Noelle Brown
Alexandra Norman
Thejas Prasad

MSDS 6372 Project 2

Intro

As a gold medalist, the MVP for multiple seasons, and the second highest scoring player in a single game in NBA history, it is indisputable that Kobe Bryant was an accomplished and dedicated player. Loyal to the Los Angeles Lakers for 20 seasons, Bryant ranks as the third scoring leader in the NBA with a total of 33,643 points scored throughout his regular season career (information found here and here). With the data that Bryant created over these 20 years, models can be created to predict whether or not he is expected to make shots given various conditions.

Data Description

With 20 years of basketball stats, Bryant has data on everything ranging from the location on the court that he took the shot to the average noise in the arena at the time of the game. The data was originally compiled with 5000 missing data points on whether or not Bryant made the shot. A list of the field names, their data types, and a brief description is provided below:

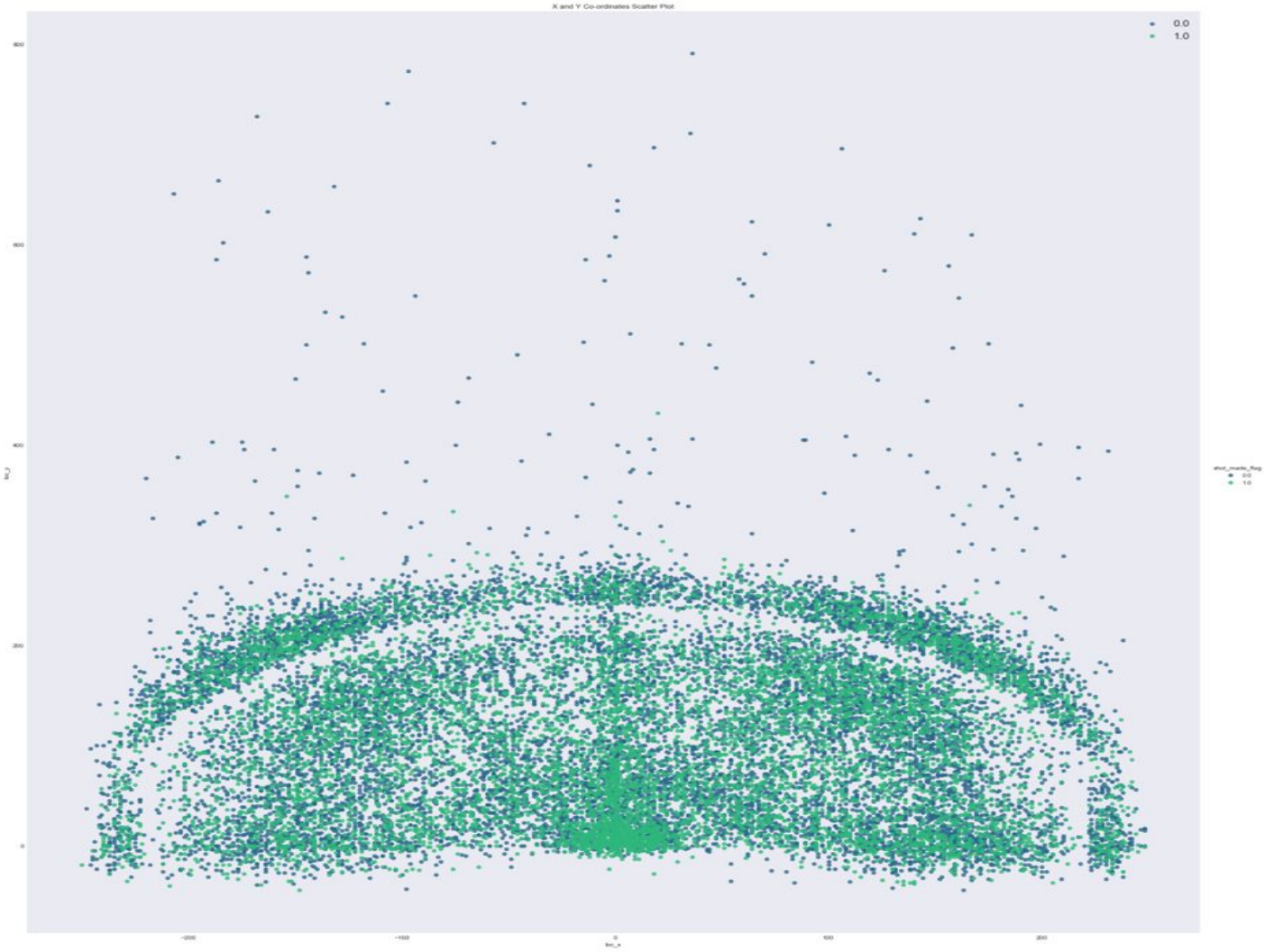
Action_type (Character): The type of action taken at the time of the shot: jump, driving dunk, layup, running jump, driving layup, reverse dunk, slam dunk, alley oop dunk, or turnaround.
Combined_shot_type (Character): The combined type of shot taken: jump, dunk, layup, hook, or tip.
Game_event_id (Numeric): The ID number of the game event.
Game_id (Numeric): The ID number of the game.
Lat (Numeric): The latitude of the location on the court where the shot was taken.
Loc_x (Numeric): The location on the x-axis on the court where the shot was taken.
Loc_y (Numeric): The location on the y-axis on the court where the shot was taken.
Lon (Numeric): The longitude of the location on the court where the shot was taken.
Minutes_remaining (Numeric): Number of minutes remaining in the period.
Period (Numeric): Which period (1-4) the shot was taken in.
Playoffs (Numeric): Whether or not the shot was taken in a playoff game (0 for no and 1 for yes).
Season (Date): The season year.
Seconds_remaining (Numeric): The number of seconds remaining in the period at the time of the shot.
Attendance (Numeric): The number of people in attendance at the game.
Avgnoisedb (Numeric): The average noise in decibels at the game.
Shot_distance (Numeric): The distance the the shot traveled to the hoop.
Shot_made_flag (Numeric): Whether or not the shot was made (0 for no and 1 for yes).
Shot_type (Character): The type of shot taken: 2 pointer or 3 pointer.
Shot_zone_area (Character): The area of the shot zone: left, left-center, right, right-center, or center.
Shot_zone_basic (Character): The shot zone: mid-range, restricted area, in the paint, or above the break 3.
Shot_zone_range (Character): The range of the shot: less than 8 feet, 8-16 feet, 16-24 feet, or 24+ feet.
Team_id (Numeric): The ID of the team that he played on at the time of the shot.
Team_name (Character): The name of the team that he played on at the time of the shot.
Game_date (Date): The date of the game at the time of the shot.
Matchup (Character): The opposing teams and the location (home or away).
Opponent (Character): The team that he played against.
Shot_id (Numeric): The ID of the shot.
Arena_temp (Numeric): The average temperature of the arena during the game in degrees Fahrenheit.

Noelle Brown
Alexandra Norman
Thejas Prasad

Exploratory Data Analysis

A few variables were added to the dataset to aid with this analysis: r, theta, total_sec_remaining, and home_flag. R and theta were created by converting the x and y coordinates to polar coordinates. This was due to the fact that the distribution of shot attempts on the axis looked circular around the hoop, which is to be expected for attempts around a basketball hoop (see figure 1 below). We suspect that polar coordinates might be more appropriate for this design than cartesian and will attempt to use this in some of our models.

Figure 1

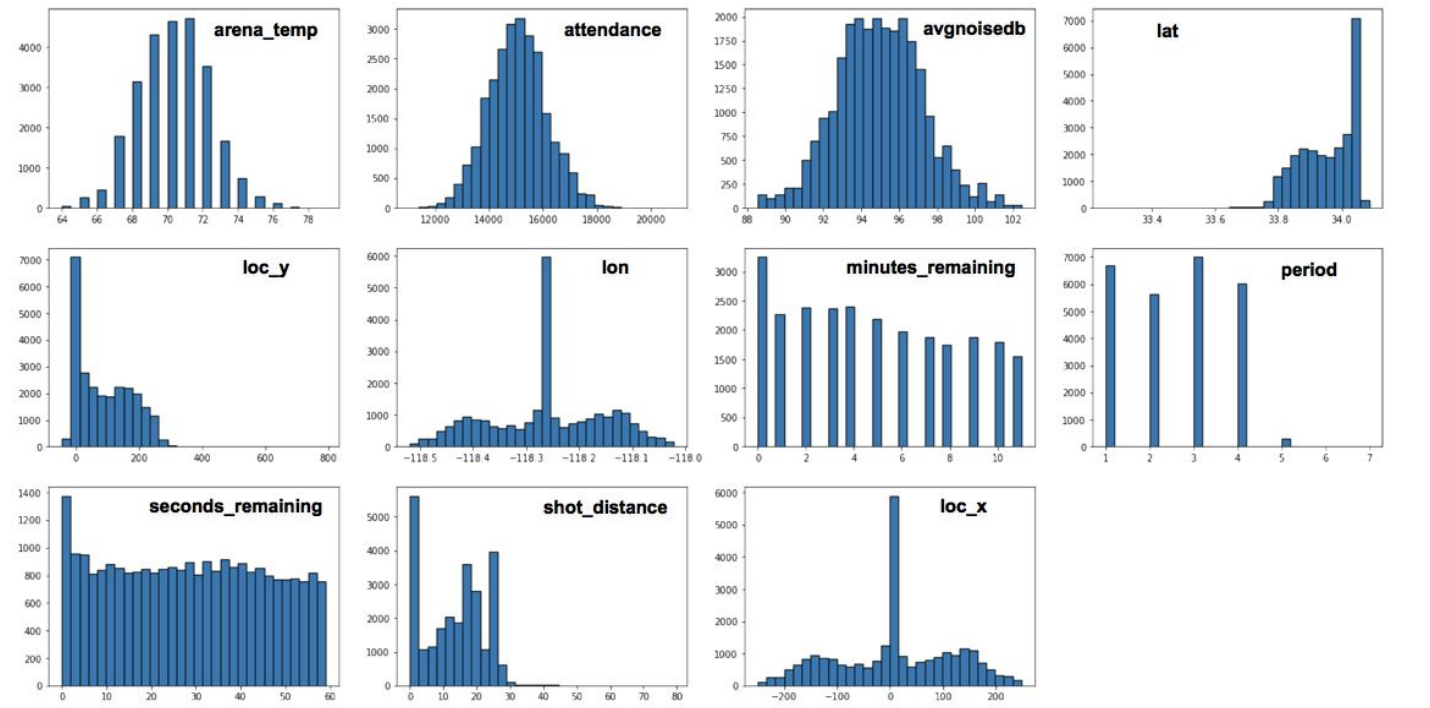


The total_sec_remaining column was created by converting the minutes remaining into seconds and then adding them to the seconds remaining information. This allows us to see the total amount of time left in the period at the time of the shot. The home_flag variable was created to identify which shots were taken at home games and which shots were taken at away games. These variables could potentially provide our analysis with important or interesting information.

After looking at plots of the data, the avgnoisedb, attendance, and loc_x variables seem fairly normally distributed, while loc_y, shot_distance, and seconds_remaining/total_sec_remaining seem slightly right skewed (see figure 2 below). However, after analyzing these variable distributions with transformations

applied, the figures are still slightly skewed. Since the data set is large enough where the central limit theorem will ensure the sampling mean distribution will approach normality. The lon and loc_x have a spike in the middle, which could be explained by the higher number of shots taken right in front of the net (dunks, free throws, layups, etc.). While these may look like they could be outliers, these observations provide us with important information, so we will keep all of the data in these variables. Therefore, we will assume normally distributed distributions and proceed. The arena_temp, minutes_remaining, and period variables are partitioned in such a way that it leads us to investigate whether these should be treated as categorical variables.

Figure 2



We analyzed any potential outliers and came to the conclusion that all of the data was part of the population and should remain in our dataset. The biggest concern we had was for “buzzer shots,” or points made at the last second of the period from a potentially long distance from the hoop. We thought this could skew the data, as the majority of these shots are missed in most games. However, after analyzing these we found that the maximum distance shot was 79 feet away from the hoop, with the penultimate distance 77 feet followed by several shots at 74 feet. Since there do not seem to be high outliers here, and these shots provide us with interesting information, we will keep these in the model. Since basketball is a relatively controlled sport, we felt as though the rest of the variables also provided necessary information that was part of the population of interest, so all of the data points were kept in the set. This is important to note when running a discriminant analysis, as this method is sensitive to outliers.

After investigating potential multicollinearity, it was confirmed that several variables could be predicted by a linear combination of the other variables. Since loc_x, loc_y, lat, lon, r, and theta all explain the same information, we do not have to include all of these in our model - only one pair. Shot_zone_area, shot_zone_range, and shot_zone_basic all provide similar information, so we will remove shot_zone_range in some models and test whether or not to keep shot_zone_area once we investigate the models. Matchup and opponent provide us with the same information, so we can remove one of these columns. Since we combined minutes_remaining and seconds_remaining, we can also remove these. Shot_distance provides us with the

same information that the radius from the polar coordinate conversion (r) provides, so we can remove one of these variables as well. We will experiment with removing different variables in different models to see if these will have any effect on the model. Additionally, shot_id, team_id, team_name, game_event_id, and game_id do not provide us with any useful information for prediction, so we will remove these from the analysis.

To ensure that the assumptions are met to run a logistic regression, we confirm that our response is binary (this assumption is met since our response is whether or not the shot was made) and our observations are independent (assumed). Additionally, there is not enough evidence to suggest that there is significant lack of fit of this model (p-value = 0.5866 from a Hosmer-Lemeshow test), so we will assume that the log odds of making a shot is linearly related to the continuous explanatory variables. Since all of the assumptions for logistic regression are met, we can proceed with this analysis.

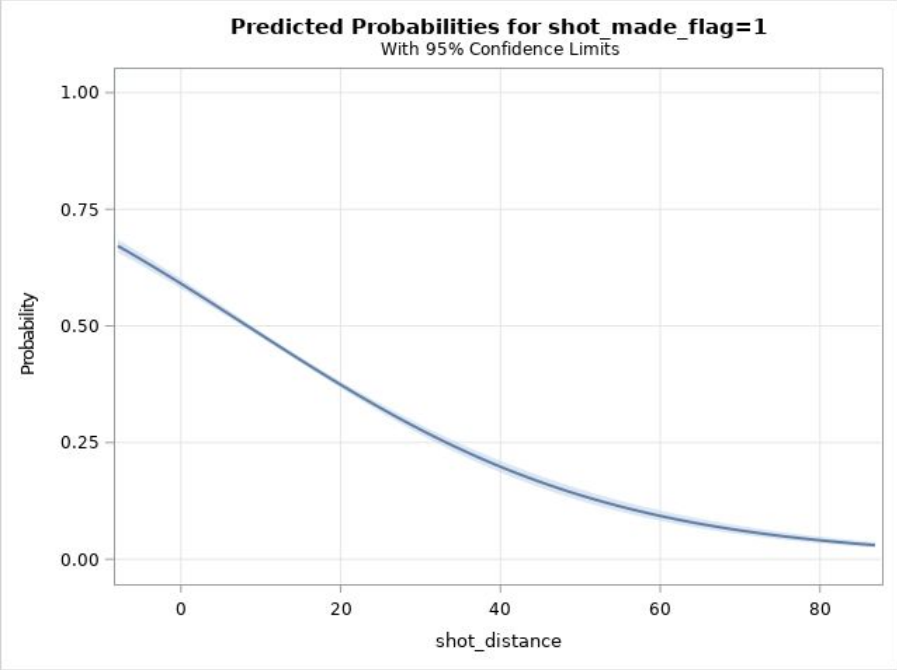
To run a linear discriminant analysis, our data needs to be normally distributed (addressed above), multivariate normally distributed (robust since we have such a large data set), independent, and have a common variance-covariance matrix. We will assume that our data is independent and proceed with this assumption. However, there is strong evidence that we do not meet the common variance-covariance matrix assumption (p-value < 0.0001 from Bartlett's test). Since this assumption is not met, we will have to run a quadratic discriminant analysis rather than a linear discriminant analysis for optimal results.

Interpretation Models / Questions

Question of Interest 1: Do the odds of Kobe making a shot decrease with respect to the distance he is from the hoop?

Analyzing soley the odds of Kobe making a shot with respect to the distance he is from the hoop, there is evidence that the odds of making a shot decrease as distance from the hoop increases. The estimated odds ratio for this relationship is 0.957, suggesting that for every one-foot increase in distance from the net, the odds of making the shot decrease by 4.3%. A 95% confidence interval for this estimation is between a 4% and 4.6% decrease in odds. Below (figure 3) is a plot of the predicted probability of making a shot versus the distance that Kobe is away from the net, which shows a clear negative relationship. This provides visual evidence that the probability of making a shot decrease as he shoots further from the hoop.

Figure 3



Noelle Brown
Alexandra Norman
Thejas Prasad

Now suppose we would like to account for a few variables that could potentially influence whether or not the shot is made in addition to the distance from the hoop. Since the angle that Kobe shoots from on the court and the type of shot that he takes could be closely related to the distance away from the net, another analysis is conducted to see if results are consistent after accounting for these variables. There is evidence that the odds of Kobe making the shot decrease by an estimated 2.5% with every one-foot increase in distance from the net after accounting for the angle that the shot was taken from and the type of shot. A 95% confidence interval for this estimation is between 2.1% and 2.9%.

Question of Interest 2: Does the probability of Kobe making a shot decrease linearly with respect to the distance he is from the hoop?

After running a Hosmer-Lemeshow goodness of fit test, there is evidence to suggest that there is significant lack of fit for this model that tests whether the probability of making a shot decreases linearly with respect to the distance Kobe is away from the net (p-value <0.0001). This suggests that the probability of making a shot is not linearly related to the distance he is from the hoop.

Question of Interest 3: Does the relationship between the distance Kobe is from the hoop and the odds of him making the shot differ if they are in the playoffs?

The odds of making a shot continue to decrease the further Kobe is from the net after accounting for whether or not his is in the playoffs. The estimated decrease in odds of making the shot is 4.3% for each one-foot increase in distance from the net after accounting for whether or not the game is in the playoffs. A 95% confidence interval for this estimated decrease is between 4.1% and 4.6%. Results are consistent after accounting for the angle of the shot as well as the type of shot, as in question of interest #1. The estimated odds of making a shot decrease by 2.5% for every one-foot increase in distance from the net after accounting for playoffs, shot angle, and shot type. A 95% confidence interval for this estimation is between 2.1% and 2.9%.

Question of Interest 4 (Bonus): Is Kobe's shooting percentage subject to a home field advantage? That is, is Kobe's shooting percentage is better or worse at home than when he is away?

There is sufficient evidence at that alpha=0.05 level of significance to support the claim that Kobe's shooting percentage is better when he is at home (p-value = 0.0006). The odds that Kobe will make a shot when he is at home is estimated to be 1.0845 times that of his away odds. A 95% confidence interval for this estimated odds ratio is (1.0324, 1.1392). This give statistical evidence that Kobe's shooting percentage is subject to a home field advantage.

Question of Interest 5 (Bonus): Is Kobe "clutch"? Kobe is often the coach's choice to take the final shot of the period or game. What do we know about his shooting percentage in these moments?

There is significant evidence that Kobe performs better during the last 5 seconds of the period than other shots throughout the period (p-value <0.0001). The estimated odds of making a shot during the last 5 seconds of the period is 2.3046 times higher than the odds of making a shot during any other time in the period. A 95% confidence interval for this estimation is (1.989, 2.6702). Interestingly, there is evidence to suggest that the odds of Kobe making a shot during the last 5 seconds of the game is lower than the odds of making any other shot in the game (p-value <0.0001). The estimated odds of making a shot during the last 5 seconds of the game is 0.4621 times lower than the odds of making a shot during any other time in the game. A 95% confidence interval for this estimate is (0.3291, 0.649). Kobe may be the best choice to put in during the final moments of most periods, but the pressure may get to him during the final shot of the game. An interesting extension to this analysis would be to see if the results are consistent during different intervals of time, such as the final 30 seconds or minute of the period or game.

Noelle Brown
Alexandra Norman
Thejas Prasad

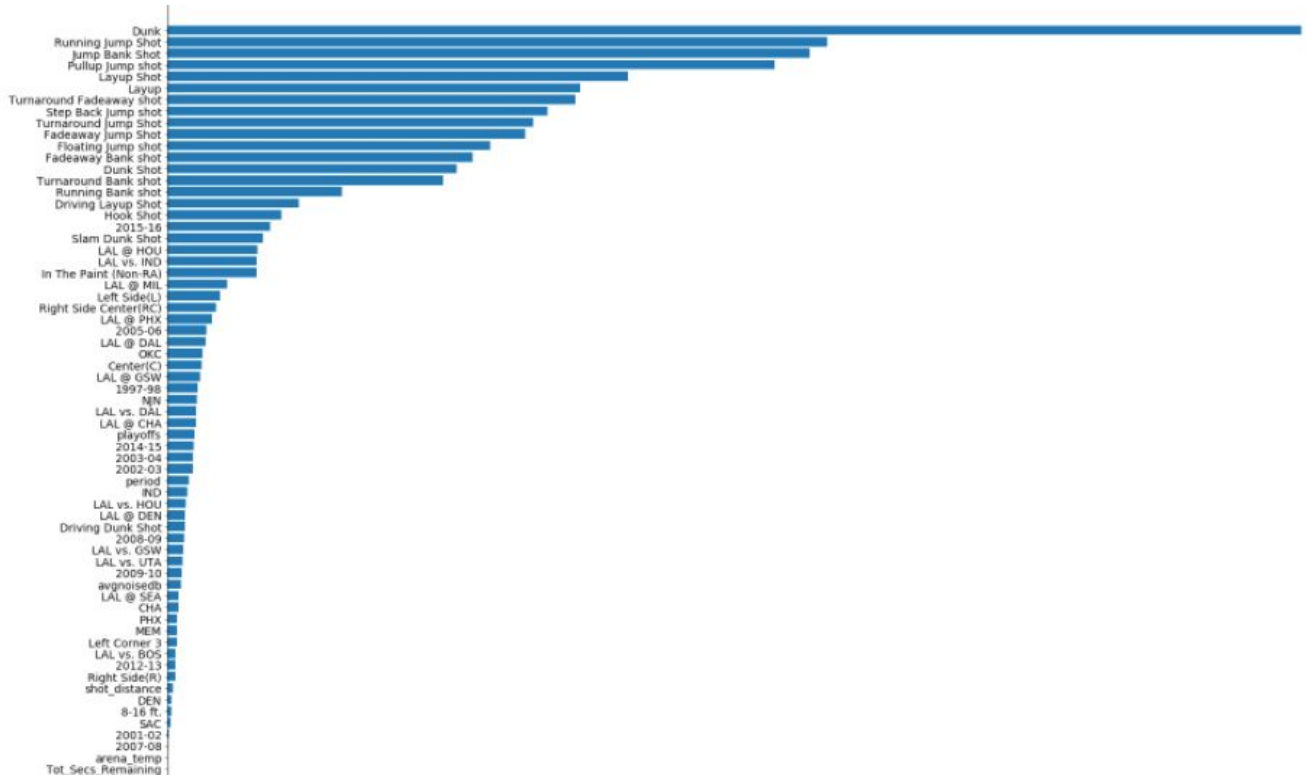
Predictive Models

Logistic Regression

Model 1: To come up with our first model, we used a forward selection technique in SAS. The selected effects were action_type, avgnoisedb, season, total_sec_remaining, arena_temp, shot_zone_basic, r, and period. This created a log loss value of 0.60477 and an area under the curve (AUC) value of 0.7045. Since our other model was created in Python, we will use a probability level of 50% in the following model statistics for consistency, since Python automatically selects this level in its output. This model produces a misclassification rate of 31.8%, a sensitivity level of 0.461, and a specificity level of 0.86.

Model 2: Our second model was created in Python and had a log loss value of 0.603598, an AUC of 0.71, a misclassification rate of 31.58%, a sensitivity of 0.4569, and a specificity level of 0.8648. What is interesting to note about this model after running a feature importance analysis is the high effect that an action type of a dunk has on the probability of making a shot. As seen below in figure 4, a dunk is the most important feature when predicting whether or not the shot will be made. This is most likely due to the fact that a dunk is a very controlled shot and is usually made when given the chance to make a shot of this type.

Figure 4



Model 3: The third logistic regression model was also created in Python. This model performed Principal Components Analysis to attempt to reduce the dimensions of the model prior to running a logistic regression. This model had the following metrics: log loss value of 0.738, an AUC of 0.4276, a misclassification rate of 53.23%, a sensitivity of 0.3316, and a specificity level of 0.5758.

Noelle Brown
Alexandra Norman
Thejas Prasad

Linear Discriminant Analysis (LDA) / Quadratic Discriminant Analysis (QDA)

The next model was created using a discriminant analysis method (see Exploratory Data Analysis section for a discussion on which technique to use based on the assumptions). Even though an assumption for LDA is not met, we will attempt this model for research and exploratory purposes and note that the predictions might not be optimal. The LDA model provided us with a los loss of 0.6044, an AUC of 0.7112, a misclassification rate of 31.556%, sensitivity of 0.4584 and specificity of 0.8641. We then attempted to first run a Principal Components Analysis and try the model using the principal components as part of the model. Interestingly, the model seemed to get worse with a misclassification rate of 53.63%. After this, we looked at a Quadratic Discriminant Analysis, which did not seem to predict as well as the LDA model even though the assumptions are met for this model. This model had a log loss of 13.578, an AUC of 0.5589, a misclassification rate of 53.23%, sensitivity of 0.3316, and specificity of 0.5759.

K-Means Clustering Analysis

Finally, we created a model using a K-Means Clustering Analysis method. This model did not do a great job minimizing the objective function; it had a log loss value of 16.919, an AUC of 0.5102, a misclassification rate of 35.499%, sensitivity of 0.4941, and specificity of 0.7649. In the future, it would be interesting to continue working on a clustering analysis method with this data and attempt to see if this model could improve on predictability and accuracy.

Model Comparison

	AUC	Misclassification Rate	Sensitivity	Specificity	Objective / Log Loss Function
Logistic Regression Model 1	0.7045	31.8%	0.461	0.86	0.60477
Logistic Regression Model 2	0.71	31.58%	0.4569	0.8648	0.603598
Logistic Regression Model 3	0.4276	53.23%	0.3316	0.5758	0.738
LDA	0.7112	31.556%	0.4584	0.8641	0.6044
K-Means Clustering	0.5101	35.499%	0.4941	0.7649	16.9194

Shooting Percentage Over Time

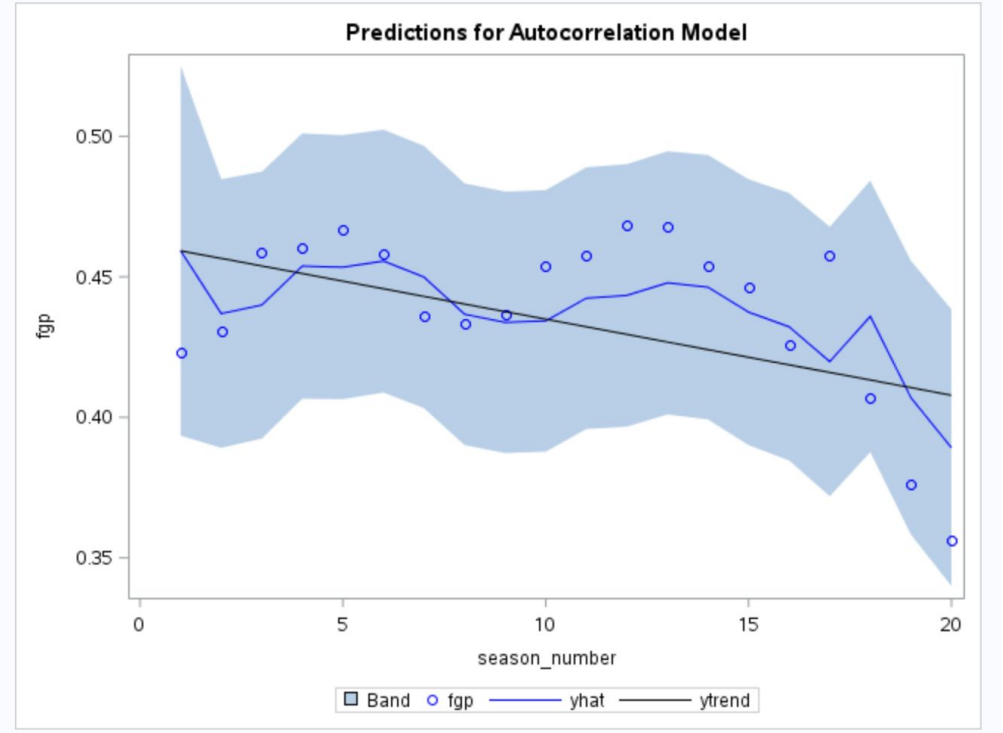
Question of Interest 6 (Bonus): Does Kobe's shooting percentage appear to get better over time?

We initially explored looking at Kobe's shooting percentage per year, but we were skeptical about the interpretability of this analysis based on the way basketball seasons are structured. Because of this, we decided to look at the shooting percentage over the seasons. The reason we decided to look at the seasons as opposed to the years is since a basketball season spans from the end of one year to the

beginning of the next year it may be a more accurate depiction of Kobe's shooting percentage over time. To be able to do an analysis on season we had to force the season variable to be a character. Once we imported the data, we created the variables season_number and fgp to be able to look at Kobe's shooting percentage over time. The season_number is simply the numbering of the seasons that Kobe played. The fgp (field goal percentage) column is the number of shots made divided by the total number of shots Kobe took.

Visually it looks like Kobe's shooting percentage has ups and downs, but overall is slightly decreasing over time, as can be seen in figure 5 below.

Figure 5



There is sufficient evidence to suggest that there is lag 1 serial correlation in the data (p-value < 0.0001). The Durbin-Watson statistic value is less than 2 (0.5426). This suggests that for every one increase in season that Kobe has played there is 0.0024 (0.24%) decrease in shooting percentage. Ultimately, Kobe's shooting percentage does not appear to get better over time - there is evidence that it seems to decline over time.

Conclusion

In conclusion, the second Logistic Regression model provided us with the optimum log loss score of 0.6036 and the largest area under the curve. This model was created using a grid search technique in Python to identify the parameters that should be included in the model. Ultimately, this model does the best job in classifying and predicting whether or not the shots were made or missed. Further analysis could be done using additional models to attempt to minimize the log loss function and improve the misclassification rate. In the

future, it would be interesting to extend this analysis to other NBA players to see if there are trends among players or teams.

Noelle Brown
Alexandra Norman
Thejas Prasad

Appendix

Code

Exploratory Data Analysis

SAS

```
/*create a total seconds remaining variable by combining minutes & seconds remaining; converts cartesian to
polar coordinates; create a home flag to indicate if game was home or away*/
data Kobe;
set KobeDataProj2;
min = minutes_remaining*60;
total_sec_remaining = min + seconds_remaining;
r = sqrt((loc_x)**2+(loc_y)**2);
if loc_x = '0' then theta=constant("pi")/2;
else theta = atan(loc_y/loc_x);
if substr(matchup,5,3) = 'vs.' then home_flag= 1;
else home_flag=0;
run;

/*split NA values and non-NA values of shot_made_flag*/
data kobetrain;
set Kobe;
where shot_made_flag ^= 'NA';
run;

data kobetest;
set Kobe;
where shot_made_flag = 'NA';
run;

/*looking at normality*/
proc univariate data=Kobe;
var loc_x loc_y attendance avgnoisedb shot_distance total_sec_remaining r theta;
histogram;
run;

/*checking for outliers*/
proc univariate data=Kobe;
var total_sec_remaining shot_distance;
run;

data test;
set Kobe;
keep shot_distance shot_id total_sec_remaining shot_made_flag;
where shot_distance > 60;
run;
proc sort data=test;
by descending shot_distance;
run;
```

```
proc print data=test;
run;

/*check assumptions: equal covariance matrix*/
proc discrim data=Kobe pool=test;
class shot_made_flag;
var loc_x loc_y minutes_remaining seconds_remaining attendance avgnoisedb shot_distance
total_sec_remaining;
run;
/*evidence for non-equal covariance: can use quadratic regression*/

/*checking for multicollinearity*/
proc corr data=Kobe;
var shot_distance lat loc_x loc_y lon minutes_remaining seconds_remaining attendance avgnoisedb
total_sec_remaining;
run;

/*remove loc_x, loc_y, lat, minutes_remaining, seconds_remaining, attendance & check again*/
proc corr data=Kobe;
var shot_distance lon avgnoisedb total_sec_remaining;
Run;
/*results: remove shot_id team_id team_name shot_zone_area shot_zone_range matchup loc_x loc_y lat lon
seconds_remaining minutes_remaining game_event_id game_id game_date*/

/*check assumptions: logistic regression*/
proc logistic data=Kobetrain descending outmodel=results plots=all;
class action_type period playoffs /*shot_zone_area*/ season shot_made_flag shot_type shot_zone_basic
opponent arena_temp;
model shot_made_flag = action_type period playoffs /*shot_zone_area*/ season shot_type shot_zone_basic
opponent arena_temp avgnoisedb total_sec_remaining r theta / selection = forward ctable lackfit;
score data=Kobe out=score1;
run;
```

Python

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

#from IPython.core.display import display, HTML
#display(HTML("<style>.container { width:100% !important; }</style>"))

# Read the data
kobe = pd.read_csv('KobeDataProj2.csv')
kobe.head()
Kobe.shape
kobe.info()

# To get a top view of missing data use heatmap
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
#axe = plt.axes()
plt.figure(figsize=(15, 7))
sns.heatmap(data = kobe.isnull(), yticklabels =False, cbar=False, cmap='viridis')
#axe.set_title('Heat Map of Full Data')

kobe_full = kobe.drop(['shot_made_flag'], axis =1)
#kobe_full.info()
Kobe_full.shape
## Create a new Data Frame to do EDA
kobe_train = kobe.dropna()
kobe_train.describe()
sns.jointplot(x= 'arena_temp', y = 'avgnoisedb', data=kobe_train)

# Kobes score / no score trend(count) aganist each teams
plt.figure(figsize=(15,5))
sns.countplot(x=kobe_train['opponent'], hue = kobe_train['shot_made_flag'], palette='plasma')

# Kobe's shooting trend in the last 5 secs with each team
plt.figure(figsize=(15,5))
sns.countplot(x=kobe_train[(kobe_train['shot_made_flag']==1) & (kobe_train['seconds_remaining'] < 6)
]['opponent'])

# Kobe's shot miss trend in the last 5 secs with each team
plt.figure(figsize=(15,5))
sns.countplot(x=kobe_train[(kobe_train['shot_made_flag']==0) & (kobe_train['seconds_remaining'] <
6)]['opponent'])

# Kobes shot range and scoring trend in playoffs
plt.figure(figsize=(15,5))
sns.countplot(x=kobe_train[kobe_train['shot_made_flag']== 1]['shot_zone_range'],
            hue = kobe_train['playoffs'],palette='viridis')

# Kobes shot range and score / no score count both (0 & 1)
plt.figure(figsize=(15,5))
sns.countplot(x=kobe_train['shot_zone_range'], hue = kobe_train['shot_made_flag'])

kobe_train['shot_made_flag'].unique()
Np.nan
kobe_train[kobe_train['shot_made_flag']== 1]['minutes_remaining'].plot.hist(bins = 30)
kobe_train['arena_temp'].plot.hist(bins = 30)
kobe_train['avgnoisedb'].plot.hist(bins = 30, ec='black')

plt.figure(figsize=(45,5))
#plt.scatter(kobe_train['loc_x'], kobe_train['loc_y'],)
sns.lmplot('loc_x', 'loc_y',kobe_train, hue='shot_made_flag', fit_reg=False,size=25, palette='rainbow' )

#plt.rcParams['figure.figsize']=(50,10)
g = sns.lmplot(x='Tot_Secs_Remaining',
          y='shot_distance', data =full[full['shot_made_flag'] == 1])
g.fig.set_size_inches(30,10)
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
# find out the correlation between variables
kobe_full.corr()

#Find highly correlated pairs
corr_matrix = kobe_full.corr().abs()
high_corr_var=np.where(corr_matrix>0.8)
high_corr_var=[(corr_matrix.index[x],corr_matrix.columns[y]) for x,y in zip(*high_corr_var) if x!=y and x<y]
High_corr_var

# Delete Correlated and other non significant Variables
kobe_full = kobe_full.drop(['team_name','team_id','game_date','game_id', 'game_event_id','lat','lon','loc_x',
                            'loc_y'], axis = 1 )
kobe_full.info()
#sns.pairplot(full_train[['Tot_Secs_Remaining','shot_distance']], hue=full_train['playoffs'])

# Deal with categorical values in full data
action_type_dmyfull = pd.get_dummies(kobe_full['action_type'], drop_first=True)
combined_shot_type_dmyfull = pd.get_dummies(kobe_full['combined_shot_type'], drop_first=True)
shot_type_dmyfull = pd.get_dummies(kobe_full['shot_type'], drop_first=True)
shot_zone_area_dmyfull = pd.get_dummies(kobe_full['shot_zone_area'], drop_first=True)
shot_zone_basic_dmyfull = pd.get_dummies(kobe_full['shot_zone_basic'], drop_first=True)
shot_zone_range_dmyfull = pd.get_dummies(kobe_full['shot_zone_range'], drop_first=True)
matchup_dmyfull = pd.get_dummies(kobe_full['matchup'], drop_first=True)
opponent_dmyfull = pd.get_dummies(kobe_full['opponent'], drop_first=True)
season_dmyfull = pd.get_dummies(kobe_full['season'], drop_first=True)

# Concatenate the Dummy variables to the full DF
kobe_full = pd.concat([kobe_full,action_type_dmyfull,combined_shot_type_dmyfull,shot_type_dmyfull,
                       shot_zone_area_dmyfull,
                       shot_zone_basic_dmyfull,shot_zone_range_dmyfull,matchup_dmyfull,
                       opponent_dmyfull,season_dmyfull],
                       axis = 1 )
kobe_full.info()

### Feature Engineering
# Create new column for total seconds remaining
kobe_full['Tot_Secs_Remaining'] = kobe_full['seconds_remaining'] +  (kobe_full['minutes_remaining'] * 60)

# Drop the categorical variables existing in the full DF, Included date 'game_date' to drop for now, need review
this.
full = kobe_full.drop(['action_type','combined_shot_type','shot_type','shot_zone_area','shot_zone_basic',
                       'shot_zone_range','matchup','opponent','season','seconds_remaining','minutes_remaining'
                        ], axis = 1 )
full.info()
full.corr()

#Find highly correlated pairs
corr_matrix = full.corr().abs()
high_corr_var=np.where(corr_matrix>0.8)
```

```
high_corr_var=[(corr_matrix.index[x],corr_matrix.columns[y]) for x,y in zip(*high_corr_var) if x!=y and x<y]
High_corr_var

# Drop the highly correlated pairs
full = full.drop(['Less Than 8 ft.','Tip Shot','Jump Shot','Restricted Area','24+ ft.','Back Court Shot','BKN'
                  ], axis = 1 )
full = pd.DataFrame(pd.concat([full, kobe[['shot_made_flag', 'shot_id']]], axis=1 ))
```

Interpretation Models / Questions

SAS

```
/*1: do odds decrease with respect to distance. Look at H-L test for 2.*/
proc logistic data = kobetrain plots=all;
class shot_made_flag (ref='0');
model shot_made_flag =  shot_distance / lackfit ctable;
run;

/*accounting for theta and type of shot*/
proc logistic data = kobetrain plots=all;
class shot_made_flag (ref='0') combined_shot_type;
model shot_made_flag = shot_distance theta combined_shot_type / lackfit ctable;
run;

/*3. accounting for playoffs*/
proc logistic data = kobetrain plots=all;
class shot_made_flag (ref='0') playoffs (ref='0');
model shot_made_flag =  shot_distance playoffs / lackfit ctable;
run;

/*accounting for playoffs, theta, type of shot*/
proc logistic data = kobetrain plots=all;
class shot_made_flag (ref='0') combined_shot_type playoffs(ref='0');
model shot_made_flag = shot_distance theta combined_shot_type playoffs / lackfit ctable;
Run;
/*bonus 1 - QOI 4*/
/*get counts for made vs. miss & home vs. away*/
proc freq data=kobetrain order=data;
tables shot_made_flag*matchup;
run;

/*input count data & run proc freq*/
proc format;
  value ExpFmt 1='Home'
               0='Away';
  value RspFmt 1='Shot Made'
               0='Shot Missed';
run;
data home_vs_away;
  input location shot Count;
  datalines;
0 0 7446
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
0 1 5766
1 0 6786
1 1 5699
;
proc sort data=home_vs_away;
  by descending location descending shot;
run;

proc freq data=home_vs_away order=data;
  format location ExpFmt. shot RspFmt.;
  tables location*shot / riskdiff(equal var=null) relrisk alpha = .05;
  weight Count;
run;

/*Bonus 2/QOI 5: order data*/
proc sort data=kobetrain;
by descending seconds_remaining descending minutes_remaining;
run;

/*get counts of <=5 and >5 seconds remaining*/
proc freq data=kobetrain order=data;
tables shot_made_flag*seconds_remaining*minutes_remaining;
run;

/*input count data & run proc freq*/
proc format;
   value ExpFmt 1='Shot in last 5 seconds of period'
                0='All other shots in period';
   value RspFmt 1='Shot Made'
                0='Shot Missed';
run;
data clutch;
   input seconds shot Count;
   datalines;
0 0 11217
0 1 13542
1 0 248
1 1 690
;
proc sort data=clutch;
   by descending seconds descending shot;
run;

proc freq data=clutch order=data;
   format seconds ExpFmt. shot RspFmt.;
   tables seconds*shot / riskdiff(equal var=null) relrisk alpha = .05;
   weight Count;
run;

/*Bonus 2 part 2: keep only period 4 in one data set and all others in another data set*/
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
data kobeclutch;
set kobetrain;
if period = '1' then delete;
if period = '2' then delete;
if period = '3' then delete;
run;

data kobeclutch2;
set kobetrain;
if period = '4' then delete;
run;

/*Order data*/
proc sort data=kobeclutch;
by descending total_sec_remaining;
run;
proc sort data=kobeclutch2;
by descending total_sec_remaining;
run;

/*get counts of <=5 and >5 seconds remaining*/
proc freq data=kobeclutch order=data;
tables shot_made_flag*total_sec_remaining;
run;
proc freq data=kobeclutch2 order=data;
tables shot_made_flag*total_sec_remaining;
run;

/*input count data & run proc freq*/
proc format;
  value ExpFmt 1='Shot in last 5 seconds of game'
               0='All other shots';
  value RspFmt 1='Shot Made'
               0='Shot Missed';
run;
data clutch2;
  input seconds shot Count;
  datalines;
0 0 14285
0 1 11560
1 0 123
1 1 46
;
proc sort data=clutch2;
  by descending seconds descending shot;
run;

proc freq data=clutch2 order=data;
  format seconds ExpFmt. shot RspFmt.;
  tables seconds*shot / riskdiff(equal var=null) relrisk alpha = .05;
```

```
   weight Count;
run;
```

Python

```
#2   The probability of Kobe making a shot decreases linearly with respect to the distance he is from the hoop.
#   If there is evidence of this, quantify this relationship.  (CIs, plots, etc.)
plt.figure(figsize=(40,15))
#sns.lmplot(data = kobe_train, x='shot_distance', y ='seconds_remaining')
sns.countplot(x=kobe_train[(kobe_train['shot_made_flag']==1)]['shot_distance'])
```

Predictive Models

Logistic Regression

SAS

```
/*logistic regression model - forward selection*/
proc logistic data=Kobetrain descending outmodel=results plots=all;
class action_type period playoffs /*shot_zone_area*/ season shot_made_flag shot_type shot_zone_basic
opponent arena_temp;
model shot_made_flag = action_type period playoffs /*shot_zone_area*/ season shot_type shot_zone_basic
opponent arena_temp avgnoisedb total_sec_remaining r theta / selection = forward ctable;
score data=Kobe out=score1;
run;

/*export data*/
proc logistic inmodel=results;
     score data=kobetest out=Score2;
  Run;
/*to calculate log loss: score data=kobetrain out=Score2*/

data submit;
set Score2;
keep shot_id P_1;
Run;

/* Log Loss Calculation */
data info;
set Score2;
prelogloss = (shot_made_flag*log(P_1) + (1-shot_made_flag)* log(1-P_1));
run;

proc means data=info sum;
var prelogloss;
output out=logloss sum=loglosssum n=obs;
run;
```

```
data info2;
set logloss;
finallogloss = ((-1/obs)*loglosssum);
run;
```

Python

```
# train data
full_train = pd.DataFrame(full.dropna())
type(full_train)
full_train.info()
# We have a slightly unbalanced target variable
full_train['shot_made_flag'].value_counts()

# test data
full_test = full[full['shot_made_flag'].isnull()]
full_test.info()

# Define x & y
X = full_train.drop(['shot_made_flag'], axis = 1)
y = full_train['shot_made_flag']
#X = full_train_fi.drop(['shot_made_flag'], axis = 1)
#y = full_train_fi['shot_made_flag']
#X.head()

# Missing values Data to be predicted
y_to_predict = pd.DataFrame(full_test.drop(['shot_made_flag'], axis =1))
y_to_predict.head()
y_to_predict.info()

# Parameter tuning Logistic regression
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,log_loss,
roc_auc_score,accuracy_score,recall_score
from sklearn.model_selection import train_test_split

# create cross validation iterator
cv = StratifiedShuffleSplit( n_splits=10, test_size=0.2)

# tune the significant parameter C
C_param_range = [0.0001, 0.001,0.01,0.1,1,10,100,1000,10000]

for i in C_param_range:
    clf_lr = LogisticRegression(penalty = 'l2', C = i,random_state = 0)
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
    scores_lr = cross_val_score(clf_lr, X, y=y, cv=cv, scoring = 'neg_log_loss', n_jobs=-1)
# The mean accuracy score and the 95% confidence interval of the score estimate
    print(i,":CI of LogisticRegression LogLoss is %0.3f +/- %0.3f" % (scores_lr.mean(), scores_lr.std()*2))

# Hyper Parameter tuning using grid search - Logistic Regression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score
seed = 7
processors=1
num_folds=3
num_instances=len(X)
scoring='neg_log_loss'
kfold = KFold(n_splits=num_folds, shuffle= True, random_state=seed)
param_grid = {'penalty': ['l1','l2'], 'C': [0.1,1, 10, 100]}
lr_grid = GridSearchCV( LogisticRegression(random_state=seed),param_grid,#refit=True,verbose=2,
                       cv = kfold,
                       scoring = scoring,
                       n_jobs = processors
                      )
lr_grid.fit(X, y)
print(lr_grid.best_score_)
print(lr_grid.best_params_)

# Model training
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)

#scale & PCA train data
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

scalar = StandardScaler()
scalar.fit(X_train)
#X_train_scaled = preprocessing.scale(X_train)
X_train_scaled = scalar.transform(X_train)
estimator = PCA(n_components=5)
X_train_pca = estimator.fit_transform(X_train_scaled)
logismodelfull = LogisticRegression(verbose=10, C= 1, penalty='l1')

#pca
logismodelfull.fit(X_train_pca, y_train)

# scale & PCA test data
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
#X_test_scaled=preprocessing.scale(X_test)
scalar.fit(X_test)
X_test_scaled = scalar.transform(X_test)
X_test_pca=estimator.fit_transform(X_test_scaled)

# pca predict the y_test
y_test_pred=logismodelfull.predict(X_test_pca)
Y_test_pred

# Traditional LR
logismodelfull.fit(X_train, y_train)

# Traditional LR Predictions
predictions = logismodelfull.predict(X_test)

#  Grid Search Predictions --- Log Loss 10.8992619888 (reduced)
predictions = lr_grid.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix,log_loss,
roc_auc_score,accuracy_score,recall_score
print(classification_report( y_test, predictions))

# pca
print(classification_report(y_test , y_test_pred))
print(confusion_matrix(y_test,predictions))

#pca
print(confusion_matrix(y_test,y_test_pred))

# save confusion matrix and slice into four pieces
confusion = confusion_matrix(y_test,predictions)
print(confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

#pca
# save confusion matrix and slice into four pieces
confusion = confusion_matrix(y_test,y_test_pred)
print(confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
```

```
FP = confusion[0, 1]
FN = confusion[1, 0]

# Get statistics
log_loss_logReg = log_loss(y_test, predictions)
print('Log Loss',log_loss(y_test, predictions))
print('\n')
print('AUC',roc_auc_score(y_test, predictions))
print('\n')
sensitivity = TP / float(FN + TP)
print('Sensitivity', sensitivity)
print('recall_score', recall_score(y_test, predictions))
print('\n')
specificity = TN / (TN + FP)
print('Specificity', specificity)
print('\n')
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('classification_error', classification_error)
print('1 - accuracy_score', 1 - accuracy_score(y_test, predictions))

#pca
log_loss_logReg = log_loss(y_test, y_test_pred)
print('Log Loss',log_loss(y_test, y_test_pred))
print('\n')
print('AUC',roc_auc_score(y_test, y_test_pred))
print('\n')
sensitivity = TP / float(FN + TP)
print('Sensitivity', sensitivity)
print('recall_score', recall_score(y_test, y_test_pred))
print('\n')
specificity = TN / (TN + FP)
print('Specificity', specificity)
print('\n')
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('classification_error', classification_error)
print('1 - accuracy_score', 1 - accuracy_score(y_test, y_test_pred))

# R/SAS Style Table Output
import statsmodels.api as sm
#train_cols = full_train.columns[1:]
#logit = sm.Logit(y , full_train[train_cols] )
logit = sm.Logit(y , X['Tot_Secs_Remaining'])
#logit = sm.Logit(y, X['shot_distance'])
#logit = sm.Logit(y, X[['shot_distance','playoffs']])
#logit = sm.Logit(y , (X['Tot_Secs_Remaining'] < 6))
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
# fit the model
result = logit.fit()
result.summary()

# odds ratios only
print (np.exp(result.params))

# Final Predictions on the Missing Values - Logistic Regression
# Predictions on the unknown
predictions_final = logismodelfull.predict(y_to_predict)
Predictions_final.shape
predictions_final_df = pd.DataFrame(columns = ['shot_made_flag' ] ) # ,'shot_id'
predictions_final_df.shape
predictions_final_df['shot_made_flag']= predictions_final
shot_id = pd.DataFrame(kobe[kobe['shot_made_flag'].isnull()]['shot_id'])
shot_id= shot_id.reset_index(drop=True)
predictions_final_df = pd.concat([predictions_final_df, shot_id], axis =1)
predictions_final_df.head()
predictions_final_df.tail()
predictions_final_df.count()
predictions_final_df['shot_id'].count()
#predictions_final_df.drop( ['shot_id','shot_made_flag'],axis =1, inplace=True)
Predictions_final_df.shape
predictions_final_df.to_csv('pred3.csv', ',', index=False)
```

Linear Discriminant Analysis (LDA) / Quadratic Discriminant Analysis (QDA)

Python

```
# Parameter Tuning for LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# create cross validation iterator
cv = StratifiedShuffleSplit( n_splits=10, test_size=0.2)

# tune the significant parameter C
shrinkage_param_range = [0, 0.25, 0.5, 0.75, 1]

for i in shrinkage_param_range:
    clf_lda = LinearDiscriminantAnalysis(solver = 'lsqr', shrinkage = i,n_components = None)
    scores_lda = cross_val_score(clf_lda, X, y=y, cv=cv, scoring = 'neg_log_loss', n_jobs=-1)
# The mean accuracy score and the 95% confidence interval of the score estimate
    print(i,":CI of LinearDiscriminantAnalysis LogLoss is %0.3f +/- %0.3f" % (scores_lda.mean(),
scores_lda.std()*2))
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
# Hyper Parameter Tuning Using Grid Search - LDA
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score

seed = 7
processors=1
num_folds=3
num_instances=len(X)
scoring='neg_log_loss'

kfold = KFold(n_splits=num_folds, shuffle= True, random_state=seed)

lda_grid = GridSearchCV ( estimator = LinearDiscriminantAnalysis(),
                          param_grid = {
                         'solver': ['lsqr'],
                         'shrinkage': [0, 0.25, 0.5, 0.75, 1],
                         'n_components': [None, 2, 5, 10]
                                    },
                          cv = kfold,
                          scoring = scoring,
                          n_jobs = processors
                        )

lda_grid.fit(X, y)

print(lda_grid.best_score_)
print(lda_grid.best_params_)
Lda_grid.estimator

# Training LDA & QDA model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
ldamodel = LinearDiscriminantAnalysis(solver = 'lsqr', shrinkage= 0)
qdamodel = QuadraticDiscriminantAnalysis()

#scale & PCA train data
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

scalar = StandardScaler()
scalar.fit(X_train)
#X_train_scaled = preprocessing.scale(X_train)
X_train_scaled = scalar.transform(X_train)
estimator = PCA(n_components=5)
```

```
X_train_pca = estimator.fit_transform(X_train_scaled)

#pca
ldamodel.fit(X_train_pca, y_train)

ldamodel.fit(X_train, y_train)
qdamodel.fit(X_train, y_train)

# LDA Predictions
predictions_lda = ldamodel.predict(X_test)

# Grid Search Predictions --- Log Loss 10.8992619888 (reduced)
predictions_lda = lda_grid.predict(X_test)

# scale & PCA test data
#X_test_scaled=preprocessing.scale(X_test)
scalar.fit(X_test)
X_test_scaled = scalar.transform(X_test)
X_test_pca=estimator.fit_transform(X_test_scaled)

# pca predict the y_test
y_test_pred=ldamodel.predict(X_test_pca)

from sklearn.metrics import classification_report, confusion_matrix,log_loss,
roc_auc_score,accuracy_score,recall_score
print(classification_report(y_test, predictions_lda))
# pca
print(classification_report(y_test , y_test_pred))
print(confusion_matrix(y_test, predictions_lda))
#pca
print(confusion_matrix(y_test,y_test_pred))

# save confusion matrix and slice into four pieces
confusion = confusion_matrix(y_test, predictions_lda)
print(confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

#pca
# save confusion matrix and slice into four pieces
confusion = confusion_matrix(y_test,y_test_pred)
print(confusion)
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

# Get statistics
log_loss_lda = log_loss(y_test, predictions_lda)
print('Log Loss',log_loss(y_test, predictions_lda))
print('\n')
print('AUC',roc_auc_score(y_test, predictions_lda))
print('\n')
sensitivity = TP / float(FN + TP)
print('Sensitivity', sensitivity)
print('recall_score', recall_score(y_test, predictions_lda))
print('\n')
specificity = TN / (TN + FP)
print('Specificity', specificity)
print('\n')
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('classification_error', classification_error)
print('1 - accuracy_score', 1 - accuracy_score(y_test, predictions_lda))

#pca Get Statistics
log_loss_logReg = log_loss(y_test, y_test_pred)
print('Log Loss',log_loss(y_test, y_test_pred))
print('\n')
print('AUC',roc_auc_score(y_test, y_test_pred))
print('\n')
sensitivity = TP / float(FN + TP)
print('Sensitivity', sensitivity)
print('recall_score', recall_score(y_test, y_test_pred))
print('\n')
specificity = TN / (TN + FP)
print('Specificity', specificity)
print('\n')
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('classification_error', classification_error)
print('1 - accuracy_score', 1 - accuracy_score(y_test, y_test_pred))

# Predict
predictions_lda_final = ldamodel.predict(y_to_predict)
predictions_lda_final.shape
predictions_lda_final_df = pd.DataFrame(columns = ['shot_made_flag' ] )
predictions_lda_final_df.shape
```

```
predictions_lda_final_df['shot_made_flag']= predictions_lda_final
shot_id_lda = pd.DataFrame(kobe[kobe['shot_made_flag'].isnull()]['shot_id'])
shot_id_lda = shot_id_lda.reset_index(drop=True)
#shot_id_lda.head()
predictions_lda_final_df = pd.concat([predictions_lda_final_df, shot_id_lda], axis =1)
predictions_lda_final_df.head()
#predictions_lda_final_df.tail()
predictions_lda_final_df.shape
#predictions_lda_final_df.drop( ['shot_id','shot_made_flag'],axis =1, inplace=True)
predictions_lda_final_df.shape
predictions_lda_final_df.to_csv('pred4.csv', ',', index=False)
```

K-Means Clustering Analysis

Python

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=2)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
km.fit(X)
X.tail()

# km.cluster_centers_
km.labels_
from sklearn.metrics import classification_report, confusion_matrix,log_loss,
roc_auc_score,accuracy_score,recall_score
print(classification_report(y, km.labels_))
print(confusion_matrix(y, km.labels_))

# save confusion matrix and slice into four pieces
confusion = confusion_matrix(y, km.labels_)
print(confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

# Get statistics
log_loss_km = log_loss(y, km.labels_)
print('Log Loss',log_loss(y, km.labels_))
print('\n')
print('AUC',roc_auc_score(y, km.labels_))
print('\n')
sensitivity = TP / float(FN + TP)
print('Sensitivity', sensitivity)
```

```
print('recall_score', recall_score(y, km.labels_))
print('\n')
specificity = TN / (TN + FP)
print('Specificity', specificity)
print('\n')
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('classification_error', classification_error)
print('1 - accuracy_score', 1 - accuracy_score(y, km.labels_))

# Predict
km.fit(y_to_predict)
km.labels_
predictions_km_final = km.labels_
predictions_km_final_df = pd.DataFrame(columns = ['shot_made_flag' ] )
predictions_km_final_df.shape
predictions_km_final_df['shot_made_flag' ] = []
predictions_km_final_df['shot_made_flag' ] =  predictions_km_final
shot_id_km = pd.DataFrame(kobe[kobe['shot_made_flag'].isnull()]['shot_id'])
shot_id_km = shot_id_km.reset_index(drop=True)
predictions_km_final_df = pd.concat([predictions_km_final_df, shot_id_km], axis =1)
predictions_km_final_df.head()
predictions_km_final_df.shape
#predictions_km_final_df.drop(predictions_km_final_df.index,axis=1, inplace=True)
# predictions_km_final_df.drop(['shot_id','shot_made_flag'],axis=1, inplace=True)
predictions_km_final_df.shape
#predictions_km_final_df.head()
predictions_km_final_df.to_csv('pred5.csv', ',', index=False)
```

Comparing All Models

Python

```
from sklearn.model_selection import KFold, cross_val_score

seed = 7
processors=1
num_folds=3
num_instances=len(X)
scoring='neg_log_loss'

kfold = KFold(n_splits=num_folds, shuffle= True, random_state=seed)

# Prepare some basic models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
# Evaluate each model in turn
results = []
names = []

for name, model in models:
    cv_results = cross_val_score(model, X, y, cv=kfold, scoring=scoring, n_jobs=processors)
    results.append(cv_results)
    names.append(name)
    print("{0}: ({1:.3f}) +/- ({2:.3f})".format(name, cv_results.mean(), cv_results.std()))

# Assess Feature Importance

#full_train_fi = full_train[['Dunk', 'Running Jump Shot','Jump Bank Shot','Pullup Jump shot','Layup Shot',
#                        'Layup','Turnaround Fadeaway shot','Step Back Jump shot','Fadeaway Jump Shot',
#                        'Turnaround Jump Shot','Floating Jump shot','Fadeaway Bank shot','Dunk Shot',
#                        'Turnaround Bank shot','Driving Layup Shot','Running Bank shot','shot_made_flag']]

#full_train_fi.head()

# Logistic Regression
feature_importance = abs(logismodelfull.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

featfig = plt.figure(figsize=(25,45))
featax = featfig.add_subplot(1, 1, 1)
featax.barh(pos, feature_importance[sorted_idx], align='center')
featax.set_yticks(pos)
featax.set_yticklabels(np.array(X.columns)[sorted_idx], fontsize=14)
featax.set_xlabel('Relative Feature Importance')

plt.tight_layout()
plt.show()
#np.array(X.columns)[sorted_idx]
#feature_importance[sorted_idx]

# LDA
feature_importance = abs(ldamodel.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

featfig = plt.figure(figsize=(25,45))
```

```
featax = featfig.add_subplot(1, 1, 1)
featax.barh(pos, feature_importance[sorted_idx], align='center')
featax.set_yticks(pos)
featax.set_yticklabels(np.array(X.columns)[sorted_idx], fontsize=14)
featax.set_xlabel('Relative Feature Importance')

plt.tight_layout()
plt.show()
```

Principal Components Analysis (PCA)

Python

```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
scalar.fit(full_train)
scaled_data = scalar.transform(full_train)#X_train
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
x_pca.shape
plt.figure(figsize=(15,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=full_train['shot_made_flag'],cmap='plasma')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
x_pca_df = pd.DataFrame(x_pca)
sns.pairplot(x_pca_df)
x_pca_df.head()

# Heat Map to get the influence of principal components with original variables
df_comp = pd.DataFrame(pca.components_,columns=full_train.columns)
df_comp.head()
plt.figure(figsize=(50,6))
sns.heatmap(df_comp,cmap='viridis')
```

Time Series based on Season

```
data WORK.KOBEDATA    ;
%let _EFIERR_ = 0; /* set the ERROR detection macro variable */
infile REFFILE delimiter = ',' MISSOVER DSD  firstobs=2 ;
informat action_type $17. ;
informat combined_shot_type $9. ;
informat game_event_id best32. ;
informat game_id best32. ;
informat lat best32. ;
informat loc_x best32. ;
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
informat loc_y best32. ;
informat lon best32. ;
informat minutes_remaining best32. ;
informat period best32. ;
informat playoffs best32. ;
informat season $7. ;
informat seconds_remaining best32. ;
informat shot_distance best32. ;
informat shot_made_flag $2. ;
informat shot_type $14. ;
informat shot_zone_area $21. ;
informat shot_zone_basic $21. ;
informat shot_zone_range $15. ;
informat team_id best32. ;
informat team_name $18. ;
informat game_date mmddyy10. ;
informat matchup $11. ;
informat opponent $3. ;
informat shot_id best32. ;
informat attendance best32. ;
informat arena_temp best32. ;
informat avgnoisedb best32. ;
format action_type $17. ;
format combined_shot_type $9. ;
format game_event_id best12. ;
format game_id best12. ;
format lat best12. ;
format loc_x best12. ;
format loc_y best12. ;
format lon best12. ;
format minutes_remaining best12. ;
format period best12. ;
format playoffs best12. ;
format season $7. ;
format seconds_remaining best12. ;
format shot_distance best12. ;
format shot_made_flag $2. ;
format shot_type $14. ;
format shot_zone_area $21. ;
format shot_zone_basic $21. ;
format shot_zone_range $15. ;
format team_id best12. ;
format team_name $18. ;
format game_date mmddyy10. ;
format matchup $11. ;
```

```
format opponent $3. ;
format shot_id best12. ;
format attendance best12. ;
format arena_temp best12. ;
format avgnoisedb best12. ;
input
action_type  $
 combined_shot_type  $
game_event_id
game_id
lat
loc_x
loc_y
lon
minutes_remaining
period
playoffs
season $
seconds_remaining
shot_distance
shot_made_flag  $
shot_type  $
shot_zone_area  $
shot_zone_basic  $
shot_zone_range  $
team_id
team_name  $
game_date
matchup  $
opponent  $
shot_id
attendance
 arena_temp
avgnoisedb
;
if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
run;

data Kobe;
set KOBEDATA;
min = minutes_remaining*60;
total_sec_remaining = min + seconds_remaining;
r = sqrt((loc_x)**2+(loc_y)**2);
if loc_x = '0' then theta=constant("pi")/2;
else theta = atan(loc_y/loc_x);
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
if substr(matchup,5,3) = 'vs.' then home_flag= 1;
else home_flag=0;
run;

/*split NA values and non-NA values of shot_made_flag*/
data kobetrain;
set Kobe;
where shot_made_flag ^= 'NA';
run;

data kobetrain2;
set kobetrain;
new = input(shot_made_flag, 2.);
  drop shot_made_flag;
  rename new=shot_made_flag;
  run;

data kobe_check;
set kobetrain2;
if season = '1996-97' then season_number = 1;
if season = '1997-98' then season_number = 2;
if season = '1998-99' then season_number = 3;
if season = '1999-00' then season_number = 4;
if season = '2000-01' then season_number = 5;
if season = '2001-02' then season_number = 6;
if season = '2002-03' then season_number = 7;
if season = '2003-04' then season_number = 8;
if season = '2004-05' then season_number = 9;
if season = '2005-06' then season_number = 10;
if season = '2006-07' then season_number = 11;
if season = '2007-08' then season_number = 12;
if season = '2008-09' then season_number = 13;
if season = '2009-10' then season_number = 14;
if season = '2010-11' then season_number = 15;
if season = '2011-12' then season_number = 16;
if season = '2012-13' then season_number = 17;
if season = '2013-14' then season_number = 18;
if season = '2014-15' then season_number = 19;
if season = '2015-16' then season_number = 20;
run;

proc print data = kobe_check;
run;

proc sql;
```

Noelle Brown
Alexandra Norman
Thejas Prasad

```
create table kobe_time as
select season_number, (shots_made/shots_per_game) as fgp, shots_per_game, shots_made from
(select season_number, count(shot_id) as shots_per_game, sum(shot_made_flag) as shots_made from
kobe_check
group by season_number);
quit;

/* Plot the Data ... */
proc sgplot data = kobe_time;
scatter x = season_number y= fgp;
series x = season_number y= fgp;
run;

/* proc glm */
proc glm data = kobe_time plots = all;
model fgp = season_number;
output out = resids r = residsOLS;
run;

proc print data=resids;
run;

/* proc autoreg */
proc autoreg data = kobe_time plots(unpack);
model fgp = season_number / dwprob;
run;

/* proc autoreg */
proc autoreg data = kobe_time plots(unpack);
model fgp = season_number / nlag=1 dwprob;
output out=Forecast p = yhat pm = ytrend lcl=lower ucl= upper;
run;

/* to get the predicted values*/
proc print data=Forecast;
run;

/* plot for full data and predictions*/
title 'Predictions for Autocorrelation Model';
proc sgplot data= Forecast;
band x = season_number upper = upper lower = lower ;
scatter x = season_number y = fgp / markerattrs=(color = blue);
series x = season_number y = yhat / lineattrs=(color = blue);
series x = season_number y = ytrend / lineattrs=(color = black);
run;
```