2110446 Data Science and Data Engineering

Preparing and Cleaning Data for Machine Learning

Kaggle Competition | Titanic Machine Learning from Disaster

> The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.
>
> One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.
>
> In this contest, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.
>
> This Kaggle Getting Started Competition provides an ideal starting place for people who may not have a lot of experience in data science and machine learning."

From the competition homepage (http://www.kaggle.com/c/titanic-gettingStarted).

Goal for this Notebook:

Show a simple example of an analysis of the Titanic disaster in Python using a full complement of PyData utilities. This is aimed for those looking to get into the field or those who are already in the field and looking to see an example of an analysis done with Python.

This Notebook will show basic examples of:

Data Handling

- Importing Data with Pandas
- Cleaning Data
- Exploring Data through Visualizations with Matplotlib

Data Analysis

- Supervised Machine learning Techniques:
  - Logit Regression Model

    - Plotting results
    - Support Vector Machine (SVM) using 3 kernels
    - Basic Random Forest
    - Plotting results

Valuation of the Analysis

- K-folds cross validation to valuate results locally
- Output the results from the IPython Notebook to Kaggle

Required Libraries:

- NumPy (http://www.numpy.org/)
- IPython (http://ipython.org/)
- Pandas (http://pandas.pydata.org/)
- SciKit-Learn (http://scikit-learn.org/stable/)
- SciPy (http://www.scipy.org/)
- StatsModels (http://statsmodels.sourceforge.net/)
- Patsy (http://patsy.readthedocs.org/en/latest/)
- Matplotlib (http://matplotlib.org/)

```
In [3]:
```

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.nonparametric.kde import KDEUnivariate
from statsmodels.nonparametric import smoothers_lowess
from pandas import Series, DataFrame
from patsy import dmatrices
from sklearn import datasets, svm
#from KaggleAux import predict as ka # see github.com/agconti/kaggleaux for more
```

Data Handling

Let's read our data in using pandas:

```
In [4]:
```

```
df = pd.read_csv("data/train.csv")
```

Show an overview of our data:

```
In [5]:
```

```
df
```

```
Out[5]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Na
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C8
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Na
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C12
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	Na

Let's take a look:

Above is a summary of our data contained in a `Pandas` `DataFrame`. Think of a `DataFrame` as a Python's super charged version of the workflow in an Excel table. As you can see the summary holds quite a bit of information. First, it lets us know we have 891 observations, or passengers, to analyze here:

```
Int64Index: 891 entries, 0 to 890
```

Next it shows us all of the columns in `DataFrame`. Each column tells us something about each of our observations, like their `name`, `sex` or `age`. These colunms are called a features of our dataset. You can think of the meaning of the words column and feature as interchangeable for this notebook.

After each feature it lets us know how many values it contains. While most of our features have complete data on every observation, like the `survived` feature here:

```
survived    891  non-null values
```

some are missing information, like the `age` feature:

```
age         714  non-null values
```

These missing values are represented as `NaN`s.

Take care of missing values:

The features `ticket` and `cabin` have many missing values and so can't add much value to our analysis. To handle this we will drop them from the dataframe to preserve the integrity of our dataset.

To do that we'll use this line of code to drop the features entirely:

```
df = df.drop(['ticket','cabin'], axis=1)
```

While this line of code removes the `NaN` values from every remaining column / feature:

```
df = df.dropna()
```

Now we have a clean and tidy dataset that is ready for analysis. Because `.dropna()` removes an observation from our data even if it only has 1 `NaN` in one of the features, it would have removed most of our dataset if we had not dropped the `ticket` and `cabin` features first.

In [6]:

```
df = df.drop(['Ticket','Cabin'], axis=1)
# Remove NaN values
df = df.dropna()
```

For a detailed look at how to use pandas for data analysis, the best resource is Wes Mckinney's book (http://shop.oreilly.com/product/0636920023784.do). Additional interactive tutorials that cover all of the basics can be found here (https://bitbucket.org/hrojas/learn-pandas) (they're free). If you still need to be convinced about the power of pandas check out this wirlwhind look (http://wesmckinney.com/blog/?p=647) at all that pandas can do.

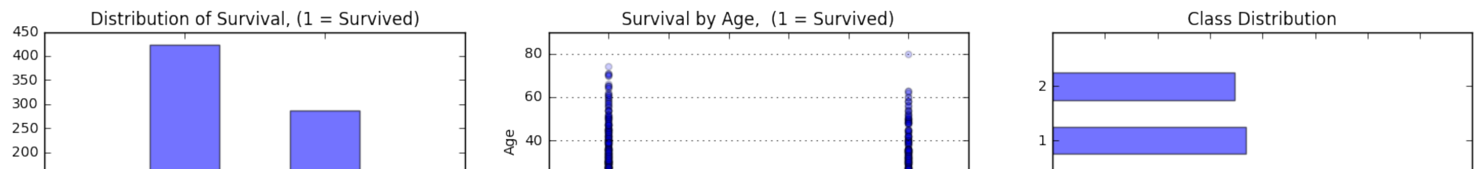
Let's take a Look at our data graphically:

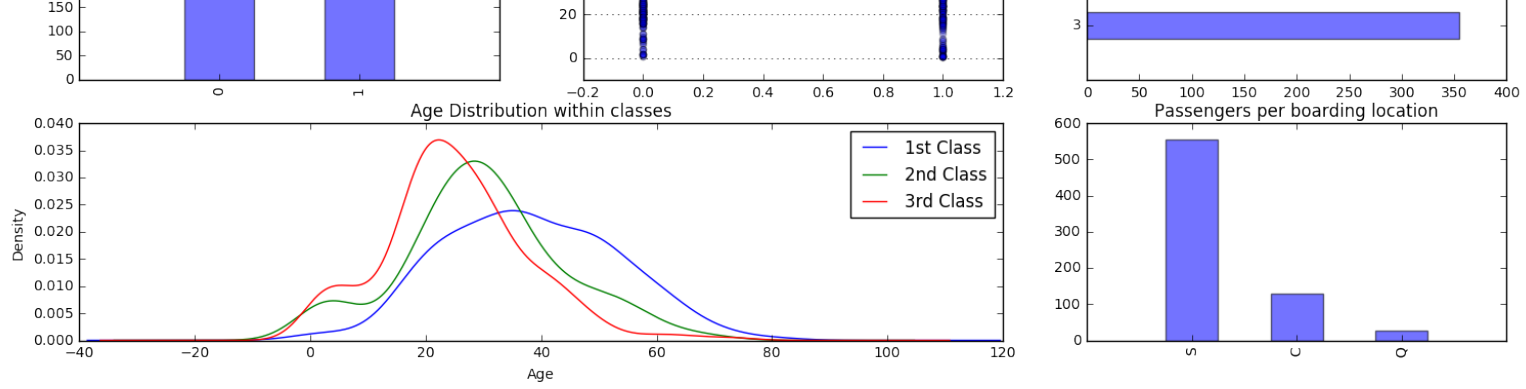
In [7]:

```
# specifies the parameters of our graphs
fig = plt.figure(figsize=(18,6), dpi=1600)
alpha=alpha_scatterplot = 0.2
alpha_bar_chart = 0.55

# lets us plot many diffrent shaped graphs together
ax1 = plt.subplot2grid((2,3),(0,0))
# plots a bar graph of those who surived vs those who did not.
df.Survived.value_counts().plot(kind='bar', alpha=alpha_bar_chart)
# this nicely sets the margins in matplotlib to deal with a recent bug 1.3.1
ax1.set_xlim(-1, 2)
# puts a title on our graph
plt.title("Distribution of Survival, (1 = Survived)")

plt.subplot2grid((2,3),(0,1))
plt.scatter(df.Survived, df.Age, alpha=alpha_scatterplot)
# sets the y axis lable
plt.ylabel("Age")
# formats the grid line style of our graphs
plt.grid(b=True, which='major', axis='y')
plt.title("Survival by Age,  (1 = Survived)")

ax3 = plt.subplot2grid((2,3),(0,2))
df.Pclass.value_counts().plot(kind="barh", alpha=alpha_bar_chart)
ax3.set_ylim(-1, len(df.Pclass.value_counts()))
plt.title("Class Distribution")

plt.subplot2grid((2,3),(1,0), colspan=2)
# plots a kernel density estimate of the subset of the 1st class passangers's age
df.Age[df.Pclass == 1].plot(kind='kde')
df.Age[df.Pclass == 2].plot(kind='kde')
df.Age[df.Pclass == 3].plot(kind='kde')
 # plots an axis lable
plt.xlabel("Age")
plt.title("Age Distribution within classes")
# sets our legend for our graph.
plt.legend(('1st Class', '2nd Class','3rd Class'),loc='best')

ax5 = plt.subplot2grid((2,3),(1,2))
df.Embarked.value_counts().plot(kind='bar', alpha=alpha_bar_chart)
ax5.set_xlim(-1, len(df.Embarked.value_counts()))
# specifies the parameters of our graphs
plt.title("Passengers per boarding location")
```

Out[7]:

```
<matplotlib.text.Text at 0x123a93128>
```





Exploratory Visualization:

The point of this competition is to predict if an individual will survive based on the features in the data like:

- Traveling Class (called pclass in the data)
- Sex
- Age
- Fare Price

Let’s see if we can gain a better understanding of who survived and died.

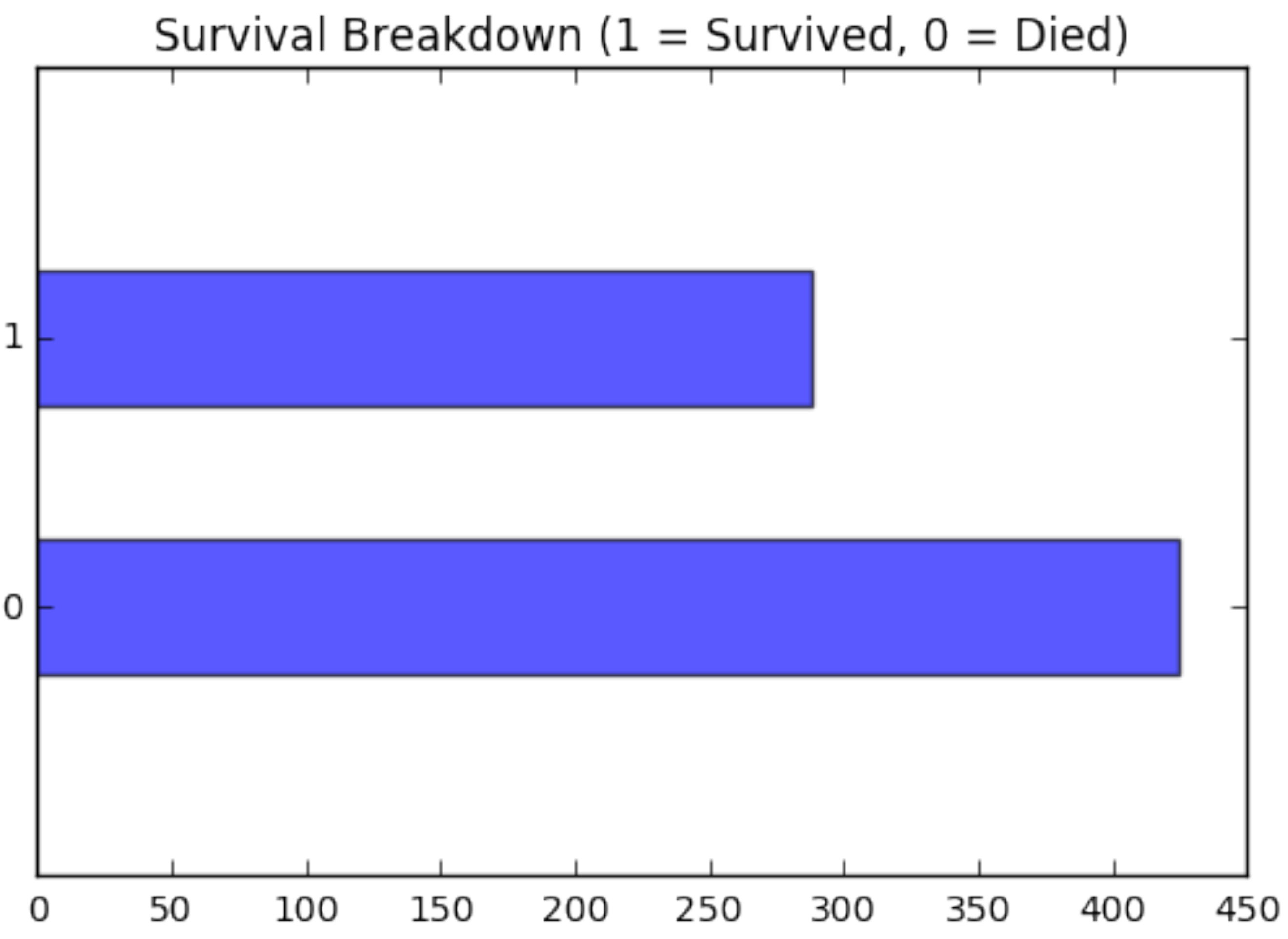
First let’s plot a bar graph of those who Survived Vs. Those who did not.

In [8]:

```
plt.figure(figsize=(6,4))
fig, ax = plt.subplots()
df.Survived.value_counts().plot(kind='barh', color="blue", alpha=.65)
ax.set_ylim(-1, len(df.Survived.value_counts()))
plt.title("Survival Breakdown (1 = Survived, 0 = Died)")
```

Out[8]:

```
<matplotlib.text.Text at 0x123baff98>


<matplotlib.figure.Figure at 0x1213656d8>
```



Now let’s tease more structure out of the data,

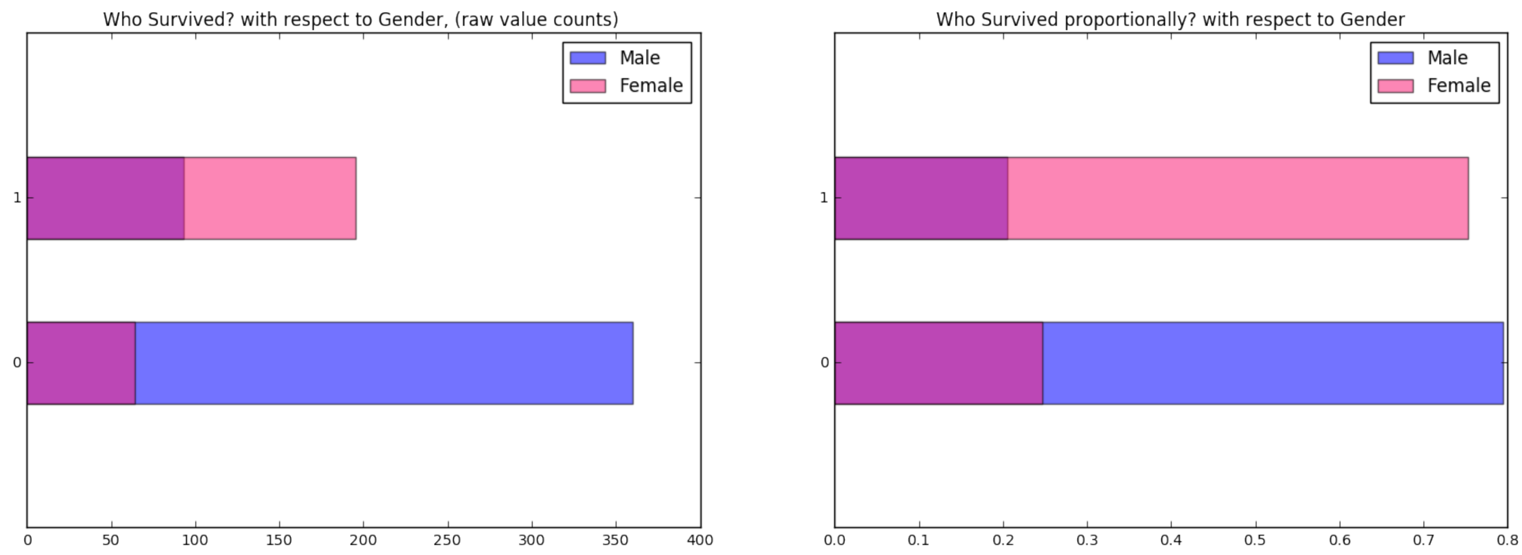
Let’s break the previous graph down by gender

```
In [9]:
```

```
fig = plt.figure(figsize=(18,6))

#create a plot of two subsets, male and female, of the survived variable.
#After we do that we call value_counts() so it can be easily plotted as a bar gra
#'barh' is just a horizontal bar graph
df_male = df.Survived[df.Sex == 'male'].value_counts().sort_index()
df_female = df.Survived[df.Sex == 'female'].value_counts().sort_index()

ax1 = fig.add_subplot(121)
df_male.plot(kind='barh',label='Male', alpha=0.55)
df_female.plot(kind='barh', color='#FA2379',label='Female', alpha=0.55)
plt.title("Who Survived? with respect to Gender, (raw value counts) "); plt.legen
ax1.set_ylim(-1, 2)

#adjust graph to display the proportions of survival by gender
ax2 = fig.add_subplot(122)
(df_male/float(df_male.sum())).plot(kind='barh',label='Male', alpha=0.55)
(df_female/float(df_female.sum())).plot(kind='barh', color='#FA2379',label='Femal
plt.title("Who Survived proportionally? with respect to Gender"); plt.legend(loc=

ax2.set_ylim(-1, 2)
```

```
Out[9]:
```

```
(-1, 2)
```



Here it’s clear that although more men died and survived in raw value counts, females had a greater survival rate proportionally (~25%), than men (~20%)

Great! But let’s go down even further:

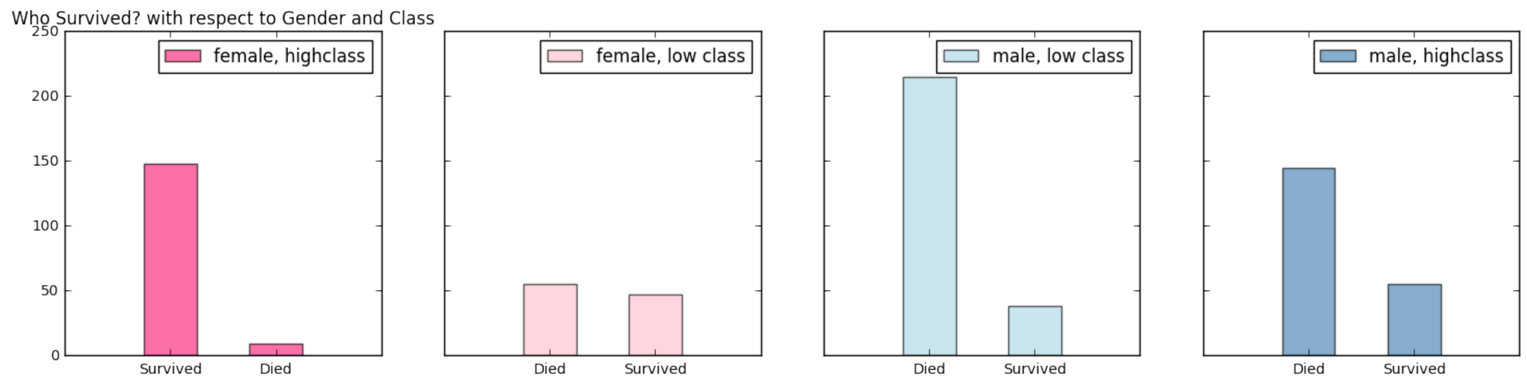
Can we capture more of the structure by using Pclass? Here we will bucket classes as lowest class or any of the high classes (classes 1 - 2). 3 is lowest class. Let’s break it down by Gender and what Class they were traveling in.

In [10]:

```
fig = plt.figure(figsize=(18,4), dpi=1600)
alpha_level = 0.65

# building on the previous code, here we create an additional subset with in the
# we created for the survived variable. I know, thats a lot of subsets. After we
# value_counts() so it it can be easily plotted as a bar graph. this is repeated
# class pair.
ax1=fig.add_subplot(141)
female_highclass = df.Survived[df.Sex == 'female'][df.Pclass != 3].value_counts()
female_highclass.plot(kind='bar', label='female, highclass', color='#FA2479', alp
ax1.set_xticklabels(["Survived", "Died"], rotation=0)
ax1.set_xlim(-1, len(female_highclass))
plt.title("Who Survived? with respect to Gender and Class"); plt.legend(loc='best

ax2=fig.add_subplot(142, sharey=ax1)
female_lowclass = df.Survived[df.Sex == 'female'][df.Pclass == 3].value_counts()
female_lowclass.plot(kind='bar', label='female, low class', color='pink', alpha=a
ax2.set_xticklabels(["Died","Survived"], rotation=0)
ax2.set_xlim(-1, len(female_lowclass))
plt.legend(loc='best')

ax3=fig.add_subplot(143, sharey=ax1)
male_lowclass = df.Survived[df.Sex == 'male'][df.Pclass == 3].value_counts()
male_lowclass.plot(kind='bar', label='male, low class',color='lightblue', alpha=a
ax3.set_xticklabels(["Died","Survived"], rotation=0)
ax3.set_xlim(-1, len(male_lowclass))
plt.legend(loc='best')

ax4=fig.add_subplot(144, sharey=ax1)
male_highclass = df.Survived[df.Sex == 'male'][df.Pclass != 3].value_counts()
male_highclass.plot(kind='bar', label='male, highclass', alpha=alpha_level, color
ax4.set_xticklabels(["Died","Survived"], rotation=0)
ax4.set_xlim(-1, len(male_highclass))
plt.legend(loc='best')
```

Out[10]:

```
<matplotlib.legend.Legend at 0x124c34b70>
```



Awesome! Now we have a lot more information on who survived and died in the tragedy. With this deeper understanding, we are better equipped to create better more insightful models. This is a typical process in interactive data analysis. First you start small and understand the most basic relationships and slowly increment the complexity of your analysis as you discover more and more about the data you're working with. Below is the progression of process laid out together:

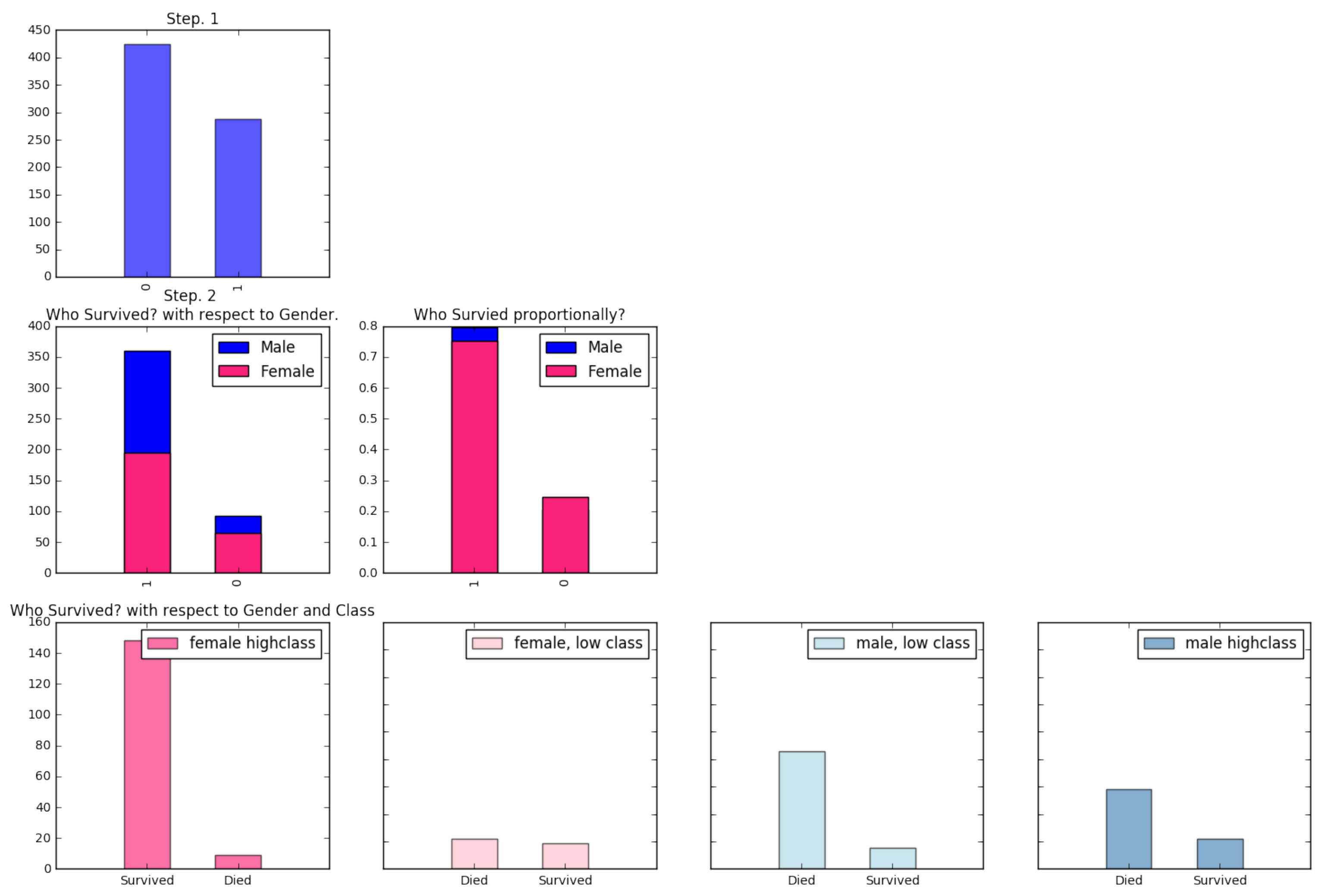
```
In [11]:
```

```
fig = plt.figure(figsize=(18,12), dpi=1600)
a = 0.65
# Step 1
ax1 = fig.add_subplot(341)
df.Survived.value_counts().plot(kind='bar', color="blue", alpha=a)
ax1.set_xlim(-1, len(df.Survived.value_counts()))
plt.title("Step. 1")

# Step 2
ax2 = fig.add_subplot(345)
df.Survived[df.Sex == 'male'].value_counts().plot(kind='bar',label='Male')
df.Survived[df.Sex == 'female'].value_counts().plot(kind='bar', color='#FA2379',l
ax2.set_xlim(-1, 2)
plt.title("Step. 2 \nWho Survived? with respect to Gender."); plt.legend(loc='bes

ax3 = fig.add_subplot(346)
(df.Survived[df.Sex == 'male'].value_counts()/float(df.Sex[df.Sex == 'male'].size
(df.Survived[df.Sex == 'female'].value_counts()/float(df.Sex[df.Sex == 'female'].
ax3.set_xlim(-1,2)
plt.title("Who Survied proportionally?"); plt.legend(loc='best')


# Step 3
ax4 = fig.add_subplot(349)
female_highclass = df.Survived[df.Sex == 'female'][df.Pclass != 3].value_counts()
female_highclass.plot(kind='bar', label='female highclass', color='#FA2479', alph
ax4.set_xticklabels(["Survived", "Died"], rotation=0)
ax4.set_xlim(-1, len(female_highclass))
plt.title("Who Survived? with respect to Gender and Class"); plt.legend(loc='best

ax5 = fig.add_subplot(3,4,10, sharey=ax1)
female_lowclass = df.Survived[df.Sex == 'female'][df.Pclass == 3].value_counts()
female_lowclass.plot(kind='bar', label='female, low class', color='pink', alpha=a
ax5.set_xticklabels(["Died","Survived"], rotation=0)
ax5.set_xlim(-1, len(female_lowclass))
plt.legend(loc='best')

ax6 = fig.add_subplot(3,4,11, sharey=ax1)
male_lowclass = df.Survived[df.Sex == 'male'][df.Pclass == 3].value_counts()
male_lowclass.plot(kind='bar', label='male, low class',color='lightblue', alpha=a
ax6.set_xticklabels(["Died","Survived"], rotation=0)
ax6.set_xlim(-1, len(male_lowclass))
plt.legend(loc='best')

ax7 = fig.add_subplot(3,4,12, sharey=ax1)
male_highclass = df.Survived[df.Sex == 'male'][df.Pclass != 3].value_counts()
male_highclass.plot(kind='bar', label='male highclass', alpha=a, color='steelblue
ax7.set_xticklabels(["Died","Survived"], rotation=0)
```

```
ax7.set_xticklabels(["Died", "Survived"], rotation=0)
ax7.set_xlim(-1, len(male_highclass))
plt.legend(loc='best')
```

```
Out[11]:
```

```
<matplotlib.legend.Legend at 0x1253dac50>
```



I've done my best to make the plotting code readable and intuitive, but if you're looking for a more detailed look on how to start plotting in matplotlib, check out this beautiful notebook here (http://nbviewer.ipython.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-4-Matplotlib.ipynb).

Now that we have a basic understanding of what we are trying to predict, let's predict it.

Supervised Machine Learning

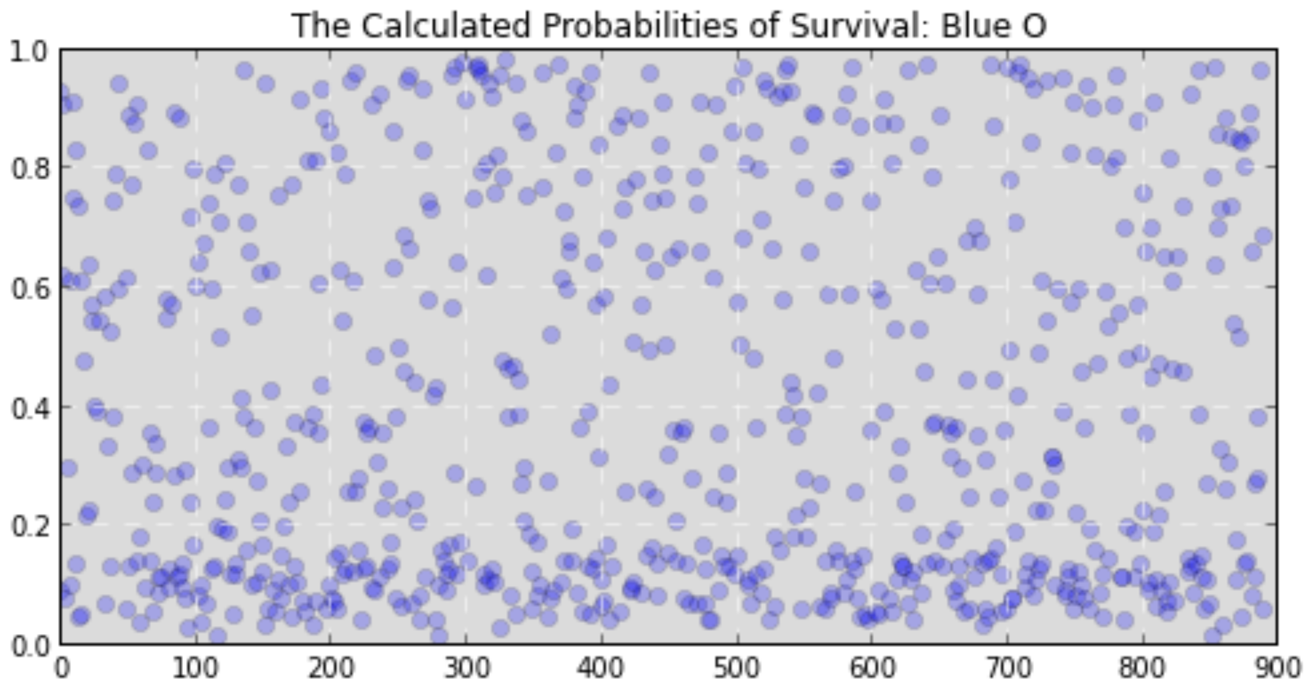
Logistic Regression:

As explained by Wikipedia:

In statistics, logistic regression or logit regression is a type of regression analysis used for predicting the outcome of a categorical dependent variable (a dependent variable that can take on a limited number of values, whose magnitudes are not meaningful but whose ordering of magnitudes may or may not be meaningful) based on one or more predictor variables. That is, it is used in estimating empirical values of the parameters in a qualitative response model. The probabilities describing the possible outcomes of a single trial are modeled, as a function of the explanatory (predictor) variables, using a logistic function. Frequently (and subsequently in this article) "logistic regression" is used to refer specifically to the problem in which the dependent variable is binary—that is, the number of available categories is two—and problems with more than two categories are referred to as multinomial logistic regression or, if the multiple categories are ordered, as ordered logistic regression. Logistic regression measures the relationship between a categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by using probability scores as the predicted values of the dependent variable. [1] As such it treats the same set of problems as does probit regression using similar techniques.

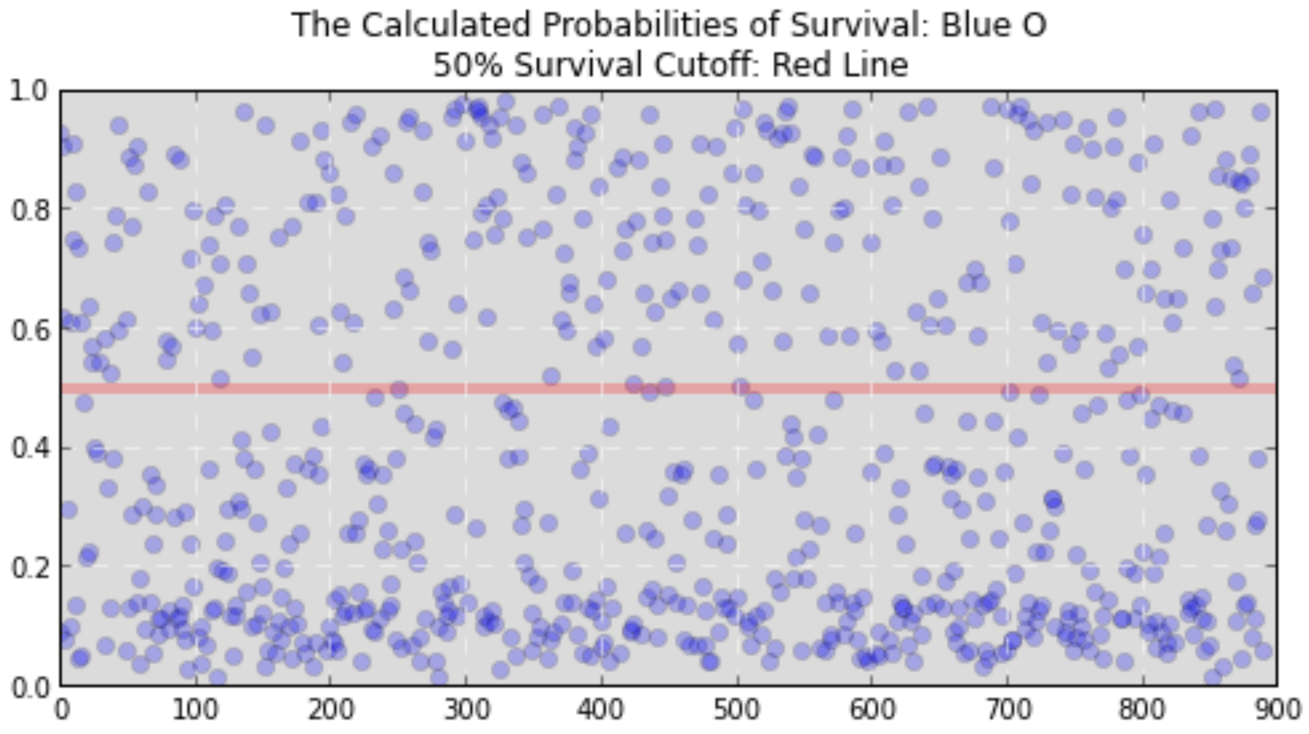
The skinny, as explained by yours truly:

Our competition wants us to predict a binary outcome. That is, it wants to know whether some will die, (represented as a 0), or survive, (represented as 1). A good place to start is to calculate the probability that an individual observation, or person, is likely to be a 0 or 1. That way we would know the chance that someone survives, and could start making somewhat informed predictions. If we did, we'd get results like this::



(*Y axis is the probability that someone survives, X axis is the passenger's number from 1 to 891.*)

While that information is useful it doesn't let us know whether someone ended up alive or dead. It just lets us know the chance that they will survive or die. We still need to translate these probabilities into the binary decision we're looking for. But how? We could arbitrarily say that our survival cutoff is anyone with a probability of survival over 50%. In fact, this tactic would actually perform pretty well for our data and would allow you to make decently accurate predictions. Graphically it would look something like this:



If you’re a betting man like me, you don’t like to leave everything to chance. What are the odds that setting that cutoff at 50% works? Maybe 20% or 80% would work better. Clearly we need a more exact way to make that cutoff. What can save the day? In steps the **Logistic Regression**.

A logistic regression follows the all steps we took above but mathematically calculates the cutoff, or decision boundary (as stats nerds call it), for you. This way it can figure out the best cut off to choose, perhaps 50% or 51.84%, that most accurately represents the training data.

The three cells below show the process of creating our Logitist regression model, training it on the data, and examining its performance.

First, we define our formula for our Logit regression. In the next cell we create a regression friendly dataframe that sets up boolean values for the categorical variables in our formula and lets our regression model know the types of inputs we're giving it. The model is then instantiated and fitted before a summary of the model's performance is printed. In the last cell we graphically compare the predictions of our model to the actual values we are trying to predict, as well as the residual errors from our model to check for any structure we may have missed.

In [12]:

```
# model formula
# here the ~ sign is an = sign, and the features of our dataset
# are written as a formula to predict survived. The C() lets our
# regression know that those variables are categorical.
# Ref: http://patsy.readthedocs.org/en/latest/formulas.html
formula = 'Survived ~ C(Pclass) + C(Sex) + Age + SibSp  + C(Embarked)'
# create a results dictionary to hold our regression results for easy analysis la
results = {}
```

In [13]:

```
# create a regression friendly dataframe using patsy's dmatrices function
y,x = dmatrices(formula, data=df, return_type='dataframe')

# instantiate our model
model = sm.Logit(y,x)

# fit our model to the training data
res = model.fit()

# save the result for outputing predictions later
results['Logit'] = [res, formula]
res.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.444388
         Iterations 6
```

Out[13]:

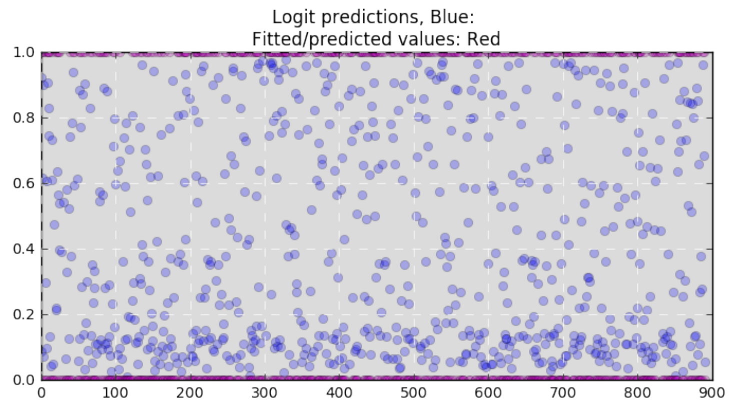
Logit Regression Results

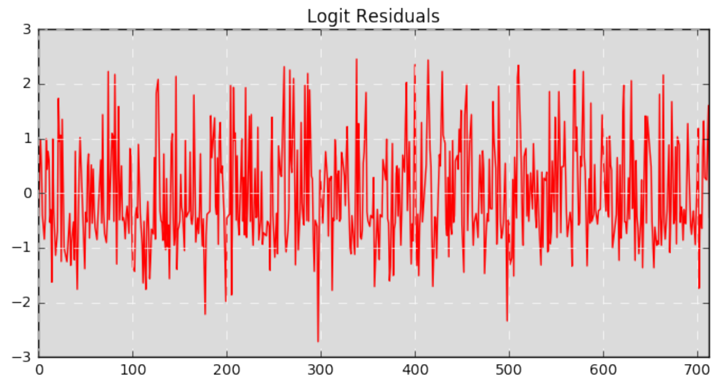
Dep. Variable:	Survived	**No. Observations:**	712
Model:	Logit	**Df Residuals:**	704
Method:	MLE	**Df Model:**	7
Date:	Sun, 04 Feb 2018	**Pseudo R-squ.:**	0.3414
Time:	12:28:56	**Log-Likelihood:**	-316.40
converged:	True	**LL-Null:**	-480.45
		LLR p-value:	5.992e-67

	coef	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
Intercept	4.5423	0.474	9.583	0.000	3.613 5.471
C(Pclass)[T.2]	-1.2673	0.299	-4.245	0.000	-1.852 -0.682
C(Pclass)[T.3]	-2.4966	0.296	-8.422	0.000	-3.078 -1.916
C(Sex)[T.male]	-2.6239	0.218	-12.060	0.000	-3.050 -2.197
C(Embarked)[T.Q]	-0.8351	0.597	-1.398	0.162	-2.006 0.335
C(Embarked)[T.S]	-0.4254	0.271	-1.572	0.116	-0.956 0.105
Age	-0.0436	0.008	-5.264	0.000	-0.060 -0.027
SibSp	-0.3697	0.123	-3.004	0.003	-0.611 -0.129

In [14]:

```
# Plot Predictions Vs Actual
plt.figure(figsize=(18,4));
plt.subplot(121, axisbg="#DBDBDB")
# generate predictions from our fitted model
ypred = res.predict(x)
plt.plot(x.index, ypred, 'bo', x.index, y, 'mo', alpha=.25);
plt.grid(color='white', linestyle='dashed')
plt.title('Logit predictions, Blue: \nFitted/predicted values: Red');

# Residuals
ax2 = plt.subplot(122, axisbg="#DBDBDB")
plt.plot(res.resid_dev, 'r-')
plt.grid(color='white', linestyle='dashed')
ax2.set_xlim(-1, len(res.resid_dev))
plt.title('Logit Residuals');
```





So how well did this work?

Lets look at the predictions we generated graphically:

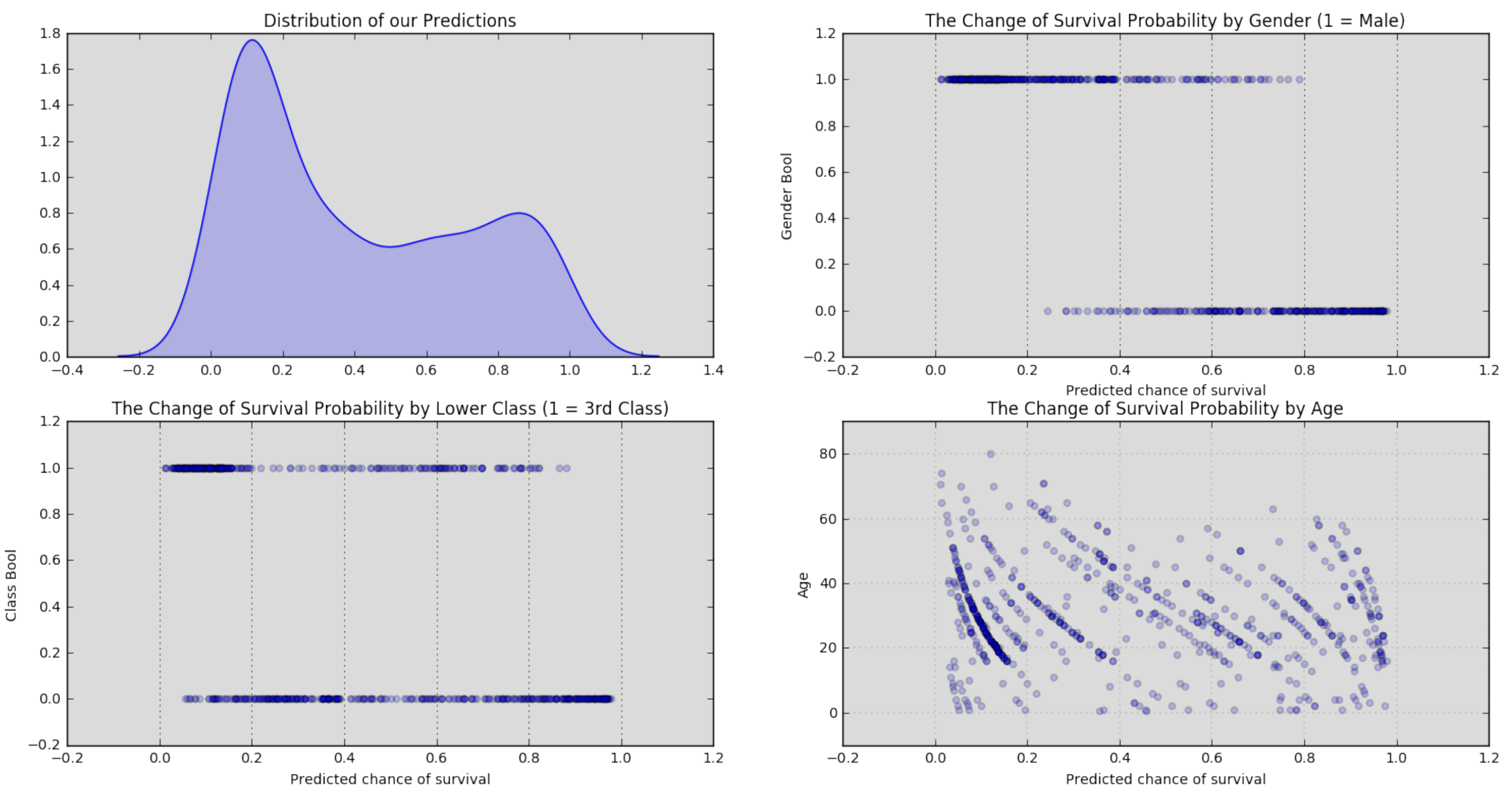
In [15]:

```
fig = plt.figure(figsize=(18,9), dpi=1600)
a = .2

# Below are examples of more advanced plotting.
# It it looks strange check out the tutorial above.
fig.add_subplot(221, axisbg="#DBDBDB")
kde_res = KDEUnivariate(res.predict())
kde_res.fit()
plt.plot(kde_res.support,kde_res.density)
plt.fill_between(kde_res.support,kde_res.density, alpha=a)
plt.title("Distribution of our Predictions")

fig.add_subplot(222, axisbg="#DBDBDB")
plt.scatter(res.predict(),x['C(Sex)[T.male]'] , alpha=a)
plt.grid(b=True, which='major', axis='x')
plt.xlabel("Predicted chance of survival")
plt.ylabel("Gender Bool")
plt.title("The Change of Survival Probability by Gender (1 = Male)")

fig.add_subplot(223, axisbg="#DBDBDB")
plt.scatter(res.predict(),x['C(Pclass)[T.3]'] , alpha=a)
plt.xlabel("Predicted chance of survival")
plt.ylabel("Class Bool")
plt.grid(b=True, which='major', axis='x')
plt.title("The Change of Survival Probability by Lower Class (1 = 3rd Class)")

fig.add_subplot(224, axisbg="#DBDBDB")
plt.scatter(res.predict(),x.Age , alpha=a)
plt.grid(True, linewidth=0.15)
plt.title("The Change of Survival Probability by Age")
plt.xlabel("Predicted chance of survival")
plt.ylabel("Age")
```

```
/Users/pongfactory/anaconda3/lib/python3.5/site-packages/statsmodels
/nonparametric/kdetools.py:20: VisibleDeprecationWarning: using a no
n-integer number instead of an integer will result in an error in th
e future
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```

Out[15]:

```
<matplotlib.text.Text at 0x1264aa748>
```


Distribution of our Predictions
The Change of Survival Probability by Gender (1 = Male)
Gender Bool
Predicted chance of survival
The Change of Survival Probability by Lower Class (1 = 3rd Class)
Class Bool
Predicted chance of survival
The Change of Survival Probability by Age
Age
Predicted chance of survival