In [1]:
```
# Lee Johnston

# Prof. Iranitalab

# DSC530-T303

# Term Project

# Bellevue University

# 3/1/24
```

In [ ]:
```
### Statistical Question:

#Is possession an overvalued/unnecessary statistic in European-Football/Soccer?
```

In [106…
```
# Loads necessary imports

from scipy.stats import pearsonr
from scipy.stats import zscore
from scipy.stats import norm
import statsmodels.formula.api as smf
from __future__ import print_function, division
import random
import statsmodels.api as sm
import pandas as pd
import numpy as np
import statistics
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import *
from imblearn.over_sampling import SMOTE

import itertools
import thinkstats2
import thinkplot
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)

%matplotlib inline
matplotlib.style.use('ggplot')

import os
```

```python
import os
```

In [3]:
```python
# Accesses the file

cup_data = pd.read_csv("2022worldcup.csv")
```

In [4]:
```python
cup_data.head()
```

Out[4]:

	Squad	# Pl	Age	Poss	MP	Starts	Min	90s	Gls	Ast	...	Gls90	Ast90	G+A90	G-PK90	G+A-PK90	xG90	xAG90
0	Argentina	24	27.5	57.4	7	77	690	7.7	15	8	...	1.96	1.04	3.00	1.43	2.48	1.96	1.00
1	Australia	20	27.8	37.8	4	44	360	4.0	3	3	...	0.75	0.75	1.50	0.75	1.50	0.58	0.48
2	Belgium	20	29.7	57.0	3	33	270	3.0	1	1	...	0.33	0.33	0.67	0.33	0.67	1.57	1.27
3	Brazil	26	27.6	56.2	5	55	480	5.3	8	6	...	1.50	1.13	2.62	1.31	2.44	2.24	1.54
4	Cameroon	22	27.2	41.7	3	33	270	3.0	4	4	...	1.33	1.33	2.67	1.33	2.67	1.14	0.66

5 rows × 32 columns

In [5]:
```python
cup_data.describe(include='all')
```

Out[5]:

	Squad	# Pl	Age	Poss	MP	Starts	Min	90s	Gls	
count	32	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.00
unique	32	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
top	Argentina	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
freq	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
mean	NaN	21.250000	27.196875	49.443750	4.000000	44.000000	369.375000	4.100000	5.312500	3.78
std	NaN	1.951013	1.177609	9.457066	1.344043	14.784473	135.763967	1.506973	4.130434	3.22
min	NaN	18.000000	24.500000	31.300000	3.000000	33.000000	270.000000	3.000000	1.000000	0.00
25%	NaN	20.000000	26.575000	42.750000	3.000000	33.000000	270.000000	3.000000	2.750000	1.00
50%	NaN	21.000000	27.300000	50.150000	3.500000	38.500000	315.000000	3.500000	4.500000	3.00
75%	NaN	22.000000	27.850000	54.775000	4.250000	46.750000	405.000000	4.475000	6.500000	5.00

75%	NaN	22.000000	27.850000	54.775000	4.250000	46.750000	405.000000	4.475000	6.500000	5.00
max	NaN	26.000000	29.700000	75.800000	7.000000	77.000000	690.000000	7.700000	16.000000	12.00

11 rows × 32 columns

In [ ]:
```
# Variables in use:

# 1.) Possession - The amount of time a side on average has control of or 'possession' of the ball.
# 2.) Goals90 - The amount of goals scored during the length of a match (90 minutes)
# 3.) xG90 - The amount of goals that are expected to have been scored due to factors such as
# positioning,shots,mistakes etc..
# 4.) Ast90 - The amount of assists in a match (not every goal will be because of an assist)
# 5.) xAG90 - The amount of assists that are expected to have been scored during a match (90 minutes)
```

In [86]:
```
# Tallies the number of statistics per column within the dataset

print(cup_data.shape)

print('_____________________')
print(cup_data.nunique())
print('_____________________')
print(cup_data[cup_data['Squad'] == 'Argentina']['Poss'].count()) # Competition finalist
print(cup_data[cup_data['Squad'] == 'France']['Poss'].count()) # Competition finalist
print('_____________________')

cup_data_hist = cup_data[cup_data['Squad'] == 'Argentina']
cup_data_act = cup_data[cup_data['Squad'] == 'France']
```

```
(31, 32)
_____________________
Squad         31
# Pl           9
Age           19
Poss          29
MP             4
Starts         4
Min            7
90s            7
Gls           13
Ast           11
G+A           17
G-PK          13
PK             4
PKatt          4
CrdY          11
CrdR           2
```

```
xG            23
npxG          23
xAG           22
npxG+xAG      25
PrgC          27
PrgP          30
Gls90         17
Ast90         17
G+A90         21
G-PK90        19
G+A-PK90      21
xG90          26
xAG90         27
xG+xAG90      29
npxG90        29
npxG+xAG90    29
dtype: int64
_____________________
1
1
_____________________
```

In [ ]:
```
### OUTLIERS:

# I used the z method to identify any outliers within the data that I would be calculating and was
# pleasantly surprised to find that there is only 1. Upon investigation the Ast90 variable is the
# the only one out of any category to have '0' is Wales. This means that they did not average even
# 0.1 assist per match. I will take this into account when calculating Ast90 data.
```

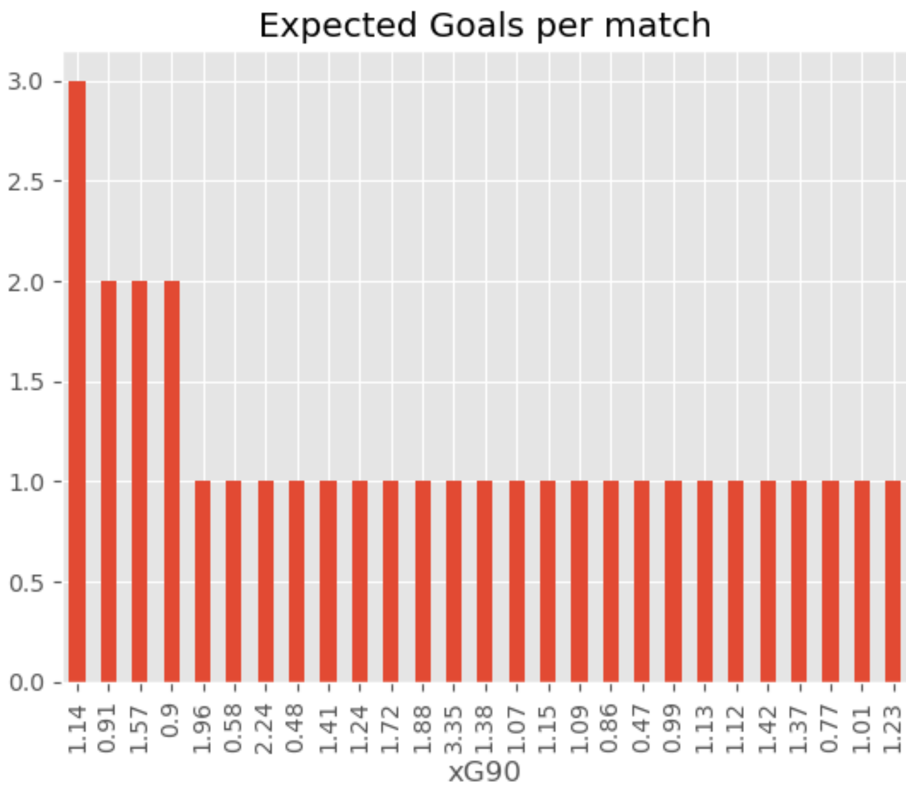
In [80]:
```
numeric_columns = ['Poss', 'Gls90', 'Ast90', 'xG90', 'xAG90']
z_scores = np.abs(zscore(cup_data[numeric_columns]))
z_threshold = 3
outliers = (z_scores > z_threshold).any(axis=1)
print("Number of Outliers:", outliers.sum())
cup_data = cup_data[~outliers]
```

```
Number of Outliers: 1
```

In [ ]:
```
### HISTOGRAMS:

# Below I demonstrate a large number of bar plots, box plots and pair plots. These are used in
# conjunction we the variables I have chosen to identify the true value in the end product of
# possession. The desired end product is assists and goals as no player as ever been accredited
# with scoring or assisting a goal without touching/possessing the ball. I plan to calculate
# this true value by finding the relationship between 'expected/x' goals/assists and actually
# goals/assists. I will then come to a conclusion and analyze this data against possession to
# test my hypothesis.
```

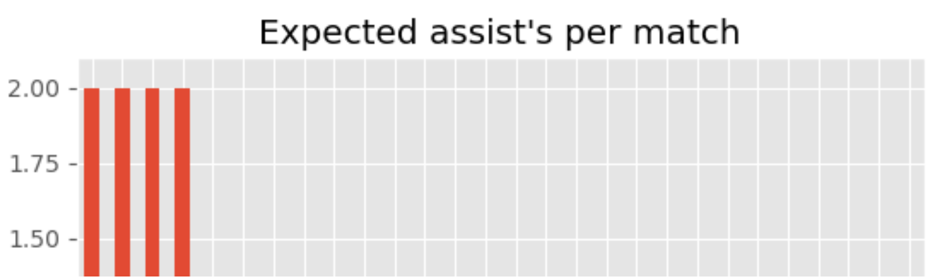
In [7]:
```
cup_data['xG90'].value_counts().plot.bar(title='Expected Goals per match')
```

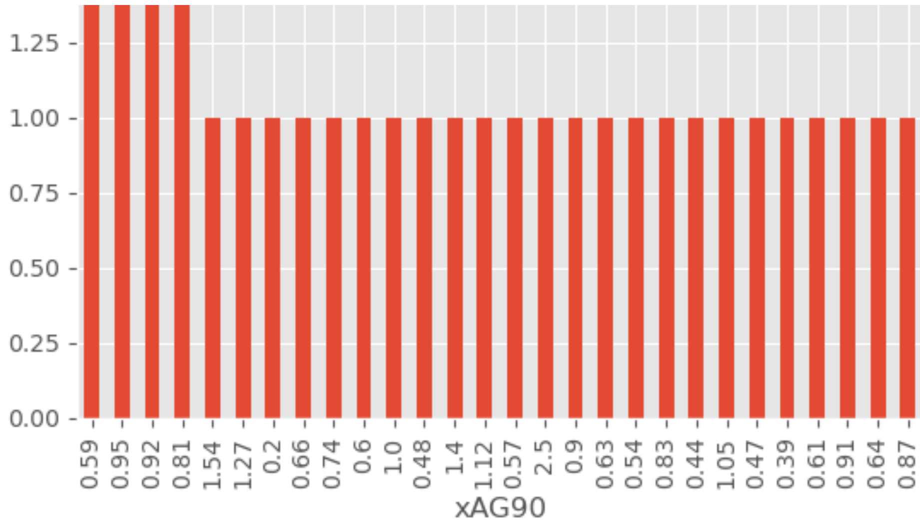
Out[7]: <Axes: title={'center': 'Expected Goals per match'}, xlabel='xG90'>



In [8]:
```
cup_data['xAG90'].value_counts().plot.bar(title="Expected assist's per match")
```

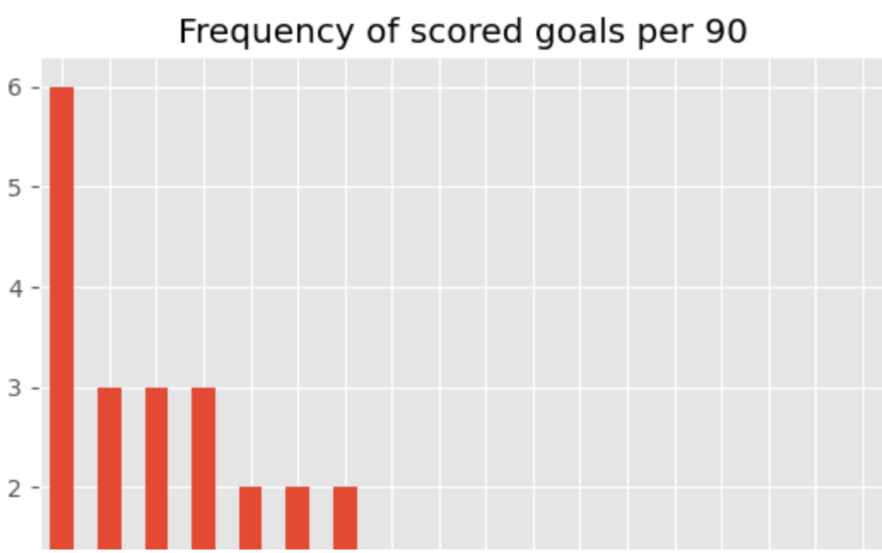
Out[8]: <Axes: title={'center': "Expected assist's per match"}, xlabel='xAG90'>

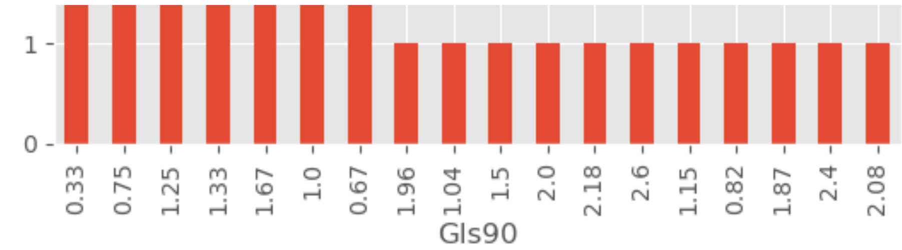




In [9]:
```
cup_data['Gls90'].value_counts().plot.bar(title="Frequency of scored goals per 90")
```

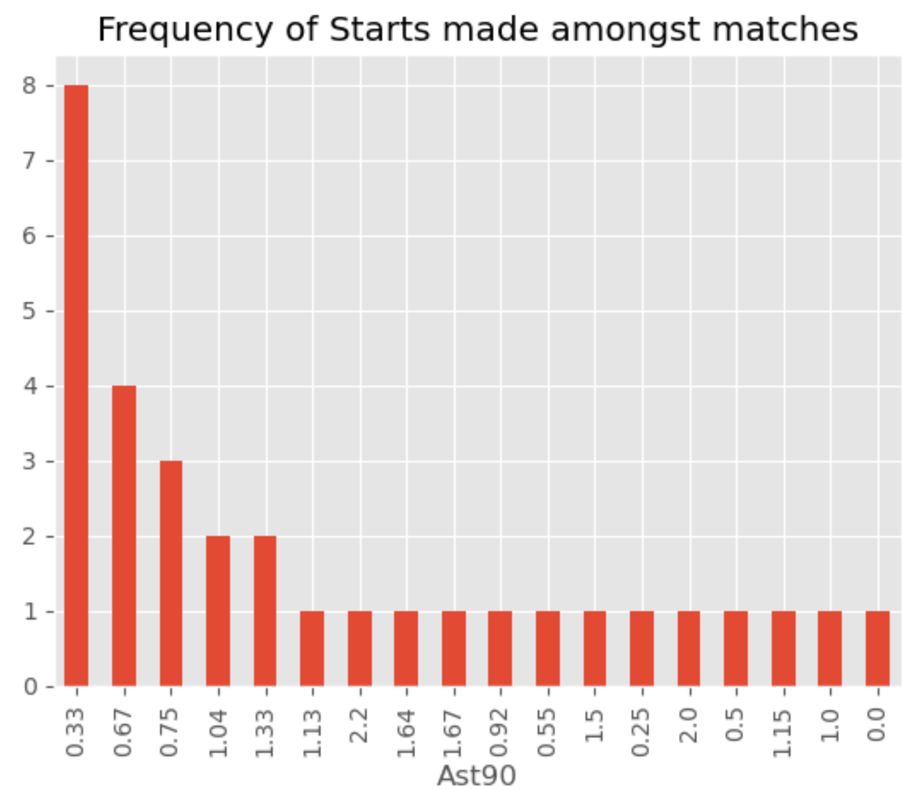
Out[9]:
```
<Axes: title={'center': 'Frequency of scored goals per 90'}, xlabel='Gls90'>
```





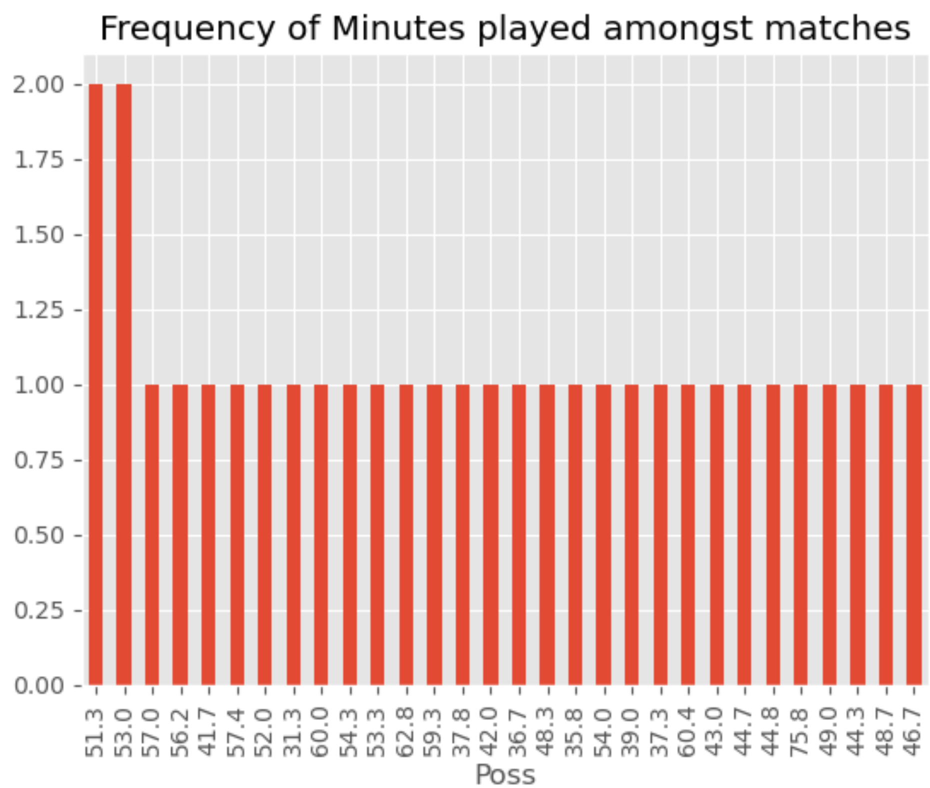
In [10]:
```
cup_data['Ast90'].value_counts().plot.bar(title="Frequency of Starts made amongst matches")
```

Out[10]: <Axes: title={'center': 'Frequency of Starts made amongst matches'}, xlabel='Ast90'>



In [11]:
```
cup_data['Poss'].value_counts().plot.bar(title="Frequency of Minutes played amongst matches")
```

Out[11]:
```
<Axes: title={'center': 'Frequency of Minutes played amongst matches'}, xlabel='Poss'>
```



In [ ]:
```
### DESCRIPTIVE STATISTICS:

# Below is the detailed report of the mean,median,mode,tail value and spread value of the targeted
# variables. The mean value of possession and xG90 will be two particular values I frequently use
# during this analysis. I do also find the value for xAG90's mode very interesting. Upon initial
# look I am surprised that per game an average of less than 1 expected assist a game shows a lack
# of effective passing. This is helpful for the first half of our hypothesis.
```

In [12]:
```
print("Mean: ", statistics.mean(cup_data.xG90))
print("Median: ", statistics.median(cup_data.xG90))
print("Mode: ", statistics.mode(cup_data.xG90))
```

```
Mean:  1.253125
Median:  1.14
Mode:  1.14
```

In [13]:
```
print("Mean: ", statistics.mean(cup_data.xAG90))
print("Median: ", statistics.median(cup_data.xAG90))
print("Mode: ", statistics.mode(cup_data.xAG90))
```

```
Mean:  0.840625
Median:  0.81
Mode:  0.95
```

In [14]:
```
print("Mean: ", statistics.mean(cup_data.Gls90))
print("Median: ", statistics.median(cup_data.Gls90))
print("Mode: ", statistics.mode(cup_data.Gls90))
```

```
Mean:  1.1953125
Median:  1.2
Mode:  0.33
```

In [15]:
```
print("Mean: ", statistics.mean(cup_data.Ast90))
print("Median: ", statistics.median(cup_data.Ast))
print("Mode: ", statistics.mode(cup_data.Ast))
```

```
Mean:  0.838125
Median:  3.0
Mode:  1
```

In [16]:
```
print("Mean: ", statistics.mean(cup_data.Poss))
print("Median: ", statistics.median(cup_data.Poss))
print("Mode: ", statistics.mode(cup_data.Poss))
```

```
Mean:  49.44375
Median:  50.15
Mode:  51.3
```

In [74]:
```
descriptive_characteristics = cup_data.describe()

spread_values = descriptive_characteristics.loc['std']
tails_values = [cup_data[column].skew() for column in ['Poss', 'Gls90', 'Ast90', 'xG90', 'xAG90']]

print("\nSpread Values:\n", spread_values)
print("\nSkewness Values:\n", tails_values)
```

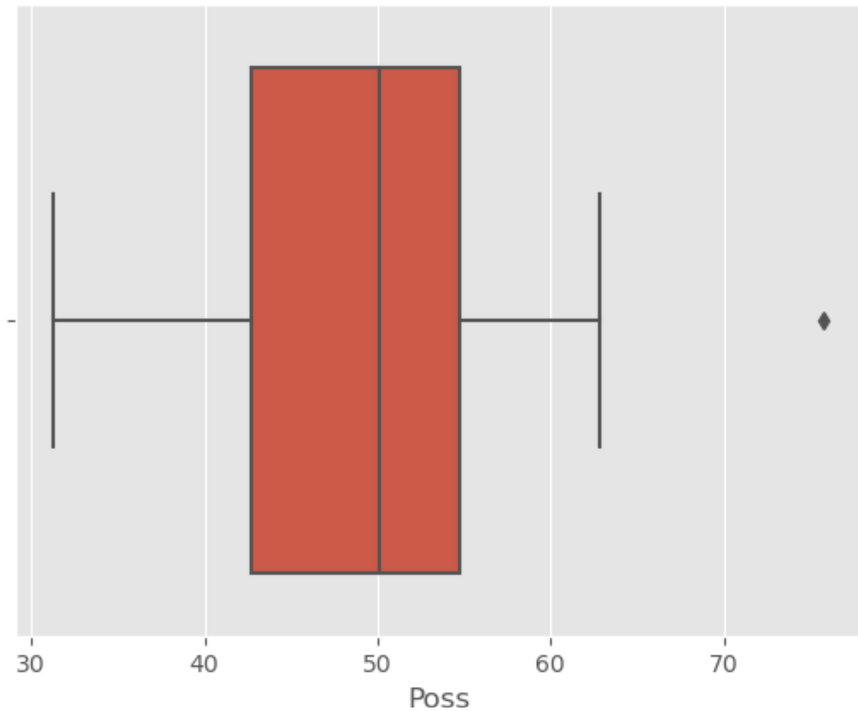
```
Spread Values:
 # Pl           1.951013
Age            1.177609
Poss           9.457066
MP             1.344043
Starts        14.784473
Min          135.763967
90s            1.506973
Gls            4.130434
Ast            3.220242
G+A            7.248401
G-PK           3.571702
PK             0.879310
PKatt          1.054464
CrdY           3.266244
CrdR           0.336011
xG             3.379635
npxG           2.860859
xAG            2.276225
```

```
npxG+xAG        5.114921
PrgC           40.185043
PrgP           78.454418
Gls90           0.656329
Ast90           0.544950
G+A90           1.167589
G-PK90          0.590517
G+A-PK90        1.107766
xG90            0.555689
xAG90           0.421364
xG+xAG90        0.960067
npxG90          0.499274
npxG+xAG90      0.920985
Name: std, dtype: float64

Skewness Values:
 [0.3729912091541292, 0.38684203930928385, 0.8216951778288208, 1.8679940832328956, 2.0968427563919017]
```

In [17]:

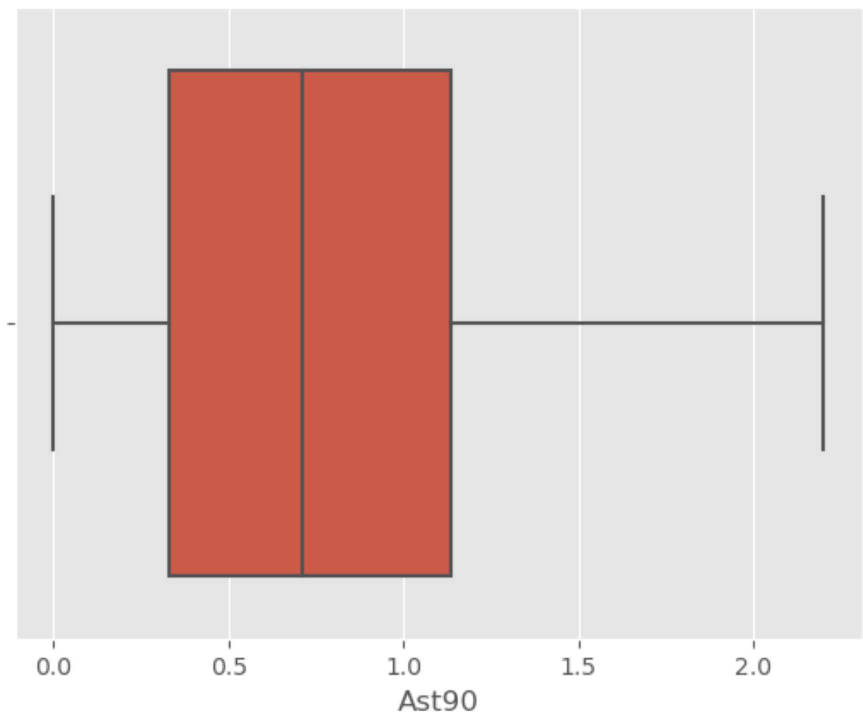
```python
sns.boxplot(x=cup_data['Poss'])
```

Out[17]: <Axes: xlabel='Poss'>



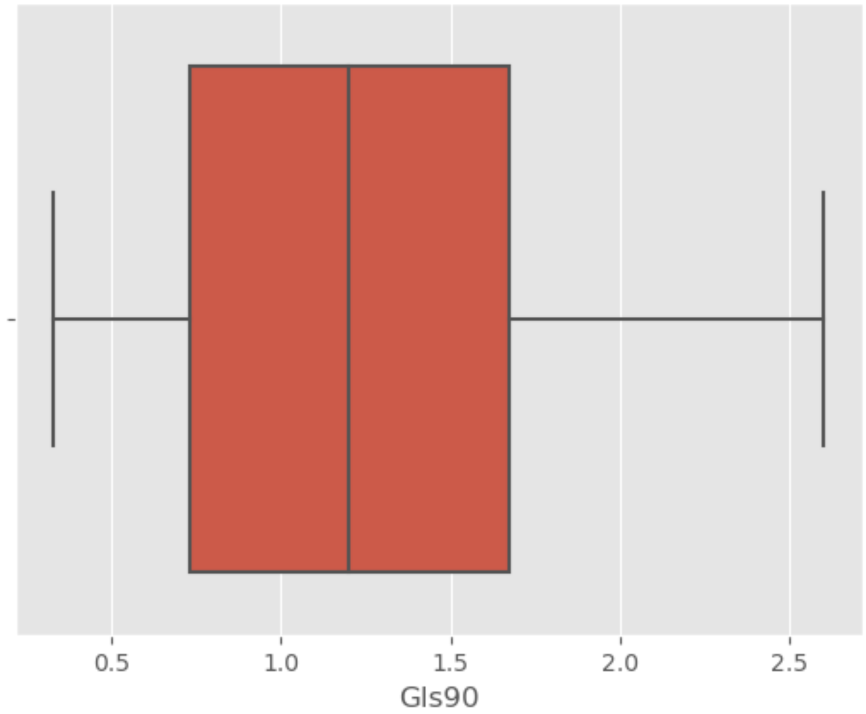
In [18]:
```
sns.boxplot(x=cup_data['Ast90'])
```

Out[18]: <Axes: xlabel='Ast90'>



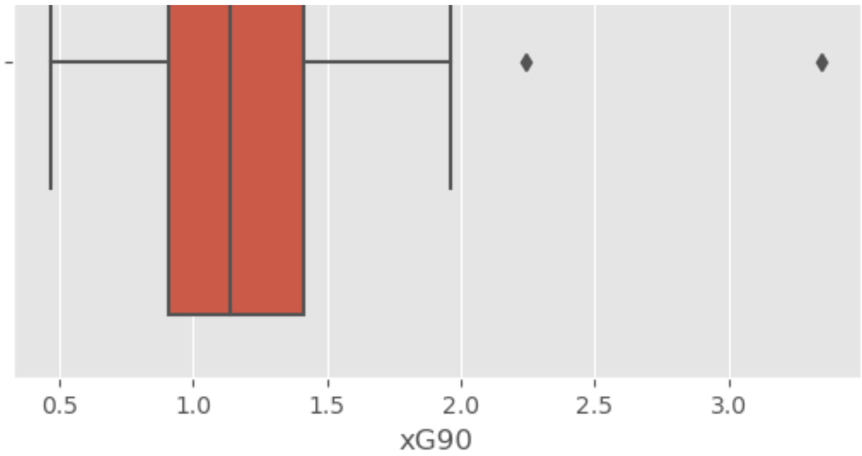
In [19]:
```
sns.boxplot(x=cup_data['Gls90'])
```

Out[19]: <Axes: xlabel='Gls90'>



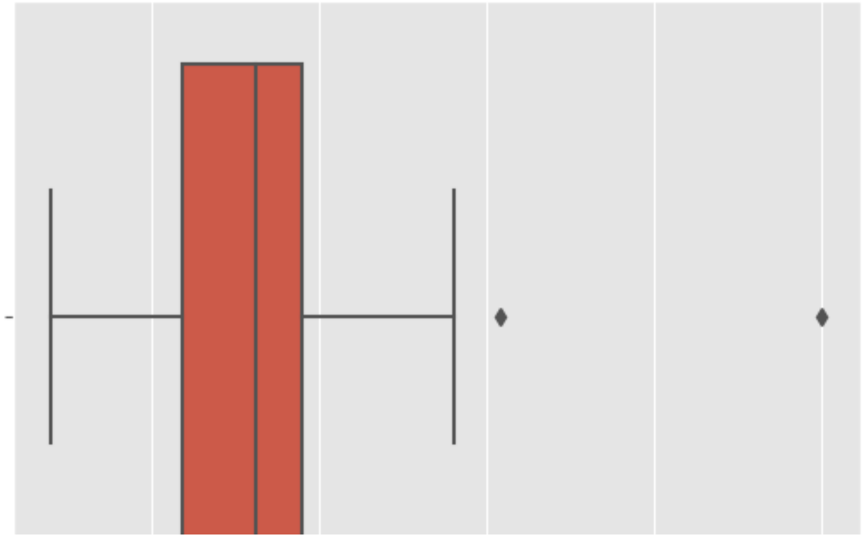
In [20]:
```
sns.boxplot(x=cup_data['xG90'])
```

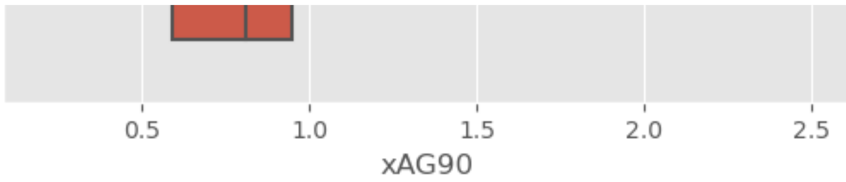
Out[20]: <Axes: xlabel='xG90'>



In [21]:
```
sns.boxplot(x=cup_data['xAG90'])
```

Out[21]: <Axes: xlabel='xAG90'>

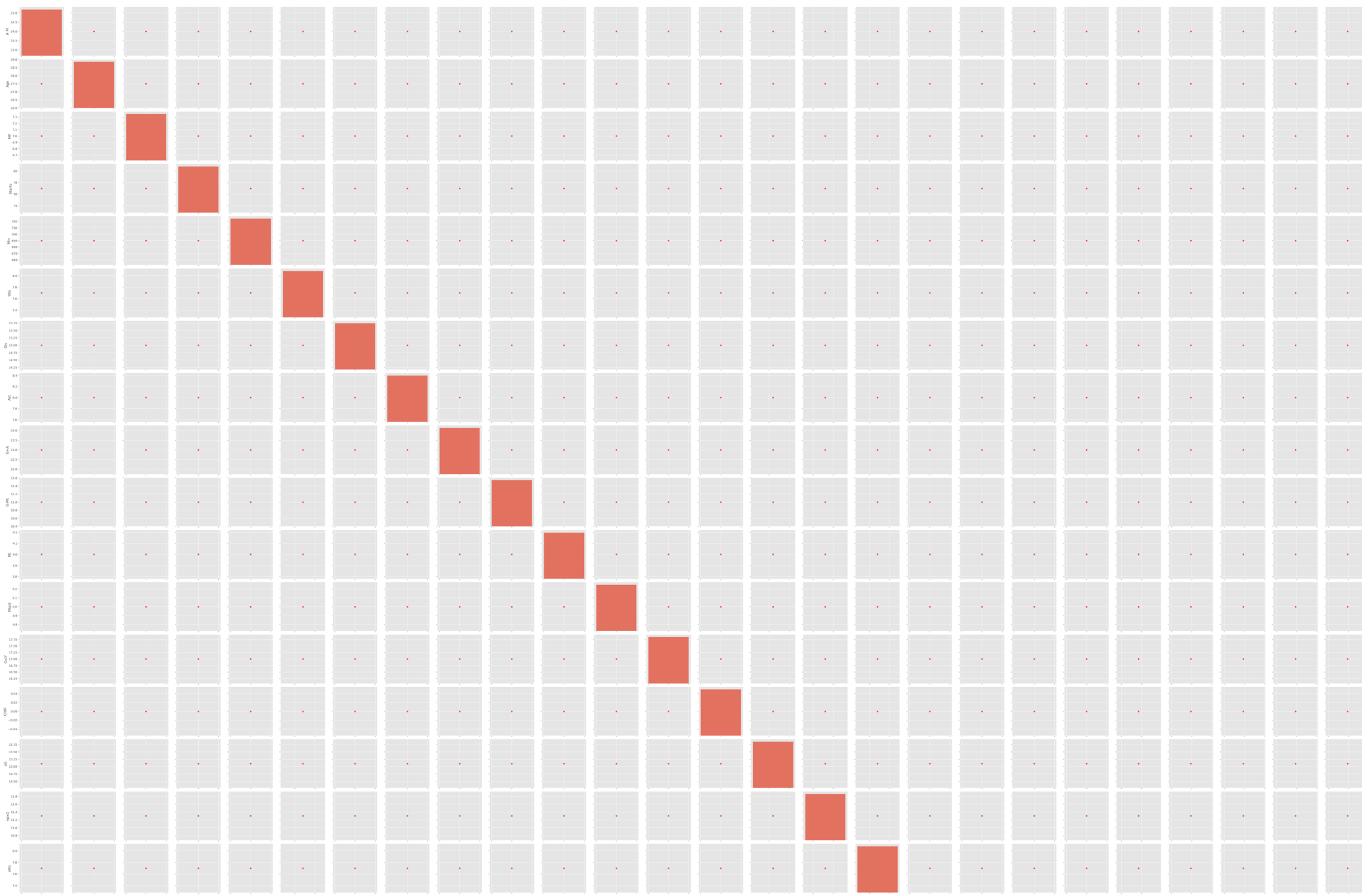


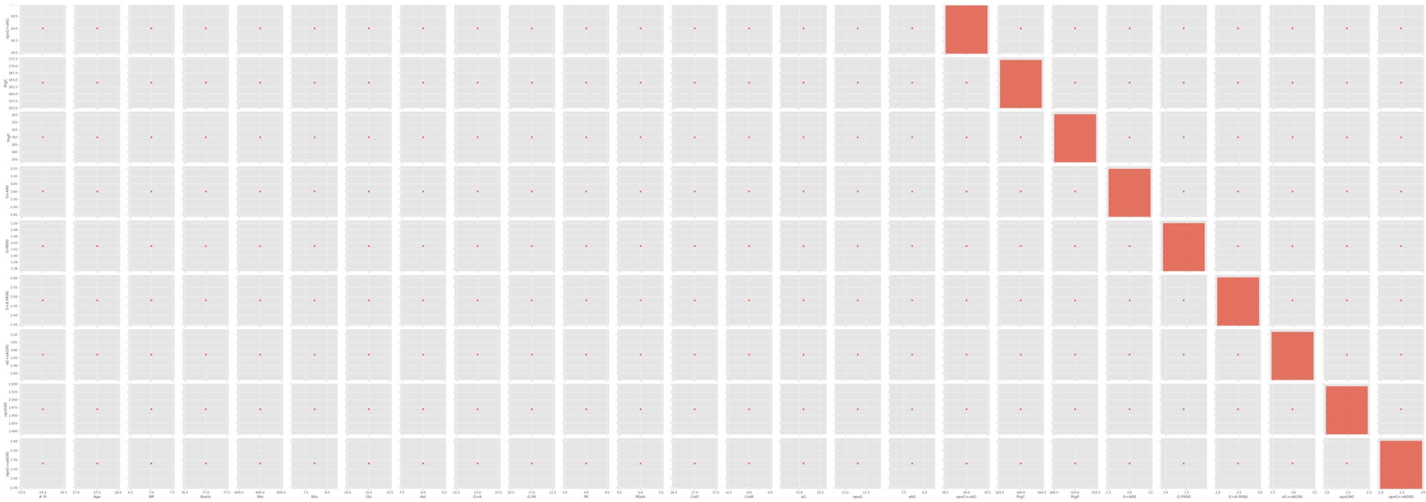


In [22]:
```
cup_data_hist = cup_data_hist.drop(['Poss', 'Gls90', 'Ast90', 'xG90', 'xAG90'], axis=1)
sns.pairplot(cup_data_hist)
```

```
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWa
rning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

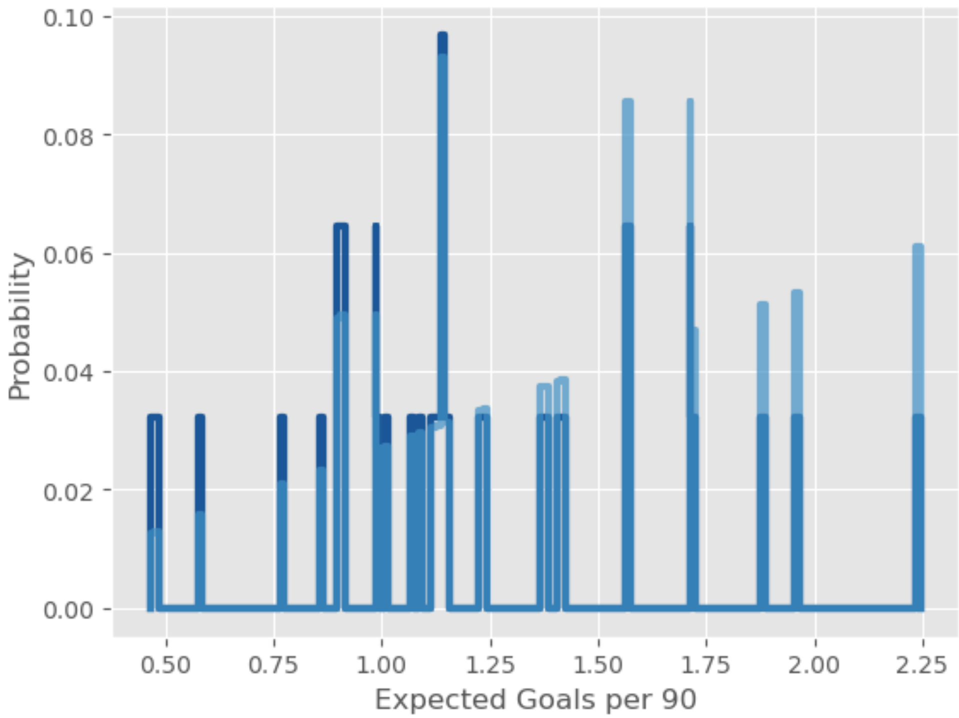
Out[22]: `<seaborn.axisgrid.PairGrid at 0x7fbf4201f010>`





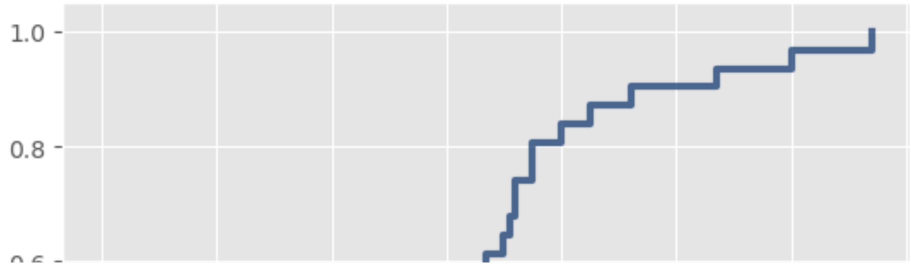
In [ ]:
```
### PMF:

# Below is the PMF I calculated for the probability of expected goals in a match. This will
# be based on a sides recorded goals and expected goals within a dataset.  The x-axis is the
# amount of goals that were expected to be scored and the y-axis is the probability of the goals
# actually being scored. The graph is represented by a difference level of result-to-probability
# with a raised level of probability that the expected goals will be executed the higher the
# quantity of xG90 rises.
```

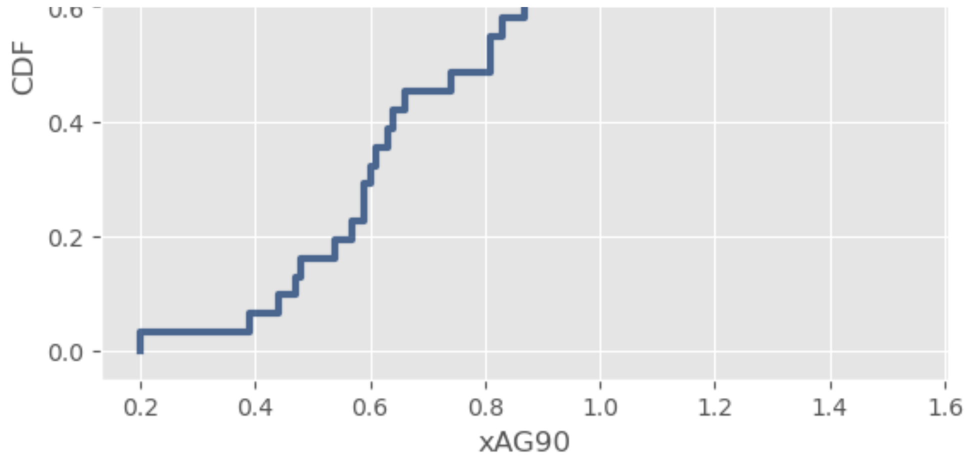
In [94]:
```
true_pmf = thinkstats2.Pmf(cup_data.xG90, label = 'True')
thinkplot.Pmf(true_pmf)
thinkplot.Config(xlabel='Expected Goals per 90', ylabel='Probability')
def Falsepmf(pmf, label):
    next_pmf = pmf.Copy(label=label)

    for x, p in pmf.Items():
        next_pmf.Mult(x, x)

    next_pmf.Normalize()
    return next_pmf
false_pmf = Falsepmf(true_pmf, label = 'False')
thinkplot.PrePlot(2)
thinkplot.Pmfs([true_pmf, false_pmf])
thinkplot.Config(xlabel='Expected Goals per 90', ylabel='Probability')
```



In [ ]:
```
### CDF:

# The x-axis shows the amount of expected assisted goals per match. The y-axis shows the level of chance
# for the concentration being equal, less than or more than the recorded amount. For the entirety of the
# chart the chance of concentration is below the expected amount.
```

In [96]:
```
cup_cdf = thinkstats2.Cdf(cup_data.xAG90, label='Expected Assisted Goals')
thinkplot.Cdf(cup_cdf)
thinkplot.Config(xlabel='xAG90', ylabel='CDF', loc='best')
```



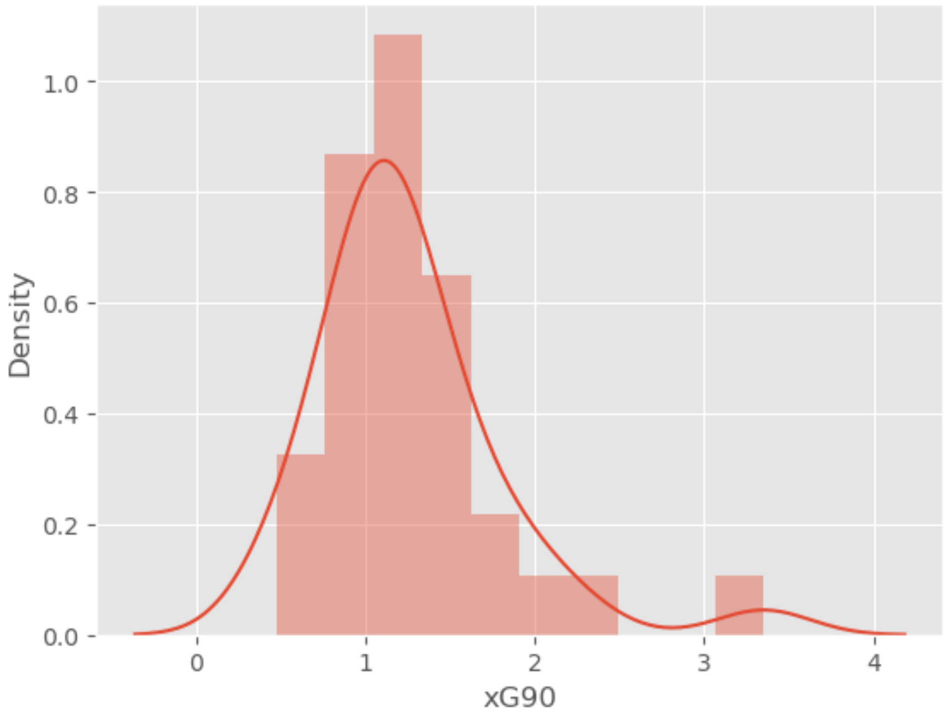


In [ ]:
```
### ANALYTICAL DISTRIBUTION:

# The first plot displays the distribution of expected goals per match while the second plot shows
# the distribution of actual goals scored per match. Similar patterns are displayed in both plots
# whilst a small rise of '3' in the expected goals plot is shown that does not arise in the actual
# goals plot. This correlates to even teams with good performances were unable to make it count, to
# the level of 3 goals. This tally would surely *almost* guarantee victory.

# (*—ALMOST—* As seen in the final)
```

In [25]:
```
sns.distplot(cup_data['xG90'])
```

```
/tmp/ipykernel_2627/713646488.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(cup_data['xG90'])
```

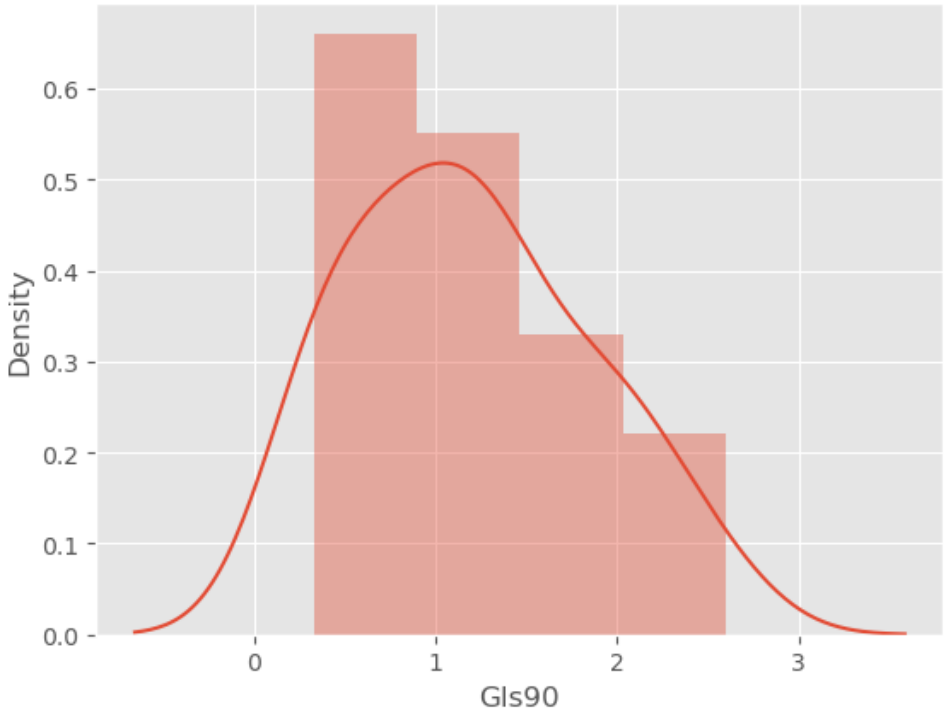
Out[25]: <Axes: xlabel='xG90', ylabel='Density'>



In [26]:
```
sns.distplot(cup_data['Gls90'])
```

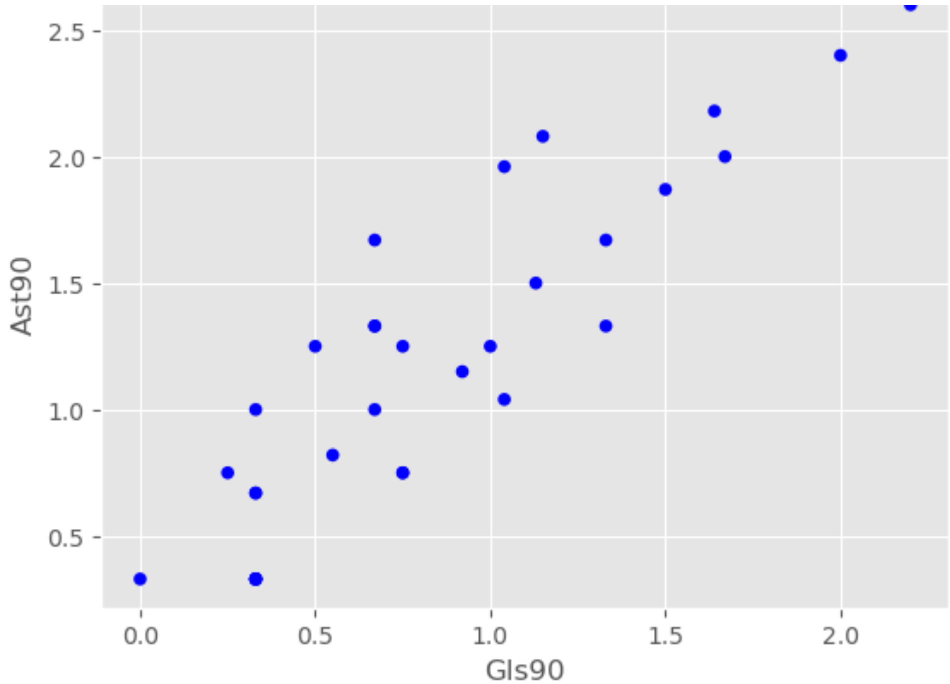
```
/tmp/ipykernel_2627/3961954287.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(cup_data['Gls90'])
```

```
Out[26]: <Axes: xlabel='Gls90', ylabel='Density'>
```

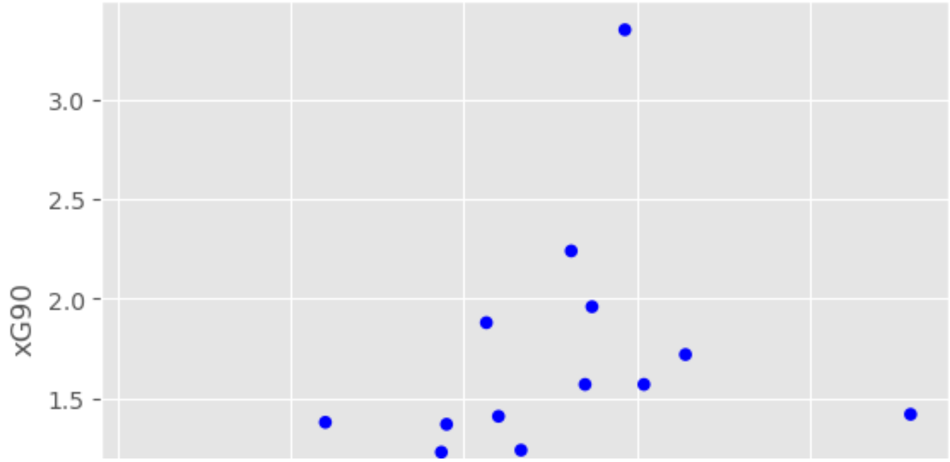


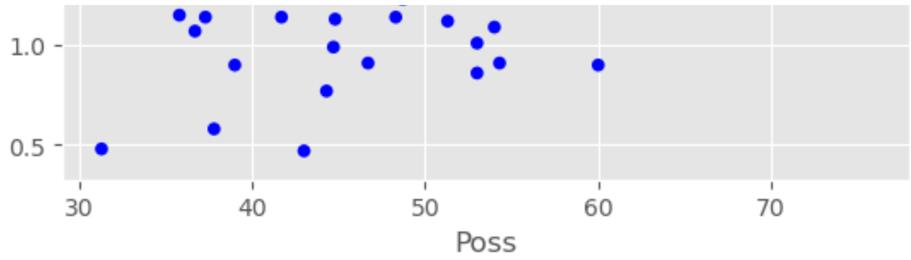
In [ ]:
```
### Scatterplots:


# These plots were created to identify the relationship between certain variables that will have a
# large impact on the datasets hypothesis. The Ast90-Gls90 is positive which accounts for a raised
# level of confidence within the datasets legitimacy. The second plot is one of the more important
# pieces of the experiment. This displays the relationship between Possesion and expected goals per
# match. Whilst you would expect the correlation to be both linear and positive it displays a general
# 'theme' of such but a vast array of outcomes that are outside of the expected path. These outcomes
# are both positive and negative but mostly lean towards the negative side other than one particularly
# seemingly large outcome resulting in an xg of almost FOUR!!! This is particularly interesting because
# it lies in the >60% percentile for possession.
```

In [27]:
```
Gls90, Ast90 = cup_data.Ast90, cup_data.Gls90
thinkplot.Scatter(Gls90, Ast90, alpha=1)
thinkplot.Config(xlabel='Gls90',
                 ylabel='Ast90',
                 legend=False)
```



In [28]:
```
Poss, xG90 = cup_data.Poss, cup_data.xG90
thinkplot.Scatter(Poss, xG90, alpha=1)
thinkplot.Config(xlabel='Poss',

                 ylabel='xG90',
                 legend=False)
```





In [ ]:
```
### Hypothesis Testing:

# The recorded p-value of the hypothesis test conducted below of 0.02 shows that the outcome
# shown is highly unlikely to be due to chance. I spearmen correlation and tested correlation
# fall in the 0.51-0.53 range displaying a slightly positive level of correlation with each-other.
# As mentioned earlier possession is necessary to score/assist but the level of importance was
# at least in my opinion, questionable.


# The correlation plot below shows that the possibility of an 'expected' goal occurring reaches
# 100% chance once a team records 20% possession whilst the first ocurance takes place at just
# below the 50% margin.
```

In [29]:
```
Poss, xG90 = cup_data.Poss, cup_data.xG90
Posses, xAG90 = cup_data.Poss, cup_data.xAG90
def Cov(xs, ys, meanx=None, meany=None):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    if meanx is None:
        meanx = np.mean(xs)
    if meany is None:
        meany = np.mean(ys)

    cov = np.dot(xs-meanx, ys-meany) / len(xs)
    return cov

def Corr(xs, ys):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    meanx, varx = thinkstats2.MeanVar(xs)
    meany, vary = thinkstats2.MeanVar(ys)

    corr = Cov(xs, ys, meanx, meany) / np.sqrt(varx * vary)
    return corr
```

```
def SpearmanCorr(xs, ys):
    xranks = pd.Series(xs).rank()
    yranks = pd.Series(ys).rank()
    return Corr(xranks, yranks)
print("Corr",Corr(xG90,Poss))
print("SpearmanCor",SpearmanCorr(xG90,Poss))
```

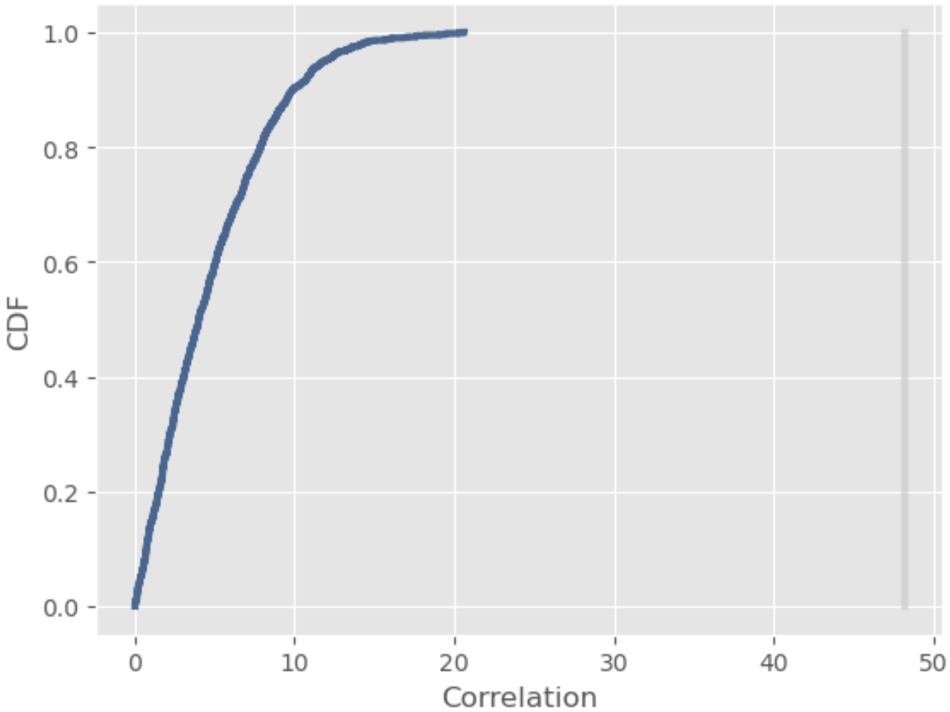
```
Corr 0.511400523745551
SpearmanCor 0.5325140429710394
```

In [76]:
```
Hypothesis_correlation, _ = pearsonr(cup_data['Poss'], cup_data['xG90'])
num_range = 1000
Tested_correlation = np.zeros(num_range)

for i in range(num_range):
    tested_possesion = np.random.permutation(cup_data['Poss'])
    tested_xg, _ = pearsonr(tested_possesion, cup_data['xG90'])
    Tested_correlation[i] = tested_xg

p_value = np.sum(np.abs(Tested_correlation) >= np.abs(Hypothesis_correlation)) / num_range

print("Hypothesis Correlation:", Hypothesis_correlation)
print("Tested P-Value:", p_value)
```

```
Hypothesis Correlation: 0.5114005237455511
Tested P-Value: 0.002
```

In [78]:
```
class TestHypothesis(thinkstats2.HypothesisTest):
    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = abs(group1.mean() - group2.mean())
        return test_stat

    def MakeModel(self):
        group1, group2 = self.data
        self.n, self.m = len(group1), len(group2)
        self.pool = np.hstack((group1, group2))

    def RunModel(self):
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data

data = cup_data.Poss.values, cup_data.xG90.values
ht = TestHypothesis(data)
pvalue = ht.PValue()
pvalue
ht.PlotCdf()
```

```
thinkplot.Config(xlabel='Correlation',
                 ylabel='CDF')
```



In [ ]:
```
### REGRESSION ANALYSIS:

# The OLS method was used to explore the relationship between the target variable 'Poss/Possesion'
# and the four other variables in question in xAG90, xG90, Gls90 and Ast90. This method produced
# an R-Squared value of 0.360 which indicates that possesion has a 36% influence on the other variables.
# The coefficient levels are significant in all variables but are especially significant in the
# expected Assisted Goals per match variable at 13.27. The F-Statistic of 3.66 is relatively close to
# 1.0 giving a sense of validity to the null hypothesis in question.
```

In [108…
```
X = sm.add_constant(cup_data[['xAG90', 'xG90', 'Gls90', 'Ast90']])
y = cup_data['Poss']

model = sm.OLS(y, X).fit()
predictions = model.predict(X)

print(model.summary())
```

```
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:                   Poss   R-squared:                       0.360
Model:                            OLS   Adj. R-squared:                  0.262
Method:                 Least Squares   F-statistic:                     3.660
Date:                Sun, 03 Mar 2024   Prob (F-statistic):             0.0171
Time:                        00:32:53   Log-Likelihood:                -106.14
No. Observations:                  31   AIC:                             222.3
Df Residuals:                      26   BIC:                             229.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         33.6587      4.599      7.318      0.000      24.205      43.113
xAG90         13.2700     12.133      1.094      0.284     -11.669      38.209
xG90           1.6584      9.397      0.176      0.861     -17.658      20.974
Gls90          0.9699      6.500      0.149      0.883     -12.391      14.331
Ast90          2.3693      7.403      0.320      0.751     -12.848      17.587
==============================================================================
Omnibus:                        7.261   Durbin-Watson:                   1.773
Prob(Omnibus):                  0.027   Jarque-Bera (JB):                5.867
Skew:                           0.764   Prob(JB):                       0.0532
Kurtosis:                       4.485   Cond. No.                         26.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```