

---

# Application of Classification Models for Predicting Shooting Performance in the NBA

A Case Study of Kobe Bryant

---

Allen Crane

Brock Friedrich

Examiner: Dr. Anthony Tanaydin

Southern Methodist University

Data Science

Applied Statistics

December 1<sup>th</sup>, 2018

**Writing period**

10. 24. 2018 – 12. 1. 2018

# Abstract

In a world with continuously increasing technological capabilities, the demand for answers increases alongside it. The domain of professional sports in particular is no exception to this trend. The very essence of competitive sports is rooted in a desire to optimize and gain every advantage over competitors. In this paper, we describe classification methodologies for predicting the performance of professional athletes through a scoped analysis of the long-time NBA allstar, Kobe Bryant. We detail three simulations modelling shooting accuracy over Bryant's 20 year career, using logistic regression and discriminant analyses to yield predictions about Bryant's theoretical future performance. We show that Bryant's potential to score is strongly dependent on his distance from the basket when shooting. We also show that Bryant's scoring performance in the post-season, as compared to that of the regular season, is consistent with his performance at any other point in the season.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Approach</b>	<b>3</b>
2.1	Problem Definition . . . . .	3
2.2	Exploratory Analysis . . . . .	5
2.2.1	Data Overview . . . . .	5
2.2.2	Multicollinearity . . . . .	8
<b>3</b>	<b>Methodology</b>	<b>11</b>
3.1	Feature Selection . . . . .	11
3.2	Model Selection . . . . .	11
3.3	Evaluation . . . . .	14
<b>4</b>	<b>Conclusion</b>	<b>15</b>
<b>5</b>	<b>Limitations and Future Directions</b>	<b>19</b>
<b>6</b>	<b>Appendix</b>	<b>21</b>
6.1	Github Repository . . . . .	21
6.2	Appendix A . . . . .	21
6.2.1	Model Comparison - Continued . . . . .	21
6.2.2	Data Dictionary . . . . .	21
6.3	SAS Code . . . . .	22
6.4	Python Code . . . . .	41



# List of Figures

1	Market Analysis . . . . .	4
2	Plots of y_loc . . . . .	7
3	Boxplot of Shot Distance . . . . .	7
4	Correlation Matrix - All features . . . . .	9
5	Correlation Matrix - Candidate Model . . . . .	12
6	Model Composition Chart . . . . .	13
7	Predicted Shot Probability over Distance to Hoop . . . . .	16
8	Performance in Regular and Post Season . . . . .	17





# List of Tables

1	Feature Summary . . . . .	5
2	Key Feature Descriptions . . . . .	6
3	Variance Inflation Factors . . . . .	8
4	Logistic Model Summary . . . . .	11
5	Log-odds with Respect to Distance . . . . .	15
6	Odds with Respect to Distance . . . . .	18
7	Percent Change over Distance . . . . .	18



# 1 Introduction

With modern computers and the popularity of the budding discipline of data science, there are numerous predictive technologies readily available. In professional sports alone, a host of disparate entities, players, organizations, and gamblers alike are hungry for new insights. Each bears an economic imperative to seek advantages and optimizations that will give them an edge over their peers. Technology, however, does not predicate useful and effective models. It is the rigorous application of sound statistical theory that yields actionable conclusions. Unfortunately, the appeal of novel algorithms and the sparkles of big data detract from the importance of a strong statistical foundation. Thus, we seek to right the ship of this rampant malpractice as it applies to this problem domain. In what follows, we explore insights we have garnered from the application of linear discriminant analysis and logistic regression, then leveraging those insights to appropriately tune the final predictive model.

Kobe Bryant marked his retirement from basketball by scoring 60 points in his final game as a member of the Los Angeles Lakers team on Wednesday, April 13, 2016. Starting to play professional basketball at the age of 17, Kobe earned the sport's highest accolades throughout his long career. Using 20 years of data on Kobe's shots made and shots missed, we wish to predict which shots will be successful.



## 2 Approach

### 2.1 Problem Definition

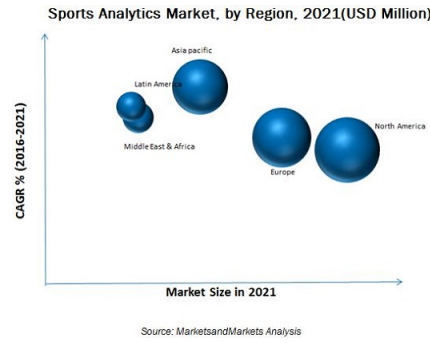
Sports analytics is, as many previously low-tech markets are, poised to boom as sports organizations are increasingly able to generate more data. Recent research

"The Sports Analytics market is expected to grow from USD 123.7 Million in 2016 to USD 616.7 Million by 2021, at a Compound Annual Growth Rate (CAGR) of 37.9

The increasing volume of on-field and off-field data generated among various sports organizations has led to an increase in managing these data to analyze them. This need is driving the adoption of sports analytics solutions. Analytics and big data technologies have found huge potential in various industries, including sports. The increasing demand of coaches, mentors, and other management officials for real-time access to the insights of relevant information presents huge potential for sports analytics market. The demand for cloud-based sports analytics solutions is also expected to increase due to the lack of budget allocation for hiring technical skills and experts to analyze data for sports organization. These are some of the major factors expected to augment the growth of the market. Moreover, in order to remain competitive, organizations are adopting sports analytics solutions."

[1]

This study was birthed out of a need to reliably quantify three key metrics specifically, each used in assessing a professional basketball player's shooting ability. Those metris are:



**Figure 1:** Market Analysis

1. Odds of making a shot as distance from the basket increases.
2. Linearity of the decline rate of the probability of making a shot with respect to the distance the shot was taken from the basket.
3. The relationship between the distance from the shooter to the basket and the odds of the shot being made is different when in the regular season verses the post-season.

Rephrasing those metrics as questions helps clarify the focus of the analysis as it relates to Kobe Bryant. Those questions are as follows:

1. Do the odds of Kobe making a shot decrease with respect to distance he is from the hoop?
2. Does the probability of Kobe making a shot decrease linearly with respect to the distance he is from the hoop?
3. Is the relationship between the distance Kobe is from the basket and the odds of him making the shot different if they are in the playoffs.

To appropriately answer the questions of interest above, we must fit a series of classifier models to the training portion of our dataset, iteratively scoring and comparing the various models against one another so that we can tune their parameters and hone in on a final feature set.

The entirety of the analysis is implemented side-by-side in Python and SAS. Below, we set the stage for conducting our analysis through an exploratory analysis of the dataset.

## 2.2 Exploratory Analysis

### 2.2.1 Data Overview

The NBA has provided a comprehensive dataset of every shot Kobe Bryant took throughout his career. Accompanying each shot record are a number of supplementary features that provide context to the positioning and environment in which a given shot was taken. Generally, the data was remarkably clean and readily manipulable. The majority of the wrangling effort was focused on synthesizing new predictors. A brief summary of the features is below.

	count	mean	std	min	25%	50%	75%	max
game_event_id	25697	249	150	2	111	253	367	653
game_id	25697	24741091	7738108	20000012	20500064	20900337	29600270	49900088
lat	25697	34	0	33	34	34	34	34
loc_x	25697	7	110	-250	-67	0	94	248
loc_y	25697	91	88	-44	4	74	160	791
lon	25697	-118	0	-119	-118	-118	-118	-118
minutes_remaining	25697	5	3	0	2	5	8	11
period	25697	3	1	1	1	3	3	7
playoffs	25697	0	0	0	0	0	0	1
seconds_remaining	25697	28	18	0	13	28	43	59
shot_distance	25697	13	9	0	5	15	21	79
shot_made_flag	25697	0	0	0	0	0	1	1
team_id	25697	1610612747	0	1610612747	1610612747	1610612747	1610612747	1610612747
shot_id	25697	15328	8860	2	7646	15336	22976	30697
attendance	25697	15041	1076	11065	14314	15048	15738	20845
arena_temp	25697	70	2	64	69	70	71	79
avgnoisedb	25697	95	2	89	93	95	96	102

**Table 1:** Feature Summary

From the available features, the variable "shot\_made\_flag" indicates whether a

particular shot was made (1) or missed (0), and is the feature that acts as our endogenous variable. The remainder of the features are the potential predictors, or exogenous variables. Since there are a manageable number of them, we can simply generate descriptive plots for each exogenous feature. Univariate distribution or frequency plots are ideal for visualizing the underlying distribution of each variable. In addition, boxplots for each exogenous feature against each level of the endogenous variable allowed us to develop our intuition about the variance within each level of the target feature.

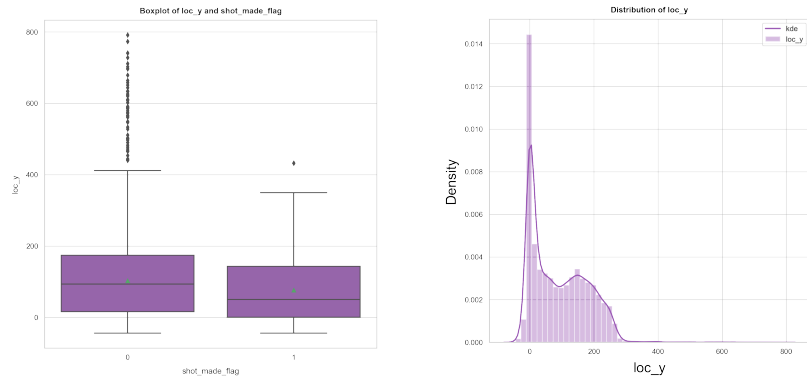
Most of the features show no signs of skewness or severe departures from normality, however, there are some exceptions.

It's clear that outliers are prevalent in some of the features, particularly those representing the distance to the hoop and the time remaining, as represented in figures 2b and 2a. Several solutions were explored to mitigate the potential impact of the outliers, none of which yielded a model with better results. Thus, all outliers were included in the final models on their original scale.

**Table 2: Key Feature Descriptions**

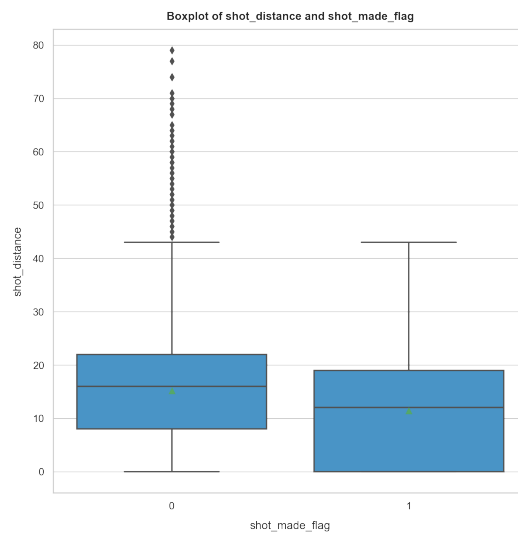
Game_event_id	Identification variable
Game_id	Identification variable
Lat and loc_y	Appear to be the same data (y-axis), based on location on the court. The tall bar indicates the frequency of shots taken at the location near the basket
Loc_x and lon	Appear to be the same data (x-axis), based on location on the court. The tall bar indicates the frequency of shots taken at the location near the basket
Minutes_remaining	Suggests that there is evidence that Kobe took more shots as the period progressed (higher shot counts closer to the end of the period)
Period	Indicates similar shot frequency across periods
Playoffs	Playoffs were a rare event in the season, hence the lower shot frequency overall
Season	Shot frequency was lower in 2 seasons
Seconds_remaining	Suggests that there is evidence that Kobe took more shots in the final seconds (higher shot counts closer to the end of the last minute in the period, correlates with shot frequency in the minutes_remaining variable above)
Shot_distance	High shot frequency at the basket, at the top of the key (field goals), and the three-point line. Shot frequency was comparatively rarer beyond the three-point line





(a) Boxplot of Shot Y Location      (b) Distplot of Shot Y Location

**Figure 2:** Plots of  $y\_loc$



**Figure 3:** Boxplot of Shot Distance

The exploratory boxplot of shot\_distance (figure 3) already hints at a potential relationship between a shot’s likelihood of being made verses the distance the shot was taken from the hoop.

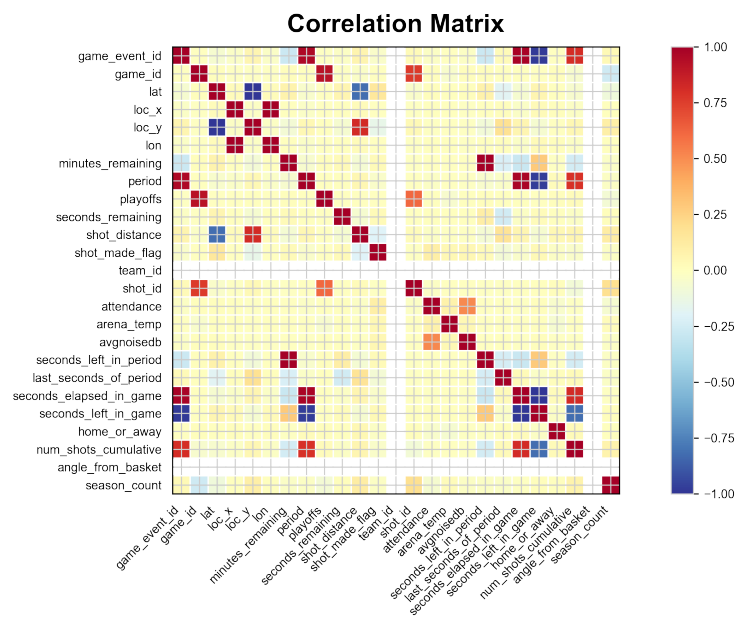
### 2.2.2 Multicollinearity

The base feature set was wrought with violations of independent variability. As noted in the table and it’s accompanying correlation matrix, approximately half of the features showed a collinear relationship with other features in the data set. This caused our approach to feature selection vary from the normal methods. We chose to ignore the more nuanced stepwise / LASSO / LarsCV methods and instead implemented a procedure to recursively drop a feature with an exorbitant variance inflation factor on each iteration until reaching a feature set that showed no multicollinearity violations. We used an elimination cutoff of the initial median VIF + 2 standard deviations. Variance inflation factors for the original features can be seen in table 3. A correlation matrix of the same feature set is displayed in figure 4.

[2]

	VIF Factor	features
0	0	Intercept
1	35	game_event_id
2	68	game_id
3	inf	lat
4	inf	loc_x
5	inf	loc_y
6	inf	lon
7	inf	minutes_remaining
8	inf	period
9	29	playoffs
10	inf	seconds_remaining
11	3	shot_distance
12	0	team_id
13	14	shot_id

**Table 3:** Variance Inflation Factors



**Figure 4:** Correlation Matrix - All features



## 3 Methodology

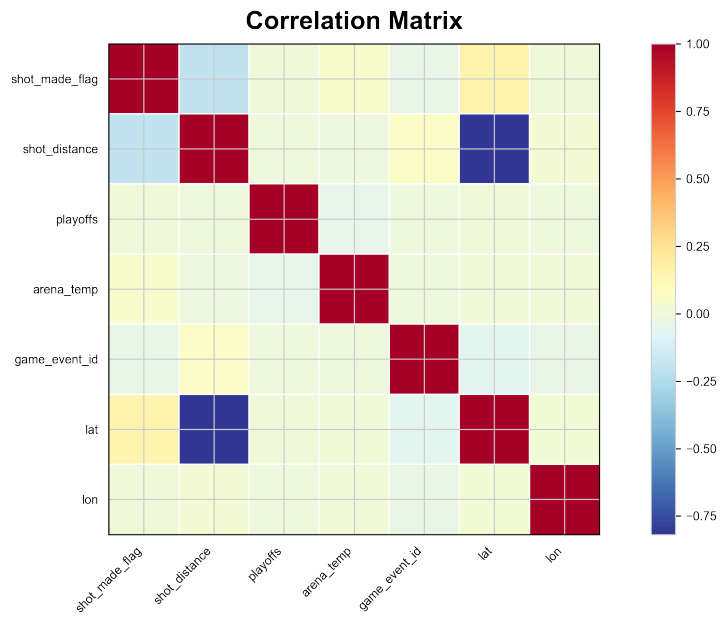
### 3.1 Feature Selection

As mentioned in the previous section, the feature selection process was dominated by the prevalent multicollinearity between many of the features. As example, collinearity was reduced to a tolerable range in the featureset of one candidate model pictured in figure 5.

### 3.2 Model Selection

Our key observation during model selection arose in differentiating the performance of Fisher's linear discriminant and binary logistic regression. While both procedures are common choices for two-level classification, and often yield similar results, there are a few critical assumptions that differ between them. The linear discriminant analysis (LDA) mandates the within-group covariance between each level of the endogenous variable be equal. Additionally, the LDA model is susceptible to the influence of extreme observations, and will potentially yield inconsistent results when outliers are not accounted for. Conversely, the logarithmic analysis has no exigence on the form of the model's predictors. The prevalence of outliers in the dataset, exposed during the exploratory analysis, was sufficient enough cause to move forward with the logistic regression as a conservative base on which we could build our predictions. Beta models using LDA can be found in the appendix (6.2).

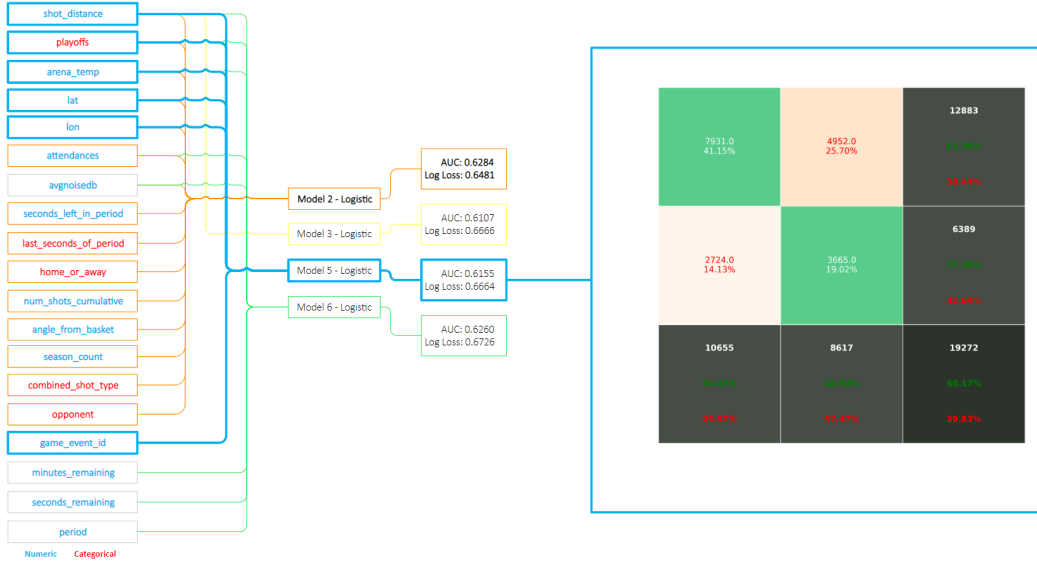
Several iterations of model selection concluded on a final model that yielded a 0.615 AUC, with exceptional sensitivity (true positive rate) and moderate specificity (true negative rate). (table 4). Categorical predictors in the final model were substituted for dummied predictors for estimate. (See regression formula - 4)



**Figure 5:** Correlation Matrix - Candidate Model

Model 5	
Log Loss	0.6664
AUC	0.6155
Sensitivity	0.744
Specificity	0.425

**Table 4:** Logistic Model Summary



**Figure 6:** Model Composition Chart

$$\text{Logit}\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 \text{shot\_distance} + \beta_2 \text{playoffs}_0 + \beta_3 \text{playoffs}_1 + \beta_4 \text{arena\_temp} + \beta_5 \text{game\_event\_id} + \beta_6 \text{lat} + \beta_7 \text{lon}$$

Regular Season (playoffs = 0)

$$\begin{aligned} &= \beta_0 + \beta_1 \text{shot\_distance} + \beta_2(1) + \beta_3(0) + \beta_4 \text{arena\_temp} + \beta_5 \text{game\_event\_id} + \beta_6 \text{lat} + \beta_7 \text{lon} \\ &= \beta_0 + \beta_1 \text{shot\_distance} + \beta_2 + \beta_4 \text{arena\_temp} + \beta_5 \text{game\_event\_id} + \beta_6 \text{lat} + \beta_7 \text{lon} \end{aligned}$$

$$\text{Logit}\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7 \quad (1)$$

Playoffs (playoffs = 1)

$$\begin{aligned} &= \beta_0 + \beta_1 \text{shot\_distance} + \beta_2(0) + \beta_3(1) + \beta_4 \text{arena\_temp} + \beta_5 \text{game\_event\_id} + \beta_6 \text{lat} + \beta_7 \text{lon} \\ &= \beta_0 + \beta_1 \text{shot\_distance} + \beta_3 + \beta_4 \text{arena\_temp} + \beta_5 \text{game\_event\_id} + \beta_6 \text{lat} + \beta_7 \text{lon} \end{aligned}$$

$$\text{Logit}\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 x_1 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7$$

A model composition chart, displayed in figure 6 shows the features selected in the champion model, along with the model's confusion matrix.

[2]

A full listing of candidate models can be seen in the appendix (6.2).

### 3.3 Evaluation

Goodness of fit measured by logarithmic loss function, where:

$$LogLoss = -(y \log(p) + (1 - y) \log(1 - p))$$

Logarithmic loss (related to cross-entropy) measures the performance of a classification model where the prediction input is a probability value between 0 and 1. Log Loss takes into account the uncertainty of a prediction based on how much it varies from the actual label, instead of simply counting if the predicted value exactly equals the true value, as is the case with accuracy. This gave us a more nuanced view into the performance of our model.



## 4 Conclusion

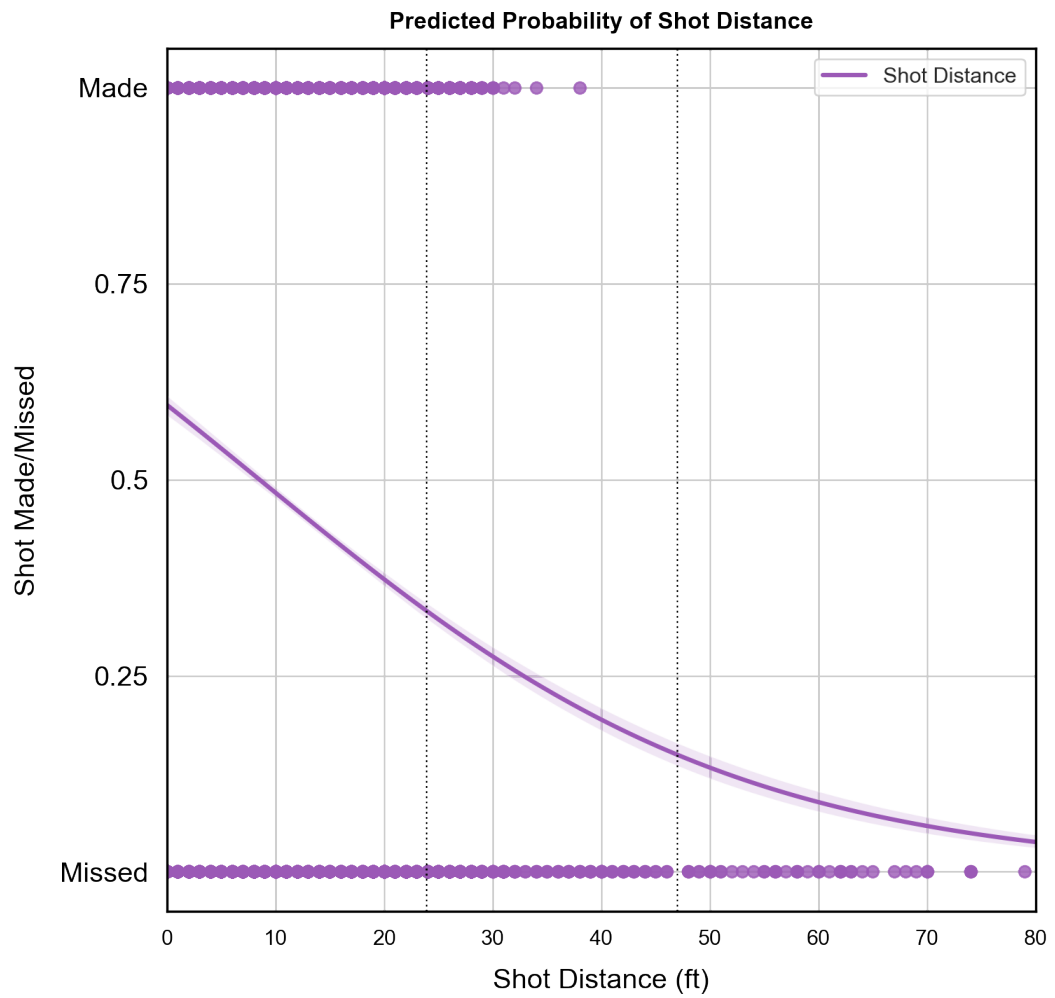
Finally, now that we've demonstrated the process of applying binary classification techniques to NBA shooting data, we can return to the original questions of interest. Our findings show a  $-1.87\% \pm 0.85\%$  ( $-2.72\%, -1.01\%$ ) step change in Kobe Bryant's shot making ability, measured with a p-value of  $< 0.001$  at  $\alpha = 0.05$  (figure 7). That is to say, for every additional foot between Bryant and the basket, his accuracy is reduced by just under 2%. Another, perhaps more familiar way to put it is in terms of odds. Kobe's odds of making a basket are

$$0.9813 \pm 0.0084 \text{ (0.9728, 0.9899)}$$

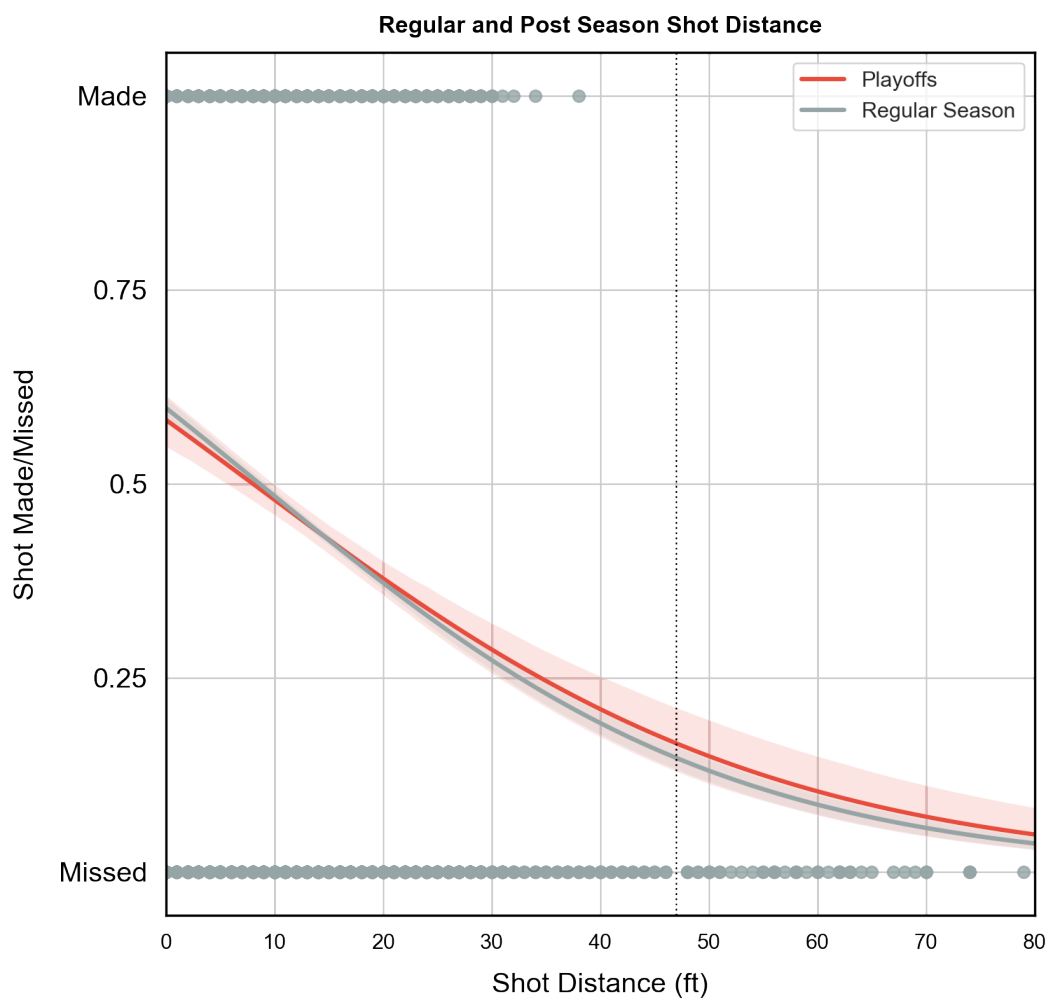
with respect to his distance from the basket.

Additionally, the model indicates a  $4.25\% \pm 9.9\%$  ( $-4.77\%, 14.15\%$ ) increase in shooting ability during the playoffs, however, the result was not statistically significant, with a p-value of 0.3671 at an  $\alpha$  level of 0.05 on the  $\chi^2$  distribution. There is not sufficient evidence to reject the null hypothesis that Kobe's performance shows no notable difference between regular and post season. (table 8)

Figure tables of Log-odds, odds, and percentage change in shooting ability with respect to distance:



**Figure 7:** Predicted Shot Probability over Distance to Hoop



**Figure 8:** Performance in Regular and Post Season

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
lat	0	0	-2	0	0	0
lon	0	0	-0	1	0	153130194710622
playoffs	71	0	1	0	0	1400174
seconds_remaining	1	0	2	0	1	1
shot_distance	0	0	-4	0	0	0
attendance	1	0	10	0	1	1
arena_temp	28	0	4	0	6	133
avgnosedb	1	0	0	1	0	6
seconds_left_in_period	1	0	1	0	1	1
last_seconds_of_period	0	0	-7	0	0	0
seconds_left_in_game	1	0	2	0	1	1
home_or_away	0	0	-1	0	0	32
num_shots_cumulative	1	0	0	1	0	2
angle_from_basket	1	0	0	1	1	1
season_count	3	0	3	0	1	5

**Table 5:** Log-odds with Respect to Distance

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
lat	0	0	-2	0	0	0
lon	0	0	-0	1	0	153130194710622
playoffs	71	0	1	0	0	1400174
seconds_remaining	1	0	2	0	1	1
shot_distance	0	0	-4	0	0	0
attendance	1	0	10	0	1	1
arena_temp	28	0	4	0	6	133
avgnosedb	1	0	0	1	0	6
seconds_left_in_period	1	0	1	0	1	1
last_seconds_of_period	0	0	-7	0	0	0
seconds_left_in_game	1	0	2	0	1	1
home_or_away	0	0	-1	0	0	32
num_shots_cumulative	1	0	0	1	0	2
angle_from_basket	1	0	0	1	1	1
season_count	3	0	3	0	1	5

**Table 6:** Odds with Respect to Distance

	Coef.	[0.025	0.975]
lat	-100	-100	-99
lon	-99	-100	15313019471062054
playoffs	6978	-99	140017285
seconds_remaining	15	-3	37
shot_distance	-85	-93	-64
attendance	2	1	2
arena_temp	2699	505	13151
avgnoisedb	28	-72	499
seconds_left_in_period	1	-1	2
last_seconds_of_period	-100	-100	-100
seconds_left_in_game	1	0	1
home_or_away	-92	-100	3078
num_shots_cumulative	5	-53	140
angle_from_basket	2	-6	10
season_count	154	31	394

**Table 7:** Percent Change over Distance



## 5 Limitations and Future Directions

The primary (and obvious) limitation to this study is the narrow subject pool from which the data was gathered. The data, being observational in nature and from a single athlete's history, allows us no ability to draw conclusions that accurately reflect a causal relationship between the predictors and the outcome variable. Additionally, the absence of a random sampling mechanism would make any inferrening our conclusions to any population, other than Kobe Bryant himself, highly suspect. Even with such a limited scope, the conclusions produced from our study serves an important purpose.

Our conclusions, presented in the previous section, yielded strong evidence of Kobe's ability to make shots dwindling as his distance from the basket grew. The general principle behind this conclusion seems obvious, and it is. Of course an athlete is going to see diminished shooting accuracy as they take longer shots. What is significant about our results is not even the quantification of that relationship between accuracy and distance directly, but it is the proof that such a relationship is reliably quantifiable and, holding all other variables constant, interpretable in a 2-dimensional space. These findings build a foundation on which future studies will be able to design experiments and investiagte broader populations. Future directions for our work may be investigating similar trends across NBA players of different vintages, skill levels, positions, and team compositions. Such an experiment, with the appropriate design, would provide definitive insight into the extensibility of our findings in this analysis.





## 6 Appendix

### 6.1 Github Repository

We are on Github! (<https://github.com/la-mar/Applied-Stats-Project-2>)  
All code used in our analysis is included in the appendix below, but we recommend viewing it in its original form in our github repository.

### 6.2 Appendix A

#### 6.2.1 Model Comparison - Continued

	Model 2	Model 3	Model 4A	Model 4B	Model 5	Model 6
	LOGIT w/PCA	LOGIT w/shot_distance	LOGIT w/shot_distance, (playoffs)	LOGIT w/shot_distance, (no playoffs)	LOGIT w/selected vars	LOGIT w/selected vars
R-Square	0.0498	0.0393	0.0338	0.0402	0.042	0.05
Max Residual S-Square	0.0006	0.0028	0.0039	0.0043	0.0062	0.0009
AIC	34035.47	34298.864	3442.4	29258.347	34237.279	34034.718
SC	34035.368	34303.172	5564.858	29278.109	34294.388	34038.100
-2 Log L	34033.37	34298.864	5038.4	29258.347	34233.279	34035.718
Area Under ROC Curve	0.6282	0.6107	0.6008	0.6124	0.6155	0.628
Log Lik.		0.00002			0.0004	0.0235

	Model 1
	GLM w/PCA
Root MSE	0.48463
Dependent Mean	0.44616
R-Square	0.0498
Adj R-Sq	0.0495
AIC	-11521
AICC	-11521
SBC	-37155
CV PRESS	6037.80515

#### 6.2.2 Data Dictionary

## 6.3 SAS Code

```
1  MSDS 6371 - Applied Statistics */
2  Allen Crane and Brock Friedrich */
3  Kobe Bryant Shot Selection      */
4  November 2018                  */
5
6  import data */
7  oc import datafile="c:\users\allen\documents\smu data science\MSDS
   ↪ 6372 - Applied Statistics\project 2\project2Data.csv"
8      dbms=dlm out=train replace;
9      delimiter=',';
10     getnames=yes;
11 n;
12
13 print data */
14 oc print data=train (obs=10);
15 n;
16
17 investigate data for variable types*/
18 oc contents data=train;
19 n;
20
21 look for missing numeric data - may need to impute */
22 oc means data = train n nmiss;
23 var _numeric_;
24 n;
25
26 investigate means of training data and any missing values */
27 oc means data = train n nmiss;
```

	Model 7
	LDA
Sensitivity (classified 1 when really 1)	0.4298
Specificity (classified 0 when really 0)	0.744
Type I Error (classified 1 when really 0)	0.256
Type II Error (classified 0 when really 1)	0.5702
Total Error	0.3962
Accuracy (1-Total Error)	0.6038

```

28 n;
29
30 univariate data analysis */
31 oc univariate data = train;
32 r season;
33 n;
34
35 note that certain "season" fields are missing */
36 ta _season;
37 set train;
38 where missing (season);
39 n;
40
41 print "season" data, where "season" is missing */
42 oc print data=_season (obs=200);
43 n;
44
45
46 more univariate data analysis */
47 s graphics on;
48 oc univariate data = train plot;
49 r recId
50 me_event_id
51 me_id
52 t
53 c_x
54 c_y
55 n
56 nutes_remaining
57 riod
58 ayoffs
59 ason
60 conds_remaining
61 ot_distance
62 ot_made_flag
63 am_id

```

```

64 me_date
65 ot_id
66 tendance
67 ena_temp
68 gnoisedb
69
70 n;
71 s graphics off;
72
73 create a time-based variable, concatenating Period, minutes
  ↪ remaining, and seconds remaining, in descending order. This one
  ↪ is for Periods remaining... */
74 ta train2;
75 set train;
76     if period = 1 then periods_remaining2 = "14";
77     else if period = 2 then periods_remaining2 = "28";
78     else if period = 3 then periods_remaining2 = "42";
79     else if period = 4 then periods_remaining2 = "57";
80     else if period = 5 then periods_remaining2 = "71";
81     else if period = 6 then periods_remaining2 = "85";
82     else if period = 7 then periods_remaining2 = "99";
83     else periods_remaining2 = period;
84 n;
85
86 print data */
87 oc print data=train2 (obs=10);
88 n;
89
90 Minutes remaining... */
91 ta train2;
92 set train2;
93     if minutes_remaining = 11 then minutes_remaining2 = "99";
94     else if minutes_remaining = 10 then minutes_remaining2 = "90";
95     else if minutes_remaining = 9 then minutes_remaining2 = "81";
96     else if minutes_remaining = 8 then minutes_remaining2 = "72";
97     else if minutes_remaining = 7 then minutes_remaining2 = "63";

```

```

98  else if minutes_remaining = 6 then minutes_remaining2 = "54";
99  else if minutes_remaining = 5 then minutes_remaining2 = "45";
100 else if minutes_remaining = 4 then minutes_remaining2 = "36";
101 else if minutes_remaining = 3 then minutes_remaining2 = "27";
102 else if minutes_remaining = 2 then minutes_remaining2 = "18";
103 else if minutes_remaining = 1 then minutes_remaining2 = "09";
104 else if minutes_remaining = 0 then minutes_remaining2 = "00";
105 else minutes_remaining2 = minutes_remaining;
106 n;
107
108 print data */
109 oc print data=train2 (obs=10);
110 n;
111
112 Seconds remaining... */
113 ta train2;
114 set train2;
115     if seconds_remaining = 59 then seconds_remaining2 = "99";
116     else if seconds_remaining = 58 then seconds_remaining2 = "97.3";
117 else if seconds_remaining = 57 then seconds_remaining2 = "95.6";
118 else if seconds_remaining = 56 then seconds_remaining2 = "94";
119 else if seconds_remaining = 55 then seconds_remaining2 = "92.3";
120 else if seconds_remaining = 54 then seconds_remaining2 = "90.6";
121 else if seconds_remaining = 53 then seconds_remaining2 = "88.9";
122 else if seconds_remaining = 52 then seconds_remaining2 = "87.3";
123 else if seconds_remaining = 51 then seconds_remaining2 = "85.6";
124 else if seconds_remaining = 50 then seconds_remaining2 = "83.9";
125 else if seconds_remaining = 49 then seconds_remaining2 = "82.2";
126 else if seconds_remaining = 48 then seconds_remaining2 = "80.5";
127     else if seconds_remaining = 47 then seconds_remaining2 = "78.9";
128 else if seconds_remaining = 46 then seconds_remaining2 = "77.2";
129 else if seconds_remaining = 45 then seconds_remaining2 = "75.5";
130 else if seconds_remaining = 44 then seconds_remaining2 = "73.8";
131 else if seconds_remaining = 43 then seconds_remaining2 = "72.2";
132 else if seconds_remaining = 42 then seconds_remaining2 = "70.5";
133 else if seconds_remaining = 41 then seconds_remaining2 = "68.8";

```

```
134 else if seconds_remaining = 40 then seconds_remaining2 = "67.1";
135 else if seconds_remaining = 39 then seconds_remaining2 = "65.4";
136 else if seconds_remaining = 38 then seconds_remaining2 = "63.8";
137 else if seconds_remaining = 37 then seconds_remaining2 = "62.1";
138 else if seconds_remaining = 36 then seconds_remaining2 = "60.4";
139 else if seconds_remaining = 35 then seconds_remaining2 = "58.7";
140 else if seconds_remaining = 34 then seconds_remaining2 = "57.1";
141 else if seconds_remaining = 33 then seconds_remaining2 = "55.4";
142 else if seconds_remaining = 32 then seconds_remaining2 = "53.7";
143 else if seconds_remaining = 31 then seconds_remaining2 = "52";
144 else if seconds_remaining = 30 then seconds_remaining2 = "50.3";
145 else if seconds_remaining = 29 then seconds_remaining2 = "48.7";
146     else if seconds_remaining = 28 then seconds_remaining2 = "47";
147 else if seconds_remaining = 27 then seconds_remaining2 = "45.3";
148 else if seconds_remaining = 26 then seconds_remaining2 = "43.6";
149 else if seconds_remaining = 25 then seconds_remaining2 = "41.9";
150 else if seconds_remaining = 24 then seconds_remaining2 = "40.3";
151 else if seconds_remaining = 23 then seconds_remaining2 = "38.6";
152 else if seconds_remaining = 22 then seconds_remaining2 = "36.9";
153 else if seconds_remaining = 21 then seconds_remaining2 = "35.2";
154 else if seconds_remaining = 20 then seconds_remaining2 = "33.6";
155 else if seconds_remaining = 19 then seconds_remaining2 = "31.9";
156 else if seconds_remaining = 18 then seconds_remaining2 = "30.2";
157 else if seconds_remaining = 17 then seconds_remaining2 = "28.5";
158 else if seconds_remaining = 16 then seconds_remaining2 = "26.8";
159 else if seconds_remaining = 15 then seconds_remaining2 = "25.2";
160 else if seconds_remaining = 14 then seconds_remaining2 = "23.5";
161 else if seconds_remaining = 13 then seconds_remaining2 = "21.8";
162 else if seconds_remaining = 12 then seconds_remaining2 = "20.1";
163 else if seconds_remaining = 11 then seconds_remaining2 = "18.5";
164 else if seconds_remaining = 10 then seconds_remaining2 = "16.8";
165 else if seconds_remaining = 9 then seconds_remaining2 = "15.1";
166     else if seconds_remaining = 8 then seconds_remaining2 = "13.4";
167 else if seconds_remaining = 7 then seconds_remaining2 = "11.7";
168 else if seconds_remaining = 6 then seconds_remaining2 = "10.1";
169 else if seconds_remaining = 5 then seconds_remaining2 = "08.4";
```

```

170 else if seconds_remaining = 4 then seconds_remaining2 = "06.7";
171 else if seconds_remaining = 3 then seconds_remaining2 = "05";
172 else if seconds_remaining = 2 then seconds_remaining2 = "03.4";
173 else if seconds_remaining = 1 then seconds_remaining2 = "01.7";
174 else if seconds_remaining = 0 then seconds_remaining2 = "00";
175 se seconds_remaining2 = seconds_remaining;
176 n;
177
178 print data */
179 oc print data=train2 (obs=10);
180 n;
181
182 concatenante data */
183 ta train2;
184 t train2;
185 s_remaining = cat(periods_remaining2, minutes_remaining2,
    ↪ seconds_remaining2);
186 n;
187
188 print data */
189 oc print data=train2 (obs=10);
190 n;
191
192 make field numeric (some components contained leading zeroes */
193 ta train2;
194 t train2;
195 n_pms_remaining = input(pms_remaining,8.);
196 n;
197
198 drop original non-numeric concetenanted data field */
199 ta train2;
200 t train2 (drop = pms_remaining);
201 n;
202 ta train2;
203
204 rename new numeric concatenated data field */

```

```

205 t train2 (rename=(
206 _pms_remaining'n='pms_remaining'n));
207 n;
208
209 print data */
210 oc print data=train2 (obs=10);
211 n;
212
213
214
215
216
217 check data - histogram */
218 s graphics on;
219 oc univariate data = train2;
220 r pms_remaining;
221 stogram;
222 n;
223 s graphics off;
224
225 check data - scatter plot */
226 oc sgplot data=train2;
227 scatter x=pms_remaining y=shot_made_flag / group=shot_made_flag;
228 n;
229
230
231
232 transform data - log transformation on shot distance and time
   ↪ remaining */
233 ta train3;
234 t train2;
235 shot_distance = log(shot_distance);
236 pms_remaining = log(pms_remaining);
237 n;
238
239

```



```

240  check data - scatter plot */
241  s graphics on;
242  oc univariate data = train3 plot;
243  r l_shot_distance l_pms_remaining;
244  n;
245  s graphics off;
246
247
248
249
250  correlation analysis */
251  s graphics on;
252  oc corr data=train2 plots=matrix(histogram);
253  r recId
254  me_event_id
255  me_id
256  t
257  c_x
258  c_y
259  n
260  nutes_remaining
261  riod
262  ayoffs
263  ason
264  conds_remaining
265  ot_distance
266  ot_made_flag
267  am_id
268  me_date
269  ot_id
270  tendance
271  ena_temp
272  gnoisedb;
273  n;
274  s graphics off;
275

```

```

276
277
278 principal component analysis */
279 s graphics on;
280 oc princomp plots=all data=train2 cov out=pca;
281 r recId
282 me_event_id
283 me_id
284 t
285 c_x
286 c_y
287 n
288 nutes_remaining
289 riod
290 ayoffs
291 ason
292 conds_remaining
293 ot_distance
294 ot_made_flag
295 am_id
296 me_date
297 ot_id
298 tendance
299 ena_temp
300 gnoisedb;
301 n;
302 s graphics off;
303
304
305 correlation analysis using train2 data vs shot made flag */
306 oc corr data=train2 plots=matrix(histogram);
307     var shot_made_flag game_event_id lat loc_y minutes_remaining
        ↪ period seconds_remaining shot_distance attendance arena_temp
        ↪ avgnoisedb;
308     run;
309

```

```

310
311  correlation analysis using principal components vs shot made flag */
312  oc corr data=pca plots=matrix(histogram);
313      var shot_made_flag prin1 - prin10;
314      run;
315
316
317  model 1 - GLM select using PCA */
318  oc glmselect data=pca plots=all seed=3;
319  del shot_made_flag =prin1-prin10 / selection = stepwise(choose=CV
    ↪  select=CV stop=CV);
320  n;
321
322
323  model 2 - Logistic using PCA */
324  s graphics on;
325  oc logistic data=pca plots(only)=(roc(id=obs) effect);
326  model shot_made_flag (event='1') =prin1-prin10 / scale=none
327                                     clparm=wald
328                                     clodds=pl
329                                     rsquare
330                                     lackfit
331                                     ctable;
332  output out = model_2_results p = Predict;
333  n;
334  s graphics off;
335
336
337  model 3 - Logistic using train2 dataset (not PCA) only by distance
    ↪  */
338  s graphics on;
339  oc logistic data=train2 plots(only)=(roc(id=obs) effect);
340  model shot_made_flag (event='1') = shot_distance / scale=none
341                                     clparm=wald
342                                     clodds=pl
343                                     rsquare

```

```

344                                     lackfit
345                                     ctable;
346 output out = model_3_results p = Predict;
347 n;
348 s graphics off;
349
350
351 create data for model 4 - data sets for playoffs and not at playoffs
    ↪ */
352
353 ta train2_playoffs;
354 t train2;
355 ere playoffs = 1;
356 n;
357
358 ta train2_no_playoffs;
359 t train2;
360 ere playoffs = 0;
361 n;
362
363
364 model 4A - Logistic using train2 dataset (not PCA) during playoffs
    ↪ */
365
366 s graphics on;
367 oc logistic data=train2_playoffs plots(only)=(roc(id=obs) effect);
368 model shot_made_flag (event='1') = shot_distance / scale=none
369                                     clparm=wald
370                                     clodds=pl
371                                     rsquare
372                                     lackfit
373                                     ctable;
374 output out = model_4A_results p = Predict;
375 n;
376 s graphics off;
377

```

```

378
379  model 4B - Logistic using train2 dataset (not PCA) during playoffs
    ↪  */
380
381  s graphics on;
382  oc logistic data=train2_no_playoffs plots(only)=(roc(id=obs) effect);
383  model shot_made_flag (event='1') = shot_distance / scale=none
384                                     clparm=wald
385                                     clodds=pl
386                                     rsquare
387                                     lackfit
388                                     ctable;
389  output out = model_4B_results p = Predict;
390  n;
391  s graphics off;
392
393
394  model 5 - Logistic using train2 dataset (not PCA) during playoffs
    ↪  */
395  s graphics on;
396  oc logistic data=train2 plots(only)=(roc(id=obs) effect);
397  model shot_made_flag (event='1') = shot_distance playoffs arena_temp
    ↪  game_event_id lat lon / scale=none
398                                     clparm=wald
399                                     clodds=pl
400                                     rsquare
401                                     lackfit
402                                     ctable;
403  output out = model_5_results p = Predict;
404  n;
405  s graphics off;
406
407
408  model 6 - Logistic using train2 dataset (not PCA) for all variables
    ↪  that had corr p < 0.0001 */
409  s graphics on;

```

```

410 oc logistic data=train2 plots(only)=(roc(id=obs) effect);
411 model shot_made_flag (event='1') = shot_distance playoffs period
    ↪ minutes_remaining seconds_remaining attendance arena_temp
    ↪ avgnoisedb / scale=none
412                                     clparm=wald
413                                     clodds=pl
414                                     rsquare
415                                     lackfit
416                                     ctable;
417 output out = model_6_results p = Predict;
418 n;
419 s graphics off;
420
421
422 Model 7 - LDA Model */
423
424 oc discrim data=train2 outstat=LDASTAT method=normal pool=yes
425             list crossvalidate;
426     class shot_made_flag;
427     priors prop;
428     var shot_distance playoffs period minutes_remaining
    ↪ seconds_remaining attendance arena_temp avgnoisedb;
429 run;
430
431
432
433
434
435
436
437
438
439 Import test data for prediction model */
440
441 import test data */

```

```

442 oc import datafile="c:\users\allen\documents\smu data science\MSDS
    ↪ 6372 - Applied Statistics\project 2\project2pred.csv"
443     dbms=dlm out=test replace;
444     delimiter=',';
445     getnames=yes;
446 n;
447
448 print data */
449 oc print data=test (obs=10);
450 n;
451
452 investigate data for variable types*/
453 oc contents data=test;
454 n;
455
456 look for missing numeric data - may need to impute */
457 oc means data = test n nmiss;
458 var _numeric_;
459 n;
460
461 investigate means of training data */
462 oc means data = test n nmiss;
463 n;
464
465 univariate data analysis */
466 oc univariate data = test;
467 r season;
468 n;
469
470 note that certain "season" fields are missing */
471 ta _seasontest;
472     set test;
473     where missing (season);
474 n;
475
476 print "season" data, where "season" is missing */

```

```

477 oc print data=_seasontest (obs=200);
478 n;
479
480 add empty predicted response field */
481 ta test2;
482 t test;
483 ot_made_flag = .;
484
485
486 print data */
487 oc print data=test2 (obs=200);
488 n;
489
490
491 Create TRAIN and TEST fields to distinguish test vs train data.
   ↪ Combine data, predict missing values, create final data set */
492
493 ta train2b;
494 t train2;
495 le = "TRAIN";
496 n;
497
498 oc print data=train2b (obs=10);
499 n;
500
501 ta test2b;
502 t test2;
503 le = "TEST";
504 n;
505
506 oc print data=test2b (obs=10);
507 n;
508
509
510
511

```



```

512  make a numeric shot_made_flag variable in test data */
513
514  ta test2c;
515  t test2b;
516  n_shot_made_flag = input(shot_made_flag,8.);
517  n;
518
519  drop original non-numeric shot_made_flag */
520  ta test2c;
521  t test2c (drop = shot_made_flag);
522  n;
523
524  rename numeric shot_made_flag */
525  ta test2c;
526  t test2c (rename=(
527  _shot_made_flag'n='shot_made_flag'n));
528  n;
529
530  drop the n_shot_made_flag variable */
531  ta test2c;
532  t test2c (drop = shot_made_flag);
533  n;
534
535  rename rannum variable to recId */
536  ta test2c;
537  t test2c (rename=(
538  annum'n='recId'n));
539  n;
540
541
542
543  combine data sets */
544
545  ta test3;
546  t train2b test2c;
547  n;

```

```

548
549 oc print data=test3 (obs=10);
550 n;
551
552 oc contents data=test3;
553 n;
554
555
556 predict response field (shot_made_flag) using desired method */
557 s graphics on;
558 oc logistic data=test3 plots(only)=(roc(id=obs) effect);
559 model shot_made_flag (event='1') = shot_distance playoffs period
    ↪ minutes_remaining seconds_remaining attendance arena_temp
    ↪ avgnosedb / scale=none
560                                     clparm=wald
561                                     clodds=pl
562                                     rsquare
563                                     lackfit
564                                     ctable;
565 tput out = model_test_results p = Predict;
566 n;
567 s graphics off;
568
569
570 check data for completeness */
571
572 oc means data = results n nmiss;
573 var _numeric_;
574 n;
575
576 oc print data=results (obs=10);
577 ere file = "TEST";
578 n;
579
580 oc contents data=results;
581 n;

```

```

582
583 oc means data=results
584   Mean Std Min Q1 Median Q3 Max;
585 n;
586
587   This is the final step that maps the predicted value into the
     ↪ shot_made_flag variable
588 d then drops all variables except shot_id and shot_made_flag. */
589
590 ta results_final;
591 tain shot_id shot_made_flag;
592 t model_test_results;
593   shot_made_flag < 1 then shot_made_flag = predict;
594 ep shot_id shot_made_flag;
595 ere file = "TEST";
596 n;
597
598 oc print data=results_final (obs=100);
599 n;
600
601 oc contents data=results_final;
602 n;
603
604
605
606
607
608
609
610
611
612
613
614
615 s graphics on;
616 oc logistic data=test3 plots(only)=(roc(id=obs) effect);

```

```

617 model shot_made_flag (event='1') = shot_distance / scale=none
618                               clparm=wald
619                               clodds=pl
620                               rsquare
621                               lackfit
622                               ctable;
623 output out = model_test3_results p = Predict;
624 n;
625 s graphics off;
626
627
628
629 model 5 - Logistic using train2 dataset (not PCA) during playoffs
    ↪ */
630 s graphics on;
631 oc logistic data=test3 plots(only)=(roc(id=obs) effect);
632 model shot_made_flag (event='1') = shot_distance playoffs arena_temp
    ↪ game_event_id lat lon / scale=none
633                               clparm=wald
634                               clodds=pl
635                               rsquare
636                               lackfit
637                               ctable;
638 output out = model_test5_results p = Predict;
639 n;
640 s graphics off;
641
642
643 model 6 - Logistic using train2 dataset (not PCA) for all variables
    ↪ that had corr p < 0.0001 */
644 s graphics on;
645 oc logistic data=test3 plots(only)=(roc(id=obs) effect);
646 model shot_made_flag (event='1') = shot_distance playoffs period
    ↪ minutes_remaining seconds_remaining attendance arena_temp
    ↪ avgnosedb / scale=none
647                               clparm=wald

```

```

648             clodds=pl
649             rsquare
650             lackfit
651             ctable;
652 output out = model_test6_results p = Predict;
653 n;
654 s graphics off;
655
656
657 ta results_final_6;
658 tain shot_id shot_made_flag;
659 t model_test6_results;
660 shot_made_flag < 1 then shot_made_flag = predict;
661 ep shot_id shot_made_flag;
662 ere file = "TEST";
663 n;

```

---

## 6.4 Python Code

---

```

1  # MSDS 6371 - Applied Statistics #
2  # Allen Crane and Brock Friedrich #
3  # Kobe Bryant Shot Selection      #
4  # November 2018                  #
5
6
7 port warnings
8 rnings.filterwarnings("ignore")
9
10 port os
11 port sys
12 port seaborn as sns
13 port matplotlib.pyplot as plt
14 port pandas as pd
15 port numpy as np
16 port statsmodels.api as sm

```

```

17 om statsmodels.stats.outliers_influence import
    ↪ variance_inflation_factor
18 om sklearn.feature_selection import RFE
19 om sklearn.discriminant_analysis import LinearDiscriminantAnalysis
20 om sklearn.model_selection import train_test_split
21 om sklearn.metrics import confusion_matrix, log_loss, roc_auc_score
22 om sklearn.linear_model import LogisticRegression
23
24 port statsmodels.formula.api as smf
25 om scipy import stats
26 port matplotlib.pyplot as plt
27 port numpy as np
28 port pandas as pd
29 om pandas.plotting import (lag_plot,
30                             autocorrelation_plot,
31                             table, scatter_matrix,
32                             boxplot)
33
34 om patsy import dmatrices
35 om math import degrees, acos
36 om scipy.spatial import distance
37
38
39 os.chdir(os.path.dirname(__file__))
40 s.path.insert(0, os.getcwd()+'/src')
41 om eda import *
42 om confusion_matrix_pretty import *
43 from plotting import *
44 om logistic_regression import *
45 om linear_discriminant_analysis import *
46
47 f cols(df: pd.DataFrame) -> list:
48     """Extract list of columns from input DataFrame and removing the
    ↪ dependent variable."""
49
50 return [x for x in df.columns.tolist() if x not in [DEPENDENT]]

```

```

51
52 f get_dummies(df: pd.DataFrame, drop_first = False):
53     """Replace catagorical variables with indicators"""
54
55     df = pd.get_dummies(df, dtype = float)
56     df.columns = df.columns \
57         .str.lower() \
58         .str.replace(" ", "_")
59     return df
60
61 f LogRegModel(data: pd.DataFrame, add_constant = False):
62     if add_constant:
63         data = sm.add_constant(data)
64         model = LogR(data, DEPENDENT)
65         model.sm = model.statsmodel()
66
67         model.yhat = model.sm.predict(model.test_x)
68         print("\n Predicted Log Loss: {}".format(
69             round(
70                 log_loss(model.test_y, model.yhat)
71                 , 4)))
72     return model
73
74 f summarize_model(model: LogR):
75     print(model.describe_features())
76     print(model.sm.summary())
77     print(model.sm.summary2())
78     print(model.sm.wald_test_terms())
79
80 Import Data
81 TA = pd.read_excel('data/project2Data.xlsx', index_col = 'recId')
82 R_PREDICTION = pd.read_excel('data/project2Pred.xlsx', index_col =
    ↪ 'rannum')
83 PENDENT = "shot_made_flag"
84
85 DUNDANT_FEATURES = [

```

```

86 'team_id', # constant term
87 'team_name', # constant term
88 'season',
89 'game_id', # violates independence
90 'matchup',
91 'shot_id',
92 'recId',
93 'shot_zone_area',
94 'shot_zone_basic',
95 'shot_zone_range',
96 'minutes_remaining',
97 'seconds_elapsed_in_game',
98 'game_event_id', # violates independence
99 'game_date', # violates independence
100 'action_type',
101     'loc_x', # collinear with lat
102     'loc_y', # collinear with lon
103
104
105
106
107
108
109 f to_latex(df):
110     pd.set_option('display.float_format', lambda x: '%.0f' % x)
111     with open('temp.txt', 'w') as f:
112         f.write(r'\resizebox{\textwidth}{!}{'+
113             df.to_latex()
114             + r'}\captionof{table}{Feature
115             ↪ Summary}\label{tbl:featuresummary}')
116     pd.set_option('display.float_format', lambda x: '%.4f' % x)
117
118 f desc(df: pd.DataFrame):
119     """Produces a summary of the input DataFrame
120

```



```

121 arguments:
122 df {pd.DataFrame} -- [description]
123
124 returns:
125 pd.DataFrame -- DataFrame of summary statistics
126 """
127
128 esc = df.describe(percentiles = None).T
129 esc['missing'] = len(df.index) - desc['count']
130 desc = desc.astype('int')
131 esc['median'] = df.median()
132 esc['missing %'] = desc.missing / len(df.index) * 100
133 return desc.T
134
135 ##### Model 0 - Predicted Log Loss: 0.6552 #####"
136
137
138 "Dataset: d0 | Prediction set: d0_pred
139   - Full Model
140 "
141
142 = prepare_data(DATA.drop(columns = ['action_type']))
143 .game_date = d0.game_date.apply(lambda x: x.toordinal())
144 = get_dummies(d0).fillna(0) # Get dummy variables for categoricals
145 _pred = wrangle_features(FOR_PREDICTION)
146 _pred.game_date = d0_pred.game_date.apply(lambda x: x.toordinal())
147 _pred = get_dummies(d0_pred).fillna(0) # Get dummy variables for
148   ↪ categoricals
149
150 _pred = d0_pred[cols(d0)]
151
152 "Fit d2"""
153
154 del0 = LogRegModel(d0)
155 mmarize_model(model0)
156 del0.roc_plot()
157

```

```

156 ##### Model 1 - Predicted Log Loss: 0.6652 #####
157
158
159 Dataset: d1 | Prediction set: d1_pred
160 - No categorical features
161 "
162 = prepare_data(DATA, drop_categorical = True) # Wrangle Data
163 .game_date = d1.game_date.apply(lambda x: x.toordinal())
164 = d1.fillna(0)
165 d1_pred = wrangle_features(FOR_PREDICTION)
166 d1_pred.game_date = d1_pred.game_date.apply(lambda x: x.toordinal())
167 d1_pred = d1_pred[cols(d1)].fillna(0)
168
169 "Fit d1"
170 del1 = LogRegModel(d1)
171 mmarize_model(model1)
172 del1.roc_plot()
173
174 ##### Model 2 - Predicted Log Loss: 0.6479 #####
175
176 Dataset: d2 | Prediction set: d2_pred
177 - Categorical features as indicators
178 - Drop redundant features
179 "
180
181 = prepare_data(DATA, drop_columns= REDUNDANT_FEATURES)
182 d2.game_date = d2.game_date.apply(lambda x: x.toordinal())
183 d2.last_seconds_of_period = d2.last_seconds_of_period.astype(int)
184 = get_dummies(d2).fillna(0) # Get dummy variables for categoricals
185 _pred = wrangle_features(FOR_PREDICTION)
186 _pred = get_dummies(d2_pred).fillna(0) # Get dummy variables for
   ↪ categoricals
187 _pred = d2_pred[cols(d2)]
188
189 "Fit d2"
190 del2 = LogRegModel(d2)

```

```

191 mmarize_model(model2)
192 del2.sm2 = model2.statsmodel_()
193 del2.sm2.fitted = model2.sm2.fit()
194 model2.sm.summary2()
195 del2.sm2.fitted.predict(model2.test_x)
196 pd.Series(model2.predict_labels(d2_pred)).set_index(d2_pred.sho)
197 model2.roc_plot()
198
199 f_train_x = model2.sm2.pdf(model2.train_x)
200 f_train_x = model2.sm2.cdf(model2.train_x)
201
202 sult = model2.sm.summary2()
203 godds = result.tables[1]
204 ds[['Coef.', '[0.025', '0.975']]] = np.exp(logodds[['Coef.', '[0.025',
    ↪ '0.975']]])
205 t_change = (odds[['Coef.', '[0.025', '0.975']]] - 1) * 100
206
207
208 rr_matrix(wrangle_features(DATA.drop(columns = ['action_type'])))
209 ot_proba(model2)
210 ot_regular_vs_post_season(model2)
211 ot_confusion_matrix(model2)
212
213
214 "
215
216 logOdds
217 -----
218
219      Coef.      Std.Err.      z      P>|z|
    ↪ [0.025      0.975]
219 t      -0.6565      0.3081 -2.1307 0.0331
    ↪ -1.2604      -0.0526
220 n      -0.0526      0.1710 -0.3076 0.7584
    ↪ -0.3879      0.2826
221 ayoffs      0.0417      0.0462 0.9019 0.3671
    ↪ -0.0489      0.1324

```

222	<i>conds_remaining</i>	0.0014	0.0009	1.5626	0.1182
	↪ -0.0004	0.0031			
223	<i>ot_distance</i>	-0.0189	0.0044	-4.2505	0.0000
	↪ -0.0275	-0.0102			
224	<i>tendance</i>	0.0002	0.0000	10.3586	0.0000
	↪ 0.0001	0.0002			
225	<i>ena_temp</i>	0.0328	0.0076	4.3008	0.0000
	↪ 0.0178	0.0477			
226	<i>gnoisedb</i>	0.0025	0.0078	0.3175	0.7509
	↪ -0.0128	0.0177			
227	<i>conds_left_in_period</i>	0.0001	0.0001	0.7850	0.4325
	↪ -0.0001	0.0002			
228	<i>st_seconds_of_period</i>	-0.8629	0.1282	-6.7335	0.0000
	↪ -1.1141	-0.6117			
229	<i>conds_left_in_game</i>	0.0001	0.0000	2.2041	0.0275
	↪ 0.0000	0.0001			
230	<i>me_or_away</i>	-0.0259	0.0305	-0.8465	0.3973
	↪ -0.0857	0.0340			
231	<i>m_shots_cumulative</i>	0.0005	0.0042	0.1257	0.9000
	↪ -0.0077	0.0087			
232	<i>gle_from_basket</i>	0.0002	0.0004	0.3873	0.6985
	↪ -0.0006	0.0009			
233	<i>ason_count</i>	0.0093	0.0034	2.7567	0.0058
	↪ 0.0027	0.0159			
234					
235	<i>Odds</i>				
236					
237		<i>Coef.</i>	<i>Std.Err.</i>	<i>z</i>	<i>P&gt; z </i> [0.025
	↪ 0.975]				
238					
239	<i>lat</i>	0.5175	0.3082	-2.1377	0.0325
	↪ 0.2829	0.9467			
240	<i>n</i>	0.9476	0.1711	-0.3144	0.7532 0.6777
	↪ 1.3251				
241	<i>ayoffs</i>	1.0214	0.0586	0.3602	0.7187 0.9105
	↪ 1.1458				

242	<i>conds_remaining</i>	1.0014	0.0009	1.5652	0.1175	0.9996
	↪ 1.0031					
243	<i>shot_distance</i>	0.9813	0.0044	-4.2570	0.0000	
	↪ 0.9728 0.9899					
244	<i>me_date</i>	1.0002	0.0003	0.5708	0.5681	0.9995
	↪ 1.0008					
245	<i>attendance</i>	1.0002	0.0000	10.3607	0.0000	
	↪ 1.0001 1.0002					
246	<i>arena_temp</i>	1.0331	0.0076	4.2622	0.0000	
	↪ 1.0177 1.0486					
247	<i>gnoisedb</i>	1.0025	0.0078	0.3238	0.7461	0.9873
	↪ 1.0180					
248	<i>conds_left_in_period</i>	1.0001	0.0001	0.7873	0.4311	0.9999
	↪ 1.0002					
249	<i>last_seconds_of_period</i>	0.4220	0.1282	-6.7317	0.0000	
	↪ 0.3283 0.5425					
250	<i>seconds_left_in_game</i>	1.0001	0.0000	2.2072	0.0273	
	↪ 1.0000 1.0001					
251	<i>me_or_away</i>	0.9739	0.0306	-0.8659	0.3865	0.9173
	↪ 1.0340					
252	<i>m_shots_cumulative</i>	1.0005	0.0042	0.1242	0.9011	0.9923
	↪ 1.0088					
253	<i>gle_from_basket</i>	1.0002	0.0004	0.3880	0.6980	0.9994
	↪ 1.0009					
254	<i>ason_count</i>	0.9419	0.1212	-0.4942	0.6212	0.7427
	↪ 1.1944					
255						
256	<i>Percent Changes</i>					
257		<i>Coef.</i>	<i>Std.Err.</i>	<i>z</i>	<i>P&gt; z </i>	
	↪ [0.025 0.975]					
258	<i>lat</i>	-48.1343	0.3081	-2.1307	0.0331	
	↪ -71.6464 -5.1247					
259	<i>n</i>	-5.1247	0.1710	-0.3076	0.7584	
	↪ -32.1487 32.6623					
260	<i>ayoffs</i>	4.2596	0.0462	0.9019	0.3671	
	↪ -4.7756 14.1521					

```

261 conds_remaining                0.1395      0.0009  1.5626 0.1182
    ↪ -0.0354  0.3147
262 shot_distance                -1.8678      0.0044 -4.2505 0.0000
    ↪ -2.7173 -1.0109
263 attendance                   0.0173      0.0000 10.3586 0.0000
    ↪ 0.0141  0.0206
264 arena_temp                   3.3317      0.0076  4.3008 0.0000
    ↪ 1.7998  4.8866
265 gnosedb                      0.2476      0.0078  0.3175 0.7509
    ↪ -1.2714  1.7900
266 conds_left_in_period         0.0061      0.0001  0.7850 0.4325
    ↪ -0.0092  0.0215
267 last_seconds_of_period       -57.8070      0.1282 -6.7335 0.0000
    ↪ -67.1786 -45.7595
268 seconds_left_in_game         0.0070      0.0000  2.2041 0.0275
    ↪ 0.0008  0.0132
269 me_or_away                  -2.5519      0.0305 -0.8465 0.3973
    ↪ -8.2134  3.4588
270 m_shots_cumulative           0.0527      0.0042  0.1257 0.9000
    ↪ -0.7654  0.8775
271 gle_from_basket             0.0150      0.0004  0.3873 0.6985
    ↪ -0.0610  0.0911
272 ason_count                   0.9308      0.0034  2.7567 0.0058
    ↪ 0.2681  1.5979
273
274 Significant features
275      Coef.  Std.Err.      z  P>|z|  [0.025  0.975]
276 t          0.5175    0.3082 -2.1377 0.0325  0.2829  0.9467
277 ot_distance    0.9813    0.0044 -4.2570 0.0000  0.9728  0.9899
278 tendance      1.0002    0.0000 10.3607 0.0000  1.0001  1.0002
279 ena_temp       1.0331    0.0076  4.2622 0.0000  1.0177  1.0486
280 st_seconds_of_period 0.4220    0.1282 -6.7317 0.0000  0.3283  0.5425
281 conds_left_in_game 1.0001    0.0000  2.2072 0.0273  1.0000  1.0001
282
283 "
284

```

```

285
286
287 " #! Interpretation
288 e p value is calculated based on the assumption that the null
    ↪ hypothesis is true.
289
290 think about it this way: "assuming the null hypothesis is true, the
    ↪ probability of the observed test statistic occurring is 0.02.
    ↪ That's not very probable. But the observed test statistic
    ↪ definitely occurred, because it was observed. Therefore, it seems
    ↪ more likely that the null hypothesis is not true, i.e. It should
    ↪ be rejected."
291
292 suming the null hypothesis is true, the probability of measuring at
    ↪ least the observed test occurring is 0.02."
293
294 "
295
296 f_test_x = sm2.pdf(model2.test_x)
297
298 " Refine Model 2 """
299
300 ld = model2.sm.wald_test_terms()
301 ld.df = wald.summary_frame()
302 ld.significant = wald.df[wald.df['P>chi2'] < 0.1].index.tolist()
303
304 " Refined Fit - Predicted Log Loss: 0.6634 """
305
306 del2r = LogRegModel(d2[wald.significant + [DEPENDENT]])
307
308 "
309 t -0.1393
310 ot_distance -0.0447
311 tendance 0.0002
312 ena_temp 0.0337
313 conds_left_in_game 0.0001

```

```

314 st_seconds_of_period    -0.8275
315 "
316
317
318
319
320 Interpret: http://www-hsc.usc.edu/~eckel/biostat2/notes/notes14.pdf
321
322 ##### Model 3 - Predicted Log Loss: 0.669 #####
323
324 "Dataset: d3 | Prediction set: d3_pred
325 - Allen's Model
326 "
327 cols = [
328     'shot_distance',
329     'playoffs',
330     'arena_temp',
331     'game_event_id',
332     'lat',
333     'lon',
334     'shot_made_flag'
335 ]
336 = DATA[d3cols]
337 _pred = FOR_PREDICTION[d3cols].drop(columns = [DEPENDENT]).fillna(0)
338
339 "Fit d3"
340 del3 = LogRegModel(d3)
341 mmarize_model(model3)
342
343
344
345 ##### Model 4 - LDA - Predicted Log Loss: 9.351
346 ↪ #####
347
347 a = LinearDiscriminantAnalysis()
348 a = lda.fit(model2.train_x, model2.train_y)

```



```

349 a_x = lda.transform(model2.train_x)
350 = lda.transform(model2.test_x)
351 labels = lda.predict(model2.test_x)
352
353 g_loss(model2.test_y, z)
354
355
356 ##### Model 5 #####
357
358 "Dataset: d5 | Prediction set: d5_pred
359   - shot_distance only predictor
360 "
361
362 = DATA[[DEPENDENT, 'shot_distance']]
363 d3_pred = FOR_PREDICTION[d3cols].drop(columns =
    ↪ [DEPENDENT]).fillna(0)
364
365 "Fit d5"
366 = sm.add_constant(d5)
367 del5 = LogR(d5, DEPENDENT)
368 del5.sm = model5.statsmodel()
369
370 del5.yhat = model5.sm.predict(model.test_x)
371
372 del5 = LogRegModel(d5)
373 = model5.sm.summary2()
374
375 = np.exp(s5.tables[1]['Coef.'])
376
377
378 ##### Model 6 - Log Loss: 0.669 #####
379
380 "Dataset: d6 | Prediction set: d6_pred
381   - Allen's model 5
382 "
383

```

```

384 = DATA[[DEPENDENT, 'shot_distance', 'playoffs', 'arena_temp',
    ↪ 'game_event_id', 'lat', 'lon']]
385
386 "Fit d6"
387 d5 = sm.add_constant(d5)
388 model5 = LogR(d5, DEPENDENT)
389
390
391 del6 = LogRegModel(d6)
392 = model6.sm.summary2()
393
394
395 = np.exp(s6.tables[1]['Coef.'])
396
397 ot_confusion_matrix(model6)
398
399 del6.sensitivity()
400 del6.specificity()
401
402
403 Data Overview
404
405 TODO: Add Univariate Plots
406 # QQ
407 # Hist
408
409
410
411
412 = d6 .select_dtypes(np.number)
413
414
415 cNoFocus = "red"
416
417
418 Correlation Matrix

```

419

420

421 `TA.shape`

422

423 *better model was at the cost of explainability and violation of*  
424 *↪ parsimony*

424

425 *"*

426 *The \_\_odds of Kobe making a shot decrease with respect to the*  
427 *↪ distance he is from the hoop\_\_. If there is evidence of this,*  
428 *↪ quantify this relationship. (CIs, plots, etc.)*

427

428 *Yes. His odds go down by -1.87% +-.85 % (-2.72% -1.01%) for every*  
429 *↪ additional foot away from the basket.*

429 *"*

430

431 *"*

432 *The \_\_probability of Kobe making a shot decreases linearly with*  
433 *↪ respect to the distance he is from the hoop\_\_. If there is*  
434 *↪ evidence of this, quantify this relationship. (CIs, plots,*  
435 *↪ etc.)*

433

434 *It doesn't. Show pdf plot.*

435

436 *near up to 23ft, but is not at zero at 23ft, so probability curve*  
437 *↪ must be curved.*

437

438

439 *"*

440

441

442

443 *"*

```

444 The relationship between the __distance Kobe is from the basket and
    ↳ the odds of him making the shot is different if they are in the
    ↳ playoffs__. Quantify your findings with statistical evidence one
    ↳ way or the other. (Tests, CIs, plots, etc.)
445
446
447
448 "
449
450
451 " Odds Ratios
452 ds ratios that are greater than 1 indicate that the event is more
    ↳ likely to occur as the predictor increases. Odds ratios that are
    ↳ less than 1 indicate that the event is less likely to occur as
    ↳ the predictor increases.
453
454 tps://www.predictiveanalyticsworld.com/patimes/on-variable-importance-in-logistic-re
455
456 The model indicates a 4.25% increase in shooting ability during the
    ↳ playoffs, however, the result was not statistically significant.
457
458 's overlap and contain zero. zome evidence but not enough to conclude
    ↳ there is a difference.
459
460
461 om sklearn.linear_model import LogisticRegression
462 om sklearn.model_selection import train_test_split
463 om sklearn.feature_selection import chi2
464 om sklearn.metrics import (
465     classification_report,
466     roc_curve,
467     auc
468
469 )
470 port pandas as pd
471 om confusion_matrix_pretty import *

```

```

472 port statsmodels.api as sm
473 om sklearn.metrics import confusion_matrix
474
475 ass LogR(LogisticRegression):
476     """Sparse extension of sklearn.linear_model.LogisticRegression.
477
478     """
479
480     def __init__(self,
481                 data: pd.DataFrame,
482                 dependent_name: None,
483                 store_covariance: bool = True,
484                 test_size: float = 0.25,
485                 fit_intercept = False):
486         super().__init__(solver = 'lbfgs',
487                         fit_intercept = fit_intercept
488                         )
489         self.yhat = None
490
491         self.train_x, self.test_x, self.train_y, self.test_y =
↪ train_test_split(data.drop(columns = [dependent_name]),
↪ data[dependent_name], test_size = test_size, random_state = 0)
492
493     def __repr__(self):
494         return super().__repr__()
495
496     def __str__(self):
497         return super().__str__()
498
499     def describe_features(self):
500
501         print(f"""
502         X: features: {len(self.train_x)}
503
504         dtypes:
505         -----""")

```

```

506
507     for k, v in self.train_x.dtypes.sort_index().items():
508         print(f'''\t\t{k:<30}{v.name:} ''')
509
510     print(f"""
511     Y: {self.train_y.name}: {self.train_y.dtype}
512     """)
513
514     def confusion_matrix(self, test = False, plot = True):
515         """Generate, and optionally plot, a confusion matrix for the
↪ test or train datasets
516
517         Keyword Arguments:
518             plot {bool} -- optionally plot the confusion matrix
↪             (default: {True})
519
520         Returns:
521             pd.DataFrame -- confusion matrix
522         """
523         if test:
524             x = self.test_x
525             y = self.test_y
526         else:
527             x = self.train_x
528             y = self.train_y
529
530         # if y is not None and x is not None:
531         self.cm = pd.DataFrame(confusion_matrix(y,
↪ self.sm2.fitted.predict(x)), index = [0, 1], columns = [0, 1])
532         if plot:
533             pretty_plot_confusion_matrix(self.cm, cmap='PuRd')
534         return self.cm
535         # else:
536         #     print('X or Y is empty. Check parameters.')
537         #     return None
538

```

```

539     def score(self) -> float:
540         """Wrapper for parent class method using xy's stored in child
    ↪ class object. Scores model fit using test data.
541
542         Returns:
543             float -- model score
544
545         """
546
547         x = self.test_x
548         y = self.test_y
549
550         score = super().score(x, y)
551         print(f'''
552         Features:
553             {' | '.join([x for x in x.columns])}
554
555         Accuracy: {score:.2%}
556
557         ''')
558         return score
559
560     def plot_separability() -> None:
561         """Plots a heatmap of fitted coefficients, highlighting
    ↪ features that are more likely seperable by a linear hyperplane.
562         """
563
564         x = self.train_x
565         if len(x.columns.tolist()) == len(self.coef_[0]):
566             fig, ax = plt.subplots(1, 1, figsize=(12, 10))
567             sns.heatmap(pd.DataFrame(self.coef_[0],
568                                     columns=[1],
569                                     index=x.columns.tolist()),
570                         ax=ax, cmap='RdBu', annot=True)
571
572             plt.title('LDA Feature Separability')

```

```

573         plt.tight_layout()
574     else:
575         print('Length of input "x" does not match number of
↪ coefficients. Refit the model using the dependents in x.')
576         return None
577
578     def fit(self):
579         """Wrapper for parent class method using xy's stored in child
↪ class object.
580
581         Fit the model.
582
583         Returns:
584             pd.Series -- array of x values projected to maximize
↪ seperation
585
586         """
587         super().fit(self.train_x, self.train_y)
588
589     def predict(self, x):
590
591         self.yhat = super().predict(x)
592         return self.yhat
593
594     def transform(self):
595         """Wrapper for parent class method using xy's stored in child
↪ class object.
596
597         Transform x to maximize seperation.
598
599         Returns:
600             pd.Series -- array of x values projected to maximize
↪ seperation
601
602         """
603         return super().transform(self.x)

```



```

604
605 def log_loss(self, x = None):
606
607     if self.yhat is not None:
608         self.yhat = self.sm.predict(x or self.test_x)
609
610     return round(log_loss(self.test_y, self.yhat), 2)
611
612 def _decision_function(self):
613     #TODO: Implement, time permitting
614     raise NotImplementedError()
615
616 def classification_report(self):
617     """Class wrapper for sklearn.metrics.classification_report
618     """
619
620     classification_report(
621         self.test_y,
622         self.yhat,
623         target_names=self.classes_.astype(str).tolist())
624
625 def statsmodel(self):
626     """Model using statsmodels library.
627
628     Returns:
629         statsmodels result object
630
631     """
632
633     return sm.Logit(self.train_y, self.train_x).fit()
634
635 def statsmodel_(self):
636     """Model using statsmodels library.
637
638     Returns:
639         statsmodels result object

```

```

640
641     """
642
643     return sm.Logit(self.train_y, self.train_x)
644
645
646 def roc_plot(self, sm = True):
647     """
648     Referenced from:
649
650     ↪ https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_cro
651     """
652
653     y_score = self.predict_labels(self.test_x)
654
655     fpr, tpr, _ = roc_curve(self.test_y, y_score)
656
657     roc_auc = auc(fpr, tpr)
658     plt.plot(fpr, tpr, lw=1, alpha=1,
659              label='ROC fold %d (AUC = %0.2f)' % (1, roc_auc))
660
661     g = plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
662                  label='Chance', alpha=.8)
663
664     plt.xlim([-0.05, 1.05])
665     plt.ylim([-0.05, 1.05])
666     plt.xlabel('False Positive Rate')
667     plt.ylabel('True Positive Rate')
668     plt.title('Receiver operating characteristic example')
669     plt.legend(loc="lower right")
670     g.set_title(f'Distribution of {y}', color = cNoFocus)
671     g.set_xlabel(f'{y}', size = 'xx-large', color = cNoFocus)
672     g.set_ylabel(f'Density', size = 'xx-large', color = cNoFocus)
673     g.set_xticklabels(g.get_xticklabels(), size = 'xx-large')
674     g.set_yticklabels(g.get_yticklabels(), size = 'xx-large')
675     g.tick_params(colors=cNoFocus)

```

```

675     g.spines['bottom'].set_color(cNoFocus)
676     g.spines['top'].set_color(cNoFocus)
677     g.spines['left'].set_color(cNoFocus)
678     g.spines['right'].set_color(cNoFocus)
679     g.xaxis.label.set_color(cNoFocus)
680     g.yaxis.label.set_color(cNoFocus)
681     return
682
683
684 def predict_labels(self, x, thresh = 0.5):
685     """Predict class labels"""
686
687     self.yhat = self.sm.fitted.predict(x)
688     pc = np.zeros(len(self.yhat))
689     pc[self.yhat > thresh] = 1
690     return pc
691
692 def sensitivity(self):
693     """Sensitivity - TP/(TP+FN)"""
694
695     cm = self.sm.pred_table()
696     sens = cm[0,0]/(cm[0,0]+cm[0,1])
697     print('Sensitivity : ', sens )
698     return sens
699
700
701 def specificity(self):
702     """Specificity = TN/(TN+FP)"""
703
704     cm = self.sm.pred_table()
705     spec = cm[1,1]/(cm[1,0]+cm[1,1])
706     print('Specificity : ', spec)
707     return spec
708
709
710

```

```

711
712 f RecursiveFeatureSelection(X, y):
713     logreg = LogisticRegression()
714     rfe = RFE(logreg, 20)
715     rfe = rfe.fit(X.fillna(0), y.values.ravel())
716     return rfe
717
718
719 Evaluate Model
720 https://www.r-bloggers.com/evaluating-logistic-regression-models/
721
722
723
724 http://blog.yhat.com/posts/logistic-regression-and-python.html
725
726 Dont use r-sq
727 https://stats.stackexchange.com/questions/3559/which-pseudo-r2-measure-is-the-one-to
728
729 TODO: Kobe last minute shots are outliers
730
731
732
733
734
735 X_train, X_test, y_train, y_test = train_test_split(
736     d3.drop(columns=[DEPENDENT]),
737     d3[DEPENDENT], test_size=0.3, random_state=0)
738 logreg = LogisticRegression(fit_intercept = True, C = 1e9)
739 logreg.fit(X_train, y_train)
740 y_pred = logreg.predict(X_test)
741 print('Accuracy of logistic regression classifier on test set:
742     ↪ {:.2f}'.format(logreg.score(X_test, y_test)))
742 log_loss(y_test, y_pred)
743 logreg.coef_
744
745 # sm

```

```

746 logit = sm.Logit(y_train, X_train)
747 logit.fit().params
748
749 port os
750 port sys
751 port seaborn as sns
752 port matplotlib.pyplot as plt
753 port pandas as pd
754 port numpy as np
755 port statsmodels.api as sm
756 om statsmodels.stats.outliers_influence import
    ↪ variance_inflation_factor
757 om sklearn.discriminant_analysis import LinearDiscriminantAnalysis
758 om sklearn.model_selection import train_test_split
759 om sklearn.metrics import confusion_matrix, log_loss
760 om sklearn.linear_model import LogisticRegression
761
762 port statsmodels.formula.api as smf
763 om scipy import stats
764 port matplotlib.pyplot as plt
765 port numpy as np
766 port pandas as pd
767 om pandas.plotting import (lag_plot,
768                             autocorrelation_plot,
769                             table, scatter_matrix,
770                             boxplot)
771
772 om patsy import dmatrices
773 om math import degrees, acos
774 om scipy.spatial import distance
775
776
777 os.chdir(os.path.dirname(__file__))
778 s.path.insert(0, os.getcwd()+'/src')
779 om eda import *
780 om confusion_matrix_pretty import *

```

```

781 from plotting import *
782
783 class LDAB(LinearDiscriminantAnalysis):
784     """Sparse extension of
    ↪ sklearn.discriminant_analysis.LinearDiscriminantAnalysis for
    ↪ handling binary class cases.
785
786     """
787
788     def __init__(self,
789                   data: pd.DataFrame,
790                   dependent_name: None,
791                   store_covariance: bool = True,
792                   solver = 'eigen',
793                   test_size: float = 0.25):
794         super().__init__(
795             # n_components = 2,
796             solver = solver,
797             store_covariance = store_covariance
798         )
799         self.yhat = None
800
801         self.train_x, self.test_x, self.train_y, self.test_y =
    ↪ train_test_split(
802             data.drop(columns = [dependent_name]),
803             data[dependent_name],
804             test_size = test_size,
805             random_state = 0)
806
807     def __repr__(self):
808         return super().__repr__()
809
810     def __str__(self):
811         return super().__str__()
812
813     def explained_variance(self) -> float:

```

```

814         """Get variance explained per discriminant.
815
816     Returns:
817         float -- explained variance ratio
818     """
819
820     print(f'''Explained variance ratio:
821     Discriminant 1: {self.explained_variance_ratio_[0]: .2f}''')
822     # return self.explained_variance_ratio_[0]
823
824     def describe_features(self):
825
826         print(f"""
827     X: features: {len(self.train_x)}
828
829         dtypes:
830         -----""")
831
832         for k, v in self.train_x.dtypes.sort_index().items():
833             print(f'\t\t{k:<30}{v.name:} ')
834
835         print(f"""
836     Y: {self.train_y.name}: {self.train_y.dtype}
837     """)
838
839     def confusion_matrix(self, test = False, plot = True):
840         """Generate, and optionally plot, a confusion matrix for the
841         ↪ test or train datasets
842
843         Keyword Arguments:
844             plot {bool} -- optionally plot the confusion matrix
845             ↪ (default: {True})
846
847     Returns:
848         pd.DataFrame -- confusion matrix
849     """

```

```

848         if test:
849             x = self.test_x
850             y = self.test_y
851         else:
852             x = self.train_x
853             y = self.train_y
854
855         # if y is not None and x is not None:
856         self.cm = pd.DataFrame(confusion_matrix(y, self.predict(x)),
↪ index = [0, 1], columns = [0, 1])
857         if plot:
858             pretty_plot_confusion_matrix(self.cm, cmap='PuRd')
859         return self.cm
860         # else:
861         #     print('X or Y is empty. Check parameters.')
862         #     return None
863
864     def score(self, x, y) -> float:
865         """Wrapper for parent class method using xy's stored in child
↪ class object. Scores model fit using test data.
866
867         Returns:
868             float -- model score
869
870         """
871
872         # x = self.test_x
873         # y = self.test_y
874
875         score = super().score(x, y)
876         print(f'''
877         Features:
878             {' | '.join([x for x in x.columns])}
879
880         Accuracy: {score:.2%}
881

```



```

882         '''
883         return score
884
885     def plot_separability() -> None:
886         """Plots a heatmap of fitted coefficients, highlighting
887         ↪ features that are more likely seperable by a linear hyperplane.
888         """
889
890         x = self.train_x
891         if len(x.columns.tolist()) == len(self.coef_[0]):
892             fig, ax = plt.subplots(1, 1, figsize=(12, 10))
893             sns.heatmap(pd.DataFrame(self.coef_[0],
894                                     columns=[1],
895                                     index=x.columns.tolist()),
896                         ax=ax, cmap='RdBu', annot=True)
897
898             plt.title('LDA Feature Separability')
899             plt.tight_layout()
900         else:
901             print('Length of input "x" does not match number of
902             ↪ coefficients. Refit the model using the dependents in x.')
903             return None
904
905     def fit(self):
906         """Wrapper for parent class method using xy's stored in child
907         ↪ class object.
908
909         Fit the model.
910
911         Returns:
912         pd.Series -- array of x values projected to maximize
913         ↪ seperation
914
915         """
916         super().fit(self.train_x, self.train_y)
917

```

```

914 def predict(self, x):
915
916     self.yhat = super().predict(x)
917     return self.yhat
918
919 def transform(self, x):
920     """Wrapper for parent class method using xy's stored in child
↪ class object.
921
922     Transform x to maximize seperation.
923
924     Returns:
925         pd.Series -- array of x values projected to maximize
↪ seperation
926
927     """
928     return super().transform(x)
929
930 def log_loss(self, x = None):
931
932     if self.yhat is not None:
933         self.yhat = self.predict(x or self.test_x)
934
935     return round(log_loss(self.test_y, self.yhat), 2)
936
937 def _decision_function(self):
938     #TODO: Implement, time permitting
939     raise NotImplementedError()
940
941 def classification_report(self):
942     """Class wrapper for sklearn.metrics.classification_report
943     """
944
945     classification_report(
946         self.test_y,
947         self.yhat,

```

```

948         target_names=self.classes_.astype(str).tolist())
949
950     def roc_plot(self):
951         """
952         Referenced from:
953
954         ↪ https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval
955         """
956
957         y_score = self.decision_function(self.test_x)
958
959         fpr, tpr, _ = roc_curve(self.test_y, y_score)
960
961         roc_auc = auc(fpr, tpr)
962         plt.plot(fpr, tpr, lw=1, alpha=1,
963                  label='ROC fold %d (AUC = %0.2f)' % (1, roc_auc))
964
965         plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
966                  label='Chance', alpha=.8)
967
968         plt.xlim([-0.05, 1.05])
969         plt.ylim([-0.05, 1.05])
970         plt.xlabel('False Positive Rate')
971         plt.ylabel('True Positive Rate')
972         plt.title('Receiver operating characteristic example')
973         plt.legend(loc="lower right")
974         plt.show()
975
976
977     """
978
979     assumptions:
980     - Each class must be:
981         - normally distributed
982         - identical cov matrices

```

```

983         - independent
984
985     "
986
987     __name__ == "__main__":
988
989
990     #! Try full model
991     model = LDAB(data.fillna(0), DEPENDENT)
992     model.fit()
993     model.explained_variance()
994     model.score()
995     model.get_confusion_matrix()
996     model.plot_separability()
997     model_fi = model.feature_importance()
998     model.log_loss()
999
1000    #! Try reduced model
1001    data_reduced = data[model_fi.index]
1002
1003    ldab_reduced = LDAB(data_reduced, DEPENDENT)
1004    ldab_reduced.fit()
1005    ldab_reduced.explained_variance()
1006    ldab_reduced.score_()
1007    ldab_reduced.get_confusion_matrix()
1008    ldab_reduced.plot_separability()
1009    ldab_reduced.log_loss()
1010
1011    disc1 = ldab_reduced.fit_transform(train_x_reduced, train_y)
1012
1013
1014    # Plot single discriminant
1015    sns.distplot(disc1)
1016
1017    # TODO: Add plots
1018

```

```

1019  #! Yes! Shows no seperation
1020
1021  sns.pairplot(temp_x,
1022             hue="shot_made_flag",
1023             palette="husl",
1024             markers = ['<', '>'],
1025             plot_kws = {
1026                 'alpha': 0.5,
1027             },
1028             diag_kws = {
1029
1030             },
1031             )
1032
1033  """
1034  Conclusion:
1035
1036  Reduced model, with only 2 features, yields results effectively
1037  ↪ equivalent to those of the full model that uses 13 features.
1038  """
1039
1040  import os
1041  import seaborn as sns
1042  import matplotlib.pyplot as plt
1043  import pandas as pd
1044  import numpy as np
1045  import statsmodels.api as sm
1046  from statsmodels.stats.outliers_influence import
1047      variance_inflation_factor
1048  from sklearn.discriminant_analysis import
1049      LinearDiscriminantAnalysis
1050  from sklearn import linear_model
1051  import statsmodels.formula.api as smf
1052  from scipy import stats
1053  import matplotlib.pyplot as plt
1054  import numpy as np

```

```

1052 import pandas as pd
1053 from pandas.plotting import (lag_plot,
1054                               autocorrelation_plot,
1055                               table, scatter_matrix,
1056                               boxplot)
1057
1058 from patsy import dmatrices
1059 from math import degrees, acos
1060 from scipy.spatial import distance
1061
1062
1063
1064 pd.options.display.max_rows = None
1065 pd.set_option('display.float_format', lambda x: '%.3f' % x)
1066 pd.set_option('large_repr', 'truncate')
1067 pd.set_option('precision',2)
1068
1069 # Matplotlib global config
1070 plt.rcParams.update({'legend.fontsize': 'x-large',
1071                      'figure.figsize': (10, 6),
1072                      'axes.labelsize': 'large',
1073                      'axes.titlesize': 'xx-large',
1074                      'xtick.labelsize': 'small',
1075                      'ytick.labelsize': 'small',
1076                      'savefig.dpi' : 300,
1077                      'savefig.format' : 'png',
1078                      'savefig.transparent' : True,
1079                      'axes.labelpad' : 10,
1080                      'axes.titlepad' : 10,
1081                      'axes.titleweight': 'bold'
1082                      })
1083
1084 # plt.style.use('seaborn-deep')
1085
1086
1087 # Define Constants

```

```

1088
1089 DEPENDENT = "shot_made_flag"
1090 PERIODS_IN_GAME = 4
1091 MIN_IN_PERIOD = 12
1092 MIN_IN_GAME = MIN_IN_PERIOD * PERIODS_IN_GAME
1093 SECONDS_IN_PERIOD = MIN_IN_PERIOD * 60
1094 SECONDS_IN_GAME = MIN_IN_GAME * 60
1095
1096
1097 """
1098 LOGISTIC MODEL:
1099
1100 - Dependent: shot_made: bool
1101
1102 """
1103
1104 """
1105 LDA MODEL:
1106
1107 - Dependent: shot_made: bool
1108
1109 """
1110
1111
1112
1113 """
1114 EDA:
1115
1116 - Potential Multicollinearity:
1117     - Court Position: lat/log
1118         x/y
1119         shot_zone_area (cat)
1120         shot_zone_basic (cat)
1121         shot_zone_range (cat)
1122
1123     - (maybe) game_date:

```

```

1124
1125
1126 - Add Features:
1127
1128     - game_count: cumulative number of games
1129       - "distance" between games is more or less equivalent
1130         ↳ (except between seasons), so representation as an
1131         ↳ ordinal continuous value is appropriate. The effects of
1132         ↳ season changes will still be captured by season_count.
1133
1134     - home_or_away: home ("vs.") or away ("@" )
1135
1136     - seconds_left_in_game: apply function
1137
1138     - seconds_left_in_period: min_remaining * 60 +
1139       ↳ seconds_remaining
1140
1141     - season_count: cumulative number of seasons
1142       - "distance" between seasons is more or less equivalent, so
1143         ↳ representation as an ordinal continuous value is
1144         ↳ appropriate.
1145
1146     - num_shots_cumulative: running total of number of shots up to
1147       ↳ the current point in the game
1148
1149     - (NotYetImplemented) shot_difficulty
1150
1151 Stretch Features:
1152
1153     - altitude: obtain from lat/long
1154
1155     - central_angle_to_basket: instead of x/y
1156
1157     - vector_length_to_basket: instead of x/y
1158
1159 Drop Features:

```



```

1153
1154     - team_id: constant
1155
1156     - team_name: constant
1157
1158     - season: replace with season_count
1159
1160     - game_id: replace with game_count
1161
1162     - matchup: redundant with opponent
1163
1164     """
1165
1166
1167
1168
1169     def desc(df: pd.DataFrame):
1170         """Produces a summary of the input DataFrame
1171
1172         Arguments:
1173             df {pd.DataFrame} -- [description]
1174
1175         Returns:
1176             pd.DataFrame -- DataFrame of summary statistics
1177         """
1178
1179         desc = df.describe().T
1180         desc['missing'] = len(df.index) - desc['count']
1181         # desc = desc.astype('int')
1182         desc['median'] = df.median()
1183         desc['missing %'] = desc.missing / len(df.index) * 100
1184         return desc.T
1185
1186     def vif(df: pd.DataFrame, dependent: str) -> pd.DataFrame:
1187         """Get Variance Inflation Factor for each feature in df via a
1188         ↪ simple, multiple regression.

```

```

1188
1189     Arguments:
1190         df {pd.DataFrame} -- dataset
1191         dependent {str} -- column name of dependent feature in df
1192
1193     Returns:
1194         pd.DataFrame -- DataFrame containing feature names and VIF
1195         ↪ measures.
1196
1197         """
1198
1199         # https://etav.github.io/python/vif_factor_python.html
1200         df = df.dropna()
1201         df = df._get_numeric_data() #drop non-numeric cols
1202
1203         #gather features
1204         features = "+".join(df.columns.drop(dependent).tolist())
1205
1206         # get y and X dataframes based on this regression:
1207         y, X = dmatrices('{y} ~'.format(dependent) + features, df,
1208         ↪ return_type='dataframe')
1209
1210         # For each X, calculate VIF and save in dataframe
1211         vif = pd.DataFrame()
1212         vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for
1213         ↪ i in range(X.shape[1])]
1214         vif["features"] = X.columns
1215
1216         return vif.round(1)
1217
1218     def angle(a: float, b: float, c: float) -> float:
1219         """ Calculate central angle for three known side lengths using
1220         ↪ Law of Cosines
1221
1222         Arguments:
1223             a {Side} -- A side length
1224             b {Side} -- B side length
1225             c {Side} -- C side length

```

```

1220
1221 Returns:
1222     Angle {float} -- central angle of A in degrees
1223     """
1224
1225     return degrees(acos((c**2 - b**2 - a**2)/(-2.0 * a * b)))
1226
1227 def central_angle(x: float, y:float) -> float:
1228     """Calculate central angle of shot using NBA court grid
1229     ↪ coordinates.
1230
1231     Arguments:
1232         x {float} -- X coordinate of shot
1233         y {float} -- Y coordinate of shot
1234
1235     Returns:
1236         float -- angle in degrees of shot
1237         """
1238
1239     # Hack
1240     if (y == 0) & (x < 0):
1241         return -90
1242
1243     if (y == 0) & (x > 0):
1244         return 90
1245
1246     if (y == 0) & (x == 0):
1247         return 0
1248
1249     # Vertices
1250     vc_a = (x, y) # shot loation
1251     vc_b = (0,0) # origin
1252     vc_c = (0, y) # reference point (0, y)
1253
1254     side_a = distance.euclidean(vc_b, vc_c)
1255     side_b = distance.euclidean(vc_a, vc_c)

```

```

1255     side_c = distance.euclidean(vc_a, vc_b)
1256
1257     # A = angle(side_a, side_b, side_c)
1258     # C = angle(side_c, side_a, side_b)
1259     B = angle(side_b, side_c, side_a)
1260
1261     return B if x > 0 else -B
1262
1263 def wrangle_features(data: pd.DataFrame) -> pd.DataFrame:
1264     feats = pd.Series(
1265         data = False,
1266         index = ['recId',
1267                 'action_type',
1268                 'combined_shot_type',
1269                 'game_event_id',
1270                 'game_id',
1271                 'lat',
1272                 'loc_x',
1273                 'loc_y',
1274                 'lon',
1275                 'minutes_remaining',
1276                 'period',
1277                 'playoffs',
1278                 'season',
1279                 'seconds_remaining',
1280                 'shot_distance',
1281                 'shot_made_flag',
1282                 'shot_type',
1283                 'shot_zone_area',
1284                 'shot_zone_basic',
1285                 'shot_zone_range',
1286                 'team_id',
1287                 'team_name',
1288                 'game_date',
1289                 'matchup',
1290                 'opponent',

```

```

1291         'shot_id',
1292         'attendance',
1293         'arena_temp',
1294         'avgnoisedb'],
1295         dtype = bool
1296     )
1297
1298     # Flag features that were passed to the function
1299     feats.loc[feats.index.isin(data.columns)] = True
1300     try:
1301         if feats.minutes_remaining & feats.seconds_remaining:
1302             data['seconds_left_in_period'] = data.minutes_remaining
1303             ↪ * 60 + data.seconds_remaining
1304         except Exception as e:
1305             print('Failed to add feature: seconds_left_in_period.
1306             ↪ ({}).format(e))
1307
1308     try:
1309         data['last_seconds_of_period'] =
1310         ↪ data.seconds_left_in_period < 2
1311         data.last_seconds_of_period =
1312         ↪ data.last_seconds_of_period.astype(int)
1313     except Exception as e:
1314         print('Failed to add feature: last_seconds_of_period.
1315         ↪ ({}).format(e))
1316
1317     try:
1318         if feats.period:
1319             data['seconds_elapsed_in_game'] = SECONDS_IN_PERIOD *
1320             ↪ data.period - data.seconds_left_in_period
1321     except:
1322         print('Failed to add feature: seconds_elapsed_in_game')
1323
1324     try:
1325         if True:

```

```

1321         data['seconds_left_in_game'] = SECONDS_IN_GAME -
↪ data.seconds_elapsed_in_game
1322     except:
1323         print('Failed to add feature: seconds_left_in_game')
1324
1325     try:
1326         if feats.matchup:
1327             data['home_or_away'] =
↪ data.matchup.str.contains("@").astype(int)
1328     except:
1329         print('Failed to add feature: home_or_away')
1330
1331     try:
1332         if feats.game_id:
1333             data['num_shots_cumulative'] =
↪ data.groupby(['game_id']).cumcount()
1334     except:
1335         print('Failed to add feature: num_shots_cumulative')
1336
1337     try:
1338         if feats.loc_x & feats.loc_y:
1339             data['angle_from_basket'] = data.apply(lambda row:
↪ central_angle(row.loc_x, row.loc_y), axis = 1)
1340     except:
1341         print('Failed to add feature: angle_from_basket')
1342
1343     try:
1344         if feats.season:
1345             # Convert season to ordered Categorical (Factor) type
1346             data.season = pd.Categorical(data.season,
↪ data.season.sort_values().unique().tolist(), ordered = True)
1347             data['season_count'] = data.season.cat.codes
1348     except:
1349         print('Failed to add feature: season_count')
1350
1351     try:

```

```

1352         if len(data.select_dtypes('object').columns):
1353             # Convert other string fields to unordered Categorical
1354             data[data.select_dtypes('object').columns.tolist()] =
↪ data.select_dtypes('object').astype('category')
1355         except:
1356             print('Failed to convert objects to categories')
1357
1358     return data
1359
1360 def drop_features(data, columns):
1361
1362     # Remove columns in the 'remove' list if they are present in
↪ the dataset
1363     data = data.drop(columns = [x for x in columns if x in
↪ data.columns])
1364     return data
1365
1366 def eigen_solver():
1367     """Assess using Eigen values and vectors
1368
1369     ↪ https://stackoverflow.com/questions/25676145/capturing-high-multi-collinearity-
1370
1371     An almost zero eigen value shows a direction with zero
↪ variation, hence collinearity.
1372
1373     """
1374     # TODO: Implement, time permitting
1375     raise NotImplementedError()
1376
1377 def check_collinearity(data: pd.DataFrame):
1378     return vif(data, DEPENDENT) \
1379         .set_index('features') \
1380         .rename(columns = {'VIF Factor' : 'VIF'}) \
1381         .sort_values(by = 'VIF', ascending = False) \
1382         .drop('Intercept')

```

```

1383
1384 def check_collinearity_recursive(data: pd.DataFrame, vifs = None):
1385     """Recursively check the multicollinearity (MC) associated
↪ with each feature. Each iteration, the feature with the largest
↪ MC is dropped if the MC is infinite or if MC > x, where x is the
↪ standard deviation of the finite VIFs of the original features. A
↪ matrix containing VIFs for each iteration is returned once an
↪ iteration is reached where MC <= x.
1386
1387     Arguments:
1388         data {pd.DataFrame} -- Matrix or DataFrame with
↪         shape(n_obs, n_features)
1389
1390     Keyword Arguments:
1391         vifs {None} -- Recursive control parameter (default:
↪         {None})
1392
1393     Returns:
1394         [pd.DataFrame] -- Matrix of VIFs per iteration. Nan (not a
↪         number) values represent features dropped from the
↪         assessment in either a previous or the current
↪         iteration.
1395     """
1396
1397     prev_vifs = vifs
1398
1399     vifs = vif(data, DEPENDENT) \
1400         .set_index('features') \
1401         .rename(columns = {'VIF Factor' : 'VIF'}) \
1402         .sort_values(by = 'VIF', ascending = False) \
1403         .drop('Intercept') # Drop intercept term
1404
1405
1406     vif0_name, vif0_val = vifs.iloc[0].name,
↪ vifs.iloc[0].values[0]
1407

```



```

1408     drop_feature = False
1409     limit = None
1410     thresh = prev_vifs.VIF[np.isfinite(prev_vifs.VIF)].max() if
↪ prev_vifs is not None else 0
1411     # If inflated feature VIF is infinite, drop the feature
1412     if vif0_val == float('inf'):
1413         drop_feature = True
1414     else:
1415         # Otherwise, drop feature if VIF within 2.5 stds.
1416         limit = (vifs[vifs != float('inf')].std()*2.5).values[0]
1417         if vif0_val > limit > thresh:
1418             drop_feature = True
1419
1420     if prev_vifs is not None:
1421         # print('\n\nprev_vifs')
1422         # print(prev_vifs)
1423         vifs = prev_vifs.join(vifs, rsuffix = '_' + str(len(vifs)))
1424
1425
1426
1427     print(f'VIF: Dropping: {vif0_name} | limit: {limit or 0:.2f} |
↪ thresh: {thresh or 0:.2f}')
1428
1429     if drop_feature:
1430         return check_collinearity_recursive(
1431             data.drop(columns = [vif0_name]),
1432             vifs = vifs
1433         )
1434
1435     print('\n\nvifs')
1436     print(vifs)
1437
1438     return vifs
1439
1440 def fix_multicollinearity(data: pd.DataFrame):

```

```

1441         """Remove multicollinear variables by assessing variance
↪      inflation factors.

1442
1443     Arguments:
1444         data {pd.DataFrame} -- (n_obs, n_features)
1445
1446     Returns:
1447         pd.DataFrame -- data
1448     """
1449     print('\n\n')
1450     vifs = check_collinearity_recursive(data)
1451     # vifs = check_collinearity(data)
1452     vifs = vifs.iloc[:, -1] # Get last column (the last iteration)
1453
1454     # remove features with high MC from data set
1455     data = data.drop(columns = vifs[vifs.isna()].index)
1456     return data
1457
1458     def prepare_data(data: pd.DataFrame, drop_categorical = False,
↪      drop_columns: list = None) -> pd.DataFrame:
1459         """Template procedue to ingest new dataset.
1460
1461     Arguments:
1462         data {pd.DataFrame} -- new dataset
1463
1464     Returns:
1465         pd.DataFrame -- dataset for further prep or analysis
1466     """
1467
1468     data = wrangle_features(data)
1469     if drop_columns is not None:
1470         data = drop_features(data, drop_columns)
1471     data = fix_multicollinearity(data)
1472
1473     # Drop remaining categoricals
1474     if drop_categorical:

```

```

1475         data = data.select_dtypes(exclude = ['object',
↳      'category'])
1476
1477         return data
1478
1479     def identify_outliers(data):
1480         pass
1481
1482
1483
1484     # action_counts =
↳      DATA['action_type'].value_counts().sort_values(ascending =
↳      False)
1485
1486     # from scipy import stats
1487     # d2[(np.abs(stats.zscore(d2)) < 3).any(axis=1)]
1488     # d2[np.abs(d2-d2.mean()) <= (3*d2.std())]
1489     # stats.trimb
1490
1491     # data = data.drop(columns = [
1492     #     'minutes_remaining',
1493     #     'seconds_remaining',
1494     #     'seconds_elapsed_in_game',
1495     #     'lat',
1496     #     'lon',
1497     #     'game_event_id',
1498     #     'period',
1499     #     'seconds_left_in_period',]
1500     # )
1501
1502     # Sort features by VIF
1503
1504     # NOTE: PLOTS
1505
1506     # fig, ax = plt.subplots(figsize=(12,8))

```

```

1507 # ax = sns.scatterplot('loc_x', 'loc_y', hue = 'shot_made_flag',
↪ data = data)
1508 # ax.set_title('Shot Location')
1509 # ax.set_xlabel('X')
1510 # ax.set_ylabel('Y')
1511 # ax.set_ylim(0, 400)
1512 # fig.savefig('figs/p2-3_price-v-months.png')
1513
1514 # fig, ax = plt.subplots(figsize=(12,8))
1515 # ax = data.boxplot()
1516 # ax.set_xticklabels(data.columns, rotation=90)
1517 # fig.tight_layout()
1518
1519 # fig, ax = plt.subplots(figsize=(12,8))
1520 # ax = data.select_dtypes(include=[np.number]).hist()
1521 # # ax.set_xticklabels(data.columns, rotation=90)
1522 # fig.tight_layout()
1523
1524 # import time
1525 # from sklearn.linear_model import LassoCV
1526 # print("Computing regularization path using the coordinate descent
↪ lasso...")
1527 # t1 = time.time()
1528 # model = LassoCV(cv=5).fit(X, y)
1529 # t_lasso_cv = time.time() - t1
1530
1531 # # Display results
1532 # m_log_alphas = -np.log10(model.alphas_)
1533
1534 # plt.figure()
1535 # ymin, ymax = 2300, 3800
1536 # plt.plot(m_log_alphas, model.mse_path_, ':')
1537 # plt.plot(m_log_alphas, model.mse_path_.mean(axis=-1), 'k',
1538 #          label='Average across the folds', linewidth=2)
1539 # plt.axvline(-np.log10(model.alpha_), linestyle='--', color='k',
1540 #          label='alpha: CV estimate')

```

```

1541
1542     # plt.legend()
1543
1544     # plt.xlabel('-log(alpha)')
1545     # plt.ylabel('Mean square error')
1546     # plt.title('Mean square error on each fold: coordinate descent '
1547     #           '(train time: %.2fs)' % t_lasso_cv)
1548     # plt.axis('tight')
1549     # plt.ylim(ymin, ymax)
1550
1551
1552     # def correct_multicollinearity(data: pd.DataFrame) ->
↪     pd.DataFrame:
1553         #         print('Anterior VIF')
1554         #         print(vif(data, DEPENDENT))
1555
1556         #         # Drop multicollinear features
1557         #         data = data.drop(columns = [
1558             #             'minutes_remaining',
1559             #             'seconds_remaining',
1560             #             'seconds_elapsed_in_game',
1561             #             'lat',
1562             #             'lon',
1563             #             'game_event_id',
1564             #             'period',
1565             #             'seconds_left_in_period',
1566
1567         #         ])
1568
1569         #         print('Posterior VIF')
1570         #         v = vif(data, DEPENDENT)
1571         #         print(v)
1572         #         return data
1573
1574     """
1575     plot a pretty confusion matrix with seaborn

```

```

1576 Created on Mon Jun 25 14:17:37 2018
1577 @author: Wagner Cipriano - wagnerbhbr - gmail - CEFETMG / MMC
1578 @repository:
    ↪ https://github.com/wcipriano/pretty-print-confusion-matrix/blob/master/confusi
1579 References:
1580 https://www.mathworks.com/help/nnet/ref/plotconfusion.html
1581
    ↪ https://stackoverflow.com/questions/28200786/how-to-plot-scikit-learn-classi
1582
    ↪ https://stackoverflow.com/questions/5821125/how-to-plot-confusion-matrix-wit
1583 https://www.programcreek.com/python/example/96197/seaborn.heatmap
1584
    ↪ https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-w
1585
    ↪ http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_
1586 """
1587
1588 #imports
1589 from pandas import DataFrame
1590 import numpy as np
1591 import matplotlib.pyplot as plt
1592 import matplotlib.font_manager as fm
1593 from matplotlib.collections import QuadMesh
1594 import seaborn as sns
1595
1596
1597 def _get_new_fig(fn, figsize=[9,9]):
1598     """ Init graphics """
1599     fig1 = plt.figure(fn, figsize)
1600     ax1 = fig1.gca() #Get Current Axis
1601     ax1.cla() # clear existing plot
1602     return fig1, ax1
1603 #
1604
1605 def _configcell_text_and_colors(array_df, lin, col, oText,
    ↪ facecolors, posi, fz, fmt, show_null_values=0):

```

```

1606     """
1607     config cell text and colors
1608     and return text elements to add and to dell
1609     @TODO: use fmt
1610     """
1611     text_add = []; text_del = [];
1612     cell_val = array_df[lin][col]
1613     tot_all = array_df[-1][-1]
1614     per = (float(cell_val) / tot_all) * 100
1615     curr_column = array_df[:,col]
1616     ccl = len(curr_column)
1617
1618     #last line and/or last column
1619     if(col == (ccl - 1)) or (lin == (ccl - 1)):
1620         #tots and percents
1621         if(cell_val != 0):
1622             if(col == ccl - 1) and (lin == ccl - 1):
1623                 tot_rig = 0
1624                 for i in range(array_df.shape[0] - 1):
1625                     tot_rig += array_df[i][i]
1626                 per_ok = (float(tot_rig) / cell_val) * 100
1627             elif(col == ccl - 1):
1628                 tot_rig = array_df[lin][lin]
1629                 per_ok = (float(tot_rig) / cell_val) * 100
1630             elif(lin == ccl - 1):
1631                 tot_rig = array_df[col][col]
1632                 per_ok = (float(tot_rig) / cell_val) * 100
1633             per_err = 100 - per_ok
1634         else:
1635             per_ok = per_err = 0
1636
1637     per_ok_s = ['%.2f%%'%(per_ok), '100%'] [per_ok == 100]
1638
1639     #text to DEL
1640     text_del.append(oText)
1641

```

```

1642         #text to ADD
1643         font_prop = fm.FontProperties(weight='bold', size=fz)
1644         text_kwargs = dict(color='w', ha="center", va="center",
↪ gid='sum', fontproperties=font_prop)
1645         lis_txt = ['%d'%(cell_val), per_ok_s, '%.2f%%'%(per_err)]
1646         lis_kwa = [text_kwargs]
1647         dic = text_kwargs.copy(); dic['color'] = 'g';
↪ lis_kwa.append(dic);
1648         dic = text_kwargs.copy(); dic['color'] = 'r';
↪ lis_kwa.append(dic);
1649         lis_pos = [(oText._x, oText._y-0.3), (oText._x, oText._y),
↪ (oText._x, oText._y+0.3)]
1650         for i in range(len(lis_txt)):
1651             newText = dict(x=lis_pos[i][0], y=lis_pos[i][1],
↪ text=lis_txt[i], kw=lis_kwa[i])
1652             #print 'lin: %s, col: %s, newText: %s' %(lin, col,
↪ newText)
1653             text_add.append(newText)
1654             #print '\n'
1655
1656         #set background color for sum cells (last line and last
↪ column)
1657         carr = [0.27, 0.30, 0.27, 1.0]
1658         if(col == ccl - 1) and (lin == ccl - 1):
1659             carr = [0.17, 0.20, 0.17, 1.0]
1660         facecolors[posi] = carr
1661
1662     else:
1663         if(per > 0):
1664             txt = '%s\n%.2f%%' %(cell_val, per)
1665         else:
1666             if(show_null_values == 0):
1667                 txt = ''
1668             elif(show_null_values == 1):
1669                 txt = '0'
1670             else:

```



```

1671         txt = '0\n0.0%'
1672     oText.set_text(txt)
1673
1674     #main diagonal
1675     if(col == lin):
1676         #set color of the text in the diagonal to white
1677         oText.set_color('w')
1678         # set background color in the diagonal to blue
1679         facecolors[posi] = [0.35, 0.8, 0.55, 1.0]
1680     else:
1681         oText.set_color('r')
1682
1683     return text_add, text_del
1684 #
1685
1686 def _insert_totals(df_cm):
1687     """ insert total column and line (the last ones) """
1688     sum_col = []
1689     for c in df_cm.columns:
1690         sum_col.append( df_cm[c].sum() )
1691     sum_lin = []
1692     for item_line in df_cm.iterrows():
1693         sum_lin.append( item_line[1].sum() )
1694     df_cm['sum_lin'] = sum_lin
1695     sum_col.append(np.sum(sum_lin))
1696     df_cm.loc['sum_col'] = sum_col
1697     #print ('\ndf_cm:\n', df_cm, '\n\n')
1698 #
1699
1700     def pretty_plot_confusion_matrix(df_cm, annot=True, cmap="Oranges",
↪     fmt='.2f', fz=11,
1701         lw=0.5, cbar=False, figsize=[8,8], show_null_values=0,
↪     pred_val_axis='y'):
1702         """
1703         print conf matrix with default layout (like matlab)
1704         params:

```

```

1705         df_cm          dataframe (pandas) without totals
1706         annot          print text in each cell
1707         cmap            Oranges,Oranges_r,YlGnBu,Blues,RdBu, ...
        ↪ see:
1708         fz             fontsize
1709         lw             linewidth
1710         pred_val_axis  where to show the prediction values (x or y
        ↪ axis)
1711
        'col' or 'x': show predicted values in
        ↪ columns (x axis) instead lines
1712         'lin' or 'y': show predicted values in
        ↪ lines (y axis)
1713
1714     """
1715     if(pred_val_axis in ('col', 'x')):
1716         xlabel = 'Predicted'
1717         ylabel = 'Actual'
1718     else:
1719         xlabel = 'Actual'
1720         ylabel = 'Predicted'
1721         df_cm = df_cm.T
1722
1723     # create "Total" column
1724     _insert_totals(df_cm)
1725
1726     #this is for print allways in the same window
1727     fig, ax1 = _get_new_fig('Conf matrix default', figsize)
1728
1729     #thanks for seaborn
1730     ax = sns.heatmap(df_cm, annot=annot, annot_kws={"size": fz},
    ↪ linewidths=lw, ax=ax1,
1731
        cbar=cbar, cmap=cmap, linecolor='w', fmt=fmt)
1732
1733     #set ticklabels rotation
1734     ax.set_xticklabels(ax.get_xticklabels(), rotation = 45,
    ↪ fontsize = 10)

```

```

1734         ax.set_yticklabels(ax.get_yticklabels(), rotation = 25,
↪     fontsize = 10)
1735
1736     # Turn off all the ticks
1737     for t in ax.xaxis.get_major_ticks():
1738         t.tick1On = False
1739         t.tick2On = False
1740     for t in ax.yaxis.get_major_ticks():
1741         t.tick1On = False
1742         t.tick2On = False
1743
1744     #face colors list
1745     quadmesh = ax.findobj(QuadMesh)[0]
1746     facecolors = quadmesh.get_facecolors()
1747
1748     #iter in text elements
1749     array_df = np.array( df_cm.to_records(index=False).tolist() )
1750     text_add = []; text_del = [];
1751     posi = -1 #from left to right, bottom to top.
1752     for t in ax.collections[0].axes.texts: #ax.texts:
1753         pos = np.array( t.get_position() ) - [0.5,0.5]
1754         lin = int(pos[1]); col = int(pos[0]);
1755         posi += 1
1756         #print ('>>> pos: %s, posi: %s, val: %s, txt: %s' %(pos,
↪     posi, array_df[lin][col], t.get_text()))
1757
1758     #set text
1759     txt_res = _configcell_text_and_colors(array_df, lin, col,
↪     t, facecolors, posi, fz, fmt, show_null_values)
1760
1761     text_add.extend(txt_res[0])
1762     text_del.extend(txt_res[1])
1763
1764     #remove the old ones
1765     for item in text_del:
1766         item.remove()

```

```

1767     #append the new ones
1768     for item in text_add:
1769         ax.text(item['x'], item['y'], item['text'], **item['kw'])
1770
1771     #titles and legends
1772     ax.set_title('Confusion matrix')
1773     ax.set_xlabel(xlbl)
1774     ax.set_ylabel(ylbl)
1775     plt.tight_layout() #set layout slim
1776     plt.show()
1777     return ax
1778
1779
1780     def plot_confusion_matrix_from_data(y_test, predictions,
↪ columns=None, annot=True, cmap="Oranges",
1781     fmt='.2f', fz=11, lw=0.5, cbar=False, figsize=[8,8],
↪ show_null_values=0, pred_val_axis='lin'):
1782     """
1783         plot confusion matrix function with y_test (actual values)
↪ and predictions (predic),
1784         without a confusion matrix yet
1785     """
1786     from sklearn.metrics import confusion_matrix
1787     from pandas import DataFrame
1788
1789     #data
1790     if(not columns):
1791         #labels axis integer:
1792         ##columns = range(1, len(np.unique(y_test))+1)
1793         #labels axis string:
1794         from string import ascii_uppercase
1795         columns = ['class %s' %(i) for i in
↪ list(ascii_uppercase)[0:len(np.unique(y_test))]]
1796
1797     confm = confusion_matrix(y_test, predictions)
1798     cmap = 'Oranges';

```

```

1799     fz = 11;
1800     figsize=[9,9];
1801     show_null_values = 2
1802     df_cm = DataFrame(confm, index=columns, columns=columns)
1803     pretty_plot_confusion_matrix(df_cm, fz=fz, cmap=cmap,
↪   figsize=figsize, show_null_values=show_null_values,
↪   pred_val_axis=pred_val_axis)
1804     #
1805
1806
1807
1808     #
1809     #TEST functions
1810     #
1811     def _test_cm():
1812         #test function with confusion matrix done
1813         array = np.array( [[13, 0, 1, 0, 2, 0],
1814                             [ 0, 50, 2, 0, 10, 0],
1815                             [ 0, 13, 16, 0, 0, 3],
1816                             [ 0, 0, 0, 13, 1, 0],
1817                             [ 0, 40, 0, 1, 15, 0],
1818                             [ 0, 0, 0, 0, 0, 20]])
1819         #get pandas dataframe
1820         df_cm = DataFrame(array, index=range(1,7), columns=range(1,7))
1821         #colormap: see this and choose your more dear
1822         cmap = 'Oranges'
1823         pretty_plot_confusion_matrix(df_cm, cmap=cmap)
1824     #
1825
1826     def _test_data_class():
1827         """ test function with y_test (actual values) and predictions
↪   (predic) """
1828         #data

```

```

1829     y_test = np.array([1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5,
↳ 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5,
↳ 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5,
↳ 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5,
↳ 1,2,3,4,5])
1830     predic = np.array([1,2,4,3,5, 1,2,4,3,5, 1,2,3,4,4, 1,4,3,4,5,
↳ 1,2,4,4,5, 1,2,4,4,5, 1,2,4,4,5, 1,2,4,4,5, 1,2,3,3,5, 1,2,3,3,5,
↳ 1,2,3,4,4, 1,2,3,4,1, 1,2,3,4,1, 1,2,3,4,1, 1,2,4,4,5, 1,2,4,4,5,
↳ 1,2,4,4,5, 1,2,4,4,5, 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5,
↳ 1,2,3,4,5])
1831     """
1832     Examples to validate output (confusion matrix plot)
1833     actual: 5 and prediction 1    >> 3
1834     actual: 2 and prediction 4    >> 1
1835     actual: 3 and prediction 4    >> 10
1836     """
1837     columns = []
1838     annot = True;
1839     cmap = 'Oranges';
1840     fmt = '.2f'
1841     lw = 0.5
1842     cbar = False
1843     show_null_values = 2
1844     pred_val_axis = 'y'
1845     #size::
1846     fz = 12;
1847     figsize = [9,9];
1848     if(len(y_test) > 10):
1849         fz=9; figsize=[14,14];
1850     plot_confusion_matrix_from_data(y_test, predic, columns,
1851         annot, cmap, fmt, fz, lw, cbar, figsize, show_null_values,
↳ pred_val_axis)
1852     #
1853
1854
1855     #

```

```
1856 #MAIN function
1857 #
1858 if(__name__ == '__main__'):
1859     print('__main__')
1860     print('_test_cm: test function with confusion matrix done\nand
↪ pause')
1861     _test_cm()
1862     plt.pause(5)
1863     print('_test_data_class: test function with y_test (actual
↪ values) and predictions (predic)')
1864     _test_data_class()
```

---





# Bibliography

- [1] Sports Analytics Market*Sports Analytics Market*. Market Research Firm, [www.marketsandmarkets.com/Market-Reports/sports-analytics-market-35276513.html](http://www.marketsandmarkets.com/Market-Reports/sports-analytics-market-35276513.html).
- [2] Confusion Matrix Pretty Print: Wagner Cipriano, [https://github.com/wcipriano/pretty-print-confusion-matrix/blob/master/confusion\\_matrix\\_pretty\\_print.py](https://github.com/wcipriano/pretty-print-confusion-matrix/blob/master/confusion_matrix_pretty_print.py)