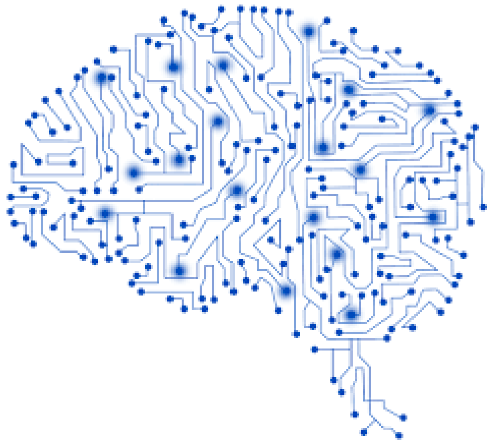
University of Minho
School of Engineering

ISLab



Dados e Aprendizagem Automática

Regressão Linear e Logística

DAA @ MEI-1º/MiEI-4º/MMC-1º – 1º Semestre

Bruno Fernandes, Filipe Gonçalves, Víctor Alves, Cesar Analide

Part IV

Contents

- Linear Regression
- Logistic Regression
- Hands On

Linear Regression

Linear Regression

- Exercise:
  - **Problem:** Development of a Machine Learning Model able to predict house prices for regions in the USA
  - **Regression Approach:** Linear Regression approach to solve this problem
  - Dataset: table with information regarding houses info. in regions of the United States, containing:
    - 'Avg. Area Income': Avg. Income of residents of the city house is located in.
    - 'Avg. Area House Age': Avg Age of Houses in same city
    - 'Avg. Area Number of Rooms': Avg Number of Rooms for Houses in same city
    - 'Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in same city
    - 'Area Population': Population of city house is located in
    - 'Price': Price that the house sold at
    - 'Address': Address for the house

	A	B	C	D	E	F	G
1	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
2	79545.45857	5.682861322	7.009188143	4.09	23086.8005	1059033.558	208
3	79248.64245	6.002899808	6.730821019	3.09	40173.07217	1505890.915	188
4	61287.06718	5.86588984	8.51272743	5.13	36882.1594	1058987.988	9127
5	63345.24005	7.188236095	5.586728665	3.26	34310.24283	1260616.807	USS
6	59982.19723	5.040554523	7.839387785	4.23	26354.10947	630943.4893	USNS

Linear Regression

Check out the data

We've been able to get some data from your neighbor for housing prices as a csv set, let's get our environment ready with the libraries we'll need and then import the data!

Import Libraries

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

Check out the Data

```
USAhousing = pd.read_csv('USA_Housing.csv')
```

Linear Regression


```
USAhousing.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

```
USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
Avg. Area Income                5000 non-null float64
Avg. Area House Age             5000 non-null float64
Avg. Area Number of Rooms       5000 non-null float64
Avg. Area Number of Bedrooms    5000 non-null float64
Area Population                 5000 non-null float64
Price                           5000 non-null float64
Address                         5000 non-null object
dtypes: float64(6), object(1)
memory usage: 273.5+ KB
```

```
USAhousing.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

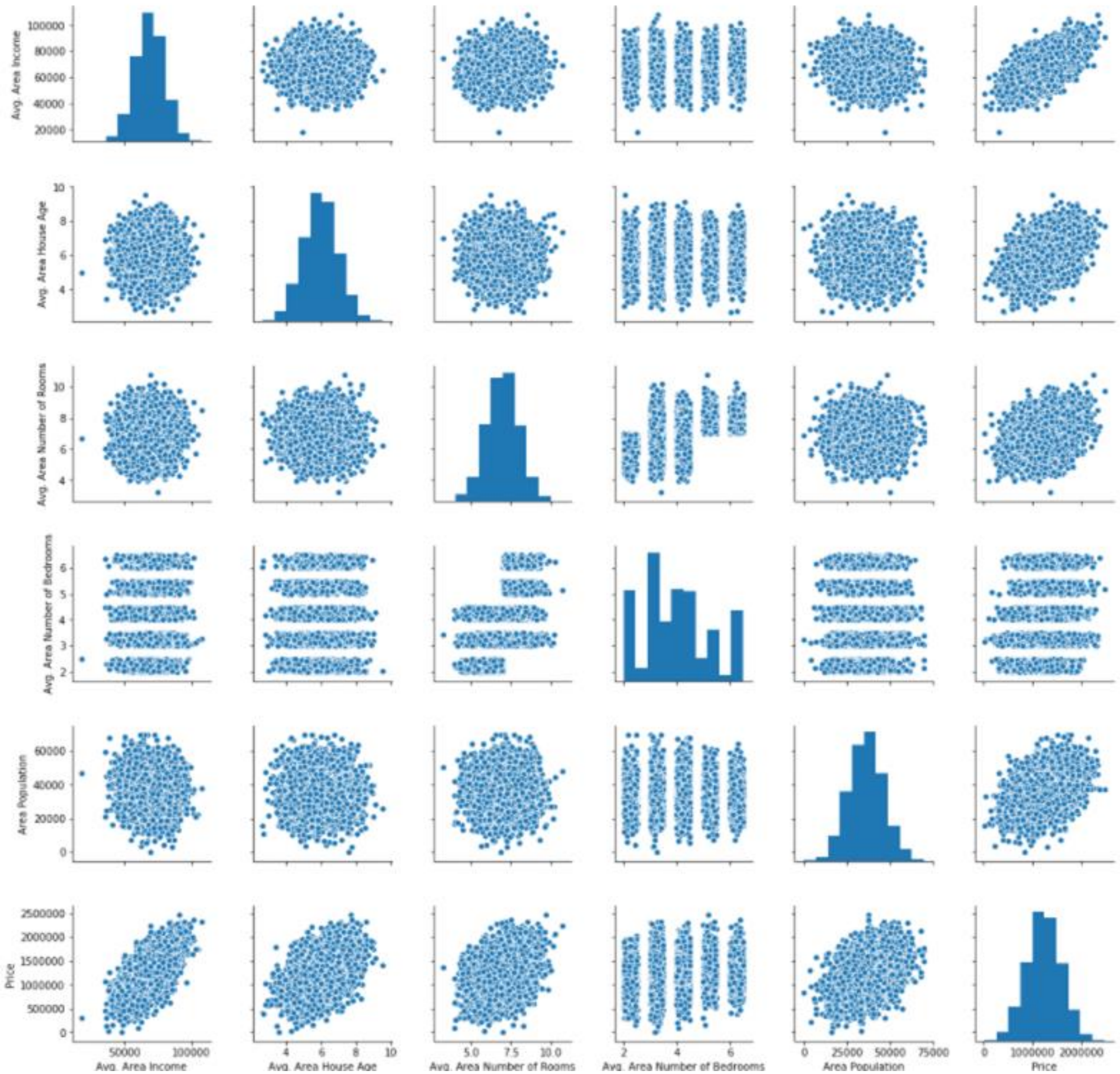
```
USAhousing.columns
```

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Linear Regression

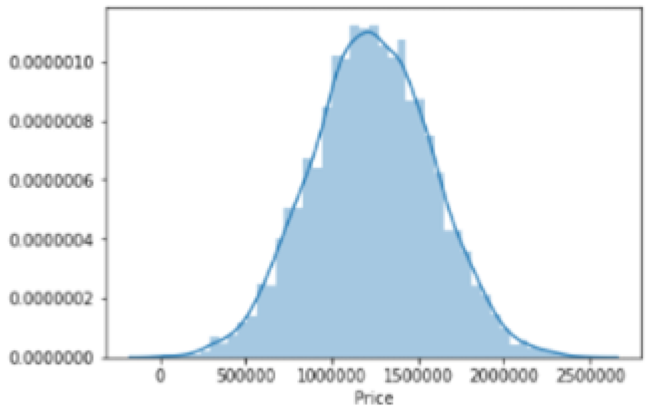

```
sns.pairplot(USAhousing)
```

```
<seaborn.axisgrid.PairGrid at 0x2972ad12b38>
```




```
sns.distplot(USAhousing['Price'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2972eb2ea90>
```




```
sns.heatmap(USAhousing.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2972f437240>
```

Linear Regression

Training a Linear Regression Model

Let's now begin to train out regression model! We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the Price column. We will toss out the Address column because it only has text info that the linear regression model can't use.

X and y arrays

```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
               'Avg. Area Number of Bedrooms', 'Area Population']]
y = USAhousing['Price']
```

Train Test Split

Now let's split the data into a training set and a testing set. We will train out model on the training set and then use the test set to evaluate the model.

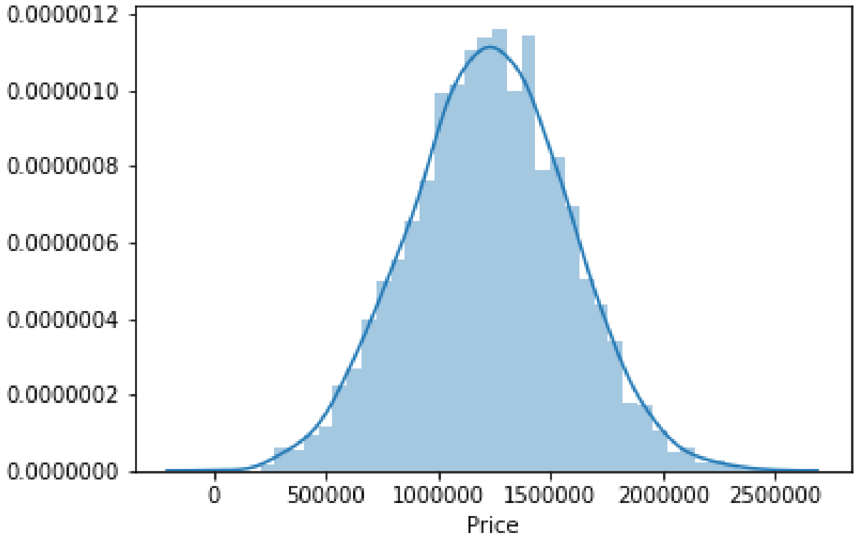
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
```

Linear Regression

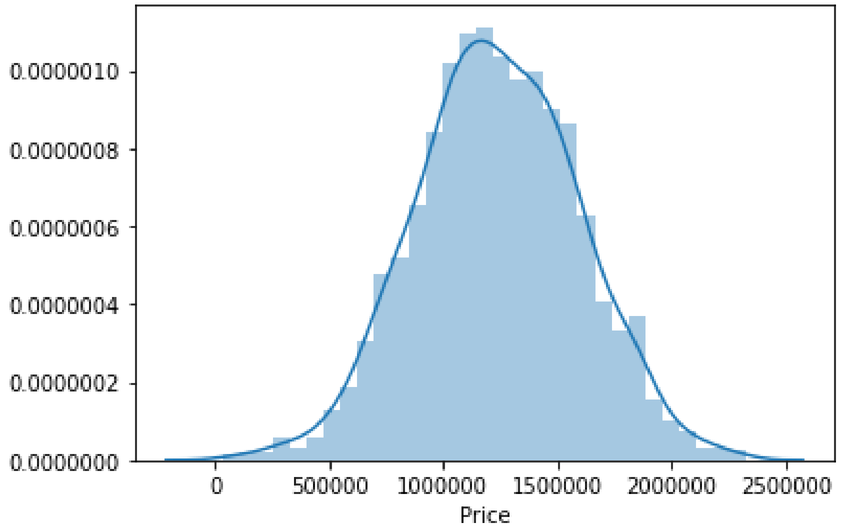

```
sns.distplot(y_train)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1c6b394ed48>




```
sns.distplot(y_test)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1c6b3ed28c8>



Linear Regression

Creating and Training the Model

```
from sklearn.linear_model import LinearRegression
```

```
lm = LinearRegression()
```

```
lm.fit(X_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Model Evaluation

Let's evaluate the model by checking out it's coefficients and how we can interpret them.

```
# print the intercept
print(lm.intercept_)
```

```
-2640159.79685
```

```
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

Interpreting the coefficients:

- Holding all other features fixed, a 1 unit increase in **Avg. Area Income** is associated with an *increase of $21.52 *.
- Holding all other features fixed, a 1 unit increase in **Avg. Area House Age** is associated with an *increase of $164883.28 *.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Rooms** is associated with an *increase of $122368.67 *.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Bedrooms** is associated with an *increase of $2233.80 *.
- Holding all other features fixed, a 1 unit increase in **Area Population** is associated with an *increase of $15.15 *.

Linear Regression

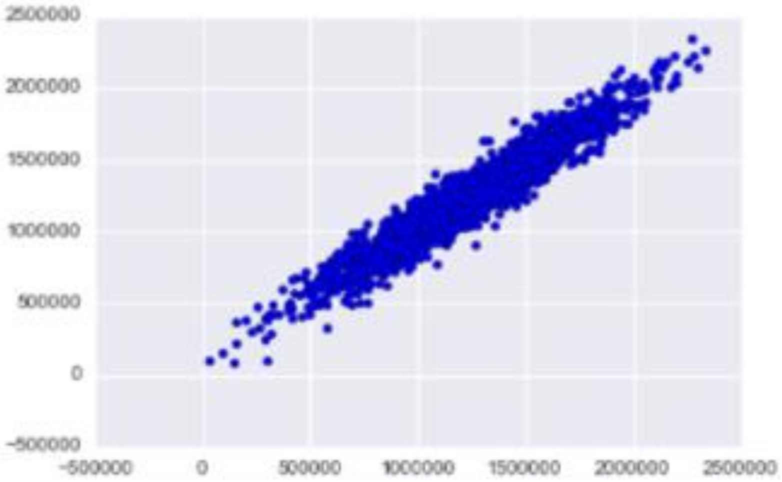
Predictions from our Model

Let's grab predictions off our test set and see how well it did!

```
predictions = lm.predict(X_test)
```

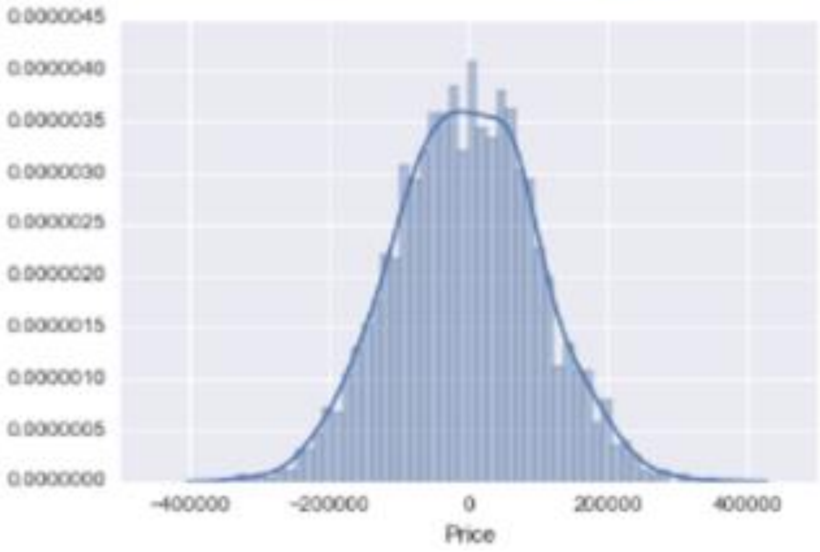
```
plt.scatter(y_test,predictions)
```

<matplotlib.collections.PathCollection at 0x142622c88>



Residual Histogram

```
sns.distplot((y_test-predictions),bins=50);
```



Linear Regression

Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

```
from sklearn import metrics
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 82288.2225191
MSE: 10460958907.2
RMSE: 102278.829223
```

Logistic Regression

Logistic Regression

- Exercise:
  - **Problem:** use machine learning to create a model that predicts which passengers survived the Titanic shipwreck
  - **Classification Approach:** Logistic Regression approach to solve this problem
  - **Dataset**: table with information regarding passengers' information, including:

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Logistic Regression

Import Libraries

Let's import some libraries to get started!

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

The Data

Let's start by reading in the titanic_train.csv file into a pandas dataframe.

```
train = pd.read_csv('titanic_train.csv')
```

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Logistic Regression

Exploratory Data Analysis

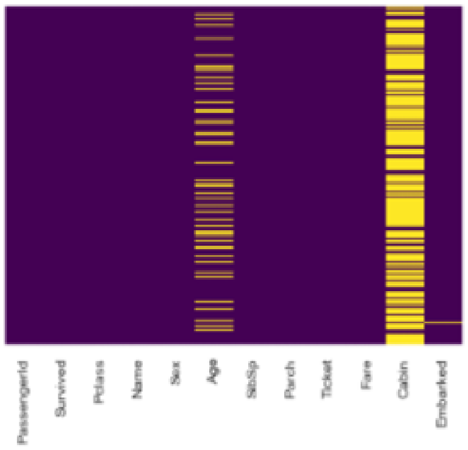
Let's begin some exploratory data analysis! We'll start by checking out missing data!

Missing Data

We can use seaborn to create a simple heatmap to see where we are missing data!

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a56f7b8>
```

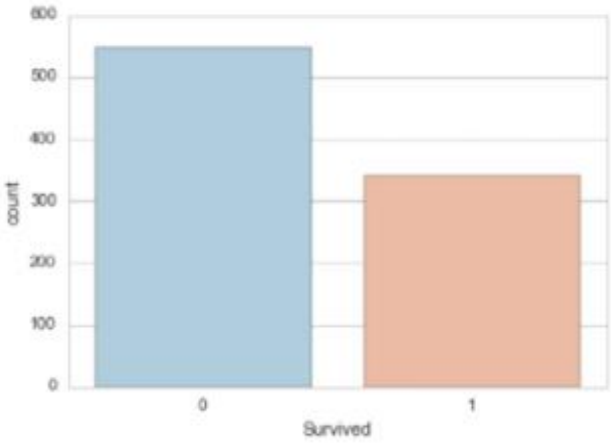


Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0"

Let's continue on by visualizing some more of the data! Check out the video for full explanations over these plots,

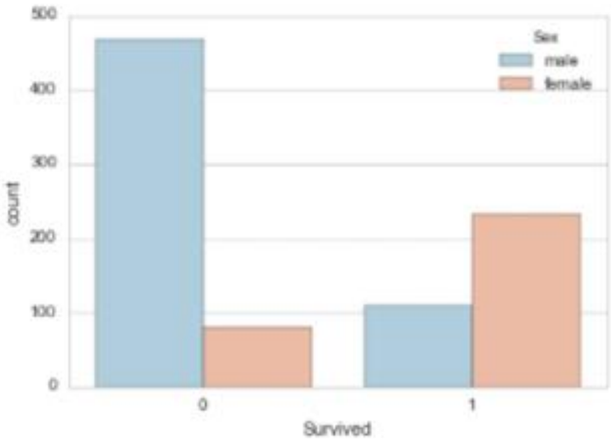
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11afae630>
```




```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

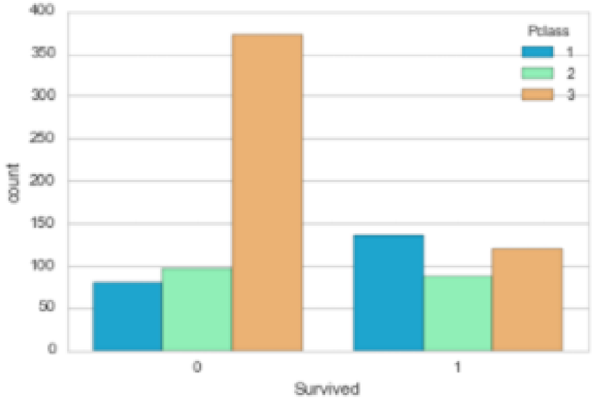
```
<matplotlib.axes._subplots.AxesSubplot at 0x11b004a20>
```



Logistic Regression

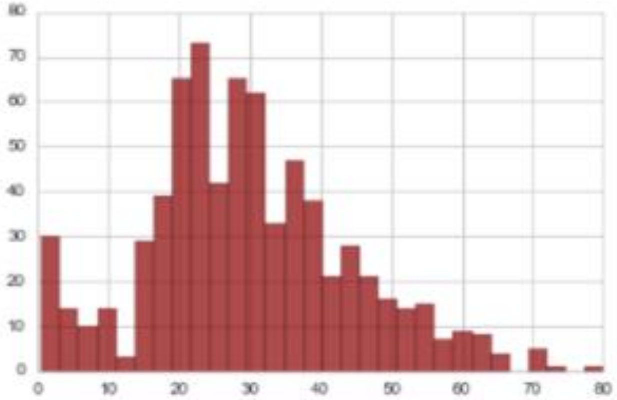

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b130f28>
```



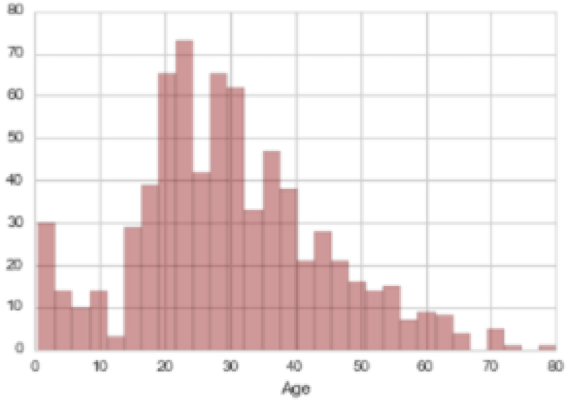

```
train['Age'].hist(bins=30,color='darkred',alpha=0.7)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b127ef0>
```



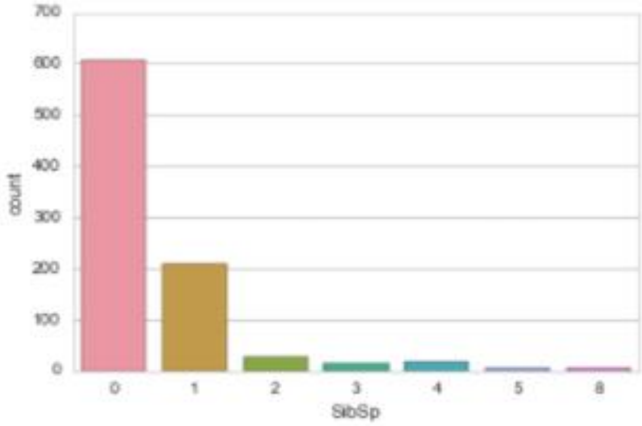
```
sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=30)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11c16f710>
```




```
sns.countplot(x='SibSp',data=train)
```

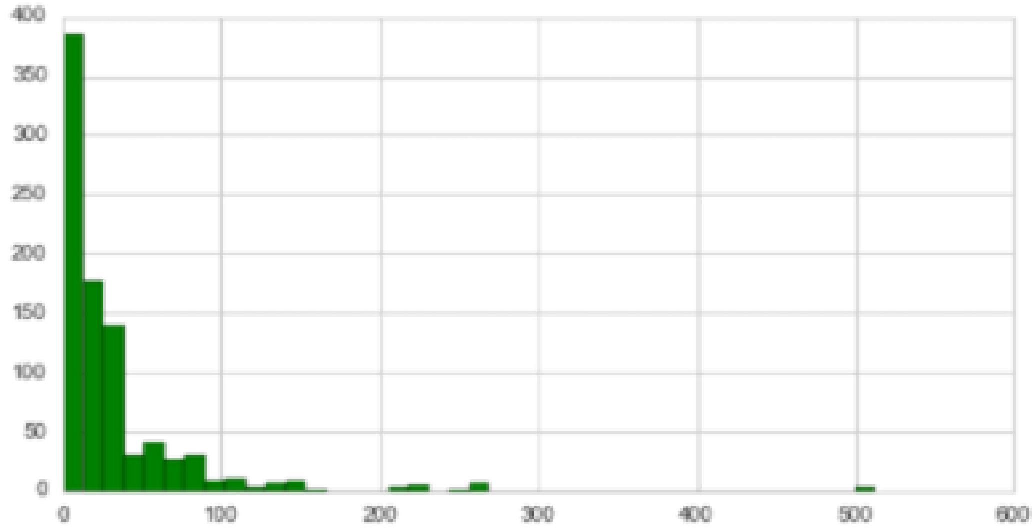
```
<matplotlib.axes._subplots.AxesSubplot at 0x11c4139e8>
```



Logistic Regression


```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x113893048>
```



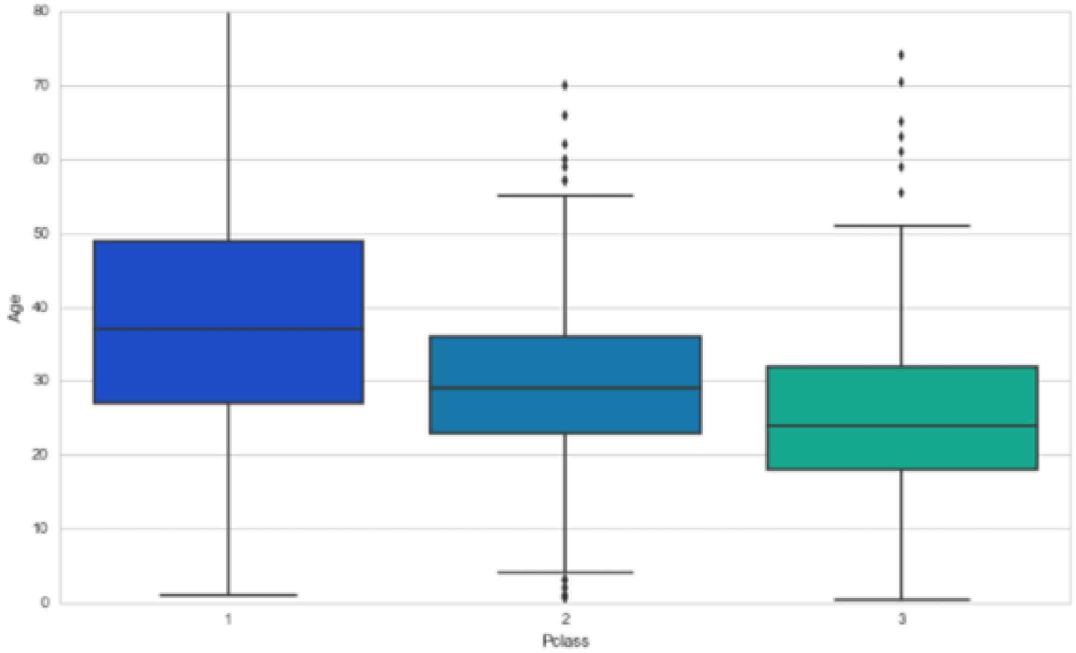
Logistic Regression

Data Cleaning

We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation). However we can be smarter about this and check the average age by passenger class. For example:

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11c901cc0>
```



We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

Logistic Regression


```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

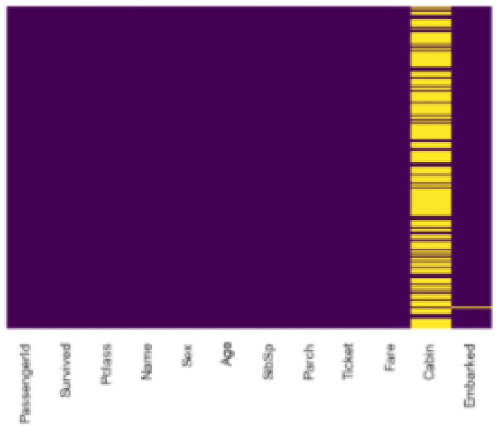
Now apply that function!

```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

Now let's check that heat map again!

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11c4dae10>
```



Great! Let's go ahead and drop the Cabin column and the row in Embarked that is NaN.

```
train.drop('Cabin',axis=1,inplace=True)
```

```
train.head()
```

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

```
train.dropna(inplace=True)
```

Logistic Regression

Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
PassengerId    889 non-null int64
Survived       889 non-null int64
Pclass         889 non-null int64
Name           889 non-null object
Sex            889 non-null object
Age            889 non-null float64
SibSp          889 non-null int64
Parch          889 non-null int64
Ticket         889 non-null object
Fare           889 non-null float64
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

```
train = pd.concat([train,sex,embark],axis=1)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	1.0	0.0	1.0
1	2	1	1	38.0	1	0	71.2833	0.0	0.0	0.0
2	3	1	3	26.0	0	0	7.9250	0.0	0.0	1.0
3	4	1	1	35.0	1	0	53.1000	0.0	0.0	1.0
4	5	0	3	35.0	0	0	8.0500	1.0	0.0	1.0

Great! Our data is ready for our model!

Logistic Regression

Building a Logistic Regression model

Let's start by splitting our data into a training set and test set.

Train Test Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

Training and Predicting

```
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
predictions = logmodel.predict(X_test)
```

Let's move on to evaluate our model!

Logistic Regression

Training and Predicting

```
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
predictions = logmodel.predict(X_test)
```

Let's move on to evaluate our model!

Evaluation

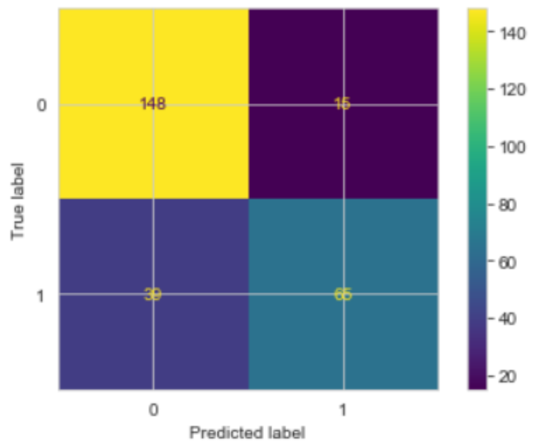
We can check precision,recall,f1-score using classification report!

```
from sklearn.metrics import classification_report, plot_confusion_matrix
```

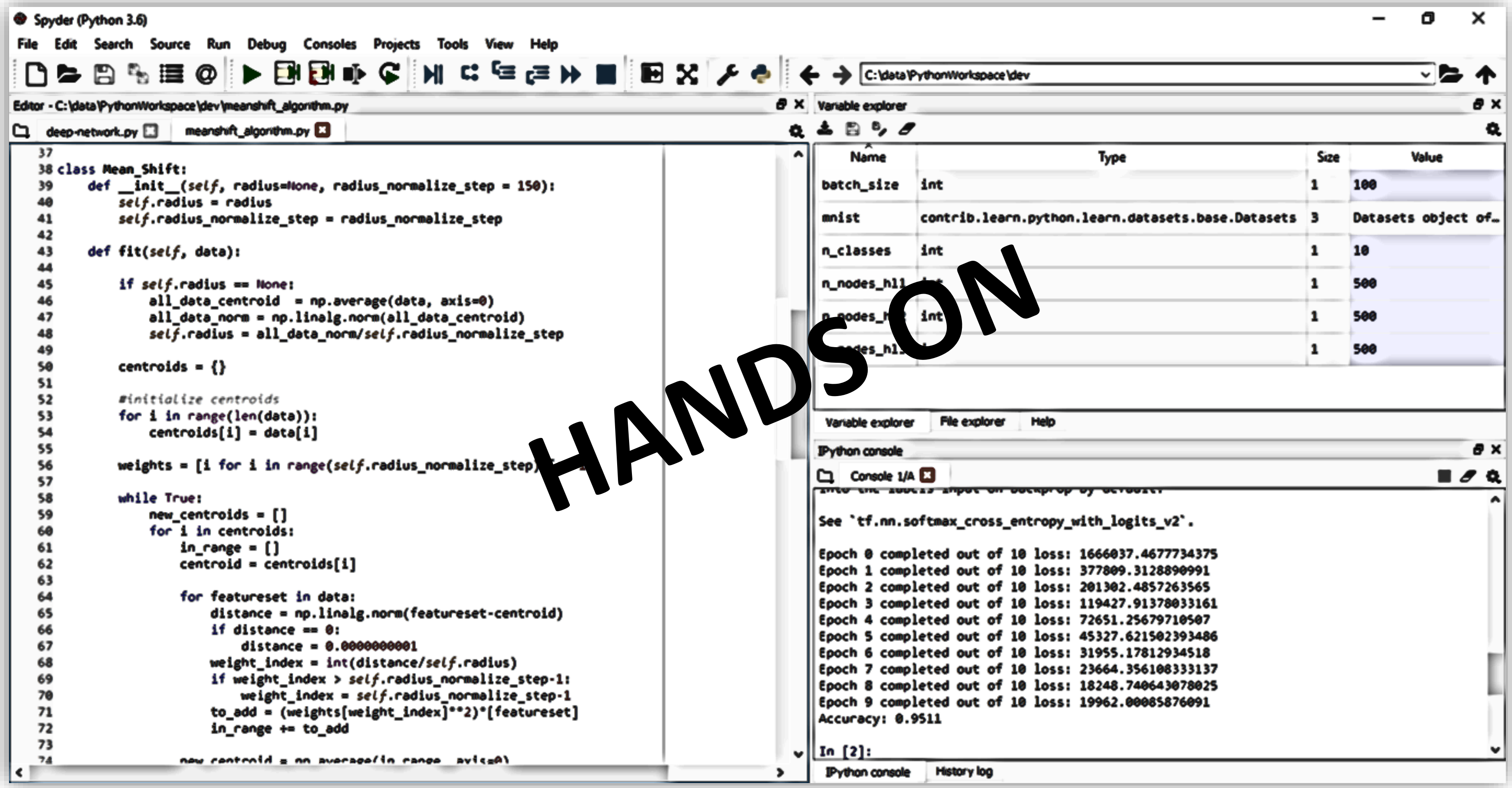
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.81      0.93      0.86       163
          1       0.85      0.65      0.74       104

avg / total       0.82      0.82      0.81       267
```

```
#Get the confusion matrix
plot_confusion_matrix(logmodel, X_test, y_test)
plt.show()
```



Hands On


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\data\PythonWorkspace\dev
Editor - C:\data\PythonWorkspace\dev\meanshift_algorithm.py
deep-network.py
meanshift_algorithm.py
class Mean_Shift:
def __init__(self, radius=None, radius_normalize_step = 150):
self.radius = radius
self.radius_normalize_step = radius_normalize_step
def fit(self, data):
if self.radius == None:
all_data_centroid = np.average(data, axis=0)
all_data_norm = np.linalg.norm(all_data_centroid)
self.radius = all_data_norm/self.radius_normalize_step
centroids = {}
#initialize centroids
for i in range(len(data)):
centroids[i] = data[i]
weights = [i for i in range(self.radius_normalize_step)
while True:
new_centroids = []
for i in centroids:
in_range = []
centroid = centroids[i]
for featureset in data:
distance = np.linalg.norm(featureset-centroid)
if distance == 0:
distance = 0.0000000001
weight_index = int(distance/self.radius)
if weight_index > self.radius_normalize_step-1:
weight_index = self.radius_normalize_step-1
to_add = (weights[weight_index]**2)*[featureset]
in_range += to_add
Variable explorer
Name Type Size Value
batch_size int 1 100
mnist contrib.learn.python.learn.datasets.base.Datasets 3 Datasets object of…
n_classes int 1 10
500
500
500
Variable explorer File explorer Help
IPython console
Console 1/A
See `tf.nn.softmax_cross_entropy_with_logits_v2`.
Epoch 0 completed out of 10 loss: 1666037.4677734375
Epoch 1 completed out of 10 loss: 377809.3128890991
Epoch 2 completed out of 10 loss: 201302.4857263565
Epoch 3 completed out of 10 loss: 119427.91378033161
Epoch 4 completed out of 10 loss: 72651.25679710507
Epoch 5 completed out of 10 loss: 45327.621502393486
Epoch 6 completed out of 10 loss: 31955.17812934518
Epoch 7 completed out of 10 loss: 23664.356108333137
Epoch 8 completed out of 10 loss: 18248.740643078025
Epoch 9 completed out of 10 loss: 19962.00085876091
Accuracy: 0.9511
In [2]:
IPython console History log
HANDS ON