Final Project#

This is the final project for Data 100 at the University of California, Berl
given access to 4 data sets, and told to pick one of the sets, and 'put
learned in this course in a more open-ended setting than the assignme

Outline

We chose to delve deeper into the basketball data set. These include
box score information for NBA players over the last 7 years. We wante
learned in order to create a classifier that can help predict a team's wi
season based on different features of the team.

Collaboration Policy

We believe that data science is a collaborative activity. This project wa
Schlotter and Eliza Page van Hamel Platerink.

Setup

In [1]:
```
#import statements needed to complete the projec

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from matplotlib.colors import ListedColormap
import seaborn as sns
import plotly.express as px

from sklearn.linear_model import LogisticRegress
from sklearn.linear_model import LinearRegressio
from sklearn.model_selection import KFold
from sklearn.base import clone
from sklearn.model_selection import train_test_s
```

Loading in the Data

We were given five possible tables to explore. Although we only endec
box score and the team box score (and predicting the winning percent
table), we loaded all of the data in originally in order to properly unders
looking at, and perfom an exploratory data analysis on what we were

Despite this, we wanted to highlight a clear understanding and interpre
sets we were used, before we properly cleaned the two sets.

Official Box Score:

- 155713 rows
- 51 columns
- 13 qualitative ordinal variables
- 10 qualitative nominal variables
- 21 quantitative continous variables
- 7 quantitative discrete variables
- each row represents a player
- for our purposes, the foreign key in this case is the 'teamAbbr' co the elements of this table we hope to use with the team box score

Team Box Score:

- 14758 rows
- 123 columns
- 10 qualitative ordinal variables
- 8 qualitative nominal variables
- 43 quantitative continous variables
- 62 quantitative discrete variables
- each row represents a game played by a team

Standings:

- The table containing the wins and losses for each team. We will u winning percentage for each team, which is what we will be predi
- 39 columns
- 3 qualitative ordinal variables
- 2 qualitative nominal variables
- 18 quantitative continous variables
- 16 quantitative discrete variables
- each row represents a game

In sports, basketball in particular, a winning percentage is the fraction or individual has won. In our case, we focused mainly on the teams.

In [2]:
```
#Downloading the provided NBA data sets
college = pd.read_csv("college.csv")
teamBoxScore = pd.read_csv("2012-18_teamBoxScore
standings = pd.read_csv("2012-18_standings.csv")
playerBoxScore = pd.read_csv("2012-18_playerBoxS
officialBoxScore = pd.read_csv("2012-18_playerBo
```

In [3]:
```
teamBoxScore.columns
```

Out[3]:
```
Index(['gmDate', 'gmTime', 'seasTyp', 'offLNm1', 'of
       'offFNm2', 'offLNm3', 'offFNm3', 'teamAbbr',
       ...
       'opptFIC40', 'opptOrtg', 'opptDrtg', 'opptEDi
ptAR',
       'opptAST/TO', 'opptSTL/TO', 'poss', 'pace'],
      dtype='object', length=123)
```

Part I – Data Cleaning

We start with cleaning our data! After some investigation, we quickly r
going to need all of the columns, or that they were not going to be exc
realized that there were no places where the data was missing (i.e. no
mess up the computation/model.

In [4]:
```
#Define a function to help us drop unneccessary

def drop(lst, data):
    '''
    Args:
        lst (list-like): names of columns to dro
        data (data frame): data frame from which

    Returns:
    The data frame, but with the columns we want
    '''
    for c in np.arange(len(lst)):
        data = data.drop(lst[c], axis = 1)

    return data

#The column labels we will be dropping from offi
dropping = ['gmDate', 'gmTime','teamLoc','teamRs
            'offLNm1', 'offFNm1', 'offLNm2', 'of
            'opptAbbr', 'opptDiv', 'opptLoc', 'o
            'playDispNm','playStat', 'playPos',

#Starting the cleaning process dropping columns
cleaned = drop(dropping, officialBoxScore)

#We only wanted to use regular season games.
cleaned = cleaned[cleaned['seasTyp'] == 'Regular

#Get the mean age of each player on each team
cleaned['playAge'] = 2020 - cleaned['playBDate']
agee = cleaned.groupby("teamAbbr").agg(np.mean)[
```

In [5]:

```
#A function to help us separate our data easily
def separateCols(data):
    '''
    Args:
        data (data frame): data frame from which

    Returns:
        A two dimensional list, separated by col
        by the mean, vs columsn we want to aggre
    '''
    cols_with_p = ["teamAbbr"]
    cols_wo_p = []
    cols = data.columns
    for i in range(len(cols)):
        if "%" in cols[i]:
            cols_with_p.append(cols[i])
        else:
            cols_wo_p.append(cols[i])
    return [cols_with_p, cols_wo_p]
```

In [6]:
```
#The columns to be dropped from teamBoxScore
dropping2 = ['gmDate', 'gmTime','teamLoc','teamR
             'offLNm1', 'offFNm1', 'offLNm2', 'o
             'opptAbbr', 'opptDiv', 'opptLoc', '


#Rename the columns so that they have % in them,
clean_tbs = teamBoxScore.copy()
clean_tbs = clean_tbs.rename(columns={
    'teamBLKR':'teamBLKR%',
    'teamPPS':'teamPPS%',
    'teamFIC':'teamFIC%',
    'teamFIC40':'teamFIC40%',
    'teamOrtg':'teamOrtg%',
    'teamDrtg':'teamDrtg%',
    'teamEDiff':'teamEDiff%',
    'teamAR':'teamAR%',
    'teamAST/TO':'teamAST/TO%',
    'teamSTL/TO':'teamSTL/TO%',
    'opptBLKR':'opptBLKR%',
    'opptPPS':'opptPPS%',
    'opptFIC':'opptFIC%',
    'opptFIC40':'opptFIC40%',
    'opptOrtgopptDrtg':'opptOrtgopptDrtg%',
    'opptEDiff':'opptEDiff%',
    'opptAR':'opptAR%',
    'opptAST/TO':'opptAST/TO%',
    'opptSTL/TO':'opptSTL/TO%',
    'poss':'poss%',
    'pace':'pace%',
    'opptOrtg':'opptOrtg%',
    'opptDrtg':'opptDrtg%'
})
clean_tbs = clean_tbs[clean_tbs['seasTyp']=="Reg
clean_tbs = drop(dropping2, clean_tbs)

#Separate the data
separated = separateCols(clean_tbs)
clean_tbs_mean = clean_tbs[separated[0]]
clean_tbs_sum = clean_tbs[separated[1]]
```

In [7]:
```
#Group the columns to be summed, and take their
sum_grouped = clean_tbs_sum.groupby("teamAbbr").
sum_grouped = sum_grouped.reset_index()

#Group the columns to have their average taken,
mean_grouped = clean_tbs_mean.groupby("teamAbbr"
mean_grouped["avg_age"] = agee
mean_grouped = mean_grouped.reset_index()
```

In [8]:

```
#We only wanted to select the standings that are
standings_mod = standings.copy()
standings_mod = standings_mod[standings_mod["gam
standings_mod = standings_mod[(standings_mod["st
                              (standings_mod["st
                              (standings_mod["st
                              (standings_mod["st
                              (standings_mod["st
                              (standings_mod["st
standings_mod_sum = standings_mod[["teamAbbr", "
                                   "homeWin", "h

standings_mod_sum = standings_mod_sum.groupby("t
standings_mod_sum = standings_mod_sum.reset_inde
```

In [9]:

```
#Join the three tables, and add a game percentag
nba = standings_mod_sum.merge(mean_grouped, left
win_percentage = nba["gameWon"] / (nba["gameWon"
nba
```

Out[9]:

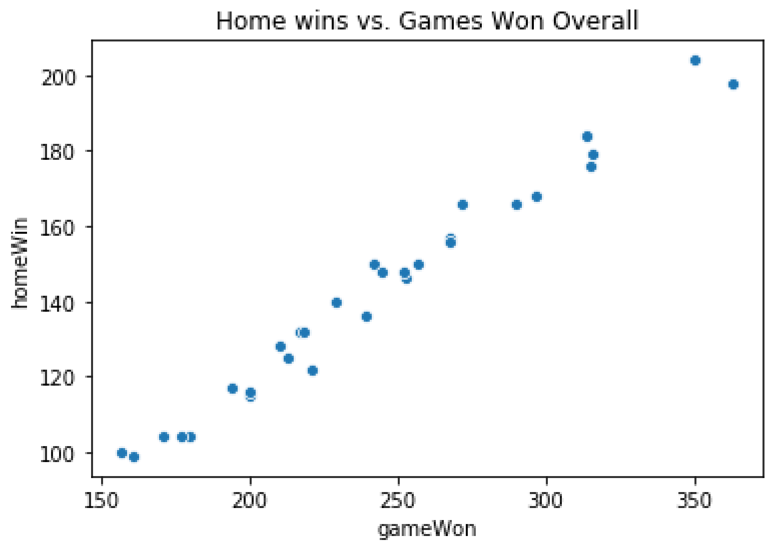
	teamAbbr	gameWon	gameLost	ptsFor	ptsAgnst	homeWin	ho
0	ATL	257	235	50090	49872	150	
1	BKN	200	292	49564	51198	115	
2	BOS	221	189	42259	41839	122	
3	CHA	217	275	49278	50053	132	
4	CHI	253	239	48797	49035	146	
5	CLE	268	224	51118	50575	157	
6	DAL	239	253	50325	50453	136	
7	DEN	242	250	52119	52260	150	
8	DET	210	282	49322	50083	128	
9	GS	363	129	54089	50468	198	
10	HOU	316	176	53445	51320	179	
11	IND	229	181	41566	40944	140	
12	LAC	315	177	52307	49960	176	
13	LAL	171	321	50315	52773	104	
14	MEM	268	224	48115	48033	156	
15	MIA	290	202	49724	48641	166	
16	MIL	213	279	49311	50452	125	
17	MIN	194	298	50670	51564	117	
18	NO	218	274	50168	51028	132	
19	NY	200	292	49002	50301	116	
20	OKC	314	178	52521	50118	184	
21	ORL	157	335	48615	51170	100	
22	PHI	161	331	48729	51533	99	
23	PHO	180	312	50453	52617	104	
24	POR	272	220	51311	50659	166	
25	SA	350	142	51099	47734	204	
26	SAC	177	315	50044	52080	104	
27	TOR	297	195	51141	49396	168	
28	UTA	245	247	48438	48192	148	
29	WAS	252	240	50207	50099	148	

30 rows × 114 columns

Feature Selection

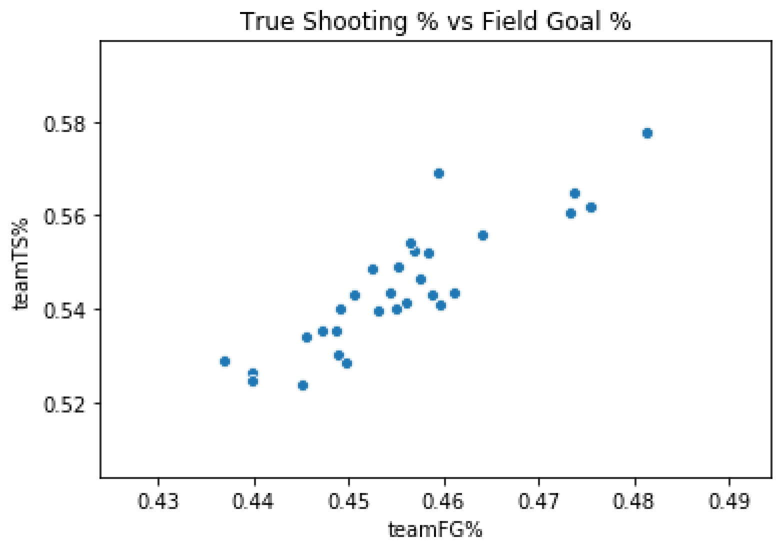
The following graph illustrates that games won and home wins are dire
is the reason that we removed awayWin, awayLoss, homeWin, and ho
GameWon and GameLost were removed because we already have cal
column, and the other features were clearly related to winning percent

In [10]:
```
sns.scatterplot(x=nba["gameWon"], y=nba["homeWin
plt.title("Home wins vs. Games Won Overall");
```



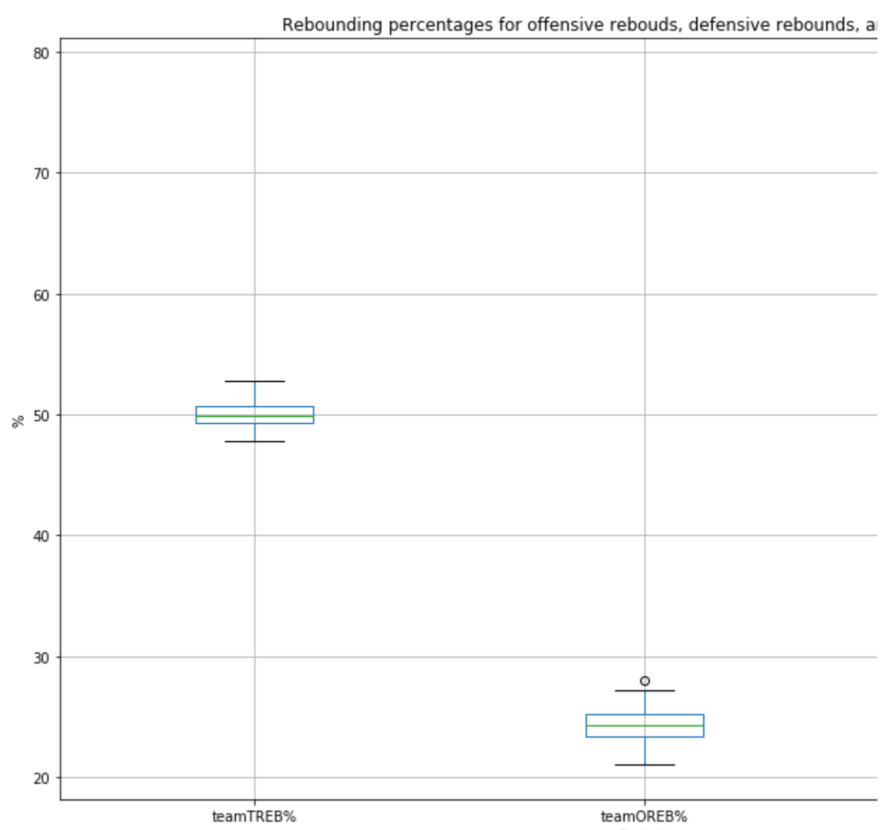
Using several shooting percentage statistical markers would also jeop
model. As the plot below illustrates, stats such as FG% and TS% are
to only select the true shooting percentage stat, which combines toge
percentage, two point percentage, and overall field goal percentage.

In [11]:
```
sns.scatterplot(x=nba["teamFG%"], y=nba["teamTS%
plt.title("True Shooting % vs Field Goal %");
```



As can be seen below, the interquartile ranges for OREB% and DREB
TREB%, leading us to only use TREB% as a feature.

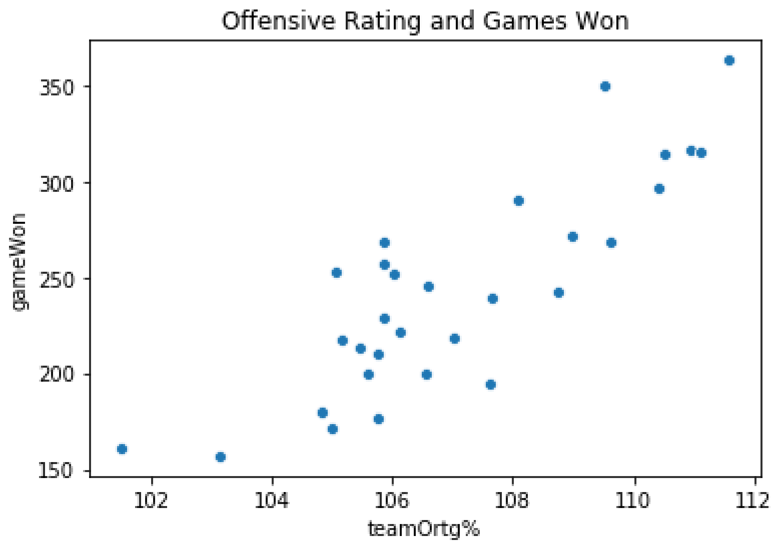
In [12]:
```
plt.figure(figsize=(15,10))
nba[["teamTREB%", "teamOREB%", "teamDREB%"]].box
plt.xlabel("Statistic")
plt.ylabel("%")
plt.title("Rebounding percentages for offensive
```

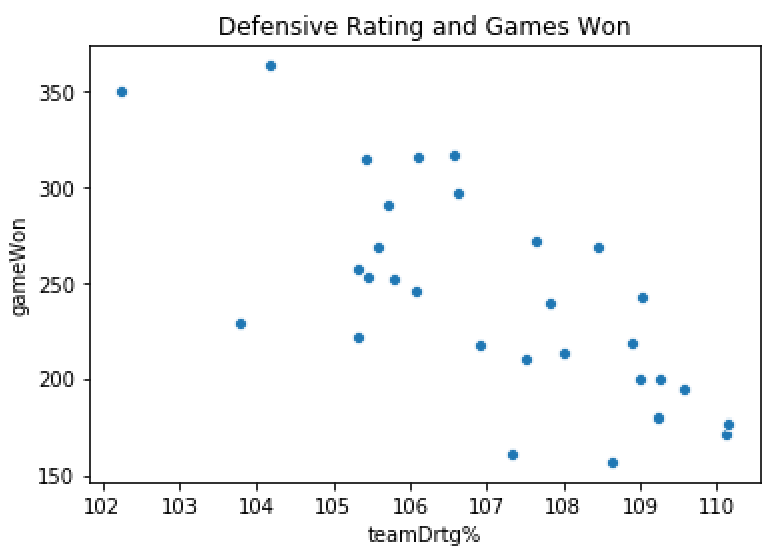


As can be seen below, offensive rating and defensive rating seem to c
percentage. I decided to keep both these features because they tell yo
team offensively and defensively, but do not directly relate to winning p
variable in any way.

In [13]:
```
sns.scatterplot(data = nba, x = 'teamOrtg%', y =
plt.title("Offensive Rating and Games Won")
plt.show()

sns.scatterplot(data = nba, x = 'teamDrtg%', y =
plt.title("Defensive Rating and Games Won")
plt.show()
```





In [14]:

```
#did not select teamEdiff% because we felt it wo
cols_to_use = ["ptsFor", "ptsAgnst", "teamTREB%"
               "teamTO%", "teamSTL%", "teamBLK%",
               "opptDrtg%", "poss%", "pace%", "av

nba_data = nba[cols_to_use]
nba_data

```

Out[14]:

	ptsFor	ptsAgnst	teamTREB%	teamASST%	teamTS%	teamTO%
0	50090	49872	48.501910	64.909878	0.551933	14.06240
1	49564	51198	49.066660	57.976684	0.542834	13.83878
2	42259	41839	49.233940	61.147066	0.540121	13.00685
3	49278	50053	49.585017	57.871976	0.528972	11.64645
4	48797	49035	50.884897	61.105707	0.526511	12.99423
5	51118	50575	50.359937	57.521463	0.552455	12.85404
6	50325	50453	47.840073	58.592095	0.549001	12.10894
7	52119	52260	51.437041	59.782251	0.546351	13.37294
8	49322	50083	50.896791	55.563472	0.523561	12.47380
9	54089	50468	50.807512	65.005040	0.577870	13.62152
10	53445	51320	50.548748	58.928676	0.569378	14.11474
11	41566	40944	50.719691	56.454443	0.539666	13.35802
12	52307	49960	49.759431	59.896971	0.564998	12.58994
13	50315	52773	49.161499	57.015367	0.533863	13.25713
14	48115	48033	50.373551	57.552933	0.530398	12.90685
15	49724	48641	49.575494	57.092192	0.560467	13.61853
16	49311	50452	48.894528	60.431702	0.540818	13.77465
17	50670	51564	49.952166	59.967525	0.541153	12.98208
18	50168	51028	49.706365	58.504467	0.543017	12.80516
19	49002	50301	49.561438	56.009931	0.535425	12.74713
20	52521	50118	52.762807	54.595635	0.555994	13.52357
21	48615	51170	48.933063	58.285206	0.528494	13.33370
22	48729	51533	48.750468	61.286530	0.524599	14.60317
23	50453	52617	49.664836	53.688707	0.535359	14.08543
24	51311	50659	50.634117	55.547467	0.548579	12.81658
25	51099	47734	50.868577	61.349135	0.562036	12.94671
26	50044	52080	49.793173	56.362162	0.540216	13.79387
27	51141	49396	50.422213	54.816162	0.554174	12.41704
28	48438	48192	51.257451	56.610987	0.543626	13.91798

	ptsFor	ptsAgnst	teamTREB%	teamASST%	teamTS%	teamTO%
29	50207	50099	50.046511	60.975315	0.543577	13.49258

Training Validation Split

We are now ready to get to work! The first thing we have to do is perfc
we can preserve some data for our testing. The data we downloaded i
available for both training models and testing the models that we train
the training data into separate training and testing datsets.

In [15]:
```
#standardize the data
standardized = (nba_data-nba_data.mean())/(nba_d
standardized
```

Out[15]:

	ptsFor	ptsAgnst	teamTREB%	teamASST%	teamTS%	teamT
0	0.110073	0.021764	-1.478276	2.310953	0.547406	1.25
1	-0.092889	0.528073	-0.920994	-0.186686	-0.124047	0.91
2	-2.911594	-3.045488	-0.755927	0.955424	-0.324237	-0.34
3	-0.203245	0.090876	-0.409492	-0.224406	-1.147005	-2.41
4	-0.388844	-0.297829	0.873198	0.940525	-1.328645	-0.36
5	0.506737	0.290192	0.355180	-0.350676	0.585954	-0.58
6	0.200750	0.243608	-2.131361	0.035012	0.331073	-1.71
7	0.892982	0.933578	1.418041	0.463758	0.135469	0.20
8	-0.186267	0.102331	0.884935	-1.056030	-1.546283	-1.15
9	1.653126	0.249336	0.796837	2.345235	2.461450	0.58
10	1.404632	0.574656	0.541495	0.156263	1.834800	1.33
11	-3.178995	-3.387227	0.710177	-0.735064	-0.357799	0.18
12	0.965524	0.055366	-0.237385	0.505086	1.511568	-0.98
13	0.196891	1.129457	-0.827409	-0.532994	-0.786050	0.03
14	-0.652000	-0.680424	0.368614	-0.339339	-1.041756	-0.49
15	-0.031152	-0.448270	-0.418889	-0.505318	1.177191	0.58
16	-0.190512	0.243227	-1.090850	0.697719	-0.272838	0.81
17	0.333872	0.667823	-0.047198	0.530502	-0.248120	-0.38
18	0.140170	0.463162	-0.289748	0.003445	-0.110502	-0.65
19	-0.309742	0.185570	-0.432759	-0.895196	-0.670796	-0.74
20	1.048098	0.115695	2.726275	-1.404687	0.847090	0.43
21	-0.459070	0.517382	-1.052825	-0.075543	-1.182253	0.14
22	-0.415082	0.655986	-1.233005	1.005665	-1.469697	2.07
23	0.250140	1.069892	-0.330729	-1.731402	-0.675671	1.29
24	0.581208	0.322266	0.625735	-1.061796	0.299920	-0.63
25	0.499405	-0.794591	0.857095	1.028218	1.293015	-0.43
26	0.092323	0.864848	-0.204089	-0.768307	-0.317206	0.84
27	0.515611	-0.159987	0.416632	-1.325244	0.712787	-1.24
28	-0.527367	-0.619712	1.240825	-0.678669	-0.065610	1.03
29	0.155218	0.108440	0.045899	0.893552	-0.069210	0.39

In [16]:
```
#Perform the train test split
nba_train_x, nba_test_x, nba_train_y, nba_test_y
```

In [17]:
```
def rmse(actual_y, predicted_y):
    '''
    Args:
        actual_y: the actual y column of our dat
        predicted_y: the y values our model pred

    Returns:
        the root mean squared error between our
    '''
    return np.sqrt(np.mean(((actual_y - predicte


def cross_validate_rmse(model, X, y):
    '''
    Args:
        model: the model we are using to predict
        X: the X values we are using to predict
        y: the actual Y values we are trying to

    Returns:
        Performs a 5 Fold cross validation, and
    '''
    model = clone(model)
    five_fold = KFold(n_splits=5)
    rmse_values = []
    for tr_ind, va_ind in five_fold.split(X):
        model.fit(X.iloc[tr_ind,:], y.iloc[tr_in
        rmse_values.append(rmse(y.iloc[va_ind],
    return np.mean(rmse_values)
```

Model One

Our first model uses all the features we previously selected in the "Fea

In [18]:
```
train_errors = []
cv_errors = []
```

In [19]:
```
x1 = cols_to_use
nba_train_x1 = nba_train_x[x1]
nba_test_x1 = nba_test_x[x1]
```

In [20]:
```
model = LinearRegression()
model.fit(nba_train_x1, nba_train_y)

nba_train_y_pred = model.predict(nba_train_x1)
```

In [21]:
```
train_error1 = rmse(nba_train_y, nba_train_y_pre
train_error1

cv_error1 = cross_validate_rmse(model, nba_train
cv_error1
```

Out[21]: 0.03254158635466488

In [22]:
```
models = {"All features":model}
train_errors.append(train_error1)
cv_errors.append(cv_error1)
```

Model Two

We took out average age, pace, and possesion. We concluded that th
not seem to be indicative of a teams winning percentage because a te
young or old. Pace and possession times/averages also felt like they c
can win games playing a fast or slow style.

In [23]:
```
x2  = ["ptsFor", "ptsAgnst", "teamTREB%", "teamA
       "teamTO%", "teamSTL%", "teamBLK%", "teamO
       "opptOrtg%", "opptDrtg%"]

nba_train_x2 = nba_train_x[x2]
nba_test_x2 = nba_test_x[x2]
```

In [24]:
```
model2 = LinearRegression()
model2.fit(nba_train_x2, nba_train_y)

nba_train_y_pred2 = model2.predict(nba_train_x2)
```

In [25]:
```
train_error2 = rmse(nba_train_y, nba_train_y_pre
train_error2

cv_error2 = cross_validate_rmse(model2, nba_trai
cv_error2
```

Out[25]: 0.02615719132886239

In [26]:
```
models["no_age+poss+pace"] = model2
train_errors.append(train_error2)
cv_errors.append(cv_error2)
```

Model Three

After further thought,we decided that points for and points against ma
efficiency of the model. Statistics such as assist percentage, offensive
help to emphasize the effiency of a team. The amount of points scored

points scored on a team could be related to pace of play. If a team sco
they are likely going to give up more points through more defensive pc
reasons, in our third model we decided to eliminate the "ptsFor" and "

In [27]:
```
x3 = ["teamTREB%", "teamASST%", "teamTS%",
      "teamTO%", "teamSTL%", "teamBLK%", "teamOr
      "teamDrtg%", "opptTS%", "opptOrtg%", "oppt

nba_train_x3 = nba_train_x[x3]
nba_test_x3 = nba_test_x[x3]
```

In [28]:
```
model3 = LinearRegression()
model3.fit(nba_train_x3, nba_train_y)
nba_train_y_pred3 = model3.predict(nba_train_x3)
```

In [29]:
```
train_error3 = rmse(nba_train_y, nba_train_y_pre
train_error3

cv_error3 = cross_validate_rmse(model3, nba_trai
cv_error3
```

Out[29]: 0.020150047765033745

In [30]:
```
models["model2_no_pts"] = model3
train_errors.append(train_error3)
cv_errors.append(cv_error3)
```

Training Error vs CV Error

As can be seen below, the first model has the lowest training error, bu
validation error, which led us to believe that the training data was bein
As a result, we removed several features (as explained earlier) and trie
were comfortable with the balance of the CV and training errors. It was
training data went down because we were concerned with how the mo
features and not overfitting the training data.

In [31]:
```
names= np.array(["Model 1", "Model 2", "Model 3"
error_df = pd.DataFrame({"Train Errors":np.array
error_df
```

Out[31]:

	Train Errors	CV Errors	Name
0	0.010352	0.032542	Model 1
1	0.012187	0.026157	Model 2
2	0.012454	0.020150	Model 3

In [32]:
```
figure = px.scatter(error_df, x="Train Errors",
figure.update_traces(textposition='top center')
```

Running our final model on the test data

Now that we are satisfied with our model, it is time to run the model or

In [33]:
```
nba_test_y_pred = model3.predict(nba_test_x3)
nba_test_y_pred
```

Out[33]:
```
array([0.71907456, 0.3408341 , 0.51735404, 0.3230970
       0.63019533])
```

In [34]:
```
test_error = rmse(nba_test_y, nba_test_y_pred)
test_error
```

Out[34]:
```
0.018201030522125333
```

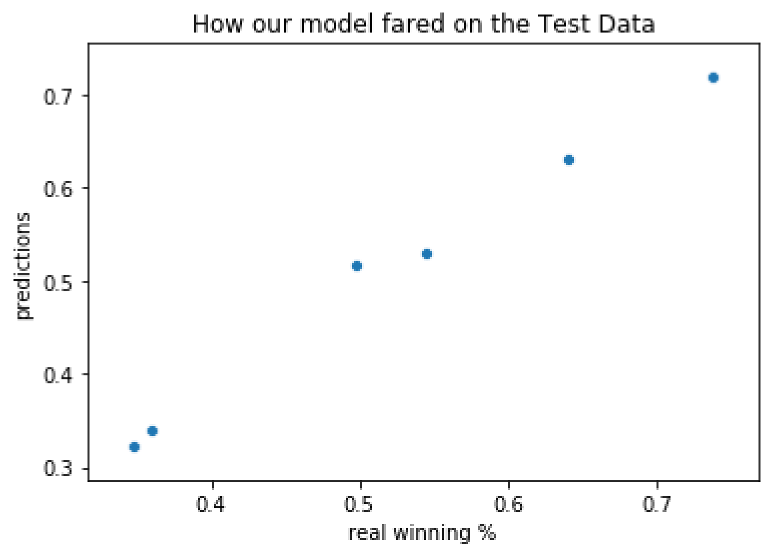
As can be seen below, our model is fairly accurate in predicting the ac gets winning percentages correct to within a couple percent of the act an 82 game season, this means that the accuracy of our model is only

In [35]:
```
results = pd.DataFrame(nba_test_y).rename(column
results["predictions"] = nba_test_y_pred
results
```

Out[35]:

	real winning %	predictions
9	0.737805	0.719075
26	0.359756	0.340834
28	0.497967	0.517354
13	0.347561	0.323097
5	0.544715	0.530450
12	0.640244	0.630195

In [36]:
```
sns.scatterplot(x="real winning %", y="predictio
plt.title("How our model fared on the Test Data"
```



Conclusion

When applying our work to the future, it is easy to see how valuable it nba team could use our model halfway through the season to predict t based on their current statistics. In addition, with more data collected, to winning in the playoffs. We had considered modeling playoff win pe decided that the sample size of 2012-2018 was too small given how fe by each team. In the future, weighting in the value of coaches, how mu its players, and how many individual star players a team has are very a we have accomplished.