counts() # binary with 1 representing a goal
```

Out[237]:
```
0    112766
1      8692
Name: goal, dtype: int64
```

In [241]:
```
# Take subset of majority class
goals_df = shots_df.loc[shots_df['goal'] == 1]

no_goals = shots_df.loc[shots_df['goal'] == 0].sample(shots_df['goa
df_sample = pd.concat([no_goals.reset_index(), goals_df.reset_index
df_sample.shape
```

Out[241]:
```
(17384, 19)
```

In [242]:
```
df_sample= df_sample.drop('index',1)
df_sample.head()
```

Out[242]:

	isHomeTeam	shotType	shotRush	arenaAdjustedShotDistance	arenaAdjustedXCordABS	aren
0	0.0	WRIST	0	62.000000	29.0	
1	1.0	WRIST	0	8.246211	81.0	
2	1.0	WRIST	0	12.165525	77.0	
3	1.0	BACK	0	6.324555	83.0	
4	1.0	WRIST	0	86.000000	11.0	

In [243]:
```
X_samp = df_sample.drop('goal',1)
y_samp = df_sample['goal']
```

In [244]:
```
X_train_sample,X_test_sample, y_train_sample, y_test_sample = train
```

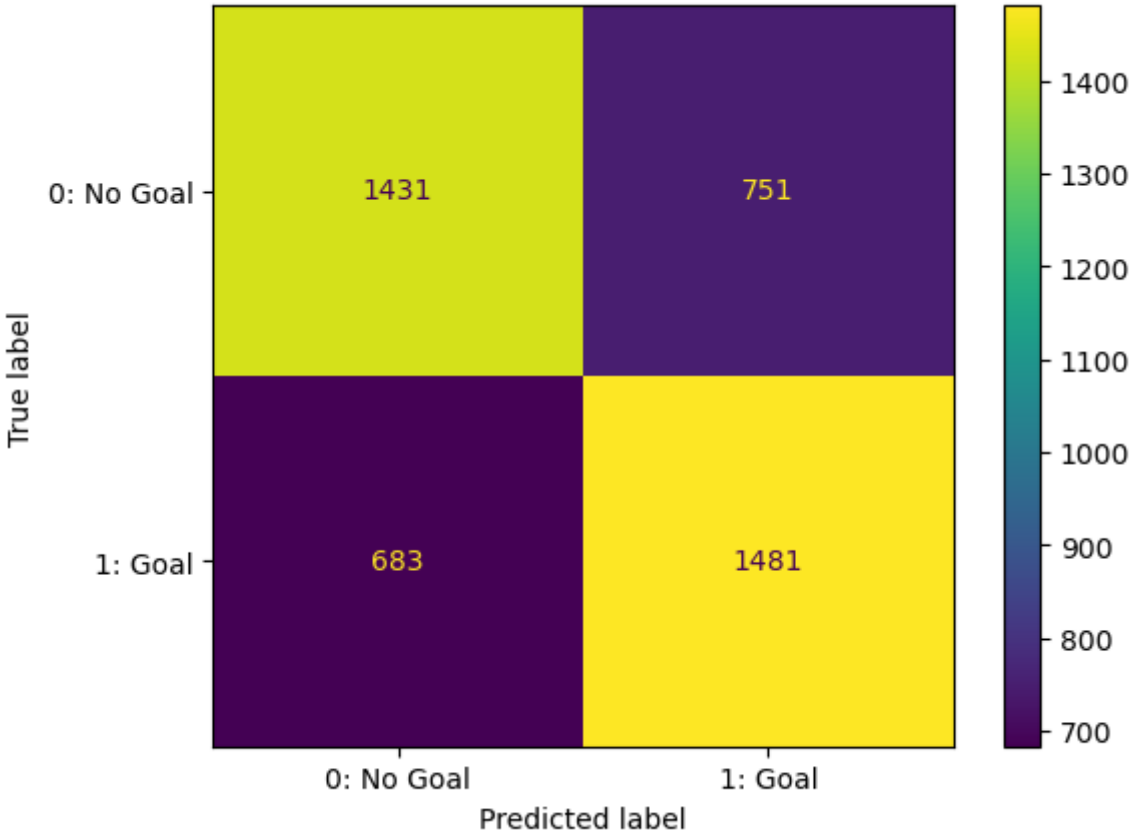
RandomForestClassifier on dataset sample

In [245]:
```
# RandomForestClassifier
steps = [('preprocess', ct),
         ('rf_clf', RandomForestClassifier())]

rf_clf_samp = Pipeline(steps)

rf_clf_samp.fit(X_train_sample,y_train_sample)
print('test: {}'.format(rf_clf_samp.score(X_test_sample,y_test_samp
print('train: {}'.format(rf_clf_samp.score(X_train_sample,y_train_s
```

```
test: 0.670041417395306
train: 1.0
```

In [246]:

```
# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
cm = confusion_matrix(y_test_sample, rf_clf_samp.predict(X_test_sam

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()

# Calculate and print the classification report
report = classification_report(y_test_sample, rf_clf_samp.predict(X
print(f'Classification report:\n{report}')
# Calculate the AUC-ROC score for the test set
auc_roc = roc_auc_score(y_test_sample, rf_clf_samp.predict_proba(X_
print(f'Test AUC-ROC score: {auc_roc:.3f}')
# Calculate the Log Loss score for the test set
log_loss_score = log_loss(y_test_sample, rf_clf_samp.predict_proba(
print(f'Test log loss score: {log_loss_score:.3f}')# Calculate the
```



```
Classification report:
              precision    recall  f1-score   support

           0       0.68      0.66      0.67      2182
           1       0.66      0.68      0.67      2164

    accuracy                           0.67      4346
   macro avg       0.67      0.67      0.67      4346
weighted avg       0.67      0.67      0.67      4346

Test AUC-ROC score: 0.737
Test log loss score: 0.601
```

Logistic Regression CV on Data subset

In [247]:

```
# Log CV
steps = [('preprocess', ct),
         ('logisticregression', LogisticRegression(max_iter = 10000
sampled_log_cv_pipeline = Pipeline(steps=steps)

#paramters to test with the grid search
log_params = {'logisticregression__solver' : ['saga','lbfgs'],
                'logisticregression__penalty': [None,'l2'],
                'logisticregression__C': [.05, 0.1,1]}

sampled_log_cv = GridSearchCV(sampled_log_cv_pipeline, param_grid=l
```

In [248]:
```
sampled_log_cv.fit(X_train_sample, y_train_sample)
```

Out[248]:
```
GridSearchCV(cv=5, error_score='raise',
             estimator=Pipeline(steps=[('preprocess',
                                        ColumnTransformer(transformer
s=[('numpipe',

Pipeline(steps=[('ss',

StandardScaler())]),

Index(['isHomeTeam', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnE
m...

Pipeline(steps=[('onehotenc',

OneHotEncoder(handle_unknown='ignore')),

('onehotnorm',

MaxAbsScaler())]),

Index(['shotType', 'game_strength_state'], dtype='object'))])),
                                       ('logisticregression',
                                        LogisticRegression(max_iter=1
0000))]),
             param_grid={'logisticregression__C': [0.05, 0.1, 1],
                         'logisticregression__penalty': [None, 'l2'],
                         'logisticregression__solver': ['saga', 'lbfg
s']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [249]:
```
# print best params
sampled_log_cv.best_params_
```

Out[249]:
```
{'logisticregression__C': 1,
 'logisticregression__penalty': 'l2',
 'logisticregression__solver': 'saga'}
```

In [251]:
```
# assign params to best_model
sampled_best_model = sampled_log_cv.best_estimator_.get_params()['l
```

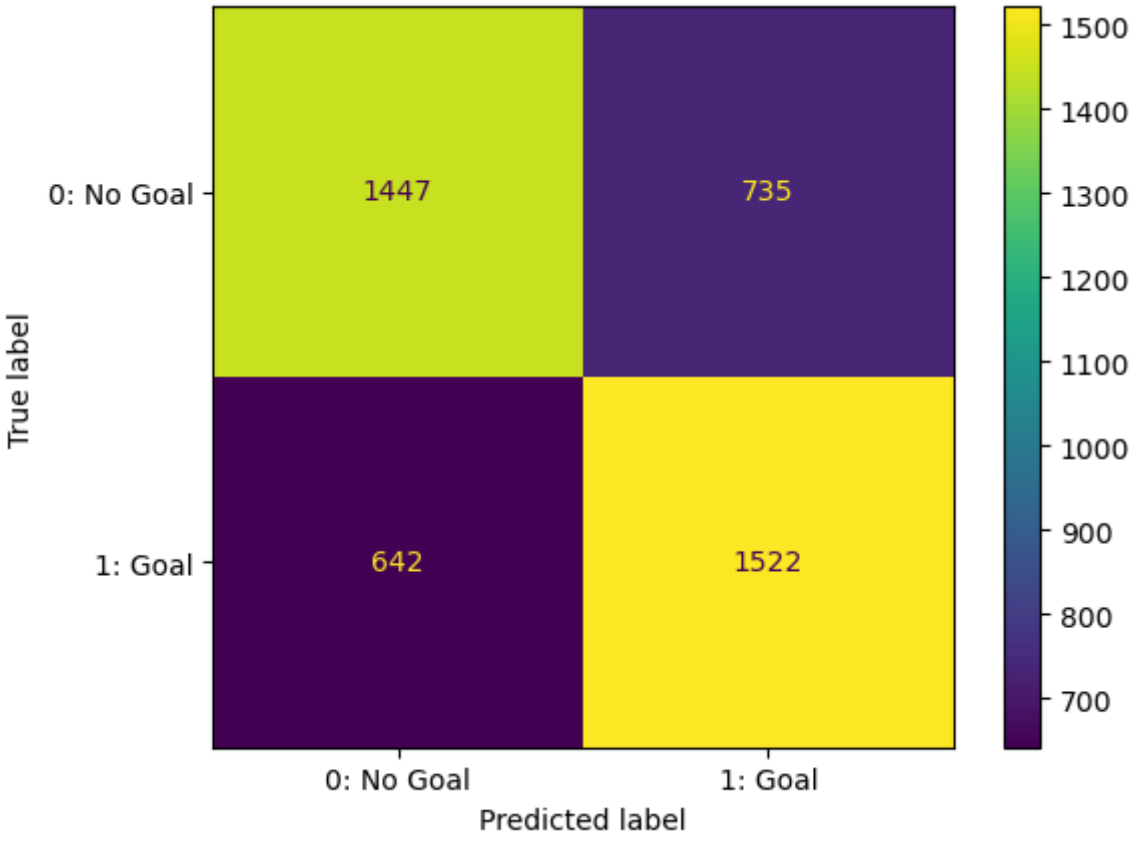
In [252]:
```
# Log Best Params w/o smote
steps = [('preprocess', ct),
        ('sampled_best_model', sampled_best_model)]

sampled_best_pipeline = Pipeline(steps=steps)
sampled_best_pipeline.fit(X_train_sample, y_train_sample)
```

Out[252]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('numpipe',
                                                  Pipeline(steps=[('s
s',
                                                                  St
andardScaler())]),
                                                  Index(['isHomeTea
m', 'shotRush', 'arenaAdjustedShotDistance',
       'arenaAdjustedXCordABS', 'arenaAdjustedYCordAbs', 'shotAngleAd
justed',
       'shotAnglePlusRebound', 'shotAnglePlusReboundSpeed', 'shotOnEm
ptyNet',
       'timeSinceLastEvent', 'distanceFromLastEv...
       'lastEventxCord_adjusted', 'lastEventyCord_adjusted',
       'speedFromLastEvent', 'offWing'],
      dtype='object')),
                                                 ('nominalpipe',
                                                  Pipeline(steps=[('o
nehotenc',
                                                                  On
eHotEncoder(handle_unknown='ignore')),
                                                                 ('o
nehotnorm',
                                                                  Ma
xAbsScaler())]),
                                                  Index(['shotType',
'game_strength_state'], dtype='object'))])),
                ('sampled_best_model',
                 LogisticRegression(C=1, max_iter=10000, solver='sag
a'))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [253]:

```
# Predict using the pipeline
y_pred_log_cv_sampled = sampled_best_pipeline.predict(X_test_sample

# Calculate the confusion matrix
label_names = ['0: No Goal', '1: Goal']
cm = confusion_matrix(y_test_sample, y_pred_log_cv_sampled)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=l
disp.plot()
plt.show()
```



In [254]:

```
# Calculate and print the classification report
report = classification_report(y_test_sample, sampled_best_pipeline
print(f'Classification report:\n{report}')
log_best_sampled_auc_roc = roc_auc_score(y_test, sampled_best_pipel
print(f'Test AUC-ROC score: {log_best_sampled_auc_roc:.3f}')
log_best_sampled_log_loss = log_loss(y_test_sample, sampled_best_pi
print(f'Test log loss score: {log_best_sampled_log_loss:.3f}')
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.69      0.66      0.68      2182
           1       0.67      0.70      0.69      2164

    accuracy                           0.68      4346
   macro avg       0.68      0.68      0.68      4346
weighted avg       0.68      0.68      0.68      4346

Test AUC-ROC score: 0.756
Test log loss score: 0.586
```

In [ ]:

```

```