Assignment is below.

In [1]:

```
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(font_scale=1.5)
import numpy as np

from pylab import rcParams
rcParams['figure.figsize'] = 20, 10


from sklearn.linear_model import LogisticRegression as Model
```

Read in the Kobe Bryant shooting data [https://www.kaggle.com/c/kobe-bryant-shot-selection (https://www.kaggle.com/c/kobe-bryant-shot-selection)]

In [2]:

```
kobe = pd.read_csv('../data/kobe.csv')
kobe.dropna(inplace=True)
```

In [3]:

```
list(kobe.columns)
```

Out[3]:

```
['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']
```

For now, use just the numerical datatypes. They are below as `num_columns`

In [4]:

```
kobe.shot_zone_area.value_counts()
```

Out[4]:

```
Center(C)                11289
Right Side Center(RC)     3981
Right Side(R)             3859
Left Side Center(LC)      3364
Left Side(L)              3132
Back Court(BC)              72
Name: shot_zone_area, dtype: int64
```

In [5]:

```
kobe.shot_zone_range.value_counts()
```

Out[5]:

```
Less Than 8 ft.    7857
16-24 ft.          6907
8-16 ft.           5580
24+ ft.            5281
Back Court Shot      72
Name: shot_zone_range, dtype: int64
```

In [6]:

```
kobe.shot_zone_basic.value_counts()
```

Out[6]:

```
Mid-Range                10532
Restricted Area           5932
Above the Break 3         4720
In The Paint (Non-RA)     3880
Right Corner 3             333
Left Corner 3              240
Backcourt                   60
Name: shot_zone_basic, dtype: int64
```

In [7]:

```
kobe
```

Out[7]:

	action_type	combined_shot_type	game_event_id	game_id	lat	loc_x
1	Jump Shot	Jump Shot	12	20000012	34.0443	-157
2	Jump Shot	Jump Shot	35	20000012	33.9093	-101
3	Jump Shot	Jump Shot	43	20000012	33.8693	138
4	Driving Dunk Shot	Dunk	155	20000012	34.0443	0
5	Jump Shot	Jump Shot	244	20000012	34.0553	-145
6	Layup Shot	Layup	251	20000012	34.0443	0

8	Jump Shot	Jump Shot	265	20000012	33.9363	-65	
9	Running Jump Shot	Jump Shot	294	20000012	33.9193	-33	
10	Jump Shot	Jump Shot	309	20000012	33.8063	-94	
11	Jump Shot	Jump Shot	4	20000019	33.9173	121	
12	Running Jump Shot	Jump Shot	27	20000019	33.9343	-67	
13	Jump Shot	Jump Shot	66	20000019	34.0403	-94	
14	Jump Shot	Jump Shot	80	20000019	33.9973	-23	
15	Jump Shot	Jump Shot	86	20000019	33.8523	62	
17	Jump Shot	Jump Shot	138	20000019	33.8183	-117	
18	Jump Shot	Jump Shot	244	20000019	33.9473	-132	
20	Jump Shot	Jump Shot	255	20000019	33.9003	3	
21	Jump Shot	Jump Shot	265	20000019	33.9173	134	
22	Running Jump Shot	Jump Shot	274	20000019	33.9343	-16	
23	Running Jump Shot	Jump Shot	299	20000019	33.8943	-109	
24	Running Jump Shot	Jump Shot	307	20000019	33.9813	-46	
25	Layup Shot	Layup	332	20000019	34.0443	0	
26	Jump Shot	Jump Shot	345	20000019	33.8483	-58	
27	Jump Shot	Jump Shot	369	20000019	33.8583	-183	
28	Jump Shot	Jump Shot	400	20000019	33.8713	85	

29	Jump Shot	Jump Shot	429	20000019	33.9573	3	
30	Running Jump Shot	Jump Shot	488	20000019	34.0403	121	
31	Jump Shot	Jump Shot	499	20000019	34.0103	127	
38	Jump Shot	Jump Shot	184	20000047	33.8603	91	
39	Jump Shot	Jump Shot	202	20000047	33.7723	-27	
...	...	...	...	...	...	...	
30661	Slam Dunk Shot	Dunk	245	49900087	34.0443	0	
30662	Jump Shot	Jump Shot	259	49900087	33.9913	-8	
30663	Jump Shot	Jump Shot	270	49900087	34.0193	106	
30665	Layup Shot	Layup	280	49900087	34.0263	-14	
30666	Jump Shot	Jump Shot	295	49900087	33.8733	-81	
30667	Jump Shot	Jump Shot	368	49900087	33.7943	40	
30669	Jump Shot	Jump Shot	425	49900087	33.9913	171	
30670	Running Jump Shot	Jump Shot	15	49900088	34.0283	-74	
30671	Driving Layup Shot	Layup	25	49900088	34.0443	0	
30672	Jump Shot	Jump Shot	29	49900088	33.9893	89	
30673	Jump Shot	Jump Shot	36	49900088	34.0443	117	
30674	Jump Shot	Jump Shot	81	49900088	33.8283	117	
30675							

	Jump Shot	Jump Shot	84	49900088	33.8283	-134
30676	Running Jump Shot	Jump Shot	98	49900088	34.0443	-141
30677	Jump Shot	Jump Shot	101	49900088	33.9013	-113
30678	Driving Layup Shot	Layup	181	49900088	34.0283	14
30679	Layup Shot	Layup	212	49900088	34.0443	0
30681	Jump Shot	Jump Shot	218	49900088	33.7833	-18
30683	Jump Shot	Jump Shot	228	49900088	33.8283	1
30684	Jump Shot	Jump Shot	231	49900088	33.9553	-96
30685	Jump Shot	Jump Shot	249	49900088	33.7943	81
30687	Jump Shot	Jump Shot	284	49900088	33.9443	40
30688	Jump Shot	Jump Shot	308	49900088	33.9833	-126
30689	Jump Shot	Jump Shot	326	49900088	33.3653	-12
30690	Jump Shot	Jump Shot	331	49900088	33.9443	-113
30691	Driving Layup Shot	Layup	382	49900088	34.0443	0
30692	Jump Shot	Jump Shot	397	49900088	33.9963	1
30694	Running Jump Shot	Jump Shot	426	49900088	33.8783	-134
30695	Jump Shot	Jump Shot	448	49900088	33.7773	31
30696	Jump Shot	Jump Shot	471	49900088	33.9723	1

In [8]:

```
kobe.shot_made_flag.value_counts(normalize=True)
```

Out[8]:

```
0.0    0.553839
1.0    0.446161
Name: shot_made_flag, dtype: float64
```

In [9]:

```
kobe.shot_made_flag.value_counts(normalize=False)
```

Out[9]:

```
0.0    14232
1.0    11465
Name: shot_made_flag, dtype: int64
```

In [10]:

```
num_columns = [col for col, dtype in zip(kobe.columns, kobe.dtypes) if dtype !=
'object']
num_columns
```

Out[10]:

```
['game_event_id',
 'game_id',
 'lat',
 'loc_x',
 'loc_y',
 'lon',
 'minutes_remaining',
 'period',
 'playoffs',
 'seconds_remaining',
 'shot_distance',
 'shot_made_flag',
 'team_id',
 'shot_id']
```

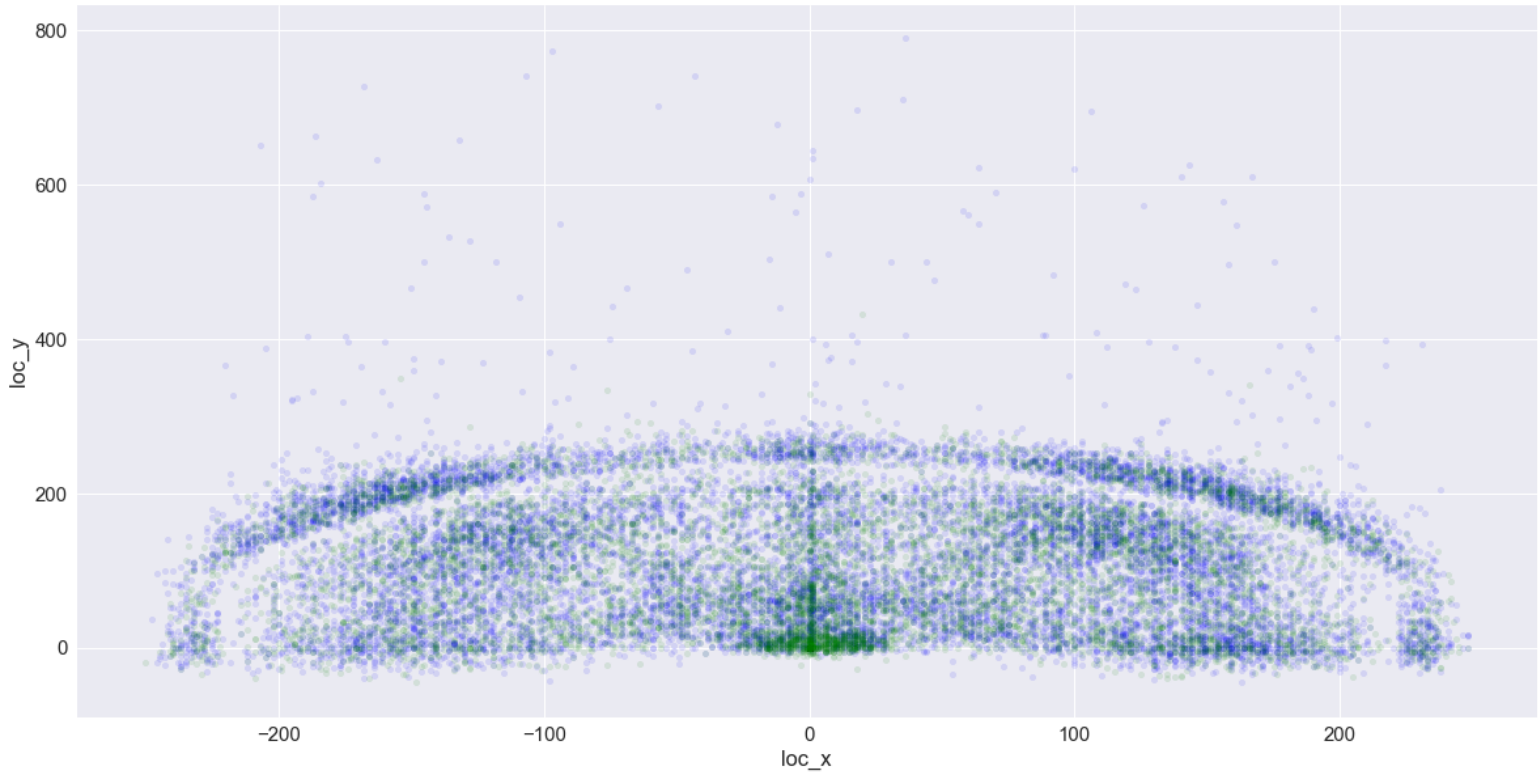
The `shot_made_flag` is the result (0 or 1) of the shot that Kobe took. Some of the values are missing (e.g. `NaN`) but we *dropped* them.

In [11]:

```
fig, ax = plt.subplots()
kobe[kobe.shot_made_flag==0].plot(kind='scatter', x='loc_x', y='loc_y', color='b
lue', alpha=0.1, ax=ax)
kobe[kobe.shot_made_flag==1].plot(kind='scatter', x='loc_x', y='loc_y', color='g
reen', alpha=0.1, ax=ax)
# plt.scatter(kobe.loc_x, kobe.loc_y, alpha=0.2)
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1025ff60>
```

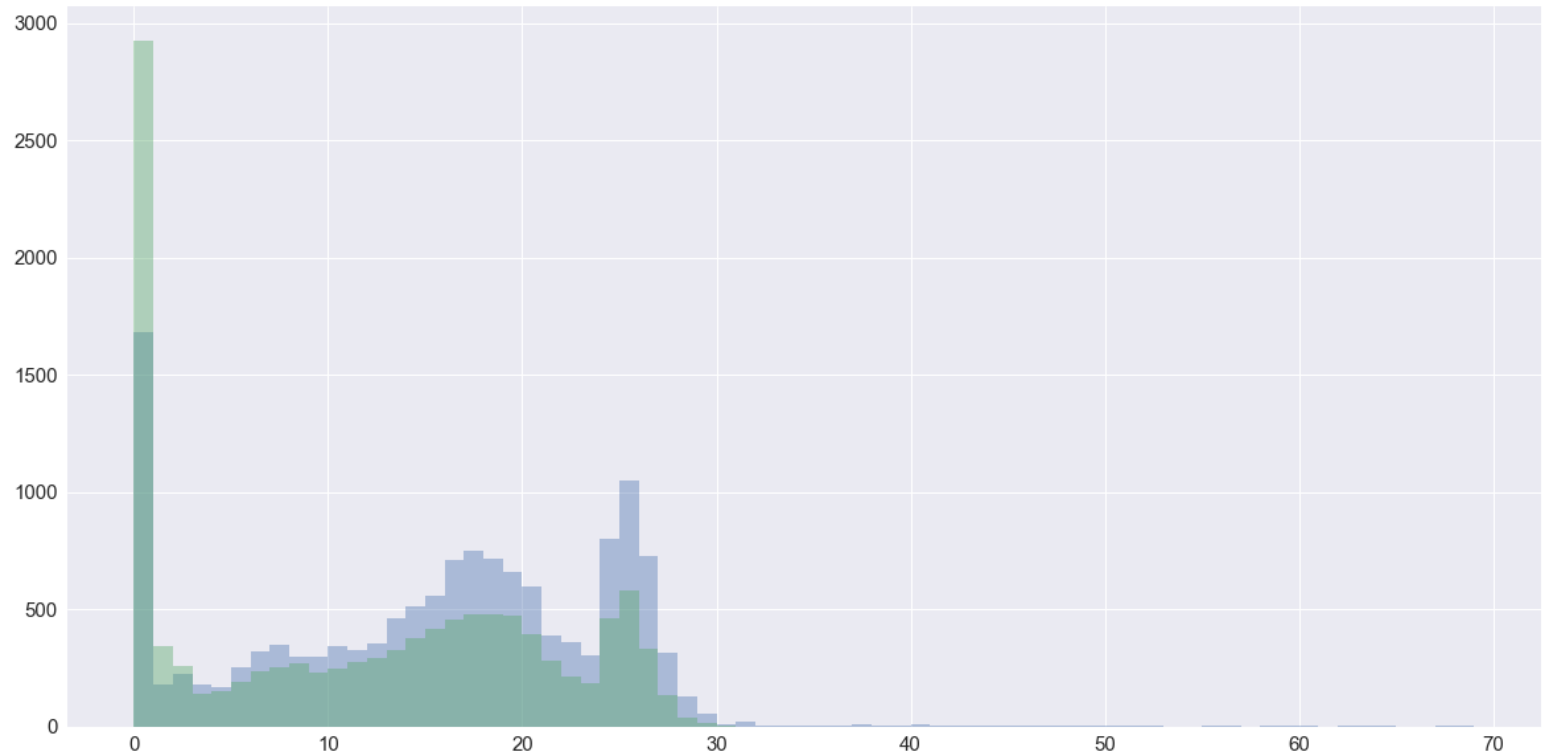


In [12]:

```
kobe[kobe.shot_made_flag==0].shot_distance.hist(bins=np.arange(0,70,1), alpha=.4
)
kobe[kobe.shot_made_flag==1].shot_distance.hist(bins=np.arange(0,70,1), alpha=.4
)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0ea1bf98>
```

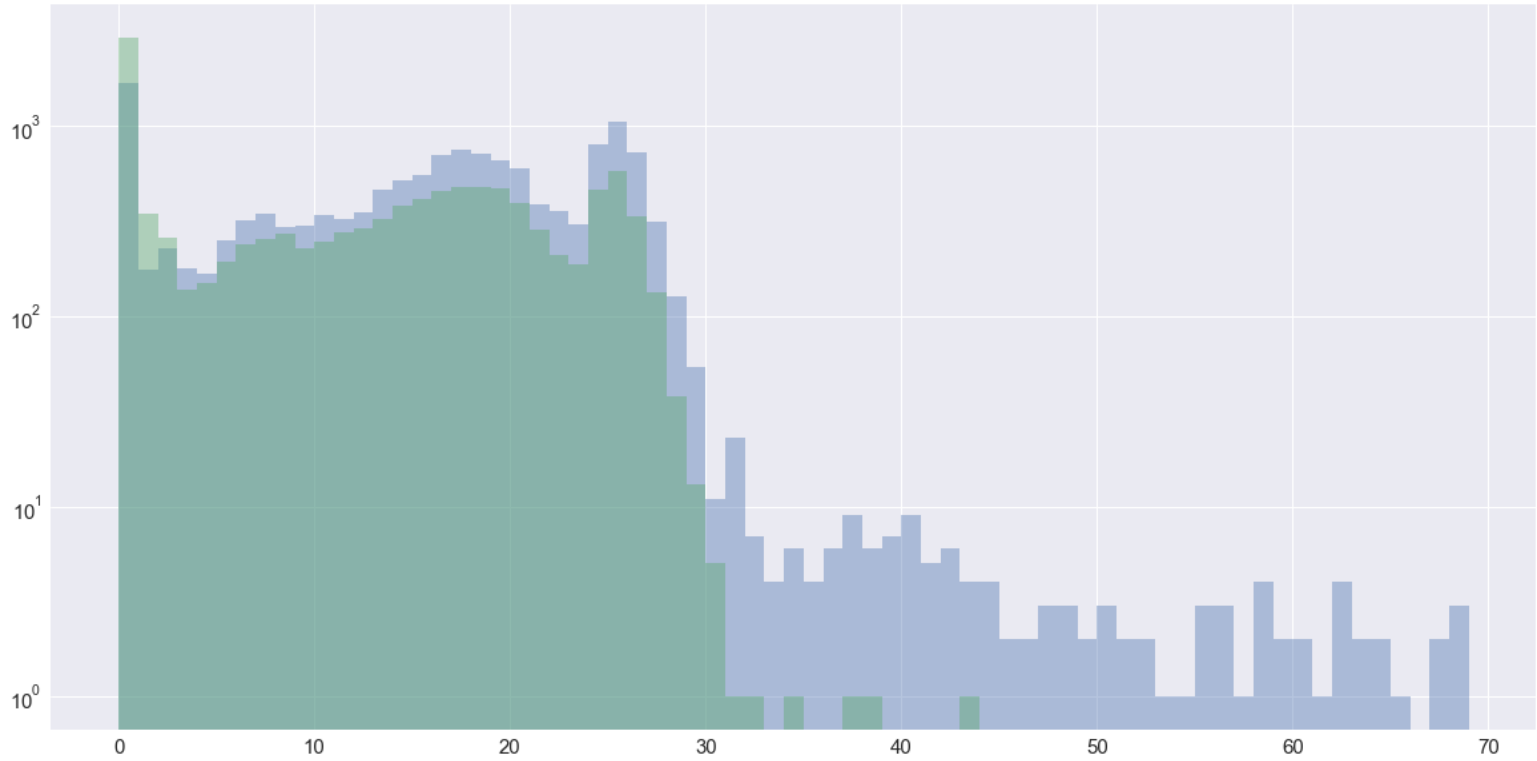


In [13]:

```
kobe[kobe.shot_made_flag==0].shot_distance.hist(bins=np.arange(0,70,1), alpha=.4
, log=True)
kobe[kobe.shot_made_flag==1].shot_distance.hist(bins=np.arange(0,70,1), alpha=.4
, log=True)
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a102b2278>
```



In [14]:

```
# fit a linear regression model and store the predictions
feature_cols = ['shot_distance', 'minutes_remaining']
X = kobe[feature_cols]
y = kobe.shot_made_flag

model = Model()
model.fit(X, y)
kobe['pred'] = model.predict(X)


from sklearn.metrics import accuracy_score
accuracy_score(kobe.shot_made_flag, kobe.pred.round())
```

Out[14]:

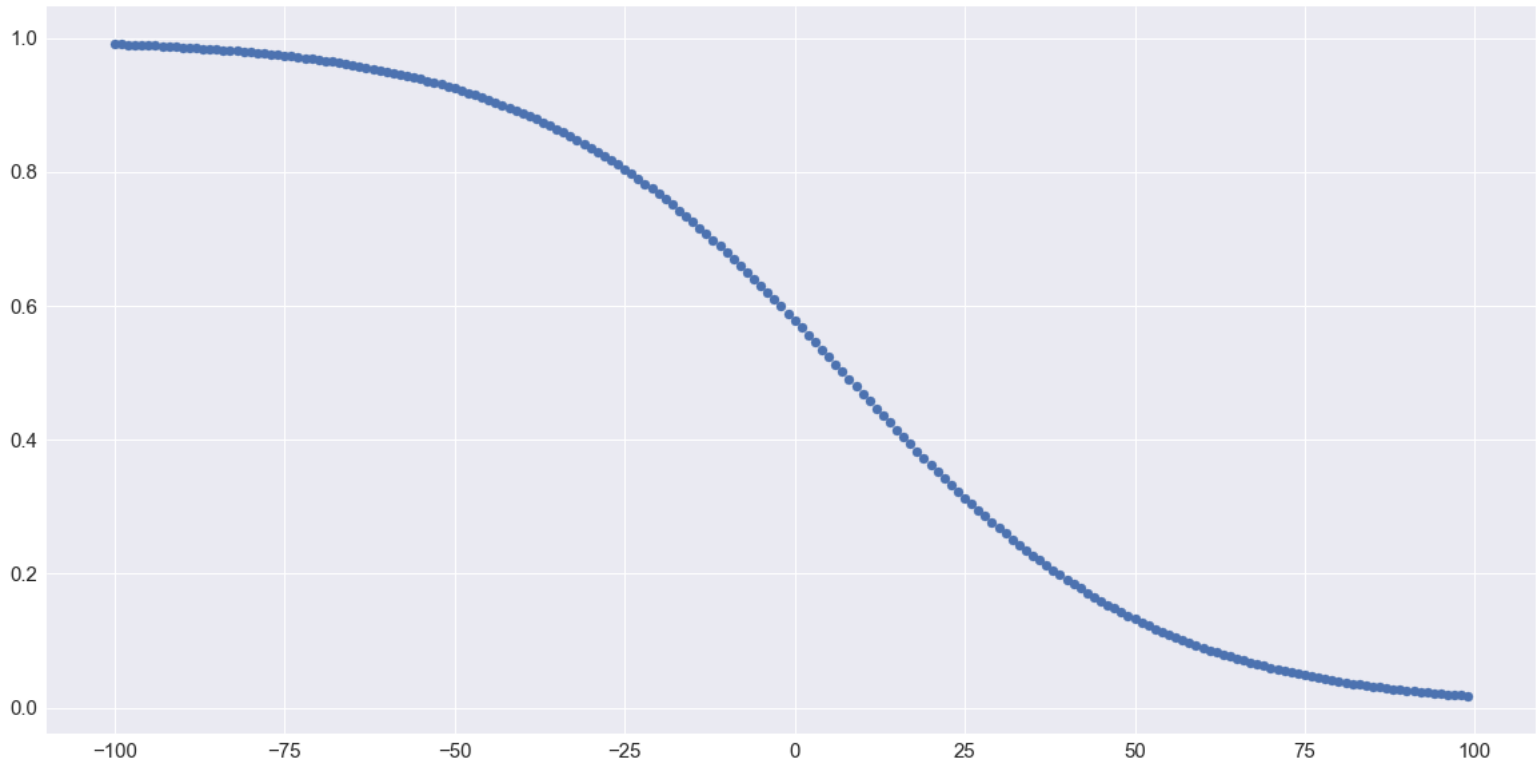
```
0.59719033350196526
```

In [15]:

```
distances = np.arange(-100, 100)
minutes = np.array([0]*200)
x_trial = np.column_stack((distances, minutes))
model.predict_proba(x_trial)
plt.scatter(distances, model.predict_proba(x_trial)[:,1])
```

Out[15]:

<matplotlib.collections.PathCollection at 0x1a11038ef0>

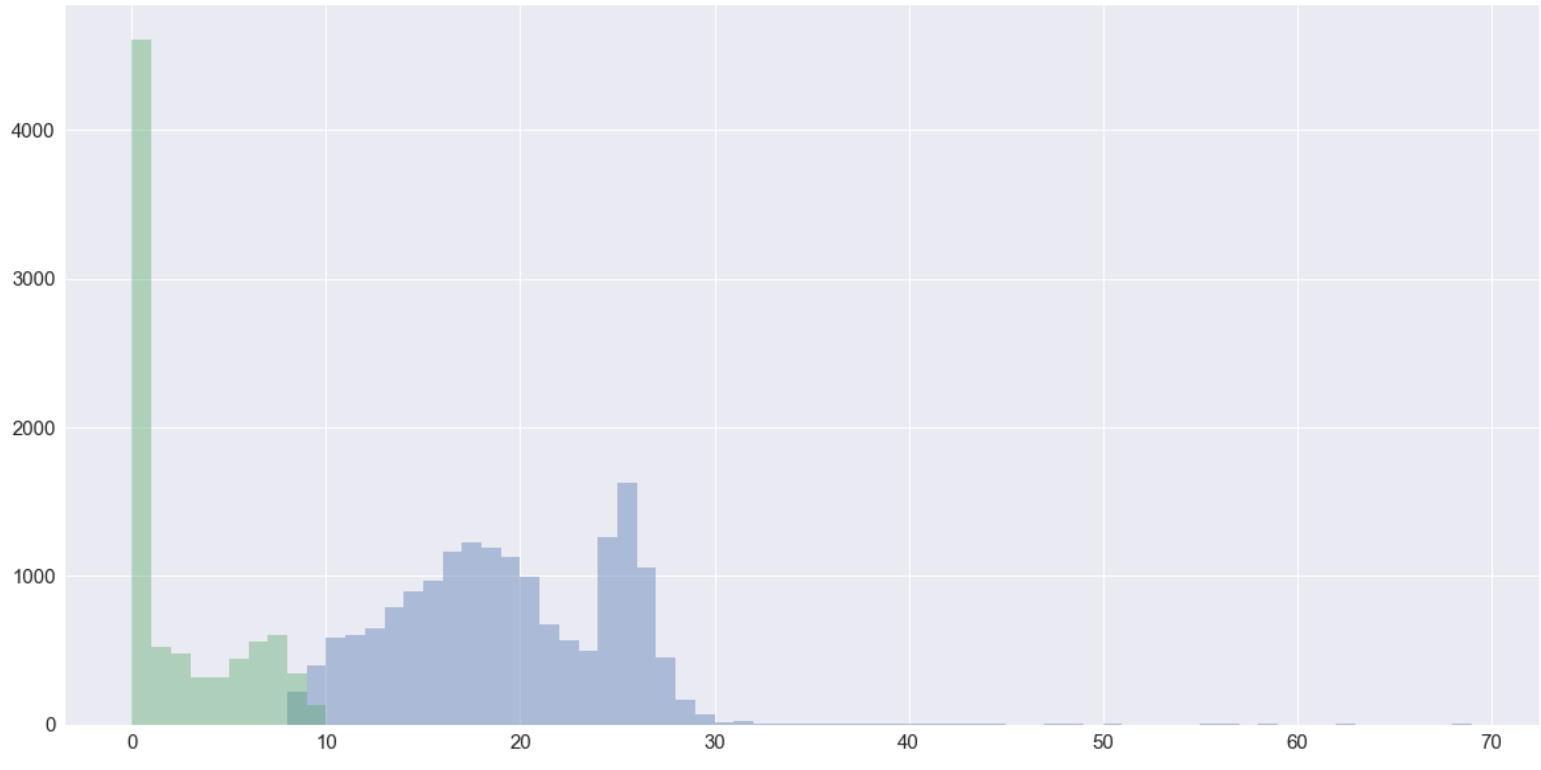


In [16]:

```
kobe[(kobe.pred==0)].shot_distance.hist(bins=np.arange(0,70,1), alpha=.4)
kobe[(kobe.pred==1)].shot_distance.hist(bins=np.arange(0,70,1), alpha=.4)
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1105ae48>
```

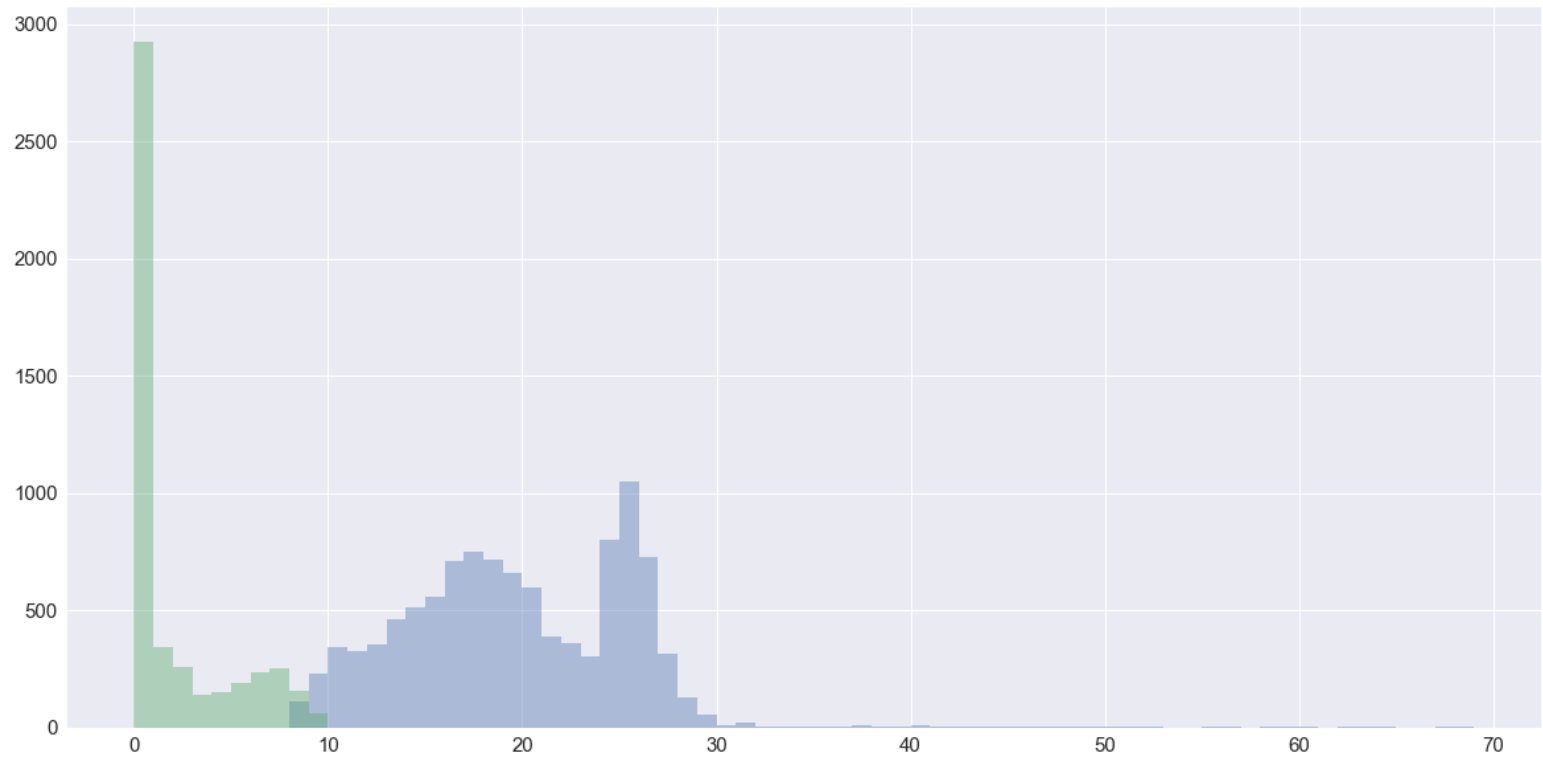


In [17]:

```
kobe[(kobe.pred==0) & (kobe.shot_made_flag==0)].shot_distance.hist(bins=np.arang
e(0,70,1), alpha=.4)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==1)].shot_distance.hist(bins=np.arang
e(0,70,1), alpha=.4)
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10eaf6a0>
```

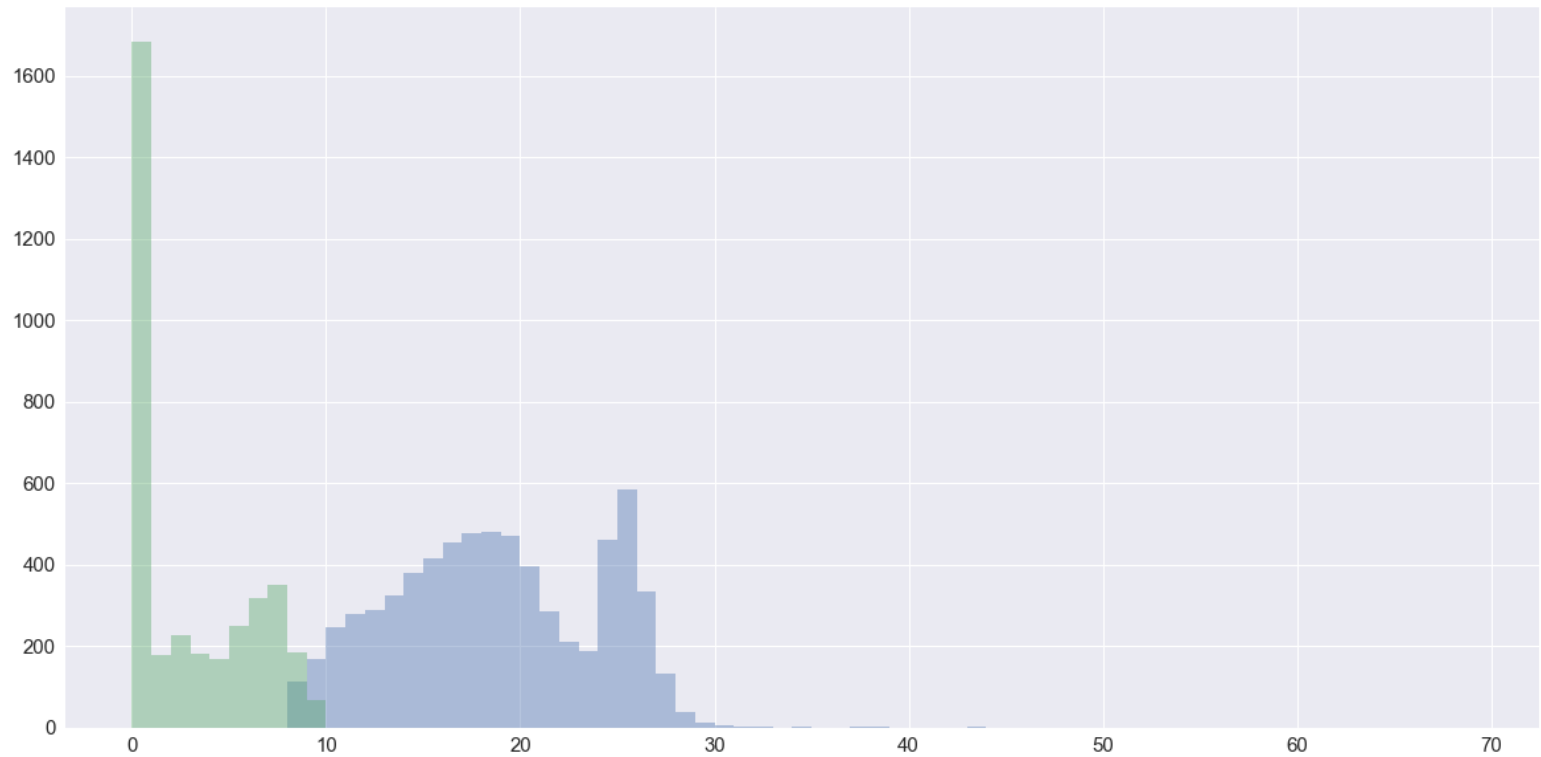


In [18]:

```
kobe[(kobe.pred==0) & (kobe.shot_made_flag==1)].shot_distance.hist(bins=np.arang
e(0,70,1), alpha=.4)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==0)].shot_distance.hist(bins=np.arang
e(0,70,1), alpha=.4)
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a16414c88>
```

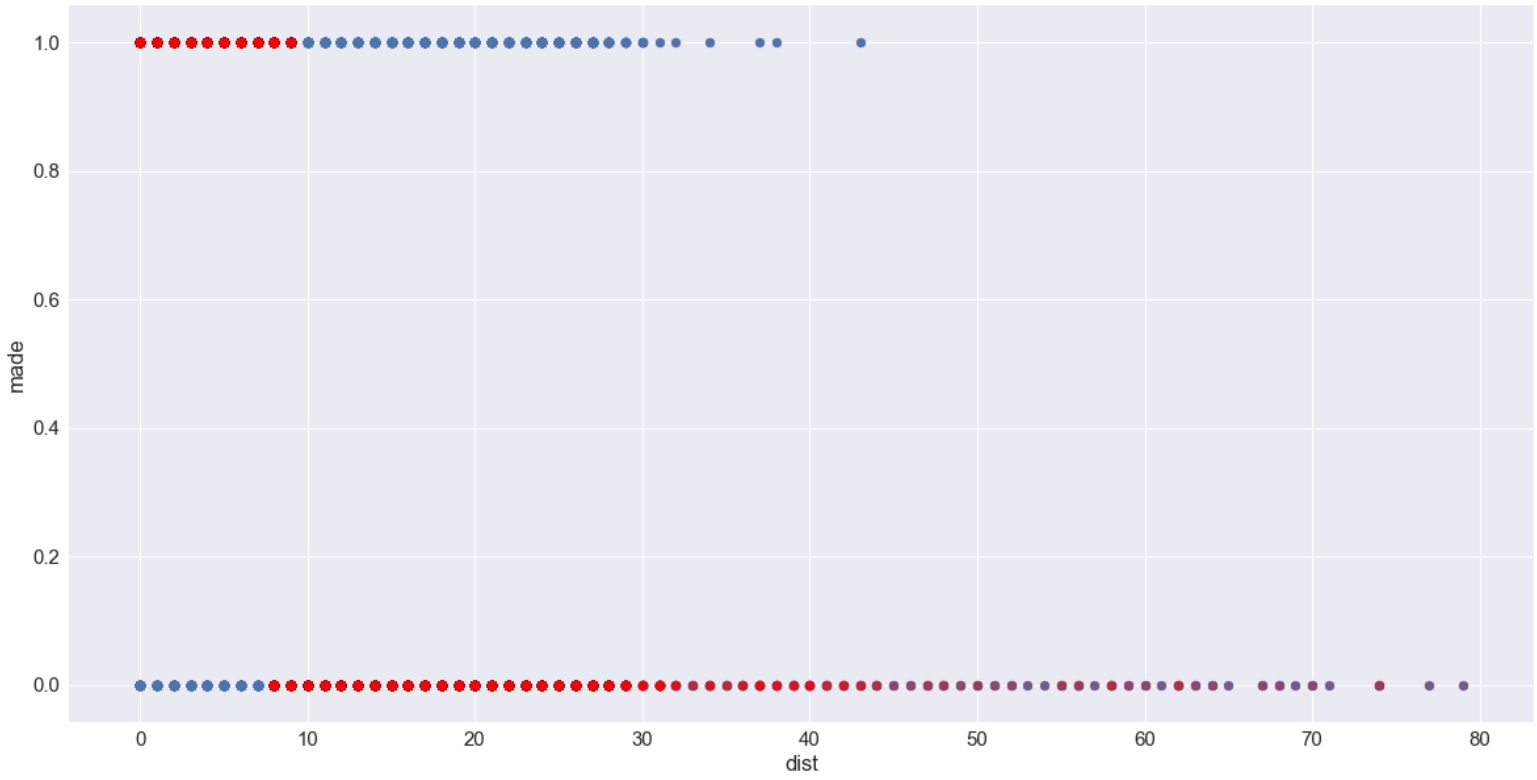


In [19]:

```
# scatter plot that includes the regression line
plt.scatter(kobe.shot_distance, kobe.shot_made_flag)
plt.scatter(kobe.shot_distance, kobe.pred, color='red', alpha=.2)
plt.xlabel('dist')
plt.ylabel('made')
```

Out[19]:

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



The following is a reminder of how the SciKit-Learn Models can be interfaced

In [20]:

```
from sklearn.linear_model import LogisticRegression as Model
# from sklearn.tree import DecisionTreeClassifier as Model
# from sklearn.ensemble import RandomForestClassifier as Model
model = Model()

from sklearn.metrics import (accuracy_score,
                             classification_report,
                             confusion_matrix, auc, roc_curve
                            )
from sklearn.metrics import *
from sklearn import cross_validation

X_train, X_test, y_train, y_test = cross_validation.train_test_split(
    X, y, test_size=0.4, random_state=0)

cross_validation.cross_val_score(model, X, y, cv=10)
```

```
/Users/jamescheever/anaconda3/lib/python3.6/site-packages/sklearn/cr
oss_validation.py:41: DeprecationWarning: This module was deprecated
in version 0.18 in favor of the model_selection module into which al
l the refactored classes and functions are moved. Also note that the
interface of the new CV iterators are different from that of this mo
dule. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Out[20]:

```
array([ 0.59237651,  0.59354337,  0.59299611,  0.59688716,  0.617509
73,
        0.58388478,  0.60334761,  0.60918645,  0.60140132,  0.583884
78])
```

Assignment

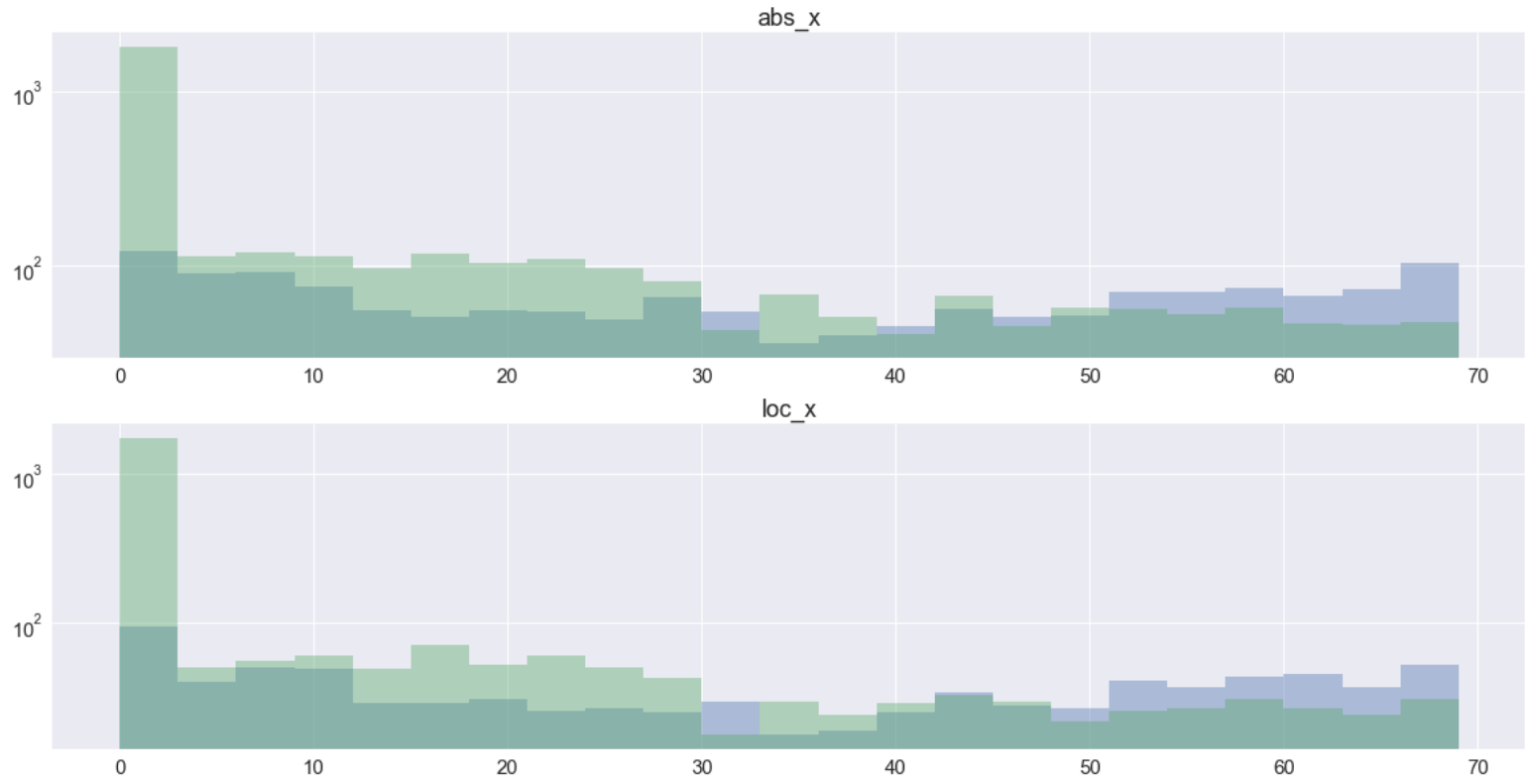
Warmup. Perform some analysis on Kobe's shot selection. Ask and answer (with charts) questions such as: Does Kobe make more shots in the 4th quarter than on average? Does Kobe make more shots from the left more than the right? What was Kobe's best year for shooting percentage? Etc. The more naunced the more you'll have a feel for the data.

1. Create a new column called `abs_x` that is equal to the absolute value of loc_x. Plot a histogram of made shots and missed shots using this variable. Explain in detail (with graphics and evidence) why this could be a better feature/column to use in a Logsitic Regression model instead of `loc_x`.

In [22]:

```
kobe['abs_x'] = abs(kobe.loc_x)

fig = plt.figure()

ax = fig.add_subplot(211)
ax.set_title('abs_x')
kobe[(kobe.pred==0) & (kobe.shot_made_flag==1)].abs_x.hist(bins=np.arange(0,70,3
), alpha=.4,log=True)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==0)].abs_x.hist(bins=np.arange(0,70,3
), alpha=.4,log=True)

ax = fig.add_subplot(212)
ax.set_title('loc_x')
kobe[(kobe.pred==0) & (kobe.shot_made_flag==1)].loc_x.hist(bins=np.arange(0,70,3
), alpha=.4,log=True)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==0)].loc_x.hist(bins=np.arange(0,70,3
), alpha=.4,log=True)

model = Model()
X = kobe[['loc_x']]
y = kobe.shot_made_flag

model = Model()
model.fit(X, y)
kobe['pred'] = model.predict(X)


from sklearn.metrics import accuracy_score
print('loc_x accuracy:' + str(accuracy_score(kobe.shot_made_flag, kobe.pred.roun
d())))

X = kobe[['abs_x']]
model = Model()
model.fit(X, y)
kobe['pred'] = model.predict(X)

from sklearn.metrics import accuracy_score
print('abs_x accuracy:' + str(accuracy_score(kobe.shot_made_flag, kobe.pred.roun
d())))
```

```
loc_x accuracy:0.55383896953
abs_x accuracy:0.592598357785
```



Explanation

abs(loc_x) could be a better predictor of a shot made if it made a difference which side of the court Kobe shot from, left or right, holding distance constant. However if distance in the horizontal direction (offset from a direct shot) is more important than abs_x will not make much of a difference in the model.

In this case, it seems there is greater prediction accuracy if you do not consider whether the shot is from the left or right of the goal.

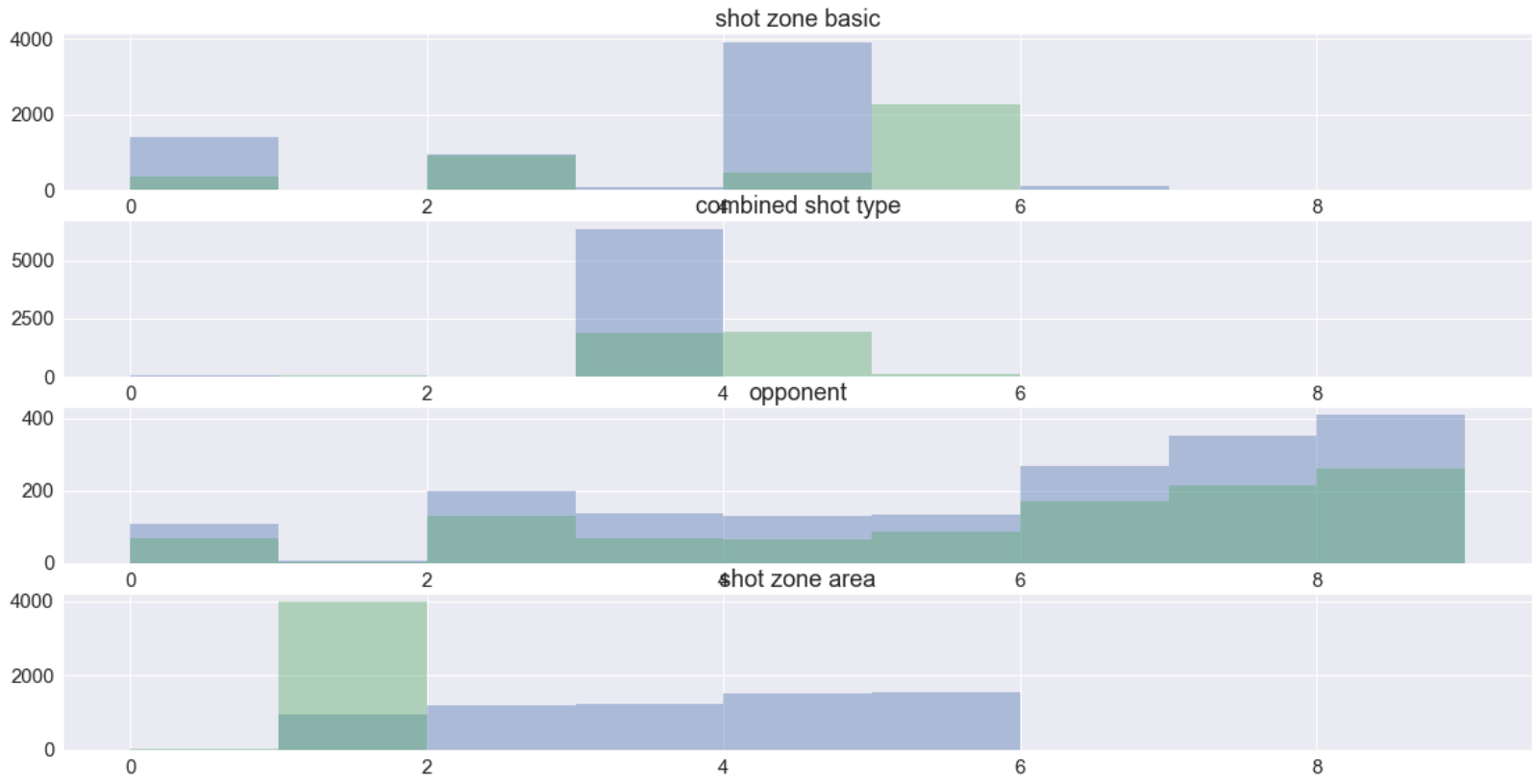
2. Convert several (including) string columns/features into numerical and attempt to use them in fitting a Logistic Regression model. Show histograms (similar to ones above) of made/missed of these new numerical features. Use these histograms to explain and justify why these features could improve the model

In [23]:

```
kobe['szb'] = kobe['shot_zone_basic'].astype('category').cat.codes
kobe['cst'] = kobe['combined_shot_type'].astype('category').cat.codes
kobe['opp'] = kobe['opponent'].astype('category').cat.codes
kobe['sza'] = kobe['shot_zone_area'].astype('category').cat.codes

fig = plt.figure()

ax = fig.add_subplot(411)
ax.set_title('shot zone basic')
kobe[(kobe.pred==0) & (kobe.shot_made_flag==1)].szb.hist(bins=np.arange(0,10,1), alpha=
.4)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==0)].szb.hist(bins=np.arange(0,10,1), alpha=
.4)


ax = fig.add_subplot(412)
ax.set_title('combined shot type')
kobe[(kobe.pred==0) & (kobe.shot_made_flag==1)].cst.hist(bins=np.arange(0,10,1), alpha=
.4)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==0)].cst.hist(bins=np.arange(0,10,1), alpha=
.4)


ax = fig.add_subplot(413)
ax.set_title('opponent')
kobe[(kobe.pred==0) & (kobe.shot_made_flag==1)].opp.hist(bins=np.arange(0,10,1), alpha=
.4)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==0)].opp.hist(bins=np.arange(0,10,1), alpha=
.4)

ax = fig.add_subplot(414)
ax.set_title('shot zone area')
kobe[(kobe.pred==0) & (kobe.shot_made_flag==1)].sza.hist(bins=np.arange(0,10,1), alpha=
.4)
kobe[(kobe.pred==1) & (kobe.shot_made_flag==0)].sza.hist(bins=np.arange(0,10,1), alpha=
.4)

model = Model()
feature_cols = ['szb', 'opp']
X = kobe[feature_cols]
y = kobe.shot_made_flag

model = Model()
model.fit(X, y)
kobe['pred'] = model.predict(X)

from sklearn.metrics import accuracy_score
accuracy_score(kobe.shot_made_flag, kobe.pred.round())
```

Out[23]:

0.60415612717437828



Explanation

We get better prediction accuracy by using a rough "principal component analysis" by reducing the feature set into a general feature whether than a specific measurement.

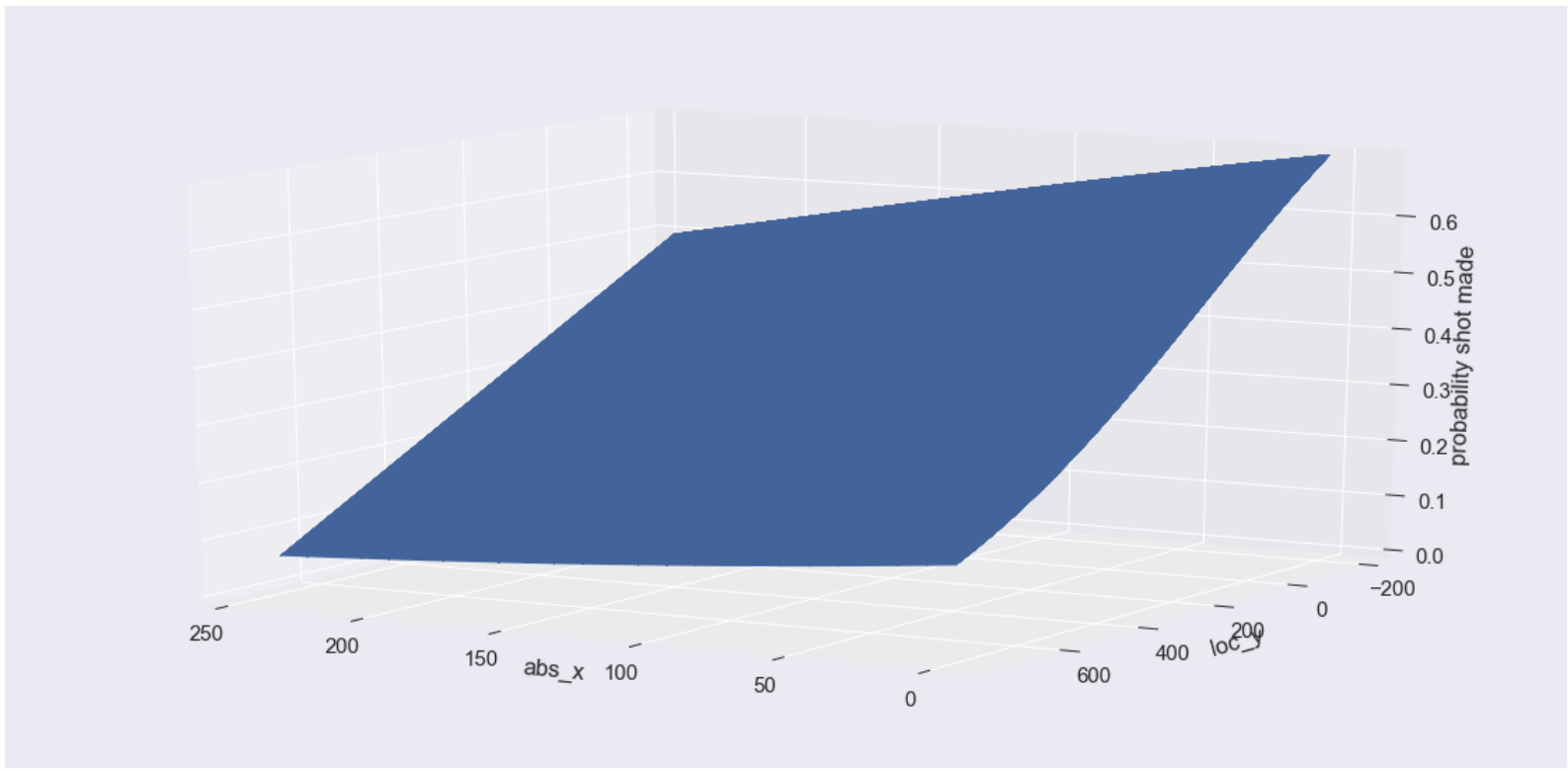
3. Show a 3 dimensional surface plot [https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html#surface-plots (https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html#surface-plots)] of probabilities from a trained Logistic Regression model using only abs_x and loc_y. The probabilities arise from a distributed grid of x values and y values as input to the predict_proba() function.

In [24]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.view_init(15, 120)

feature_cols = ['abs_x', 'loc_y']
X = kobe[feature_cols]
y =  kobe.shot_made_flag
model = LogisticRegression()
model.fit(X,y)

x = []
y = []

for x1 in np.arange(0, 250,10):
    for y1 in np.arange(-200, 800,50):
        x.append(x1)
        y.append(y1)

X,Y = np.meshgrid(x,y)
zs = np.column_stack((np.ravel(X), np.ravel(Y)))
p = model.predict_proba(zs)
Z = np.array([v[1] for v in p])
Z = Z.reshape(X.shape)

surf = ax.plot_surface(X,Y,Z,linewidth=0)
ax.set_zlabel('probability shot made')
ax.set_xlabel('abs_x')
ax.set_ylabel('loc_y')
ax.set_zlim(0.0, 0.7)
```

Out[24]:

(0.0, 0.7)


probability shot made
0.6
0.5
0.4
0.3
0.2
0.1
0.0
250
200
150
abs_x
100
50
0
600
400
200
0
−200