Data Science

Yogesh Kulkarni

PCA

Too much of anything is good for nothing!

What happens when a data set has too many variables ? Here are few possible situations which you might come across:

- You find that most of the variables are correlated.
- You lose patience and decide to run a model on whole data. This returns poor accuracy and you feel terrible.
- You become indecisive about what to do
- You start thinking of some strategic method to find few important variables

High Dimensionality

Curse of Dimensionality

- Data-sets typically high dimensional
- Images of 20x20 bitmaps have 400 dimensions. Mega pixel images has far too much.
- In text, all words are features, say, 10^6.

Curse of Dimensionality

- Machine Learning methods are statistical, and data sparse.
- As the Dimensionality grows, fewer observations per region, ie density.
- Actual/True content is in the very small subset of this high dimensional space.
- More dense the space, better for the algorithm.

Dimensionality Reduction Algorithms

- How'd you identify highly significant variable(s) out 1000 or 2000?
- Feature selection: Select only relevant features (based on domain, correlation, etc)
- Feature Extraction: Compose new features based on combination of all existing ones.

Dimensionality Reduction Algorithms

- Principal Components Analysis (PCA) is a Feature Extraction method.
- Projects data from high-dimensional space into a lower-dimensional space

Dimensionality Reduction Example

Body Measurements

Say that we have a data-set of the following measurements from a large set of human volunteers with the following variables:

- height
- weight
- waist size
- shoe size
- length of right arm
- length of left arm

Body Measurements

- length of torso
- hat size
- left hand ring size
- right hand ring size

Technically we have 10 variables, though most of the variation in the data-set can be summarized by at most 2-3 variables.
What can you do? Domain knowledge?

Reduction in Dimensions

In decreasing order of variation, consider the following measurements that can be derived from these 10 variables

- height: captures a large amount of the variation in the total dataset.
- body mass index: should be relatively uncorrelated with overall height, captures much of the next largest variation in the data.
- ratio of torso length to total height: attempts to capture the remaining variation based on how height is distributed over a given individuals frame.

PCA Algorithm

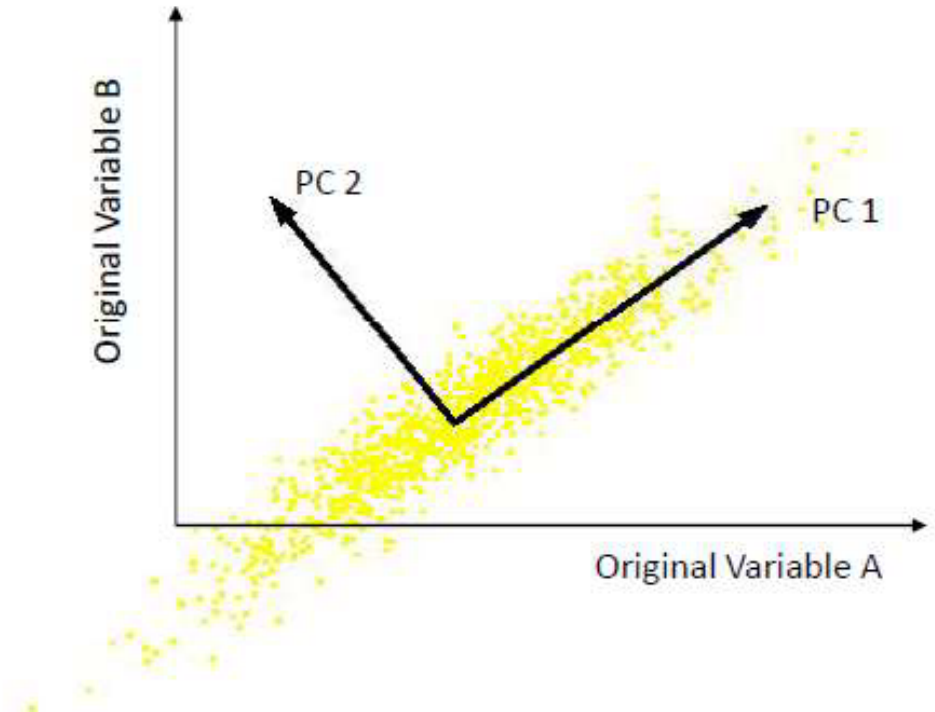
PCA

- One of the oldest (1901!)
- To understand “important” dimensions

PCA

- PCA looks at the data and tries to find a direction-line along which data is spread the most.
- Once done, you look for another (perpendicular) dimension which has second most variation.
- These are called as Principal components.

PCA in a nutshell



- Orthogonal directions of greatest variance in data
- Projections along PC1 discriminate the data most along any one axis

PCA

- First principal component is the direction of greatest variability (co-variance) in the data
- Second is the next orthogonal (uncorrelated) direction of greatest variability. So first remove all the variability along the first component, and then find the next direction of greatest variability
- And so on

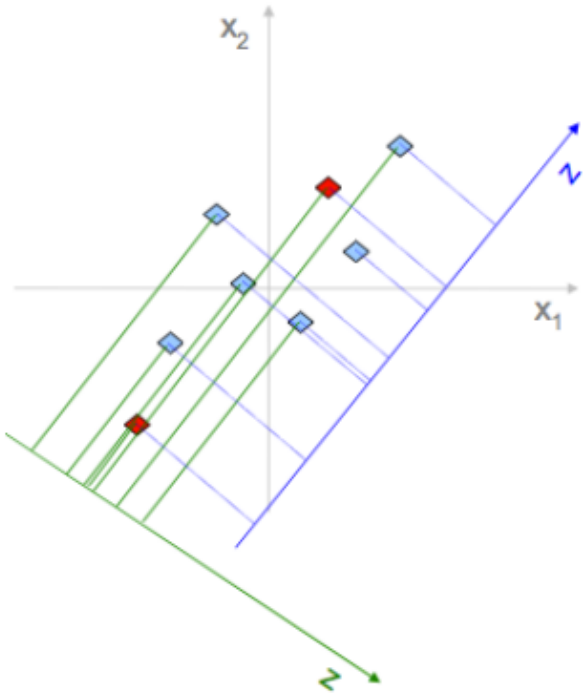
PCA

Principle

- Linear projection method to reduce the number of parameters
- Transfer a set of correlated variables into a new set of uncorrelated variables
- Map the data into a space of lower Dimensionality
- Form of unsupervised learning

Why greatest variability?

- Example: reduce 2-dimensional data to 1
- Data points in 2D x_1, x_2 space are represented by projection points on z axis, so, only 1-D distances.
- See how two red points on different z axes are ad different distances.
- The one with max variance preserves original intent-structure.



PCA in Steps

PCA in Steps

- Center the data by subtracting mean.
- Subtracting the mean makes variance and covariance calculation easier by simplifying their equations.
- The variance and co-variance values are not affected by the mean value.
- Compute Co-variance matrix.

PCA in Steps

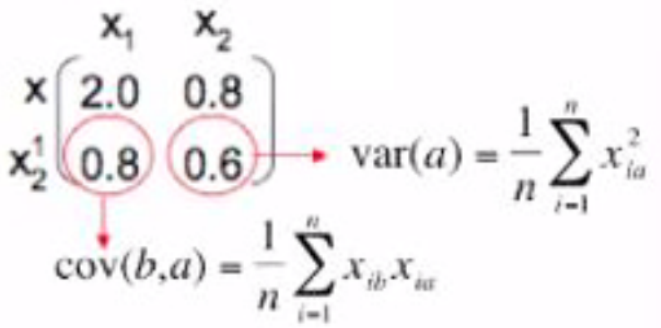
"Center" the data at zero: $x_{i,a} = x_{i,a} - \mu$

– subtract mean from each attribute

Compute covariance matrix Σ

– covariance of dimensions x_1 and x_2:

- do x_1 and x_2 tend to increase together?
- or does x_2 decrease as x_1 increases?



(Reference: Principal Component Analysis - Victor Lavrenko)

PCA in Steps

First way

- Initialize seed vector
- Successive multiplication with Co-variance matrix
- Transforms the vector
- Finally, the Principal Components.

Multiply a vector by Σ: $\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} -1 \\ +1 \end{pmatrix} \rightarrow \begin{pmatrix} -1.2 \\ -0.2 \end{pmatrix}$ again $\rightarrow \begin{pmatrix} -2.5 \\ -1.0 \end{pmatrix} \rightarrow \begin{pmatrix} -6.0 \\ -2.7 \end{pmatrix} \rightarrow \begin{pmatrix} -14.1 \\ -6.4 \end{pmatrix} \rightarrow \begin{pmatrix} -33.3 \\ -15.1 \end{pmatrix}$

– turns towards direction of variance

slope: 0.400 0.450 0.454 0.454

PCA in Steps

Another way

- Calculate the eigen vectors and eigenvalues of the co-variance matrix (Manually, Python, Matlab, etc)
- Since the non-diagonal elements in this co-variance matrix are positive, we should expect that both the x and y variable increase together.

Want vectors **e** which aren't turned: Σ **e** = λ **e**

– e ... eigenvectors of Σ, λ ... corresponding eigenvalues

– principal components = eigenvectors w. largest eigenvalues

PCA in Steps

- Retain top Principal components
- Project original X data on them

PCA Example Working

PCA Example in Steps

Subtract the mean

- From each of the data dimensions.
- All the x values have their $\bar{x}$ subtracted and y values have $\bar{y}$ subtracted from them.
- This produces a data set whose mean is zero.
- Subtracting the mean makes variance and covariance calculation easier by simplifying their equations.
- The variance and co-variance values are not affected by the mean value.

PCA Example in Steps

DATA:

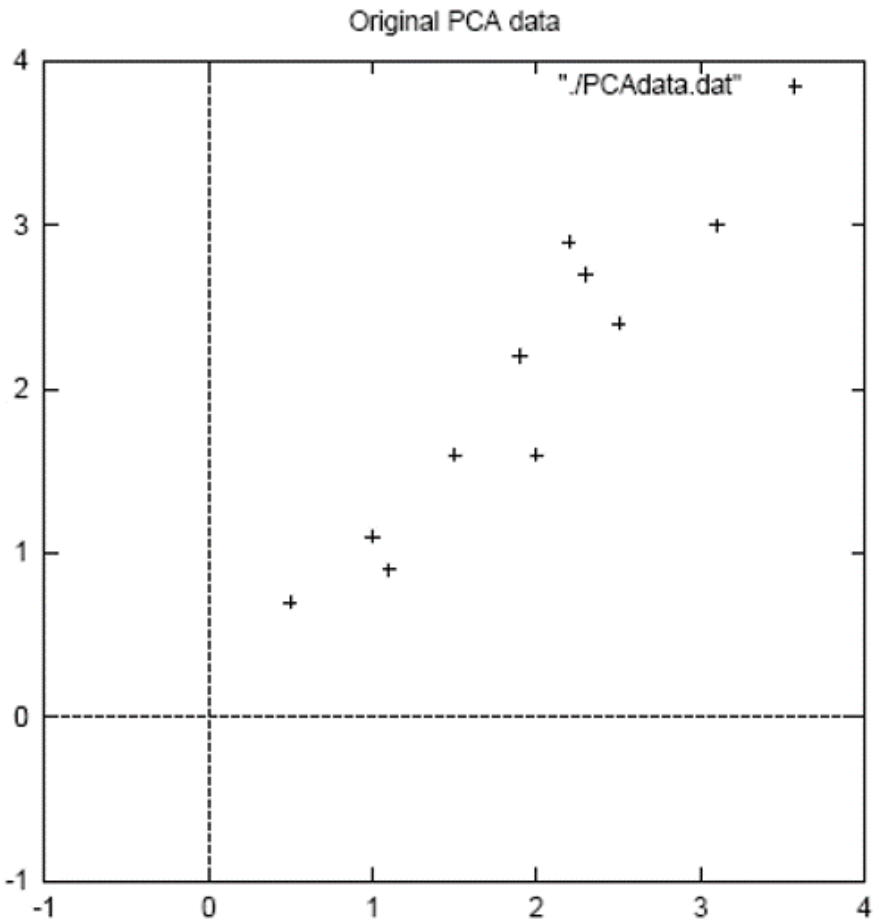
x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

ZERO MEAN DATA:

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

PCA Example in Steps

Original Data



PCA Example in Steps

- Calculate the co-variance matrix (Manually, Python, Matlab, etc)

$$\text{cov} = \begin{bmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{bmatrix}$$

- Since the non-diagonal elements in this co-variance matrix are positive, we should expect that both the x and y variable increase together.

PCA Example in Steps

- Calculate the eigen-vectors and eigenvalues of the co-variance (Manually, Python, Matlab, etc)

```
eigenvalues = .0490833989,  1.28402771
eigenvectors =      -.735178656   -.677873399,
                    .677873399  -.735178656
```

- Note they are perpendicular to each other

PCA Example in Steps

- Highest eigenvalue is the principle component.
- Here, it is pointed down the middle of the data.
- Order eigen vectors by eigenvalue, highest to lowest.
- The order of significance.
- Keep important ones.
- Small eigenvalues ignorable

PCA Example in Steps

$$NewData = RowFeatureVector_{nf \times nf} \times RowZeroMeanData_{nRows \times nftransposed}$$

- RowFeatureVector: Matrix with eigen-vectors as rows, first on top
- RowZeroMeanData is the mean-adjusted data transposed

PCA Example in Steps

DATA:

x	y
-.827970186	-.175115307
1.77758033	.142857227
-.992197494	.384374989
-.274210416	.130417207
-1.67580142	-.209498461
-.912949103	.175282444
.0991094375	-.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-.162675287

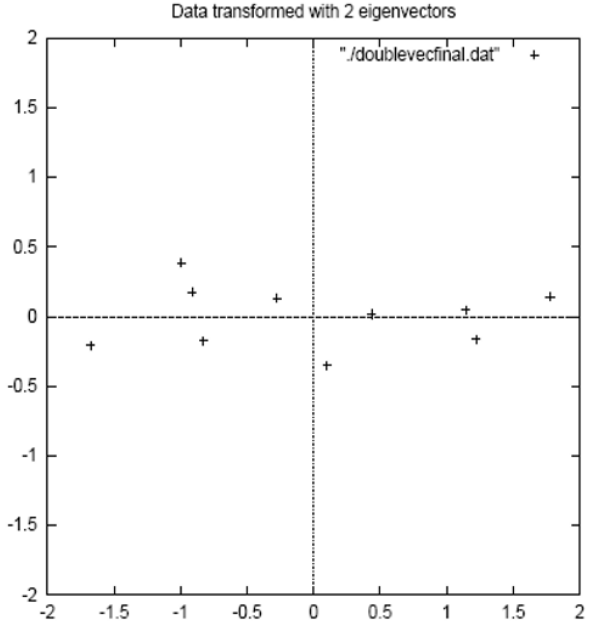
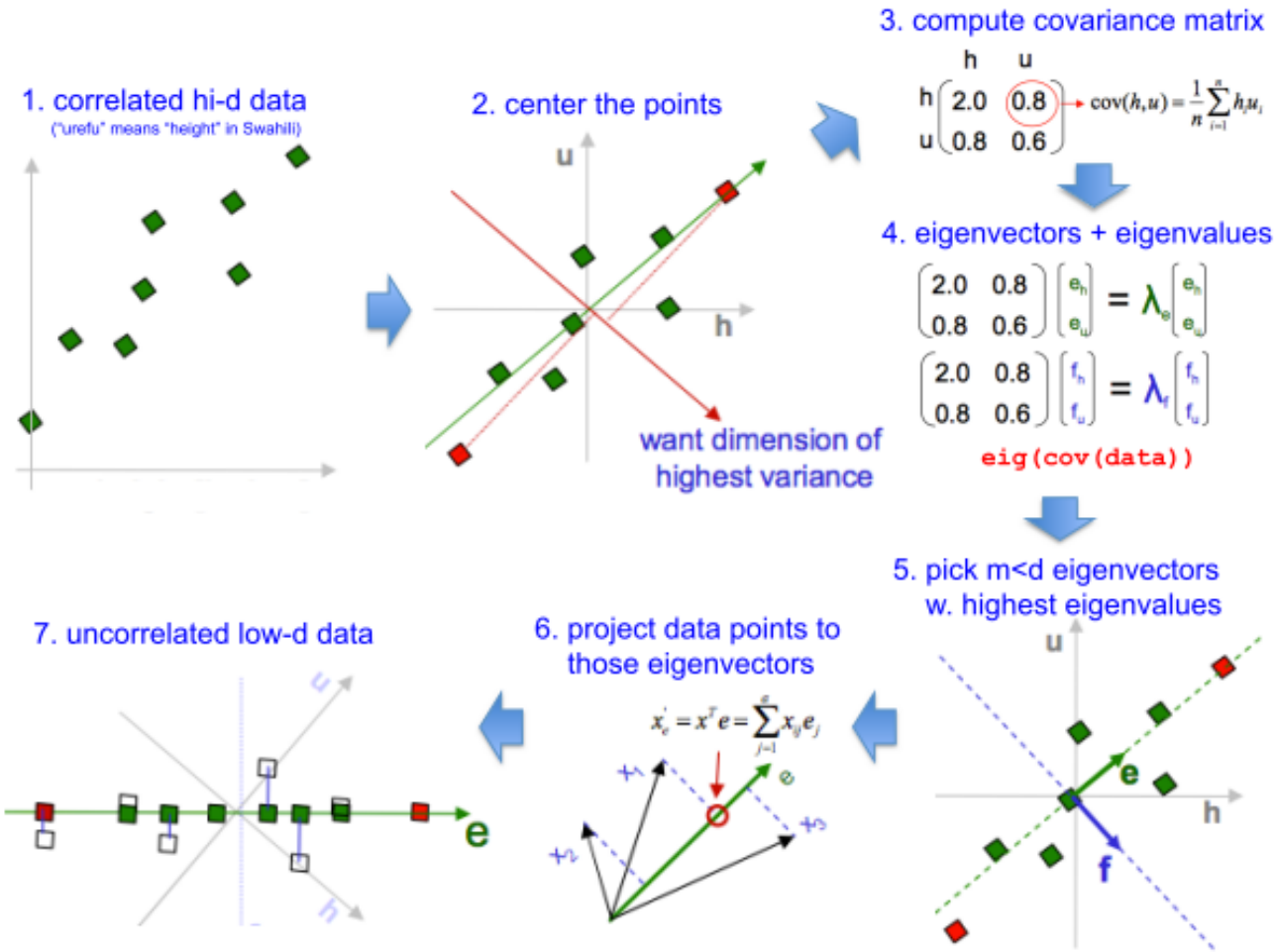


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

PCA in a nutshell

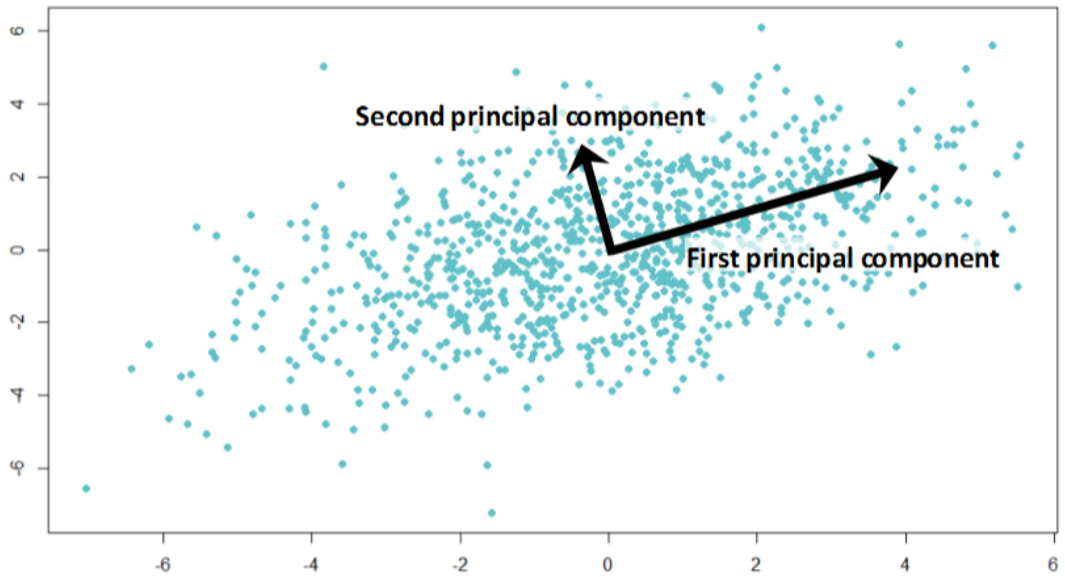


PCA

- A principal component is a normalized linear combination of the original predictors in a data set.
- The first principal component can be written as:
  $Z^1 = w^{11}X^1 + w^{21}X^2 + \ldots + w^{p1}X^p$
- It captures the maximum variance in the data set.
- It determines the direction of highest variability in the data.
- Larger the variability captured in first component, larger the information captured by component.
- No other component can have variability higher than first principal component.

PCA

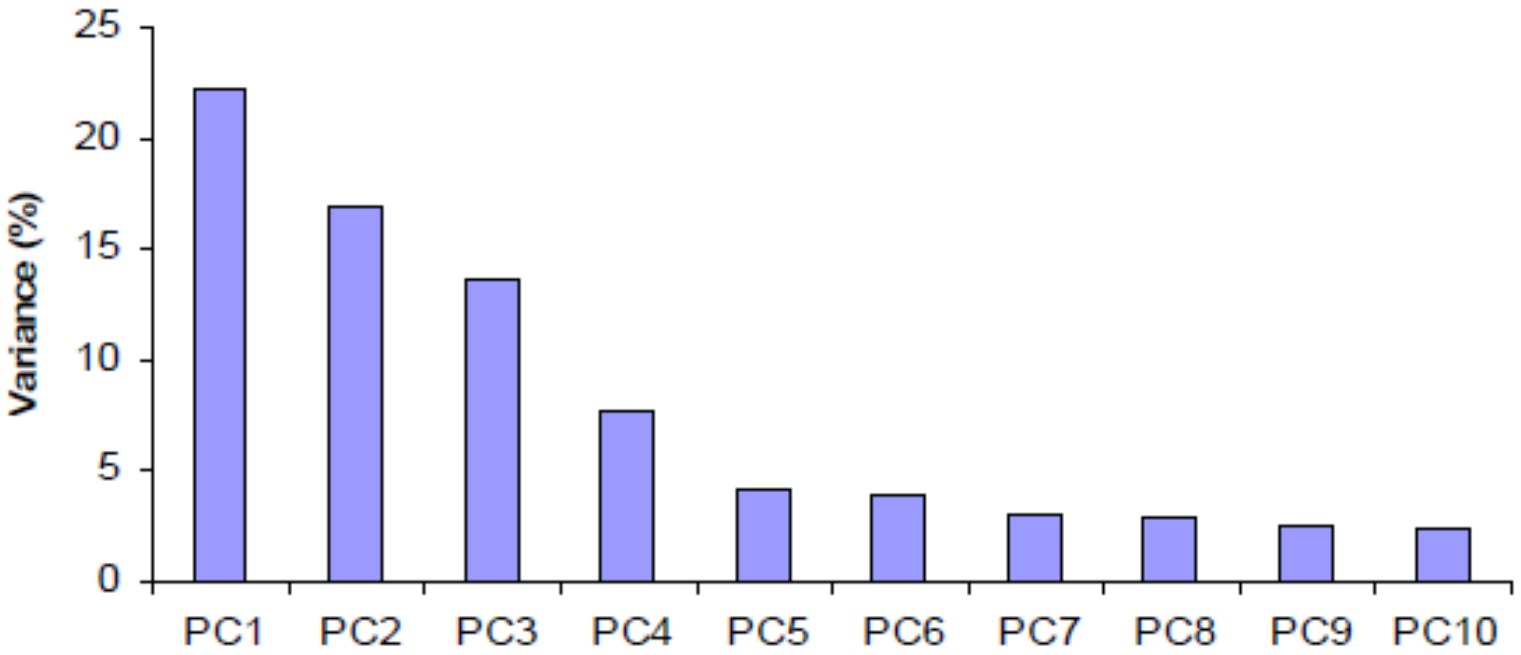
- The first principal component results in a line which is closest to the data i.e. it minimizes the sum of squared distance between a data point and the line.
- Similarly, we can compute the second principal component also.
  $Z^2 = w^{12}X^1 + w^{22}X^2 + \ldots + w^{p2}X^p$
- If the two components are uncorrelated, their directions should be orthogonal



- All succeeding principal component capture the remaining variation without being correlated with the previous component.

PCA

Can ignore the components of lesser significance.



You do lose some information, but if the eigenvalues are small, you don't lose much.

Principal Component Analysis

Advantages

- Useful Pre-processing step
- Reduces computational complexity
- Noise reduction

Principal Component Analysis

Limitations

- Linear manifold only
- Co-variance Matrix huge, finding eigen-vectors is slow
- SVD comes to rescue

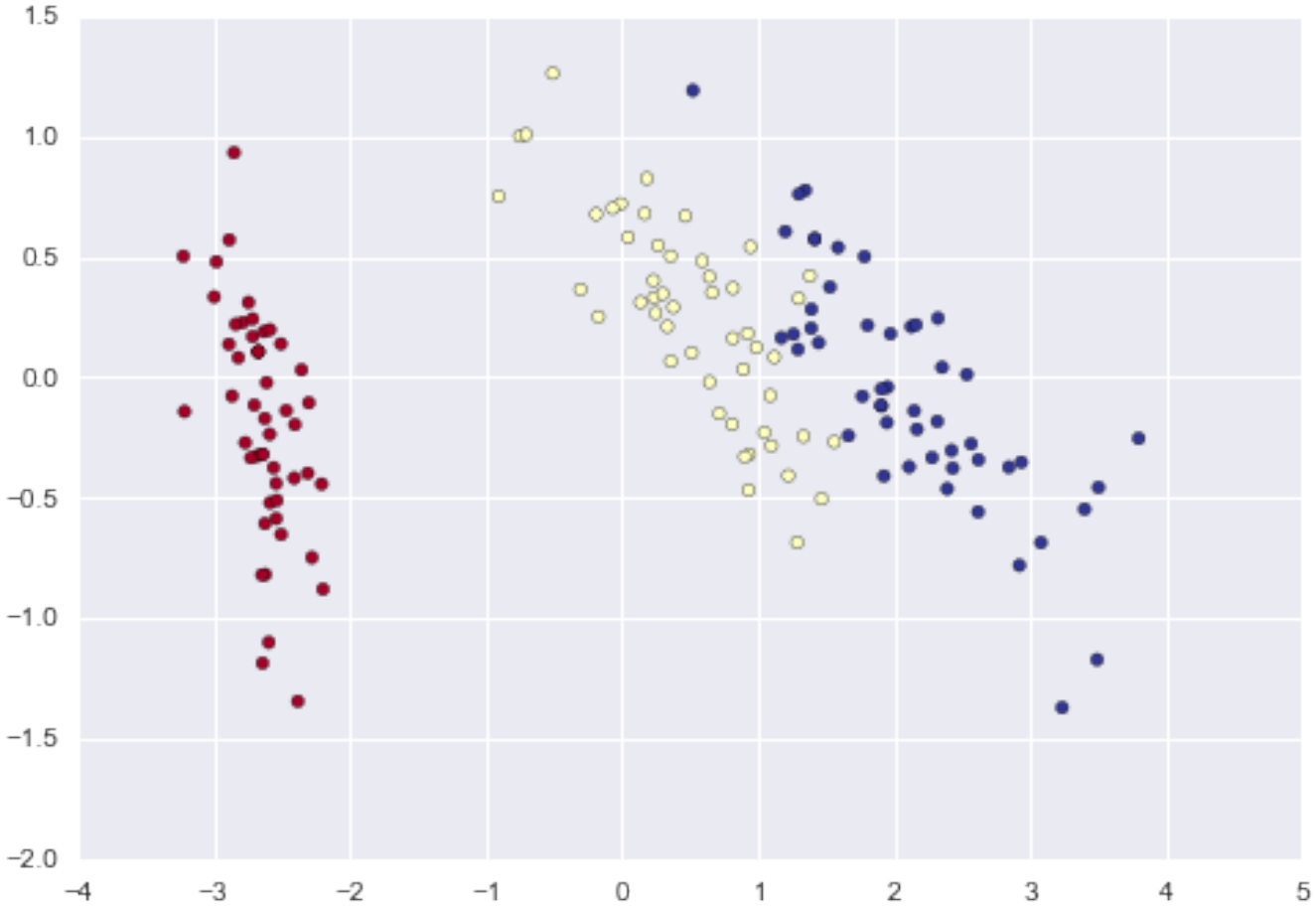
PCA with Scikit-Learn

PCA

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn; seaborn.set()
from sklearn import datasets
from sklearn.decomposition import PCA
import pylab as pl

iris = datasets.load_iris()
X, y = iris.data, iris.target

pca = PCA(n_components=2)
pca.fit(X)
X_reduced = pca.transform(X)
print("Reduced dataset shape:", X_reduced.shape)

pl.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y,
           cmap='RdYlBu')

print("Meaning of the 2 components:")
for component in pca.components_:
    print(" + ".join("%.3f x %s" % (value, name)
                     for value, name in zip(component,
                                            iris.feature_names)))
```

(Ref: Understanding PCA - Part 2 (High-dimensional data) - Karan Rajwanshi)

PCA Plotting



(Ref: Understanding PCA - Part 2 (High-dimensional data) - Karan Rajwanshi)

Case Study: Titanic Disaster

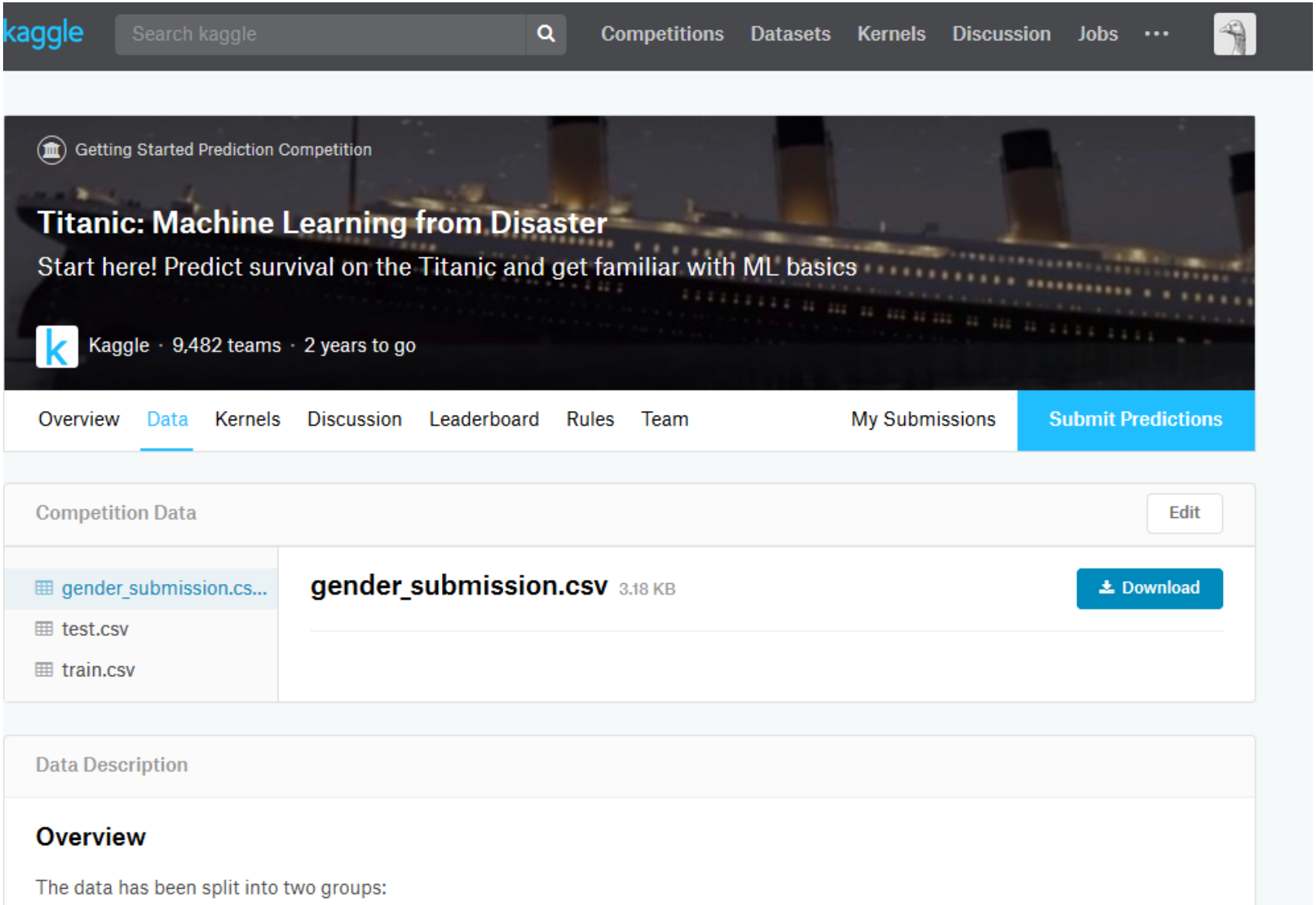
The Disaster

The Titanic competition

- The sinking of the Titanic is one of the most infamous shipwrecks in history.
- It sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew.
- One of the reasons for loss of life was that there were not enough lifeboats for the passengers and crew.
- Some groups of people were more likely to survive than others, such as women, children, and the upper-class.

Kaggle

Kaggle Competition



What is Kaggle?

- Kaggle is a site where people create algorithms and compete against machine learning practitioners around the world.
- Your algorithm wins the competition if it's the most accurate on a particular data set.
- Kaggle is a fun way to practice your machine learning skills.
- Indian counterpart is "Analytics Vidhya".
- Many competitions and good prizes.

The Titanic competition

- Kaggle has created a number of competitions designed for beginners.
- One of the most popular of these competitions is Titanic competition
- In this competition, we have a data set of different information about passengers onboard the Titanic, and we see if we can use that information to predict whether those people survived or not.

The Titanic competition

- Each Kaggle competition has two key data files that you will work with - a training set and a testing set.
- The training set contains data we can use to train our model.
- It has a number of feature columns which contain various descriptive data, as well as a column of the target values we are trying to predict: in this case, Survival.
- The testing set contains all of the same feature columns, but is missing the target value column. Additionally, the testing set usually has fewer observations (rows) than the training set.

Data

Data Dictionary

Below are the descriptions contained in that data dictionary

- PassengerID: A column added by Kaggle to identify each row
- Survived: Passenger survived or not (0=No, 1=Yes)
- Pclass: The class of the ticket (1=1st, 2=2nd, 3=3rd)
- Sex: The passenger's sex
- Age: The passenger's age in years
- SibSp: The number of siblings or spouses
- Parch: The number of parents or children
- Ticket: The passenger's ticket number
- Fare: The fare the passenger paid
- Cabin: The passenger's cabin number
- Embarked: The port where the passenger embarked (C=Cherbourg, Q=Queenstown, S=Southampton)

Load the data

```
import pandas as pd

test = pd.read_csv("titanic_test.csv")
train = pd.read_csv("titanic_train.csv")

print("Dimensions of train: {}".format(train.shape))
print("Dimensions of test: {}".format(test.shape))

Dimensions of train: (891, 12)
Dimensions of test: (418, 11)
```

Data Exploration

Exploring the data

Let's take a look at the first few rows of the train dataframe.

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S

Exploring the data

Let's take a look at the last few rows of the train dataframe.

```
1 train.tail()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32	0	0	370376	7.75	NaN	Q

Column Data Types

df_train.dtypes

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
```

- Type 'object' is a string for pandas, which poses problems with machine learning algorithms.
- If we want to use these as features, we'll need to convert these to number representations.

Dataframe information

df_train.info()

```
Data columns (total 12 columns):
PassengerId     891 non-null int64
Survived        891 non-null int64
Pclass          891 non-null int64
Name            891 non-null object
Sex             891 non-null object
Age             714 non-null float64
SibSp           891 non-null int64
Parch           891 non-null int64
Ticket          891 non-null object
Fare            891 non-null float64
Cabin           204 non-null object
Embarked        889 non-null object
```

- Age, Cabin, and Embarked are missing values.
- Cabin has too many missing values.

Descriptive Statistics

df_train.describe()

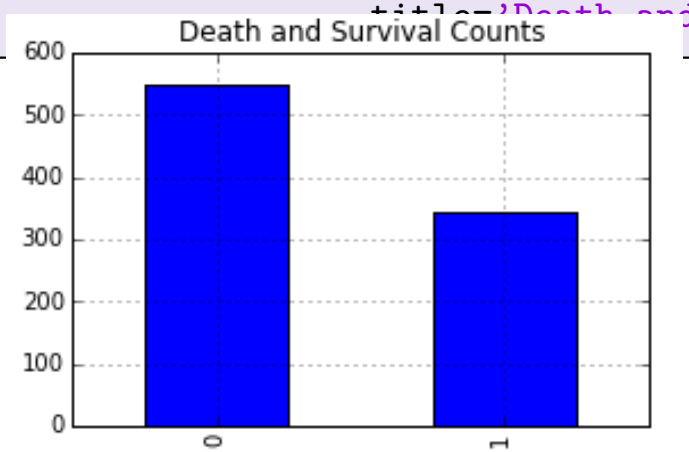
- Age, Cabin, and Embarked are missing values.
- Cabin has too many missing values.

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Plot Features

Plot death and survival counts

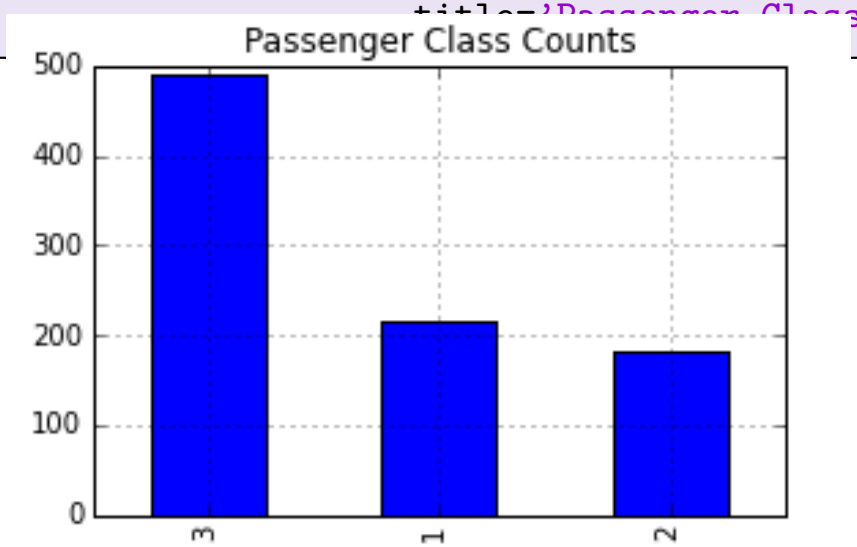
```
# Set up a grid of plots
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (3, 2)

plt.subplot2grid(fig_dims, (0, 0))
df_train['Survived'].value_counts().plot(kind='bar',
                                         title='Death and Survival Counts')
```



Plot Features

Plot Pclass counts

```
plt.subplot2grid(fig_dims, (0, 1))
df_train['Pclass'].value_counts().plot(kind='bar',
                                       title='Passenger Class Counts')
```



Plot Features

Plot Sex counts

```
plt.subplot2grid(fig_dims, (1, 0))
df_train['Sex'].value_counts().plot(kind='bar',
                                    title='Gender Counts')
plt.xticks(rotation=0)
```

Plot Embarked counts

```
plt.subplot2grid(fig_dims, (1, 1))
df_train['Embarked'].value_counts().plot(kind='bar',
                                         title='Ports of Embarkation Counts')
```

Plot the Age histogram

```
plt.subplot2grid(fig_dims, (2, 0))
df_train['Age'].hist()
plt.title('Age Histogram')
```

Cross Tab

We'll determine which proportion of passengers survived based on their passenger class.. Generate a cross tab of Pclass and Survived:

```
pclass_xt = pd.crosstab(df_train['Pclass'], df_train['Survived'])
pclass_xt
```

Survived	0	1
Pclass		
1	80	136
2	97	87
3	372	119

Feature Engineering

Generate a mapping of Sex from a string to a number representation:

```
sexes = sorted(df_train['Sex'].unique())
genders_mapping = dict(zip(sexes, range(0, len(sexes) + 1)))
genders_mapping

{'female': 0, 'male': 1}
```

Feature Engineering

Transform Sex from a string to a number representation:

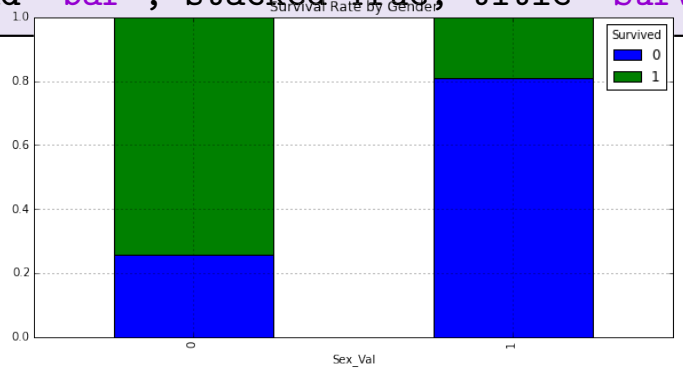
```
df_train['Sex_Val'] = df_train['Sex'].map(genders_mapping).astype(int)
df_train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Sex_Val
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C	0
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S	0
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S	1

Exploring the data

Plot a normalized cross tab for Sex_Val and Survived:

```
sex_val_xt = pd.crosstab(df_train['Sex_Val'], df_train['Survived'])
sex_val_xt_pct = sex_val_xt.div(sex_val_xt.sum(1).astype(float), axis=0)
sex_val_xt_pct.plot(kind='bar', stacked=True, title='Survival Rate by Gender')
```



The majority of females survived, but not males.

Exploring the data

Any insights by looking at both Sex and Pclass? Count males and females in each Pclass:

```
passenger_classes = sorted(df_train['Pclass'].unique())

for p_class in passenger_classes:
    print 'M: ', p_class, len(df_train[(df_train['Sex'] == 'male') &
                              (df_train['Pclass'] == p_class)])
    print 'F: ', p_class, len(df_train[(df_train['Sex'] == 'female') &
                              (df_train['Pclass'] == p_class)])

M:  1 122
F:  1 94
M:  2 108
F:  2 76
M:  3 347
F:  3 144
```

Exploring the data

Plot survival rate by Sex:

```
# Plot survival rate by Sex
females_df = df_train[df_train['Sex'] == 'female']
females_xt = pd.crosstab(females_df['Pclass'], df_train['Survived'])
females_xt_pct = females_xt.div(females_xt.sum(1).astype(float), axis=0)
females_xt_pct.plot(kind='bar',  stacked=True,
                    title='Female Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')
```

Exploring the data

Plot survival rate by Pclass:

```
# Plot survival rate by Pclass
males_df = df_train[df_train['Sex'] == 'male']
males_xt = pd.crosstab(males_df['Pclass'], df_train['Survived'])
males_xt_pct = males_xt.div(males_xt.sum(1).astype(float), axis=0)
males_xt_pct.plot(kind='bar',    stacked=True,
                  title='Male Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')
```

The vast majority of females in First and Second class survived. Males in First class had the highest chance for survival.

Cleaning the data

The Embarked column might be an important feature but it is missing a couple data points which might pose a problem for machine learning algorithms:

```
df_train[df_train['Embarked'].isnull()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Sex_Val
61	62	1	1	Icard, Miss. Amelie	female	38	0	0	113572	80	B28	NaN	0
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62	0	0	113572	80	B28	NaN	0

Cleaning the data

Prepare to map Embarked from a string to a number representation:

```
# Get the unique values of Embarked
embarked_locs = sorted(df_train['Embarked'].unique())

embarked_locs_mapping = dict(zip(embarked_locs,
                                 range(0, len(embarked_locs) + 1)))
embarked_locs_mapping

{nan: 0, 'C': 1, 'Q': 2, 'S': 3}
```

Cleaning the data

Transform Embarked from a string to a number representation to prepare it for machine learning algorithms:

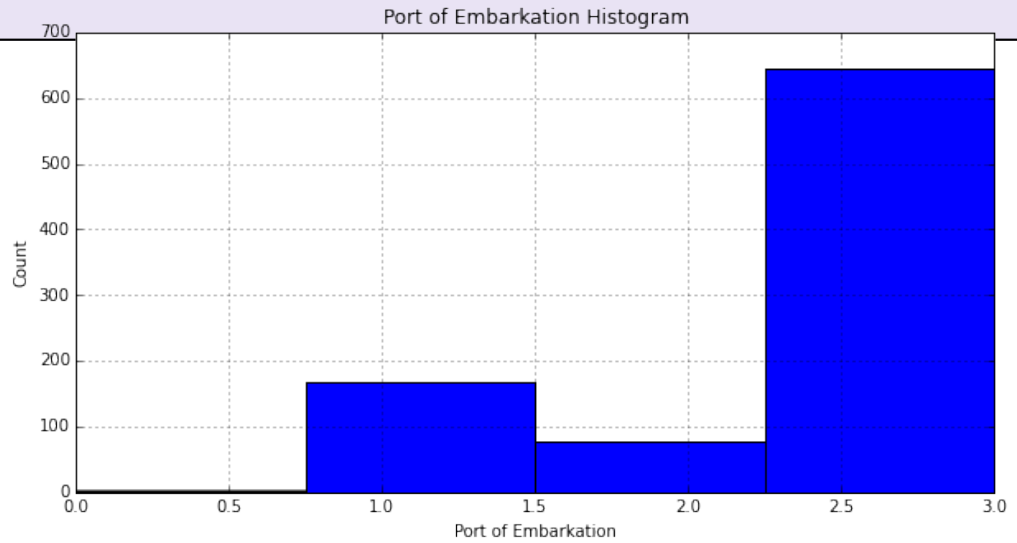
```
df_train['Embarked_Val'] = df_train['Embarked'] \
                           .map(embarked_locs_mapping) \
                           .astype(int)

df_train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Sex_Val
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C	0
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S	0
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S	1

Cleaning the data

Plot the histogram for Embarked_Val:

```
df_train['Embarked_Val'].hist(bins=len(embarked_locs), range=(0, 3))
plt.title('Port of Embarkation Histogram')
plt.xlabel('Port of Embarkation')
plt.ylabel('Count')
plt.show()
```



Cleaning the data

Since the vast majority of passengers embarked in 'S': 3, we assign the missing values in Embarked to 'S':

```
if len(df_train[df_train['Embarked'].isnull()] > 0):
    df_train.replace({'Embarked_Val' :
                    { embarked_locs_mapping[nan] : embarked_locs_mapping['S']
                    }
                },
                inplace=True)
```

Cleaning the data

Verify we do not have any more NaNs for Embarked_Val:

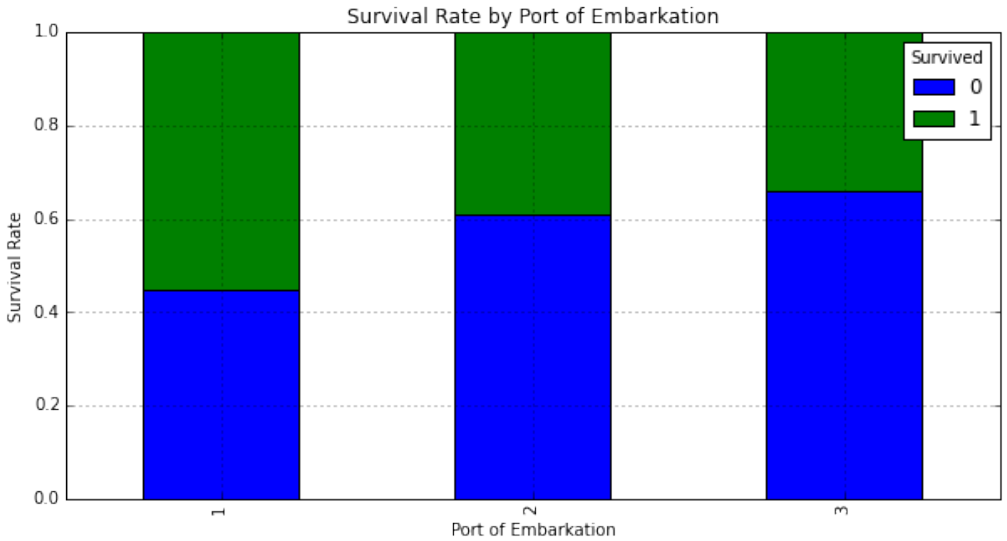
```
embarked_locs = sorted(df_train['Embarked_Val'].unique())
embarked_locs

array([1, 2, 3])
```

Exploring the data

Plot a normalized cross tab for Embarked_Val and Survived:

```
embarked_val_xt = pd.crosstab(df_train['Embarked_Val'], df_train['Survived'])
embarked_val_xt_pct = \
    embarked_val_xt.div(embarked_val_xt.sum(1).astype(float), axis=0)
embarked_val_xt_pct.plot(kind='bar', stacked=True)
plt.title('Survival Rate by Port of Embarkation')
plt.xlabel('Port of Embarkation')
plt.ylabel('Survival Rate')
```

Exploring the data



It appears those that embarked in location 'C': 1 had the highest rate of survival.

Feature Engineering

The Age column seems like an important feature–unfortunately it is missing many values. Filter to view missing Age values:

```
1 df_train[df_train['Age'].isnull()][['Sex','Pclass','Age']].head()
```

	Sex	Pclass	Age
5	male	3	NaN
17	male	2	NaN
19	female	3	NaN
26	male	3	NaN
28	female	3	NaN

Feature Engineering

Determine the Age typical for each passenger class by Sex_Val. We'll use the median instead of the mean because the Age histogram seems to be right skewed.

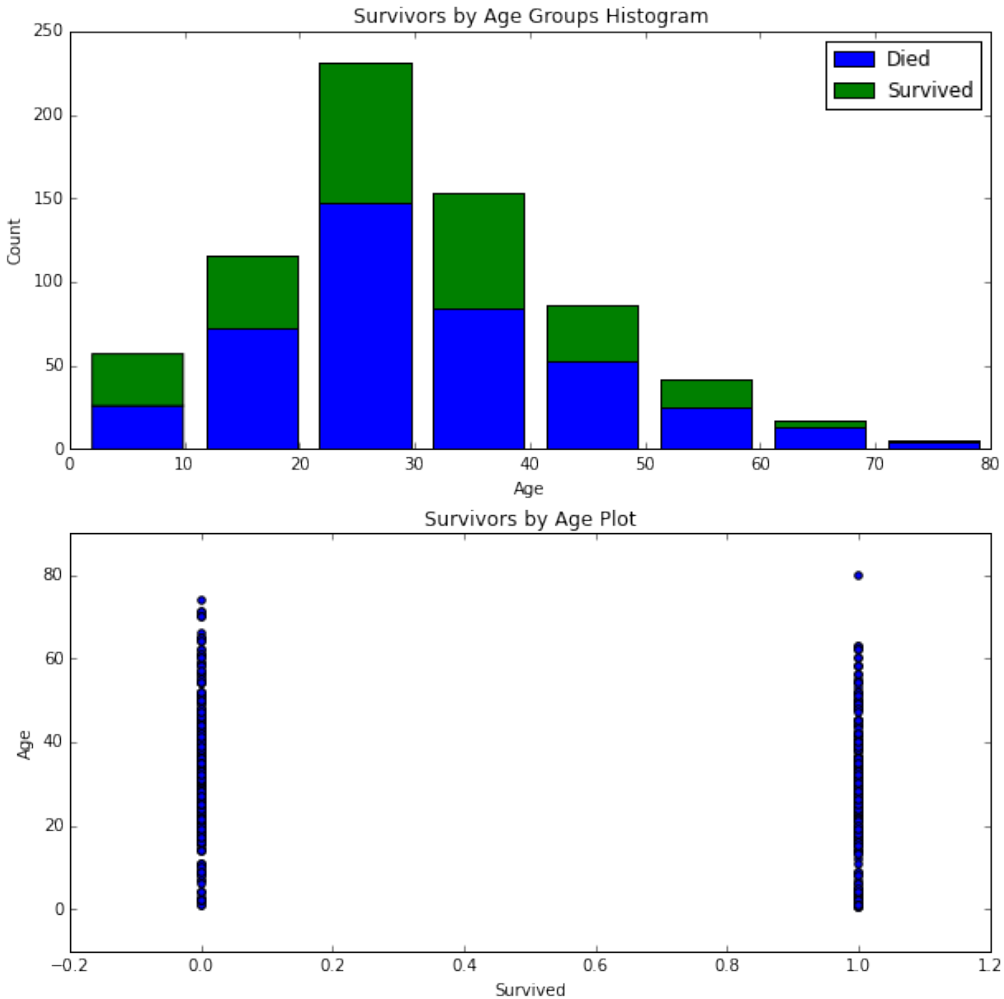
```
# To keep Age in tact, make a copy of it called AgeFill
# that we will use to fill in the missing ages:
df_train['AgeFill'] = df_train['Age']

df_train['AgeFill'] = df_train['AgeFill'] \
                      .groupby([df_train['Sex_Val'], df_train['Pclass']]) \
                      .apply(lambda x: x.fillna(x.median()))
```

Feature Engineering

Plot a normalized cross tab for AgeFill and Survived:

```
# Set up a grid of plots
fig, axes = plt.subplots(2, 1, figsize=fizsize_with_subplots)

# Histogram of AgeFill segmented by Survived
df1 = df_train[df_train['Survived'] == 0]['Age']
df2 = df_train[df_train['Survived'] == 1]['Age']
max_age = max(df_train['AgeFill'])
axes[0].hist([df1, df2],
             bins=max_age / bin_size,
             range=(1, max_age),
             stacked=True)
axes[0].legend(('Died', 'Survived'), loc='best')
axes[0].set_title('Survivors by Age Groups Histogram')
axes[0].set_xlabel('Age')
axes[0].set_ylabel('Count')
```

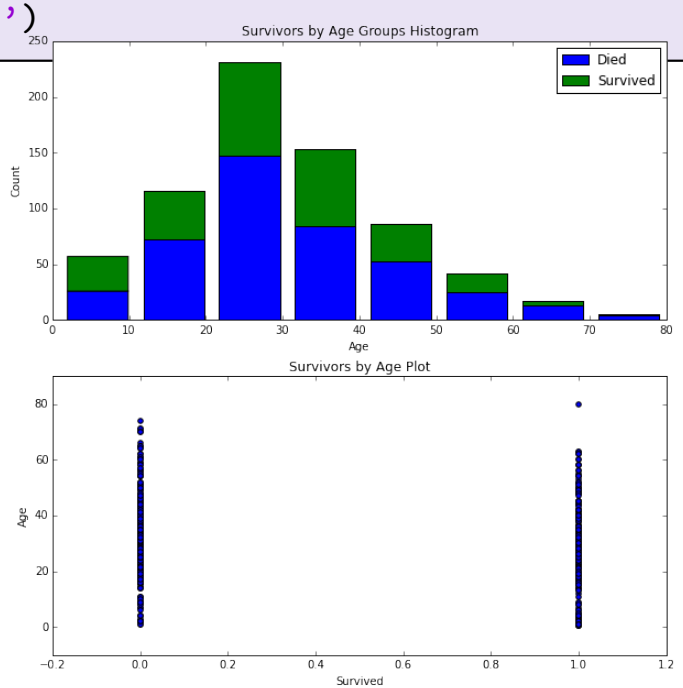
Feature Engineering



Feature Engineering

Plot a normalized cross tab for AgeFill and Survived:

```
# Scatter plot Survived and AgeFill
axes[1].scatter(df_train['Survived'], df_train['AgeFill'])
axes[1].set_title('Survivors by Age Plot')
axes[1].set_xlabel('Survived')
axes[1].set_ylabel('Age')
```

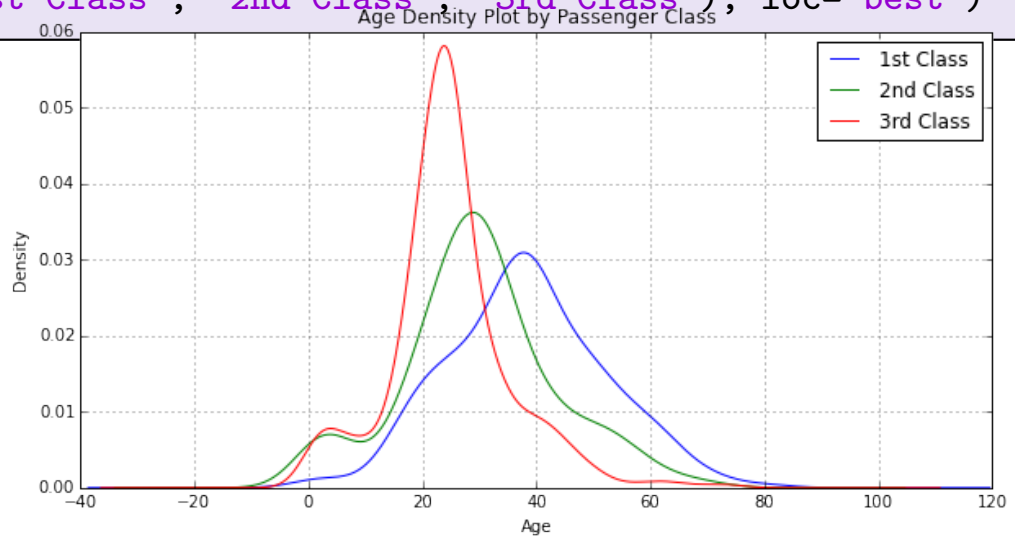


Unfortunately, the graphs above do not seem to clearly show any insights.

Feature Engineering

Plot AgeFill density by Pclass:

```
for pclass in passenger_classes:
    df_train.AgeFill[df_train.Pclass == pclass].plot(kind='kde')
plt.title('Age Density Plot by Passenger Class')
plt.xlabel('Age')
plt.legend(('1st Class', '2nd Class', '3rd Class'), loc='best')
```



Feature Engineering

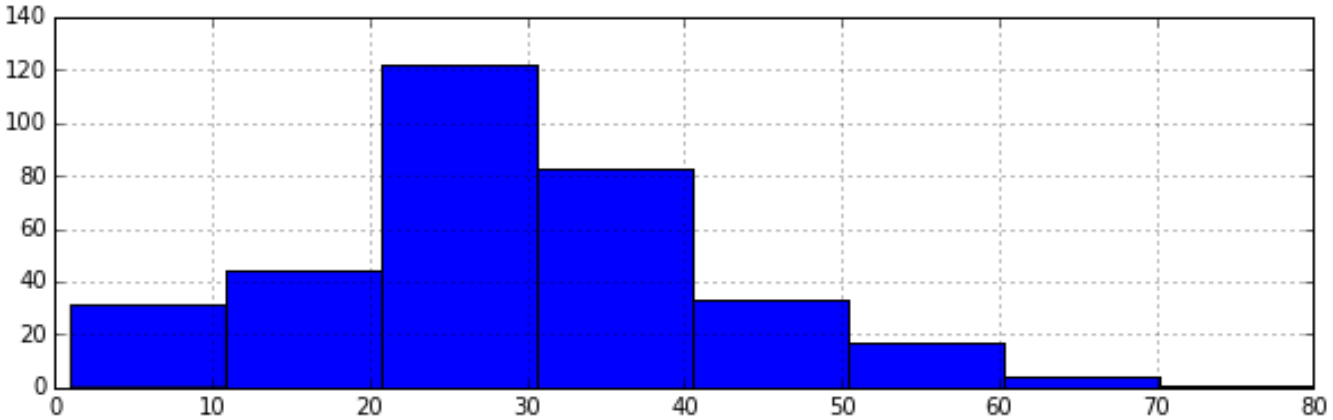
- The first class passengers were generally older then second class passengers, which in turn were older than third class passengers.
- We've determined that first class passengers had a higher survival rate than second class passengers, which in turn had a higher survival rate than third class passengers.

Feature Engineering

Plots

```
# Set up a grid of plots
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (3, 1)

# Plot the AgeFill histogram for Survivors
plt.subplot2grid(fig_dims, (0, 0))
survived_df = df_train[df_train['Survived'] == 1]
survived_df['AgeFill'].hist(bins=max_age / bin_size, range=(1, max_age))
```

Feature Engineering



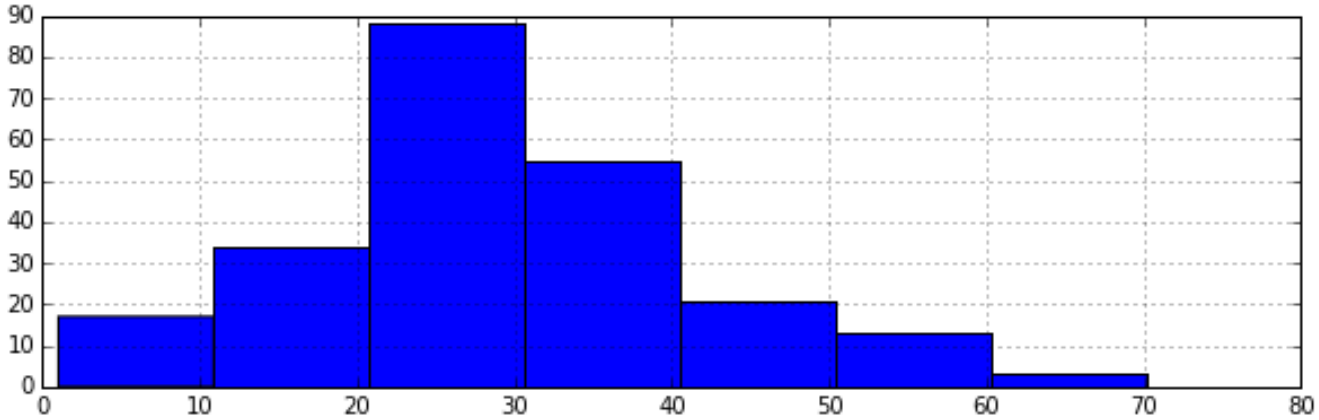
We see that most survivors come from the 20's to 30's age ranges and might be explained by the following two graphs.

Feature Engineering

Plots

```
# Plot the AgeFill histogram for Females
plt.subplot2grid(fig_dims, (1, 0))
females_df = df_train[(df_train['Sex_Val'] == 0) & (df_train['Survived'] == 1)]
females_df['AgeFill'].hist(bins=max_age / bin_size, range=(1, max_age))
```

Feature Engineering

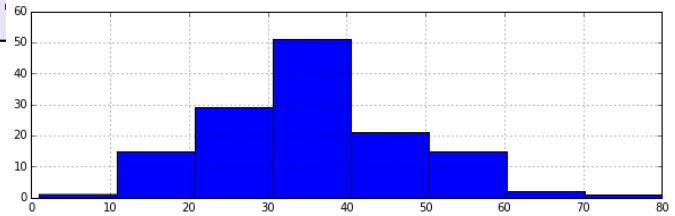


Shows most females are within their 20's.

Feature Engineering

Plots

```
# Plot the AgeFill histogram for first class passengers
plt.subplot2grid(fig_dims, (2, 0))
class1_df = df_train[(df_train['Pclass'] == 1) & (df_train['Survived'] == 1)]
class1_df['AgeFill'].hist(bins=max_age / bin_size, range=(1, max_age))
```



Shows most first class passengers are within their 30's.

Feature Engineering

Define a new feature FamilySize that is the sum of Parch (number of parents or children on board) and SibSp (number of siblings or spouses):

```
df_train['FamilySize'] = df_train['SibSp'] + df_train['Parch']
df_train.head()
```

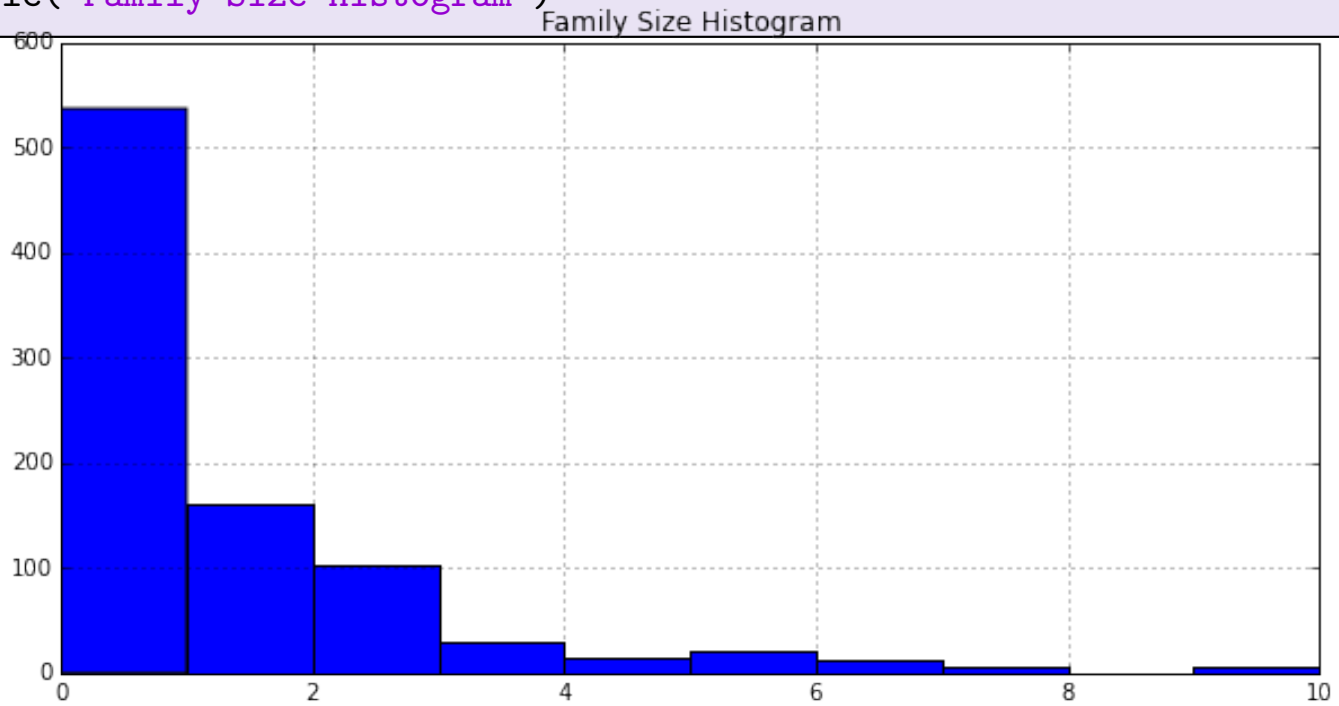
Feature Engineering

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Sex_Val	I
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S	1	:
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C	0	'
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S	0	:
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S	0	:
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S	1	:

Feature Engineering

Plot a histogram of FamilySize:

```
df_train['FamilySize'].hist()
plt.title('Family Size Histogram')
```

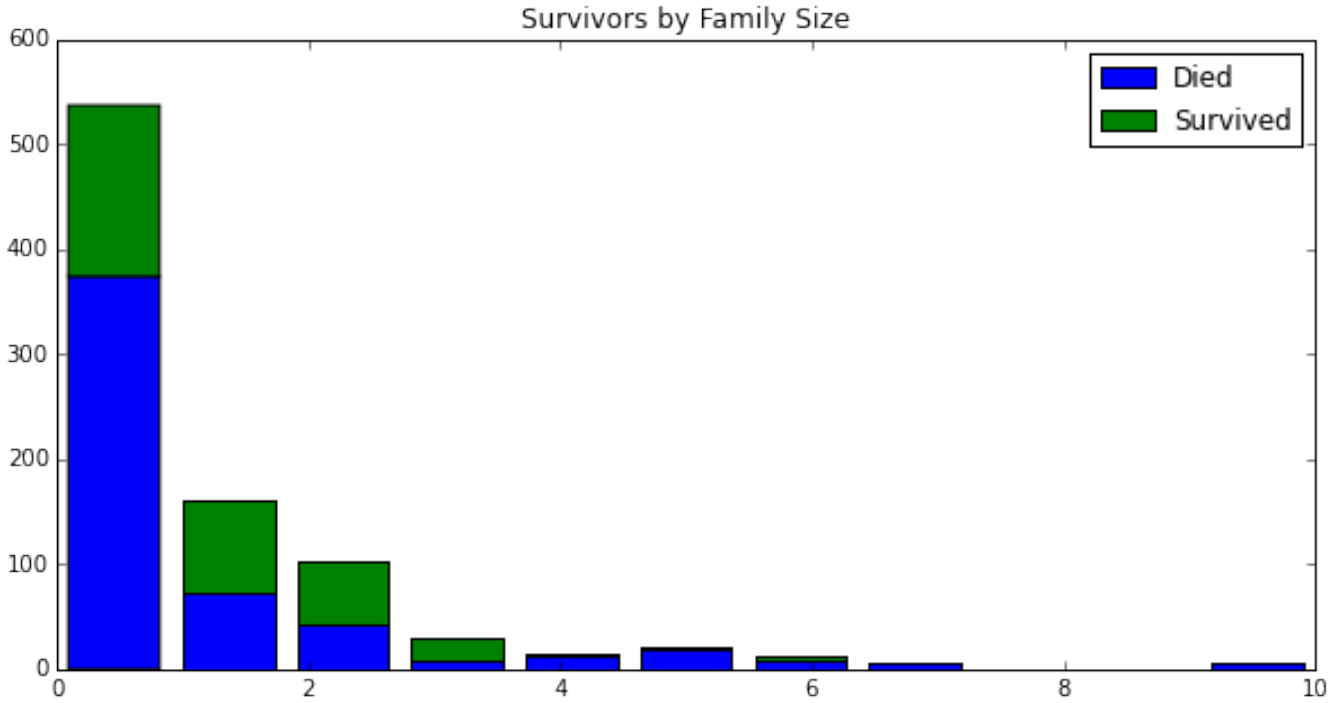


Feature Engineering

Plot a histogram of AgeFill segmented by Survived:

```
family_sizes = sorted(df_train['FamilySize'].unique())
family_size_max = max(family_sizes)
df1 = df_train[df_train['Survived'] == 0]['FamilySize']
df2 = df_train[df_train['Survived'] == 1]['FamilySize']
plt.hist([df1, df2],
         bins=family_size_max + 1,
         range=(0, family_size_max),
         stacked=True)
plt.legend(('Died', 'Survived'), loc='best')
plt.title('Survivors by Family Size')
```

Feature Engineering



It is not immediately obvious what impact FamilySize has on survival.

Final Data Preparation for Machine Learning

Many machine learning algorithms do not work on strings and they usually require the data to be in an array, not a DataFrame. Show only the columns of type 'object' (strings):

```
df_train.dtypes[df_train.dtypes.map(lambda x: x == 'object')]

Name        object
Sex         object
Ticket      object
Cabin       object
Embarked    object
dtype: object
```

Final Data Preparation for Machine Learning

Drop the columns we won't use:

```
df_train = df_train.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'],
                         axis=1)
```

Final Data Preparation for Machine Learning

Drop the columns we won't use:

- The Age column since we will be using the AgeFill column instead.
- The SibSp and Parch columns since we will be using FamilySize instead.
- The PassengerId column since it won't be used as a feature.
- The Embarked_Val as we decided to use dummy variables instead.

```
df_train = df_train.drop(['Age', 'SibSp', 'Parch', 'PassengerId',
     'Embarked_Val'], axis=1)
```

Final Data Preparation for Machine Learning

Convert the DataFrame to a numpy array:

```
train_data = df_train.values
train_data

array([[  0.    ,   3.    ,   7.25  , ...,   1.    ,  22.    ,   1.    ],
       [  1.    ,   1.    ,  71.2833, ...,   0.    ,  38.    ,   1.    ],
       [  1.    ,   3.    ,   7.925 , ...,   1.    ,  26.    ,   0.    ],
       ...,
       [  0.    ,   3.    ,  23.45  , ...,   1.    ,  21.5   ,   3.    ],
       [  1.    ,   1.    ,  30.    , ...,   0.    ,  26.    ,   0.    ],
       [  0.    ,   3.    ,   7.75  , ...,   0.    ,  32.    ,   0.    ]])
```

Modeling

Random Forest: Training

Create the random forest object:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100)
```

Random Forest: Training

Fit the training data and create the decision trees:

```
# Training data features, skip the first column 'Survived'
train_features = train_data[:, 1:]

# 'Survived' column values
train_target = train_data[:, 0]

# Fit the model to our training data
clf = clf.fit(train_features, train_target)
score = clf.score(train_features, train_target)
"Mean accuracy of Random Forest: {0}".format(score)

'Mean accuracy of Random Forest: 0.980920314254'
```

Predicting

Random Forest: Predicting

Read the test data:

```
df_test = pd.read_csv('../data/titanic/test.csv')
df_test.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Note the test data does not contain the column 'Survived', we'll use our trained model to predict these values.

Random Forest: Predicting

Clean the test data similar to training data:

```
# Data wrangle the test set and convert it to a numpy array
df_test = clean_data(df_test, drop_passenger_id=False)
test_data = df_test.values

# Get the test data features, skipping the first column 'PassengerId'
test_x = test_data[:, 1:]

# Predict the Survival values for the test data
test_y = clf.predict(test_x)
```

Kaggle Submission

Random Forest: Prepare for Kaggle Submission

Create a DataFrame by combining the index from the test data with the output of predictions, then write the results to the output:

```
df_test['Survived'] = test_y
df_test[['PassengerId', 'Survived']] \
    .to_csv('../data/titanic/results-rf.csv', index=False)
```

Evaluate Model Accuracy

Get an idea of accuracy before submitting to Kaggle. We'll split our training data, 80% will go to "train" and 20% will go to "test":

```
from sklearn import metrics
from sklearn.cross_validation import train_test_split

train_x, test_x, train_y, test_y = train_test_split(train_features,
                                                    train_target,
                                                    test_size=0.20,
                                                    random_state=0)
print (train_features.shape, train_target.shape)
print (train_x.shape, train_y.shape)
print (test_x.shape, test_y.shape)
((891, 8), (891,))
((712, 8), (712,))
((179, 8), (179,))
```

Evaluate Model Accuracy

Use the new training data to fit the model, predict, and get the accuracy score:

```
clf = clf.fit(train_x, train_y)
predict_y = clf.predict(test_x)

from sklearn.metrics import accuracy_score
print ("Accuracy = %.2f" % (accuracy_score(test_y, predict_y)))

Accuracy = 0.83
```

Evaluate Model Accuracy

View the Confusion Matrix:

```
confusion_matrix = metrics.confusion_matrix(test_y, predict_y)


print ("           Predicted")
print ("         |  0  |  1  |")
print ("         |-----|-----|")
print ("       0 | %3d | %3d |" % (confusion_matrix[0, 0],
                                    confusion_matrix[0, 1]))
print ("Actual   |-----|-----|")
print ("       1 | %3d | %3d |" % (confusion_matrix[1, 0],
                                    confusion_matrix[1, 1]))
print ("         |-----|-----|")
```

Evaluate Model Accuracy

Display the classification report:

```
from sklearn.metrics import classification_report
print(classification_report(test_y,
                            predict_y,
                            target_names=['Not Survived', 'Survived']))

              precision    recall  f1-score   support
Not Survived       0.84      0.89      0.86       110
    Survived       0.81      0.72      0.76        69
 avg / total       0.83      0.83      0.82       179
```

Next?

To improve accuracy

Feature Preparation, Selection, and Engineering

- How to determine which features in your model are the most-relevant to your predictions
- Ways to reduce the number of features used to train your model and avoid over-fitting
- Techniques to create new features to improve the accuracy of your model

To improve accuracy

Model Selection and Tuning

- How other classification algorithms work
- About hyper-parameters, and how to select the hyper-parameters that give the best predictions
- How to compare different algorithms to improve the accuracy of your predictions

Thanks . . .

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com