In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

一、 輸入資料

In [2]:

```
csv_file = "data.csv"
df1 = pd.read_csv(csv_file)
df1.head()
```

Out[2]:

	action_type	combined_shot_type	game_event_id	game_id	lat	loc_x	loc_y	
0	Jump Shot	Jump Shot	10	20000012	33.9723	167	72	-118
1	Jump Shot	Jump Shot	12	20000012	34.0443	-157	0	-118
2	Jump Shot	Jump Shot	35	20000012	33.9093	-101	135	-118
3	Jump Shot	Jump Shot	43	20000012	33.8693	138	175	-118
4	Driving Dunk Shot	Dunk	155	20000012	34.0443	0	0	-118

5 rows × 25 columns

問題1：預測Kobe是否進球

目的：在資料中，**shot_made_flag**是指有無進球，而我們要利用原本的資料來預測是否會進球。

二、 資料整理，挑選對"進球"有用的資料作使用

1. 刪除空白的值

In [3]:

```
nona = df1[pd.notnull(df1['shot_made_flag'])]
```

2. 查看資料 (action_type、combined_shot_type、shot_type)

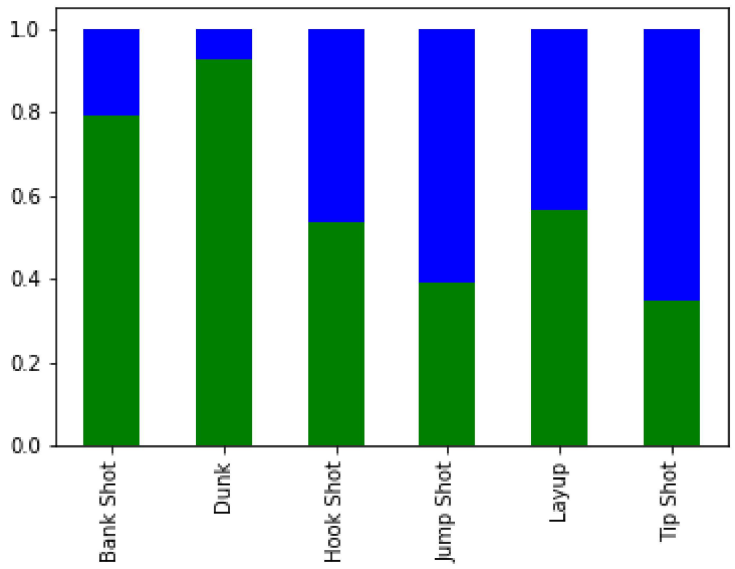
In [4]:

```
field_goal = nona['combined_shot_type'][nona['shot_made_flag']==1]
non_field_goal = nona['combined_shot_type'][nona['shot_made_flag']==0]
count = field_goal.value_counts()
print('出手率：')
print(count/count.sum())
count_non = non_field_goal.value_counts()
p = count/(count+count_non)
print('命中率：')
print(p)
p.plot.bar(color='g')
(1-p).plot.bar(color='b',bottom=p)
```

```
出手率：
Jump Shot    0.672307
Layup        0.223375
Dunk         0.085478
Bank Shot    0.008286
Hook Shot    0.005931
Tip Shot     0.004623
Name: combined_shot_type, dtype: float64
命中率：
Bank Shot    0.791667
Dunk         0.928030
Hook Shot    0.535433
Jump Shot    0.391071
Layup        0.565093
Tip Shot     0.348684
Name: combined_shot_type, dtype: float64
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fffa740198>
```

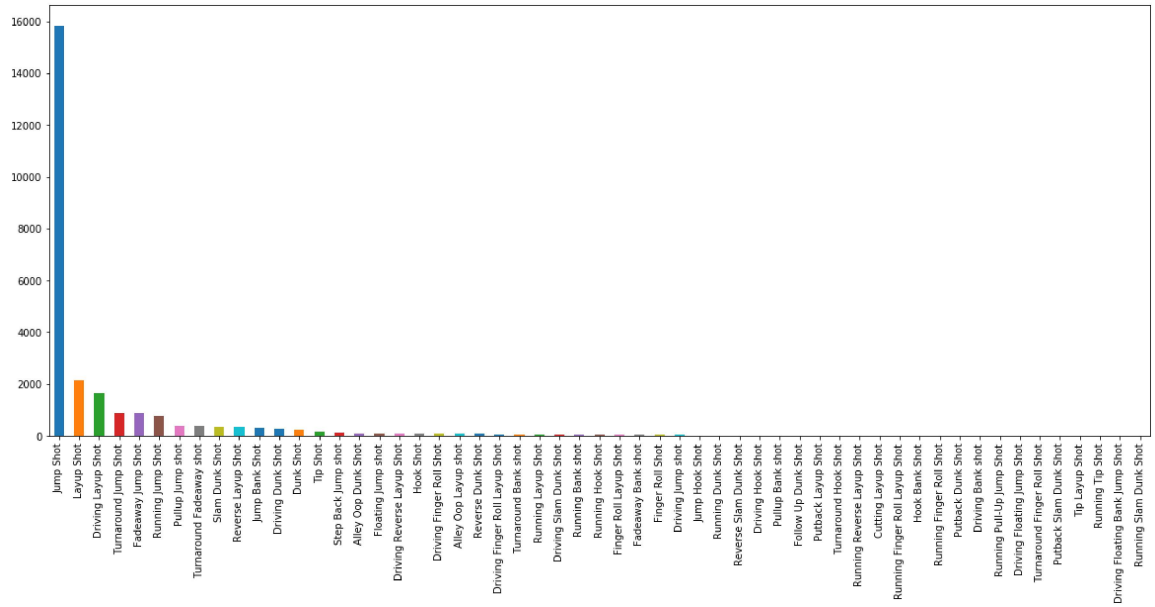


In [5]:

```
#投籃的方式
count = nona.action_type.value_counts()
count.plot.bar(figsize=(20, 8))
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fffa7fc748>
```



Kobe 的出手以 **Jump Shot** 為主，佔了約 **67.2%**，命中率為 **39.1%**，為所有種類中命中率算低的。

出手前三名是**Jump Shot**、**Layup**、**Dunk**，距離跟命中率成反比，投籃命中率隨著籃框的距離減少而增加，可想而知，距離越遠的越難進球。

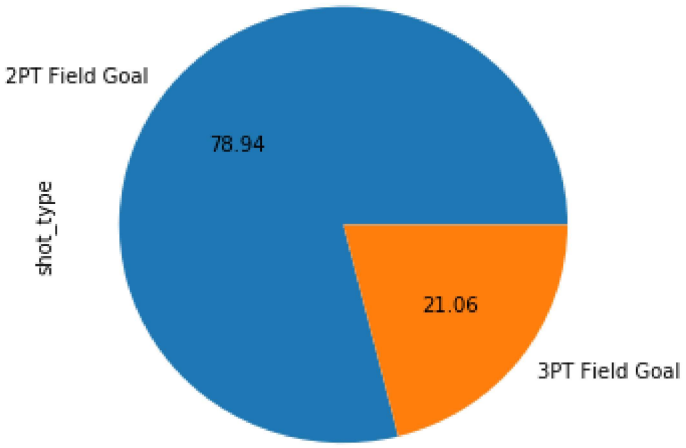
In [6]:

```
#兩分球或三分球
count = nona.shot_type.value_counts()
print(count)
count.plot.pie(autopct='%.2f',figsize=(5, 5))
```

```
2PT Field Goal    20285
3PT Field Goal     5412
Name: shot_type, dtype: int64
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fffaca2128>
```



用什麼方式投籃會影響進球的狀態，所以要考慮進去。

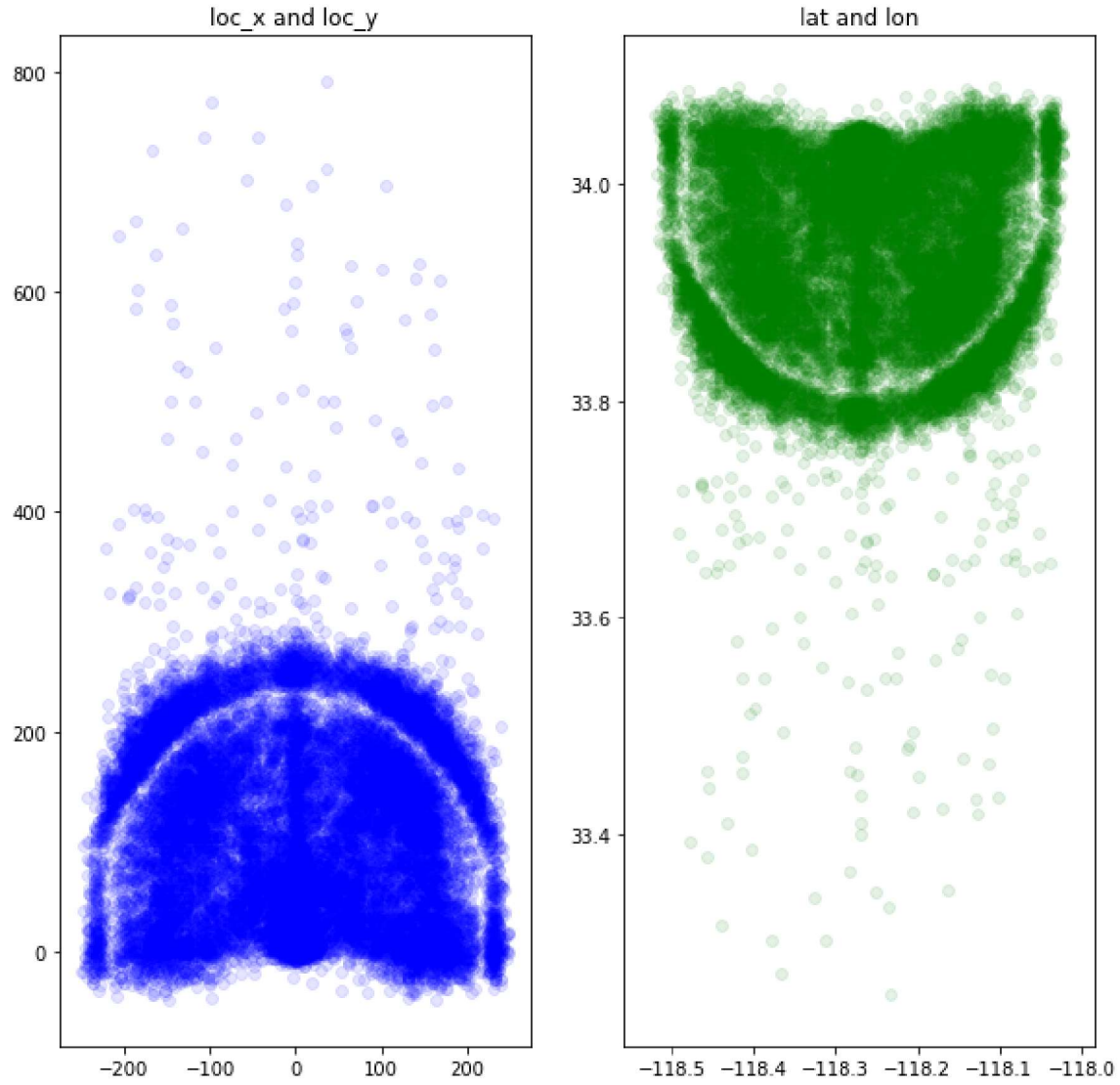
3. 查看資料 (loc_x 、loc_y、lat、lon)

In [7]:

```
alpha =0.1
plt.figure(figsize=(10,10))

#loc_x and loc_y
plt.subplot(121)                    #畫在一行兩列的第一幅圖裡
plt.scatter(nona.loc_x,nona.loc_y,color='blue',alpha=alpha)
plt.title('loc_x and loc_y')

#lat and lon
plt.subplot(122)
plt.scatter(nona.lon,nona.lat,color='green',alpha=alpha)
plt.title('lat and lon')
```

Out[7]:

```
Text(0.5,1,'lat and lon')
```



畫圖出來像籃球場，可發現是**1**、**2**節與**3**、**4**節投籃的位置，兩者本質上是一樣的，所以選擇使用**loc_x** 、**loc_y**。

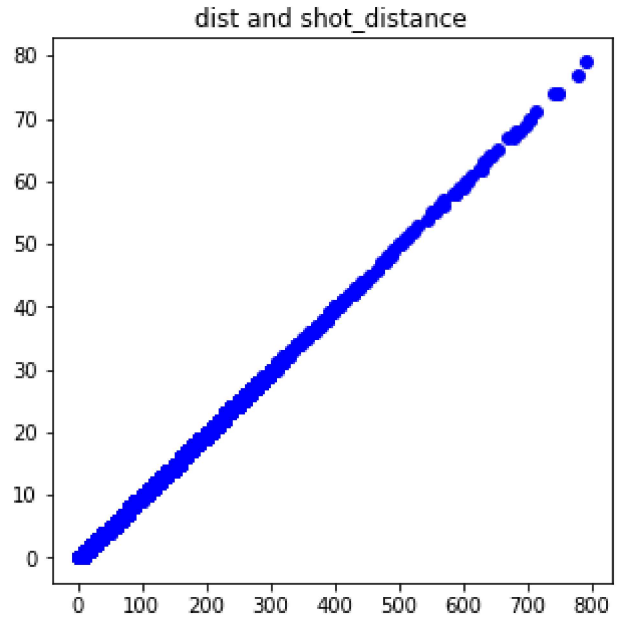
而球場是半圓形的，把座標位置轉換為極坐標來表示。

4. 比較 **dist** 和 **shot_distance**：**dist**為距離籃框的距離，**angle**為投籃的角度，新增**dist**、**angle**變數。

In [8]:

```
nona['dist']=np.sqrt(nona['loc_x']**2+nona['loc_y']**2)     #新增一列dist
loc_x_zero= (nona['loc_x'] == 0)                            #篩掉loc_x=0的值
nona['angle']=np.array([0]*len(nona))
nona['angle'][~loc_x_zero]=np.arctan(nona['loc_y'][~loc_x_zero]/nona['loc_x'][~loc_x_ze
ro])   #  ~ = False
nona['angle'][loc_x_zero]=np.pi / 2
plt.figure(figsize=(5,5))
plt.scatter(nona.dist,nona.shot_distance,color='blue')
plt.title('dist and shot_distance')
plt.show()
```

```
C:\Users\rital\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
C:\Users\rital\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing import
s until
C:\Users\rital\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
C:\Users\rital\Anaconda3\lib\site-packages\pandas\core\generic.py:7626: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
C:\Users\rital\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y:2961: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\Users\rital\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """
```



有加角度的 **dist** 和本來資料裡的 **shot_distance** 為正相關，所以可以擇一使用，我選擇使用 **dist** ，因為多了角度的數據，可讓分析更精準。

5. 查看資料 **(matchup、opponent)**

In [9]:

```
pd.DataFrame({'matchup':nona.matchup, 'opponent':nona.opponent})
```

Out[9]:

	matchup	opponent
1	LAL @ POR	POR
2	LAL @ POR	POR
3	LAL @ POR	POR
4	LAL @ POR	POR
5	LAL @ POR	POR
6	LAL @ POR	POR
8	LAL @ POR	POR
9	LAL @ POR	POR
10	LAL @ POR	POR
11	LAL vs. UTA	UTA
12	LAL vs. UTA	UTA
13	LAL vs. UTA	UTA
14	LAL vs. UTA	UTA
15	LAL vs. UTA	UTA
17	LAL vs. UTA	UTA
18	LAL vs. UTA	UTA
20	LAL vs. UTA	UTA
21	LAL vs. UTA	UTA
22	LAL vs. UTA	UTA
23	LAL vs. UTA	UTA
24	LAL vs. UTA	UTA
25	LAL vs. UTA	UTA
26	LAL vs. UTA	UTA
27	LAL vs. UTA	UTA
28	LAL vs. UTA	UTA
29	LAL vs. UTA	UTA
30	LAL vs. UTA	UTA
31	LAL vs. UTA	UTA
38	LAL @ VAN	VAN
39	LAL @ VAN	VAN
...	...	...
30661	LAL @ IND	IND
30662	LAL @ IND	IND
30663	LAL @ IND	IND
30665	LAL @ IND	IND
30666	LAL @ IND	IND
30667	LAL @ IND	IND

	matchup	opponent
30669	LAL @ IND	IND
30670	LAL vs. IND	IND
30671	LAL vs. IND	IND
30672	LAL vs. IND	IND
30673	LAL vs. IND	IND
30674	LAL vs. IND	IND
30675	LAL vs. IND	IND
30676	LAL vs. IND	IND
30677	LAL vs. IND	IND
30678	LAL vs. IND	IND
30679	LAL vs. IND	IND
30681	LAL vs. IND	IND
30683	LAL vs. IND	IND
30684	LAL vs. IND	IND
30685	LAL vs. IND	IND
30687	LAL vs. IND	IND
30688	LAL vs. IND	IND
30689	LAL vs. IND	IND
30690	LAL vs. IND	IND
30691	LAL vs. IND	IND
30692	LAL vs. IND	IND
30694	LAL vs. IND	IND
30695	LAL vs. IND	IND
30696	LAL vs. IND	IND

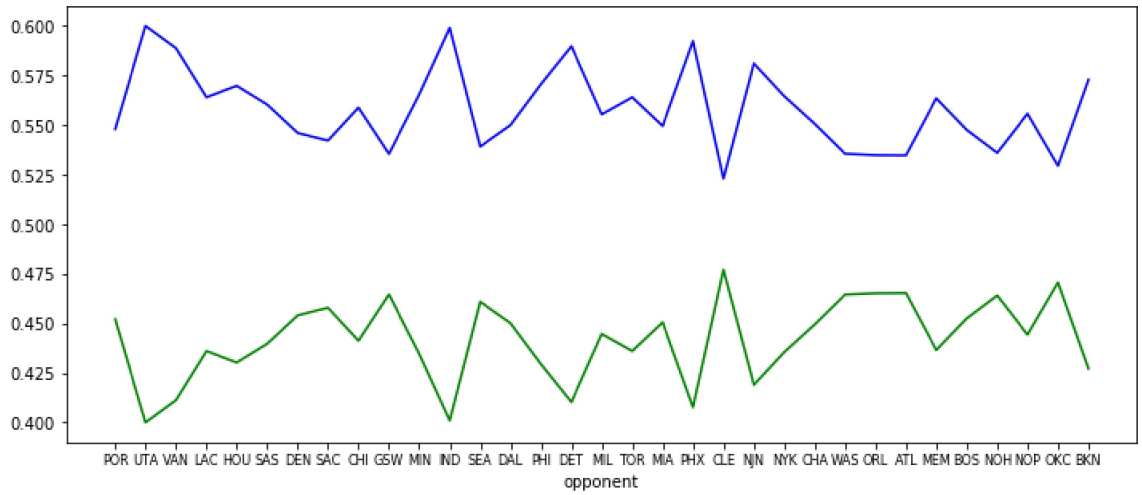
25697 rows × 2 columns

In [10]:

```
field_goal = nona['opponent'][nona['shot_made_flag']==1]
non_field_goal = nona['opponent'][nona['shot_made_flag']==0]
count = field_goal.value_counts()
count_non = non_field_goal.value_counts()
p = count/(count+count_non)

op = plt.figure(figsize=(12,5))
op_plt = op.add_subplot(1, 1, 1)
op_plt.plot(p,color='g')
op_plt.plot((1-p),color='b')
op_plt.set_xticklabels(nona['opponent'].unique(), fontsize=8)
op_plt.set_xlabel('opponent')
```

Out[10]:

```
Text(0.5,0,'opponent')
```



兩者資料類型相似，選擇使用**opponent**，碰到不一樣的對手也會影響進球的狀態，所以要考慮進去。

6. 查看資料 (minutes_remaining、seconds_remaining)

In [11]:

```
nona['remaining_time']=nona['minutes_remaining']*60+nona['seconds_remaining']
```

```
C:\Users\rital\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

把剩餘時間做合併 **(**單位：秒**)**。

7. 查看資料 (season)

In [12]:

```
print(nona['season'].unique())
```

```
['2000-01' '2001-02' '2002-03' '2003-04' '2004-05' '2005-06' '2006-07'
 '2007-08' '2008-09' '2009-10' '2010-11' '2011-12' '2012-13' '2013-14'
 '2014-15' '2015-16' '1996-97' '1997-98' '1998-99' '1999-00']
```

In [13]:

```
nona['season'] = nona['season'].apply(lambda x: int(x.split('-')[0]) )
nona['season'].unique()
```

```
C:\Users\rital\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[13]:

```
array([2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
       2011, 2012, 2013, 2014, 2015, 1996, 1997, 1998, 1999], dtype=int64)
```

將賽季轉為數值型的資料，只用賽季的年份來代表，而每個時期的狀態不一樣，所以要考慮進去。

利用維基百科上的資訊 ，可以將 **Kobe** 的生涯分成 **6** 群：

(1) 1996-1999 新人期，從高中球隊到進入**NBA**，成為史上最年輕的先發球員

(2) 1999-2003 高峰期，湖人隊連奪三次**NBA**總冠軍，與其他明星球員被譽為「四大天王」

(3) 2003-2004 低潮期，遇上性侵事件和球員內鬥，讓此球季為柯比生涯的最低點

(4) 2004-2007 轉折期，球隊好轉，柯比在此季創下湖人隊的單季最高個人得分的球隊紀錄

(5) 2007-2013 巔峰期，獲得多項殊榮：**NBA**最有價值球員獎項、多次總決賽**MVP**.....等等

(6) 2013-2016 大傷至退休時期

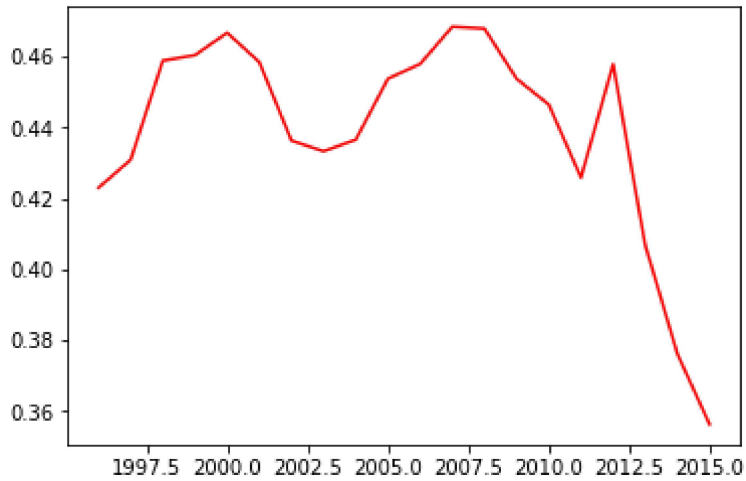
In [14]:

```
field_goal = nona['season'][nona['shot_made_flag']==1]
non_field_goal = nona['season'][nona['shot_made_flag']==0]
count = field_goal.value_counts()
count_non = non_field_goal.value_counts()
p = count/(count+count_non)
print('命中率：')
print(p)
p.plot(color='r')
```

```
命中率：
1996    0.422977
1997    0.430864
1998    0.458824
1999    0.460366
2000    0.466667
2001    0.458431
2002    0.436285
2003    0.433260
2004    0.436557
2005    0.453742
2006    0.457885
2007    0.468389
2008    0.467855
2009    0.453725
2010    0.446417
2011    0.425847
2012    0.457831
2013    0.406780
2014    0.376054
2015    0.356223
Name: season, dtype: float64
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fffa9bc550>
```



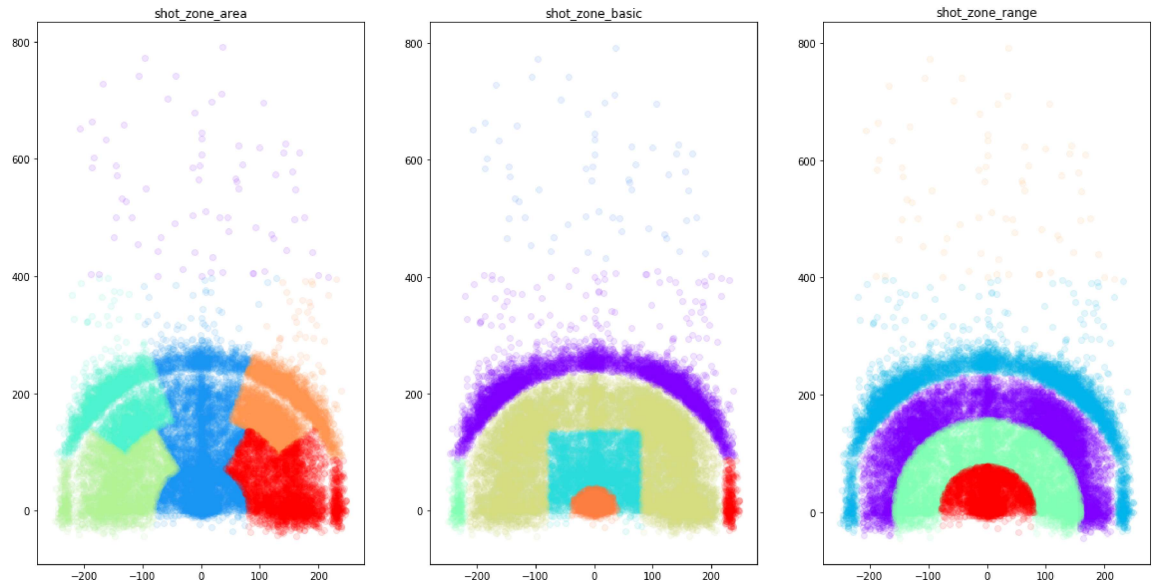
8. 查看資料 (shot_zone_area、shot_zone_basic、shot_zone_range)

In [15]:

```
import matplotlib.cm as cm
plt.figure(figsize=(20,10))

def scatter_plot_by_category(feat):
    alpha = 0.1
    gs = nona.groupby(feat)                          #分組
    colors  = cm.rainbow(np.linspace(0, 1, len(gs)))
    for g, c in zip(gs, colors ):                        #zip兩兩對印
        plt.scatter(g[1].loc_x, g[1].loc_y, color=c, alpha=alpha)

plt.subplot(131)
scatter_plot_by_category('shot_zone_area')
plt.title('shot_zone_area')

plt.subplot(132)
scatter_plot_by_category('shot_zone_basic')
plt.title('shot_zone_basic')

plt.subplot(133)
scatter_plot_by_category('shot_zone_range')
plt.title('shot_zone_range')
```

Out[15]:

```
Text(0.5,1,'shot_zone_range')
```



由圖可看出，這些代表球場上的區域，而剛剛的**dist**和**angle**可以代替使用，所以這些不需要採用。

三、整理資料，把有用的資料做彙整。

1. 刪除名稱與其他無用數據

In [16]:

```
drops = ['shot_id', 'team_id', 'team_name', 'shot_zone_area', 'shot_zone_range', 'shot_
zone_basic', \
         'matchup', 'lon', 'lat', 'seconds_remaining', 'minutes_remaining', \
         'shot_distance', 'loc_x', 'loc_y', 'game_event_id', 'game_id', 'game_date','pl
ayoffs']
for drop in drops:
    nona = nona.drop(drop, axis=1)
nona.head()
```

Out[16]:

	action_type	combined_shot_type	period	season	shot_made_flag	shot_type	opponent
1	Jump Shot	Jump Shot	1	2000	0.0	2PT Field Goal	POR
2	Jump Shot	Jump Shot	1	2000	1.0	2PT Field Goal	POR
3	Jump Shot	Jump Shot	1	2000	0.0	2PT Field Goal	POR
4	Driving Dunk Shot	Dunk	2	2000	1.0	2PT Field Goal	POR
5	Jump Shot	Jump Shot	3	2000	0.0	2PT Field Goal	POR

2. 將類別資料轉為數值資料

In [17]:

```
categorical_vars = ['action_type', 'combined_shot_type', 'shot_type', 'opponent', 'peri
od', 'season']
for var in categorical_vars:
    nona = pd.concat([nona, pd.get_dummies(nona[var], prefix=var)], axis=1)
    nona = nona.drop(var, axis=1)
nona.head(5)
```

Out[17]:

	shot_made_flag	dist	angle	remaining_time	action_type_Alley Oop Dunk Shot	action_type_A Oop Layup
1	0.0	157.000000	-0.000000	622	0	
2	1.0	168.600119	-0.928481	465	0	
3	0.0	222.865430	0.903063	412	0	
4	1.0	0.000000	1.570796	379	0	
5	0.0	145.416643	0.075717	572	0	

5 rows × 127 columns

3. 把測試與訓練數據做區隔

In [18]:

```
from sklearn.model_selection import train_test_split
from sklearn import cross_validation as cv

X = nona.drop('shot_made_flag', axis=1)
Y = nona['shot_made_flag']
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.33, random_state=0)

fold = cv.KFold(len(y_train), n_folds=10, shuffle=False, random_state=0)
grid = {'C': 10.0 ** np.arange(-3, 3)}
```

```
C:\Users\rital\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and funct
ions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

四、用以下幾種方法來預測

1. LogisticRegression

In [19]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.grid_search import GridSearchCV

clf = LogisticRegression(penalty='l2', max_iter=1000)
gs = GridSearchCV(clf,param_grid = grid, scoring='roc_auc', cv=fold)
gs.fit(X_train, y_train)
print("所有C值下之結果：",gs.grid_scores_)
print("最好的C值：",gs.best_params_)
print("LR模型準確度：",gs.best_score_)
```

```
C:\Users\rital\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: Depr
ecationWarning: This module was deprecated in version 0.18 in favor of the
model_selection module into which all the refactored classes and functions
are moved. This module will be removed in 0.20.
  DeprecationWarning)

所有C值下之結果： [mean: 0.66422, std: 0.00853, params: {'C': 0.001}, mean:
0.68891, std: 0.00601, params: {'C': 0.01}, mean: 0.68884, std: 0.00567, p
arams: {'C': 0.1}, mean: 0.68828, std: 0.00576, params: {'C': 1.0}, mean:
0.68780, std: 0.00590, params: {'C': 10.0}, mean: 0.68783, std: 0.00586, p
arams: {'C': 100.0}]
最好的C值： {'C': 0.01}
LR模型準確度： 0.6889131467637993
```

In [20]:

```
print('預測的結果：',gs.predict(X_test))
```

```
預測的結果： [0. 0. 0. ... 0. 0. 1.]
```

2. LinearDiscriminantAnalysis (線性判別分析)

LDA：將數據點通過投影的方式，投影到維度更低的空間中，使投影後的點會按照類別區分，而**LDA**的目標是讓不同類別之間的距離越遠越好，同一類別的距離越近越好。

In [41]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

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

lda_grid.fit(X_train, y_train)

print("最好的參數：",lda_grid.best_params_)
print("LDA模型準確度：",lda_grid.best_score_)
```

最好的參數： {'n_components': None, 'shrinkage': 0, 'solver': 'lsqr'}
LDA模型準確度： 0.6816914498141264

In [39]:

```
print('預測的結果：',lda_grid.predict(X_test))
```

預測的結果： [0. 0. 0. ... 0. 0. 1.]

3. 比較兩種方法的結果

In [42]:

```
print("LR準確度：",gs.score(X_test,y_test))
print("LDA準確度：",lda_grid.score(X_test,y_test))
```

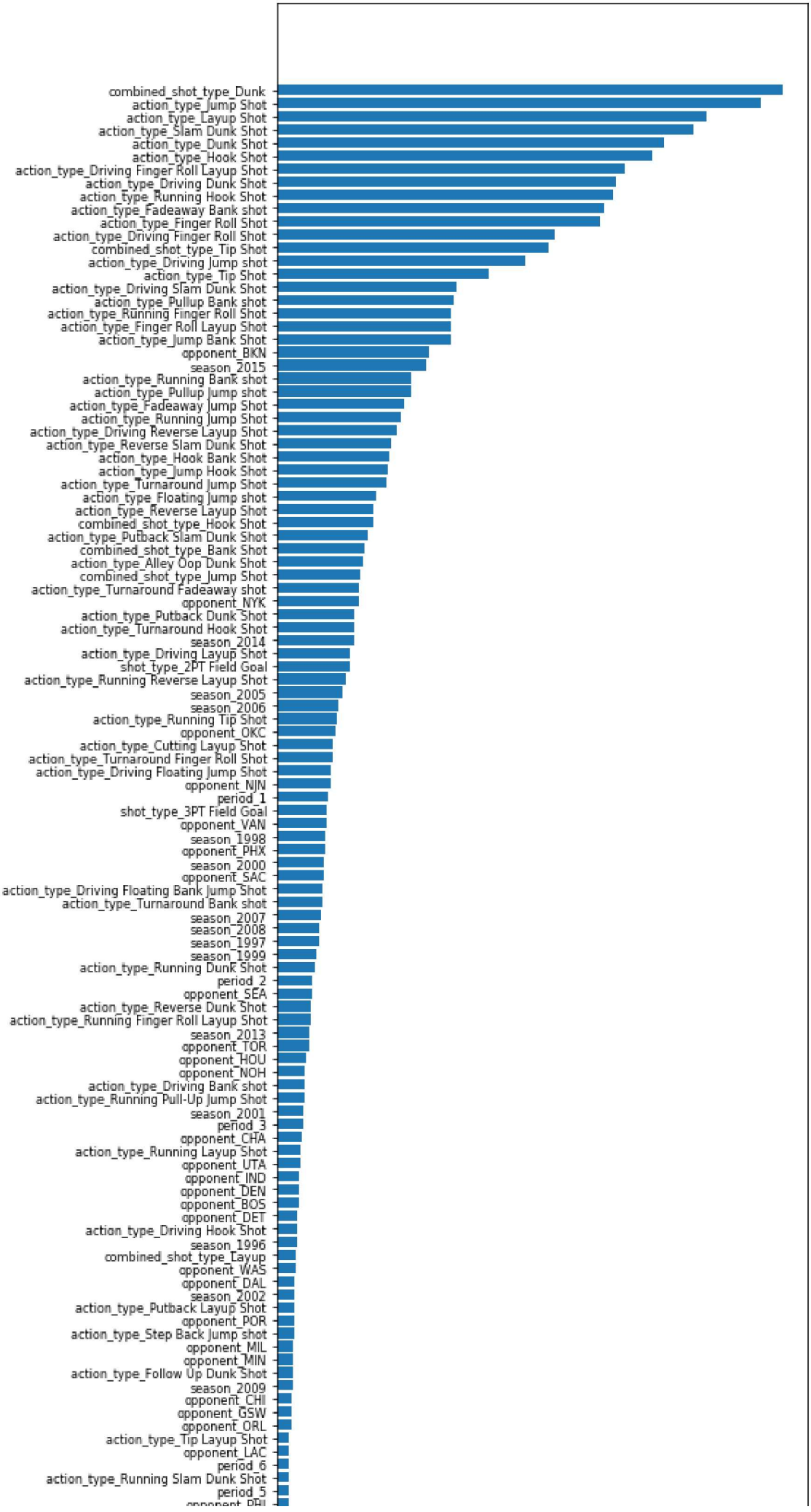
LR準確度： 0.684317096528529
LDA準確度： 0.6806980308925834

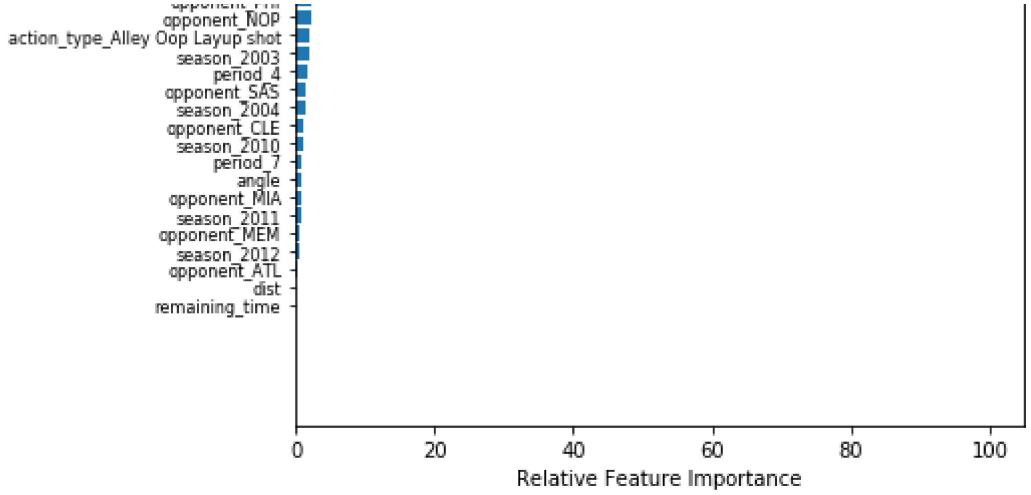
問題2：如何守住Kobe？

先看對於**Kobe**進球的重要特徵排行，以下以迴歸係數來做標準，看哪個特徵影響最大

In [24]:

```
model = LogisticRegression(penalty='l2', max_iter=1000)
clf = model.fit(X, Y)

feature_importance = abs(clf.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)    #由小排到大
pos = np.arange(sorted_idx.shape[0])

featfig = plt.figure(figsize=(8,18))
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=8)
featax.set_xlabel('Relative Feature Importance')

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


combined_shot_type_Dunk
action_type_Jump Shot
action_type_Layup Shot
action_type_Slam Dunk Shot
action_type_Dunk Shot
action_type_Hook Shot
action_type_Driving Finger Roll Layup Shot
action_type_Driving Dunk Shot
action_type_Running Hook Shot
action_type_Fadeaway Bank shot
action_type_Finger Roll Shot
action_type_Driving Finger Roll Shot
combined_shot_type_Tip Shot
action_type_Driving Jump shot
action_type_Tip Shot
action_type_Driving Slam Dunk Shot
action_type_Pullup Bank shot
action_type_Running Finger Roll Shot
action_type_Finger Roll Layup Shot
action_type_Jump Bank Shot
opponent_BKN
season_2015
action_type_Running Bank shot
action_type_Pullup Jump shot
action_type_Fadeaway Jump Shot
action_type_Running Jump Shot
action_type_Driving Reverse Layup Shot
action_type_Reverse Slam Dunk Shot
action_type_Hook Bank Shot
action_type_Jump Hook Shot
action_type_Turnaround Jump Shot
action_type_Floating Jump shot
action_type_Reverse Layup Shot
combined_shot_type_Hook Shot
action_type_Putback Slam Dunk Shot
combined_shot_type_Bank Shot
action_type_Alley Oop Dunk Shot
combined_shot_type_Jump Shot
action_type_Turnaround Fadeaway shot
opponent_NYK
action_type_Putback Dunk Shot
action_type_Turnaround Hook Shot
season_2014
action_type_Driving Layup Shot
shot_type_2PT Field Goal
action_type_Running Reverse Layup Shot
season_2005
season_2006
action_type_Running Tip Shot
opponent_OKC
action_type_Cutting Layup Shot
action_type_Turnaround Finger Roll Shot
action_type_Driving Floating Jump Shot
opponent_NJN
period_1
shot_type_3PT Field Goal
opponent_VAN
season_1998
opponent_PHX
season_2000
opponent_SAC
action_type_Driving Floating Bank Jump Shot
action_type_Turnaround Bank shot
season_2007
season_2008
season_1997
season_1999
action_type_Running Dunk Shot
period_2
opponent_SEA
action_type_Reverse Dunk Shot
action_type_Running Finger Roll Layup Shot
season_2013
opponent_TOR
opponent_HOU
opponent_NOH
action_type_Driving Bank shot
action_type_Running Pull-Up Jump Shot
season_2001
period_3
opponent_CHA
action_type_Running Layup Shot
opponent_UTA
opponent_IND
opponent_DEN
opponent_BOS
opponent_DET
action_type_Driving Hook Shot
season_1996
combined_shot_type_Layup
opponent_WAS
opponent_DAL
season_2002
action_type_Putback Layup Shot
opponent_POR
action_type_Step Back Jump shot
opponent_MIL
opponent_MIN
action_type_Follow Up Dunk Shot
season_2009
opponent_CHI
opponent_GSW
opponent_ORL
action_type_Tip Layup Shot
opponent_LAC
period_6
action_type_Running Slam Dunk Shot
period_5



由上圖可知，**combined_shot_type_Dunk**、**action_type_Jump Shot**、**action_type_Layup Shot**為最主要影響進球的前三名，而前二十名都是以**"**投球的動作**"**為主要的影響。

在前面有提到，**Jump Shot**是**Kobe**出手率高但失誤率也最高的， 所以如果我們要守住**Kobe**的話，可以從他**Jump Shot**下防守，盡量不要讓他做**Dunk**的動作。