Linear Regression with Python

We have information about house prices in regions of the United States in `USA_Housing.csv` .

The data contains the following columns:

- `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

Check out the data ¶

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Check out the Data

In [2]:
```
USAhousing = pd.read_csv('USA_Housing.csv')
```

In [3]:
```
USAhousing.head()
```

Out[3]:

	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

In [4]:
```
USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

In [5]:
```
USAhousing.describe()
```

Out[5]:

	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

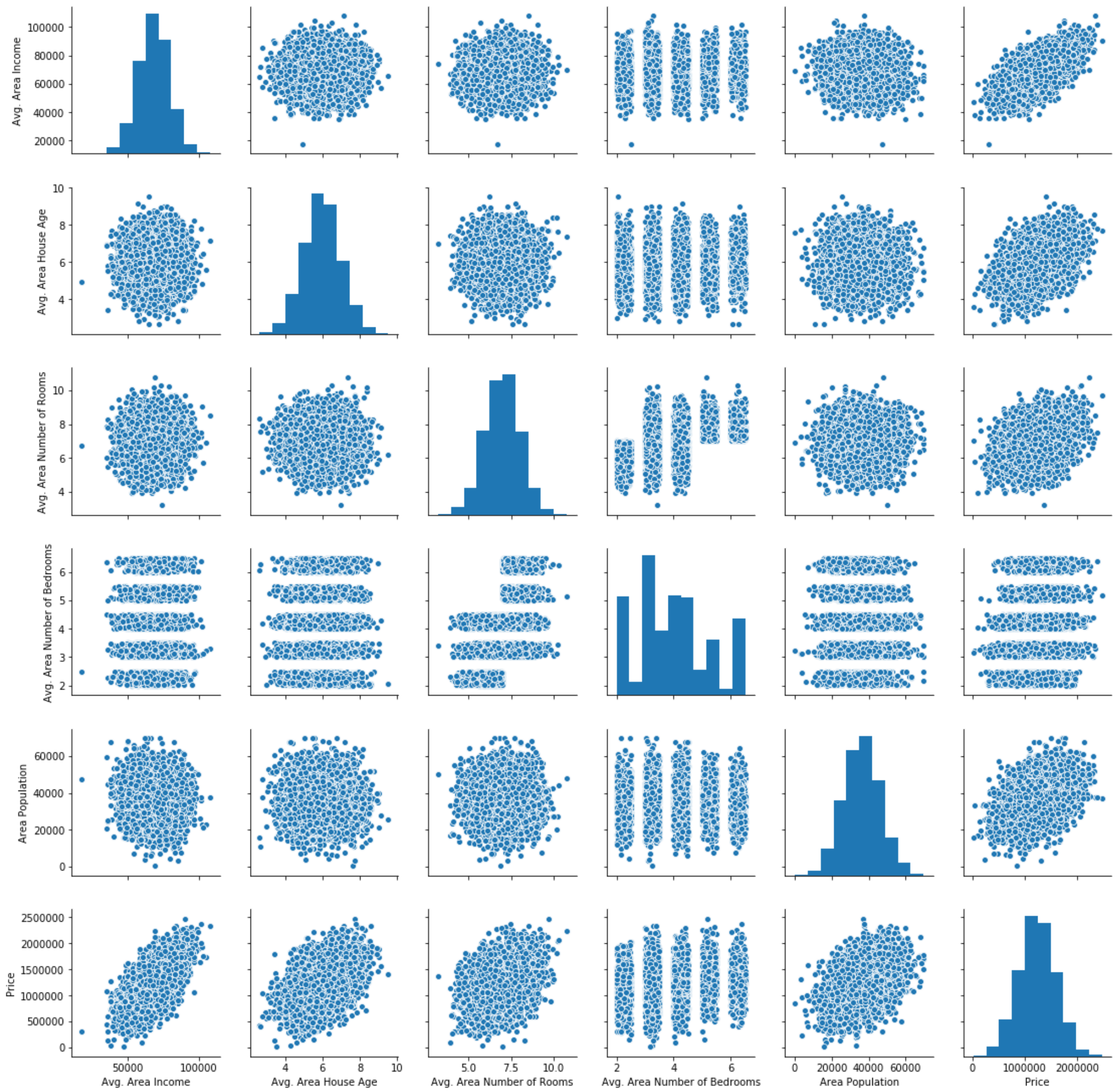
In [6]:
```
USAhousing.columns
```

Out[6]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Let's create some plot to check out the data!

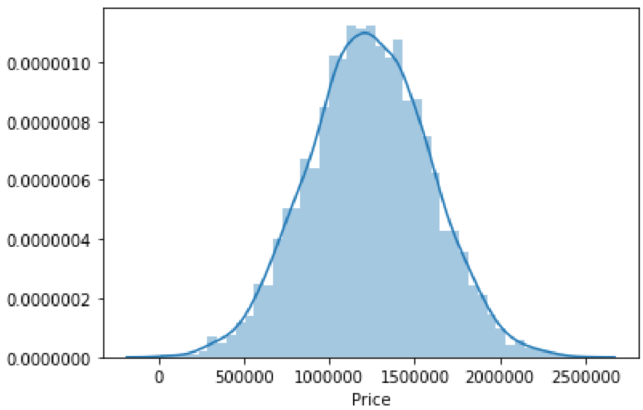
In [7]:
```
sns.pairplot(USAhousing)
```

Out[7]: <seaborn.axisgrid.PairGrid at 0x1a1cf15c50>

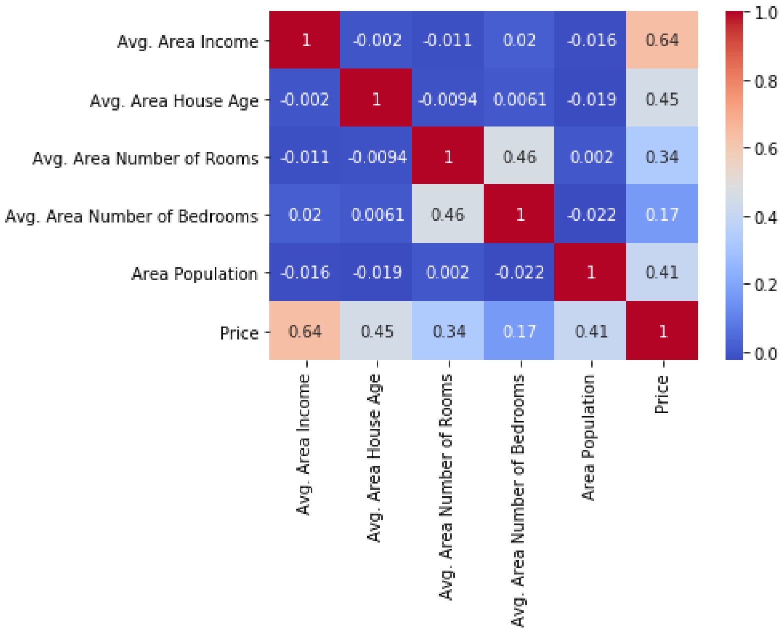


In [8]:
```
sns.distplot(USAhousing['Price'])
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1f25e990>



In [9]:
```
sns.heatmap(USAhousing.corr(), annot=True, cmap='coolwarm')
```
Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1d6a2910>
```



Training a Linear Regression Model

We 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.

X and y arrays

In [10]:
```
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.

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
```

Creating and Training the Model

In [13]:
```
from sklearn.linear_model import LinearRegression
```

In [14]:
```
lm = LinearRegression()
```

In [15]:
```
lm.fit(X_train, y_train)
```

Out[15]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Model Evaluation

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

In [16]:
```
# print the intercept
print(lm.intercept_)
```

```
-2640159.796851911
```

In [17]:
```
coeff_df = pd.DataFrame(lm.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

Out[17]:

	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** .

Does this make sense? Probably not because the data is made up. If you want real data to repeat this sort of analysis, check out the [boston dataset (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html)](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html):

```
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.DESCR)
boston_df = boston.data
```

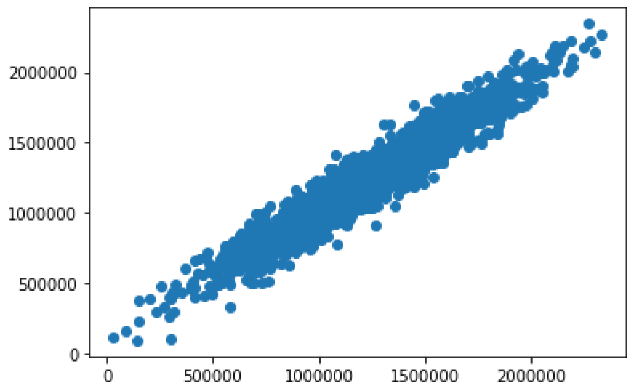
Predictions from our Model

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

In [18]:
```
predictions = lm.predict(X_test)
```

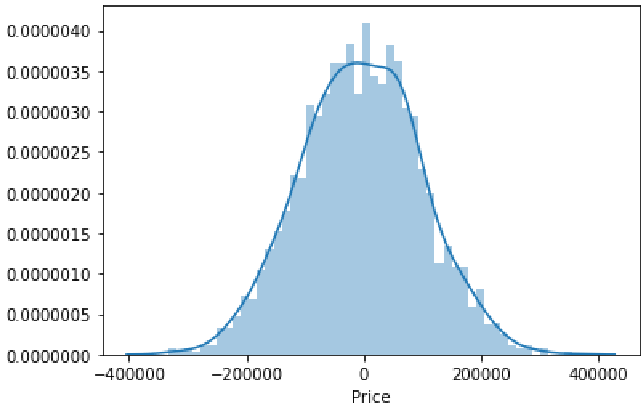
In [19]:
```
plt.scatter(y_test, predictions)
```

Out[19]: <matplotlib.collections.PathCollection at 0x1a1fc06a90>



Residual Histogram

In [20]:
```
sns.distplot((y_test - predictions), bins=50);
```



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.

In [21]:
```
from sklearn import metrics
```

In [22]:
```
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: 82288.22251914957
MSE: 10460958907.209501
RMSE: 102278.82922291153
```

Linear Regression Project - Solutions

A company is trying to decide whether to focus their efforts on their mobile app experience or their website.

Imports

Import pandas, numpy, matplotlib, and seaborn. Then set %matplotlib inline

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

We'll work with the `Ecommerce Customers`. It has Customer info, such as Email, Address, and their color Avatar. Then it also has numerical value columns:

- Avg. Session Length: Average session of in-store style advice sessions.
- Time on App: Average time spent on App in minutes
- Time on Website: Average time spent on Website in minutes
- Length of Membership: How many years the customer has been a member.

Read in the `Ecommerce Customers` csv file as a DataFrame called customers.

In [2]:
```
customers = pd.read_csv("Ecommerce Customers")
```

Check the head of customers, and check out its info() and describe() methods.

In [3]:
```
customers.head()
```

Out[3]:

	Email	Address	Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.655651	39.577668	4.082621	58
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.109461	37.268959	2.664034	39
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.330278	37.110597	4.104543	48
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.717514	36.721283	3.120179	58
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189	37.536653	4.446308	59

In [4]:
```
customers.describe()
```

Out[4]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	33.053194	12.052488	37.060445	3.533462	499.314038
std	0.992563	0.994216	1.010489	0.999278	79.314782
min	29.532429	8.508152	33.913847	0.269901	256.670582
25%	32.341822	11.388153	36.349257	2.930450	445.038277
50%	33.082008	11.983231	37.069367	3.533975	498.887875
75%	33.711985	12.753850	37.716432	4.126502	549.313828
max	36.139662	15.126994	40.005182	6.922689	765.518462

In [5]:
```
customers.info()
```

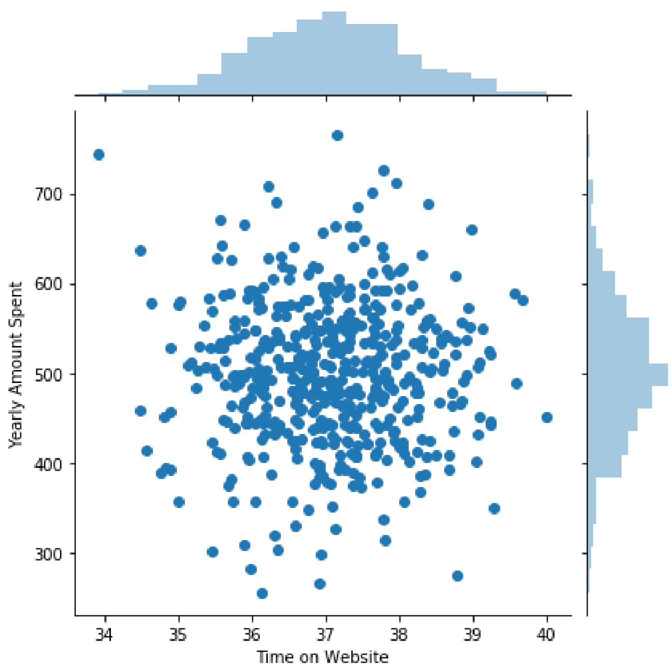
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Email                 500 non-null    object 
 1   Address               500 non-null    object 
 2   Avatar                500 non-null    object 
 3   Avg. Session Length   500 non-null    float64
 4   Time on App           500 non-null    float64
 5   Time on Website       500 non-null    float64
 6   Length of Membership  500 non-null    float64
 7   Yearly Amount Spent   500 non-null    float64
dtypes: float64(5), object(3)
memory usage: 31.4+ KB
```

Exploratory Data Analysis

Use seaborn to create a jointplot to compare the Time on Website and Yearly Amount Spent columns. Does the correlation make sense?

In [6]:
```
sns.jointplot(x='Time on Website', y='Yearly Amount Spent', data=customers)
```

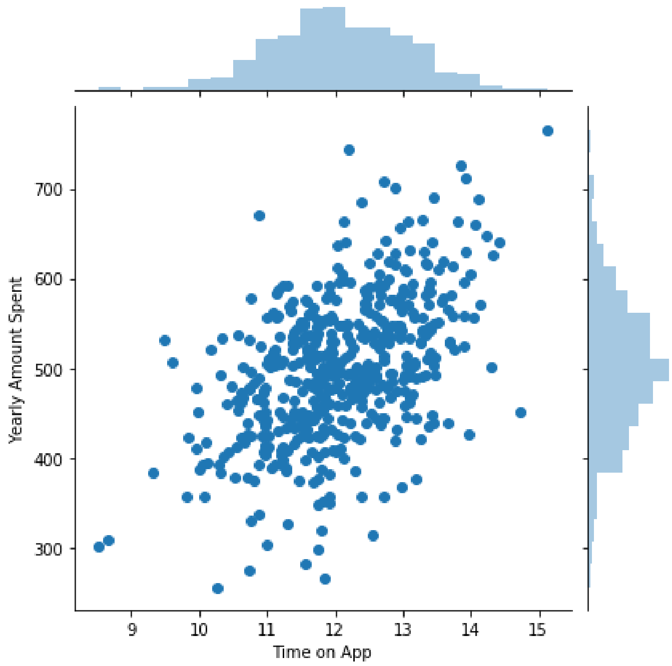
Out[6]: <seaborn.axisgrid.JointGrid at 0x1a1682fa10>



Do the same but with the Time on App column instead.

In [7]:
```
sns.jointplot(x='Time on App', y='Yearly Amount Spent', data=customers)
```

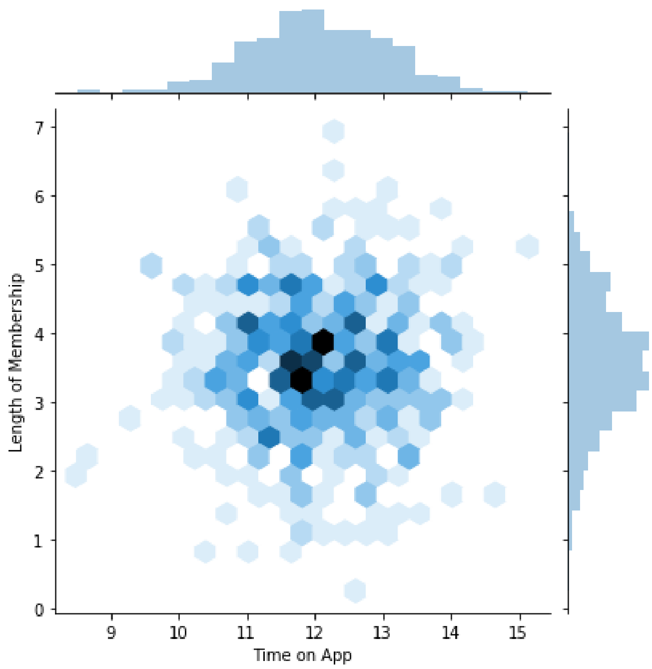
Out[7]: <seaborn.axisgrid.JointGrid at 0x1a17304b90>



Use jointplot to create a 2D hex bin plot comparing Time on App and Length of Membership.

In [8]:
```
sns.jointplot(x='Time on App', y='Length of Membership', kind='hex', data=customers)
```

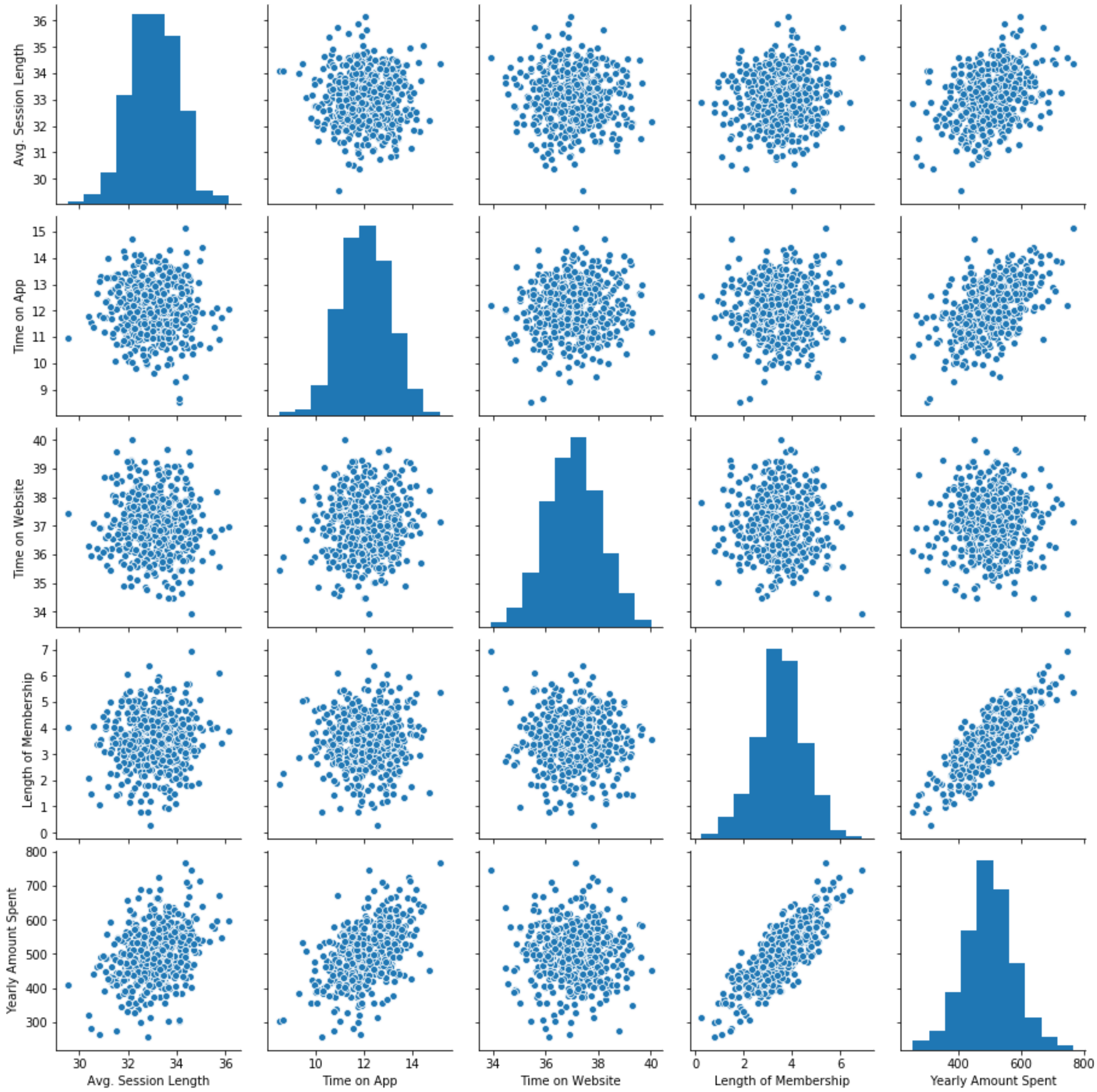
Out[8]: <seaborn.axisgrid.JointGrid at 0x1a174b0d50>



Use pairplot to recreate the plot below.

In [9]:
```
sns.pairplot(customers)
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x1a1771b350>

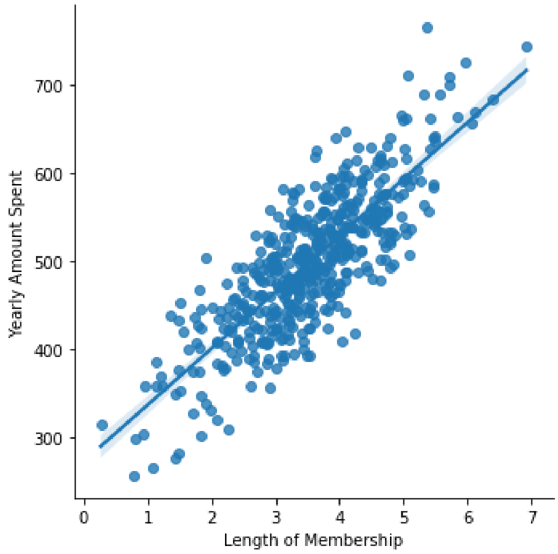


Based off this plot what looks to be the most correlated feature with Yearly Amount Spent?

In [10]:
```
# Length of Membership
```

Create a linear model plot (using seaborn's lmplot) of Yearly Amount Spent vs. Length of Membership.

```
In [11]: sns.lmplot(x='Length of Membership', y='Yearly Amount Spent', data=customers)
Out[11]: <seaborn.axisgrid.FacetGrid at 0x1a186719d0>
```



Training and Testing Data

Split the data into training and testing sets.

Set a variable `X` equal to the numerical features of the customers and a variable `y` equal to the `Yearly Amount Spent` column.

```
In [12]: y = customers['Yearly Amount Spent']
```

```
In [13]: X = customers[['Avg. Session Length', 'Time on App','Time on Website', 'Length of Membershi
p']]
```

Split the data into training and testing sets. Set `test_size=0.3` and `random_state=101`

```
In [14]: from sklearn.model_selection import train_test_split
```

```
In [15]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

Training the Model

Import `LinearRegression`

```
In [16]: from sklearn.linear_model import LinearRegression
```

Create an instance of a `LinearRegression()` model named `lm`.

```
In [17]: lm = LinearRegression()
```

Train/fit `lm` on the training data.

```
In [18]: lm.fit(X_train, y_train)
Out[18]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Print out the coefficients of the model

In [19]:
```
# The coefficients
print('Coefficients: \n', lm.coef_)
```

```
Coefficients:
 [25.98154972 38.59015875  0.19040528 61.27909654]
```

Predicting Test Data

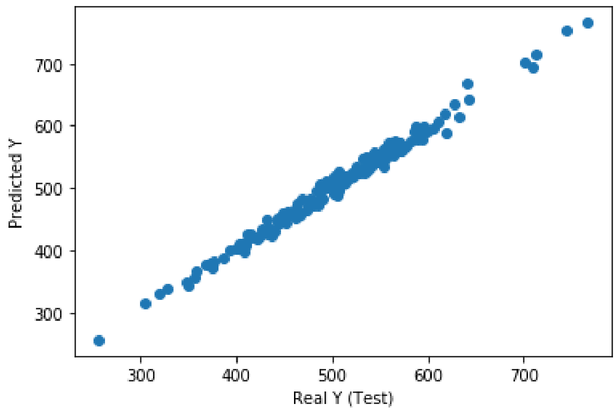
Use `lm.predict()` to predict off the `X_test` set of the data.

In [20]:
```
predictions = lm.predict(X_test)
```

Create a scatterplot of the real test values versus the predicted values.

In [21]:
```
plt.scatter(y_test, predictions)
plt.xlabel('Real Y (Test)')
plt.ylabel('Predicted Y')
```

Out[21]:
```
Text(0, 0.5, 'Predicted Y')
```



Evaluating the Model

Calculate the Mean Absolute Error, Mean Squared Error, and the Root Mean Squared Error.

In [22]:
```
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)))
```

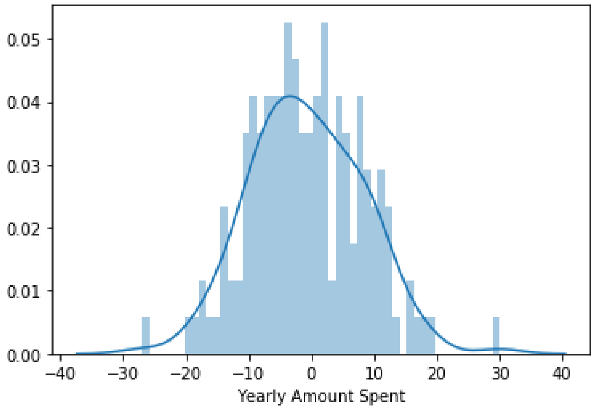
```
MAE: 7.228148653430853
MSE: 79.81305165097487
RMSE: 8.933815066978656
```

Residuals

Plot a histogram of the residuals and make sure it looks normally distributed.

In [23]:
```
sns.distplot((y_test - predictions), bins=50)
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19299990>



Conclusion

Should we focus our effort on mobile app or website development?

Recreate the dataframe below.

In [24]:
```
coeffecients = pd.DataFrame(lm.coef_, X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

Out[24]:

	Coeffecient
Avg. Session Length	25.981550
Time on App	38.590159
Time on Website	0.190405
Length of Membership	61.279097

Logistic Regression

For this lecture we will be working with the [Titanic Data Set from Kaggle (https://www.kaggle.com/c/titanic)](https://www.kaggle.com/c/titanic).

We'll be trying to predict a classification- survival or deceased.

Import Libraries

In [77]:
```
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.

In [78]:
```
train = pd.read_csv('titanic_train.csv')
```

In [79]:
```
train.head()
```

Out[79]:

	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

Exploratory Data Analysis

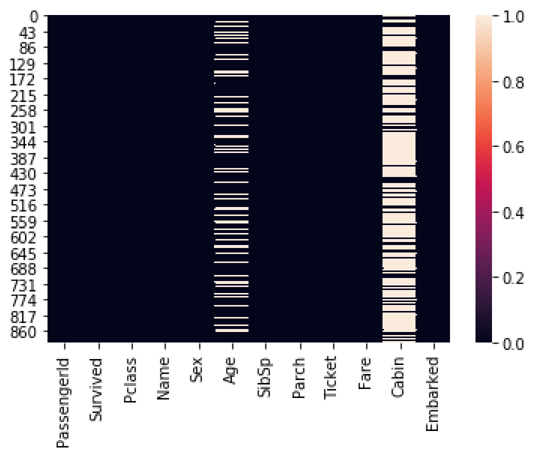
Let's start by checking for missing data!

Missing Data

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

In [80]:
```
sns.heatmap(train.isnull())
```

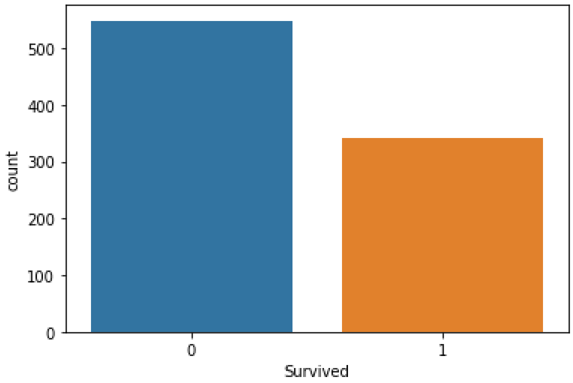
Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x1a27e66510>



Roughly 20 percent of the Age data is missing, likely small enough for reasonable replacement with some form of imputation. Cabin, instead, is missing too many values.

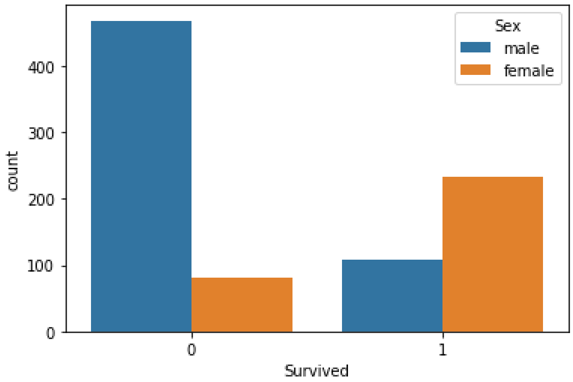
In [81]:
```
sns.countplot(x='Survived', data=train)
```

Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x1a282c6790>



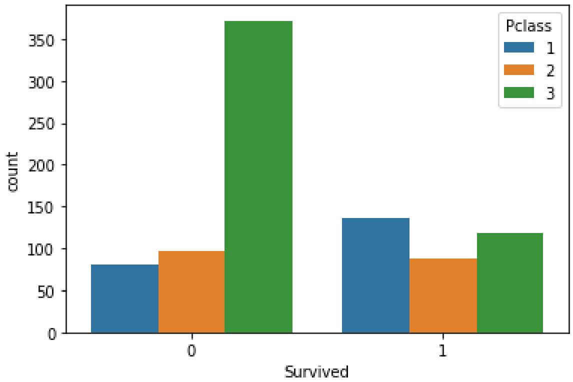
In [82]:
```
sns.countplot(x='Survived', hue='Sex', data=train)
```

Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x1a28328d10>

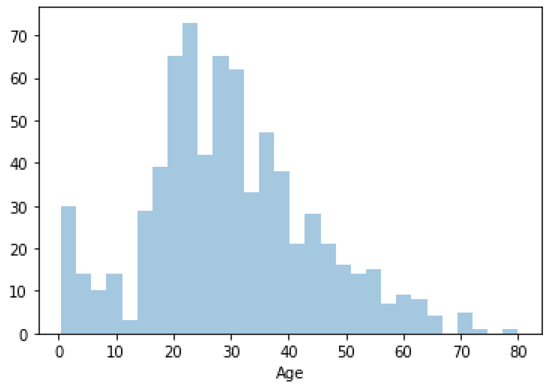


In [83]:
```
sns.countplot(x='Survived', hue='Pclass', data=train)
```

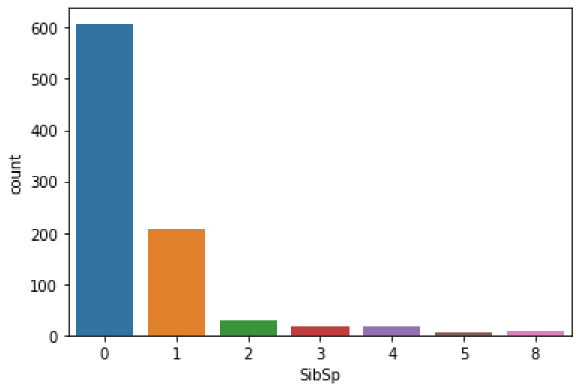
Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x1a283908d0>



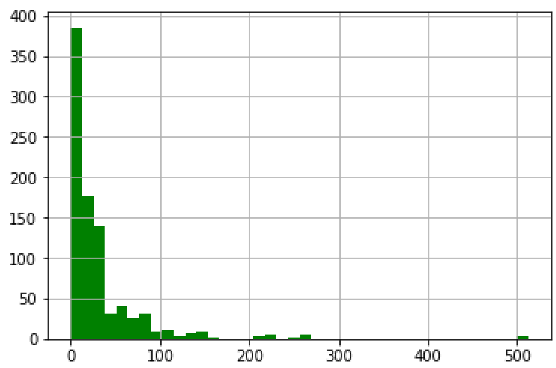
In [84]:
```
sns.distplot(train['Age'].dropna(), kde=False, bins=30)
```
Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2841c810>



In [85]:
```
sns.countplot(x='SibSp', data=train)
```
Out[85]: <matplotlib.axes._subplots.AxesSubplot at 0x1a284e0450>



In [86]:
```
train['Fare'].hist(color='green', bins=40)
```
Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x1a285e0f10>

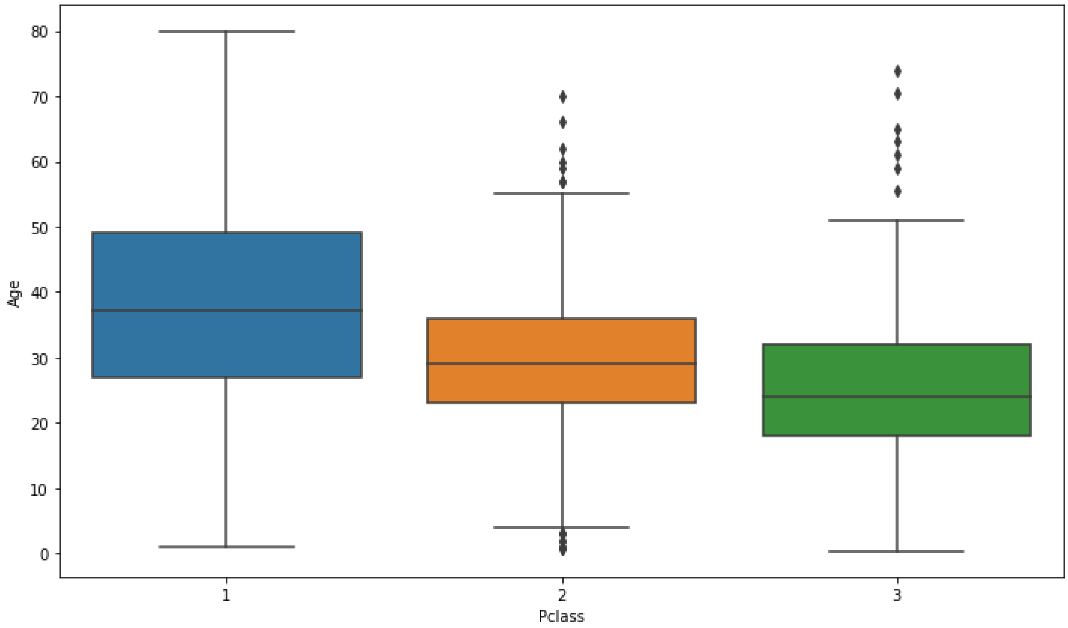


Data Cleaning

We want to fill in missing age data instead of just dropping the missing age data rows. We can fill in the mean, or even the average age by class.

In [87]:
```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass', y='Age', data=train)
```

Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2872a910>



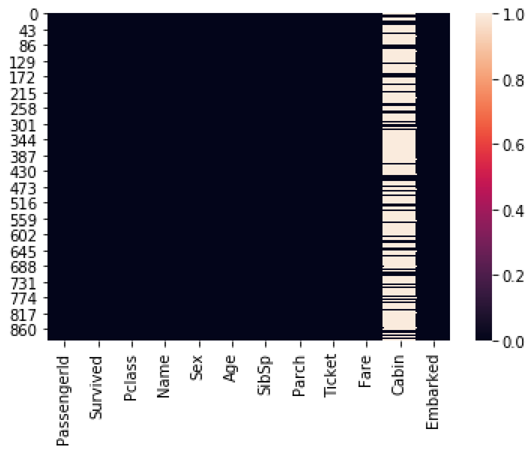
We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

In [88]:
```
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
```

In [89]:
```
train['Age'] = train[['Age','Pclass']].apply(impute_age, axis=1)
```

In [90]:
```
sns.heatmap(train.isnull())
```

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x1a28837b10>



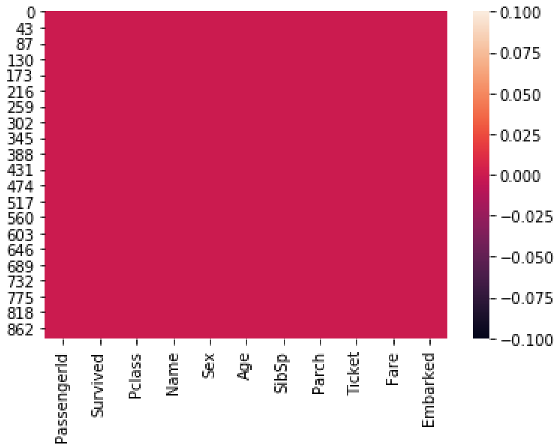
Let's drop the Cabin column and any other row with NaN.

In [91]:
```
train.drop('Cabin', axis=1, inplace=True)
```

In [92]:
```
train.dropna(inplace=True)
```

In [93]:
```
sns.heatmap(train.isnull())
```

Out[93]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a28adead0>
```



Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas.

In [94]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  889 non-null    int64
 1   Survived     889 non-null    int64
 2   Pclass       889 non-null    int64
 3   Name         889 non-null    object
 4   Sex          889 non-null    object
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64
 7   Parch        889 non-null    int64
 8   Ticket       889 non-null    object
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [95]:
```
pd.get_dummies(train['Sex'], drop_first=True)
```
Out[95]:

	male
0	1
1	0
2	0
3	0
4	1
...	...
886	1
887	0
888	0
889	1
890	1

889 rows × 1 columns

In [97]:
```
sex = pd.get_dummies(train['Sex'], drop_first=True)
embark = pd.get_dummies(train['Embarked'], drop_first=True)
```
In [98]:
```
train.drop(['Sex','Embarked','Name','Ticket'], axis=1, inplace=True)
```
In [99]:
```
train = pd.concat([train, sex, embark], axis=1)
```
In [100]:
```
train.head()
```
Out[100]:

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

In [106]:
```
train.drop('PassengerId', axis=1, inplace=True)
train.head()
```
Out[106]:

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

Building a Logistic Regression model

Train Test Split

In [107]:
```
from sklearn.model_selection import train_test_split
```
In [108]:
```
X = train.drop('Survived', axis=1)
y = train['Survived']
```
In [109]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.30,
                                                    random_state=101)
```

Training and Predicting

In [110]:
```
from sklearn.linear_model import LogisticRegression
```

In [114]:
```
logmodel = LogisticRegression(max_iter=1000)
logmodel.fit(X_train, y_train)
```

Out[114]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [115]:
```
predictions = logmodel.predict(X_test)
```

Evaluation

In [116]:
```
from sklearn.metrics import classification_report
```

In [117]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.82      0.92      0.87       163
           1       0.85      0.69      0.76       104

    accuracy                           0.83       267
   macro avg       0.84      0.81      0.82       267
weighted avg       0.83      0.83      0.83       267
```

In [118]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predictions)
```

Out[118]:
```
array([[150,  13],
       [ 32,  72]])
```

You might want to explore other feature:

- Try grabbing the Title (Dr.,Mr.,Mrs,etc..) from the name as a feature
- Maybe the Cabin letter could be a feature
- Is there any info you can get from the ticket?

Logistic Regression Project - Solutions

In this project we will be working with a fake advertising data set, indicating whether or not a particular internet user clicked on an Advertisement on a company website. This data set contains the following features:

- `Daily Time Spent on Site` : consumer time on site in minutes
- `Age` : cutomer age in years
- `Area Income` : Avg. Income of geographical area of consumer
- `Daily Internet Usage` : Avg. minutes a day consumer is on the internet
- `Ad Topic Line` : Headline of the advertisement
- `City` : City of consumer
- `Male` : Whether or not consumer was male
- `Country` : Country of consumer
- `Timestamp` : Time at which consumer clicked on Ad or closed window
- `Clicked on Ad` : 0 or 1 indicated clicking on Ad

Import Libraries

Import a few libraries you think you'll need

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

Read in the `advertising.csv` file and set it to a data frame called `ad_data` .

In [2]:
```
ad_data = pd.read_csv('advertising.csv')
```

Check the head of ad_data

In [3]:
```
ad_data.head()
```

Out[3]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0

Use `info()` and `describe()` on `ad_data`

In [4]:
```
ad_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Daily Time Spent on Site  1000 non-null   float64
 1   Age                       1000 non-null   int64  
 2   Area Income               1000 non-null   float64
 3   Daily Internet Usage      1000 non-null   float64
 4   Ad Topic Line             1000 non-null   object 
 5   City                      1000 non-null   object 
 6   Male                      1000 non-null   int64  
 7   Country                   1000 non-null   object 
 8   Timestamp                 1000 non-null   object 
 9   Clicked on Ad             1000 non-null   int64  
dtypes: float64(3), int64(3), object(4)
memory usage: 78.2+ KB
```

In [5]:
```
ad_data.describe()
```

Out[5]:

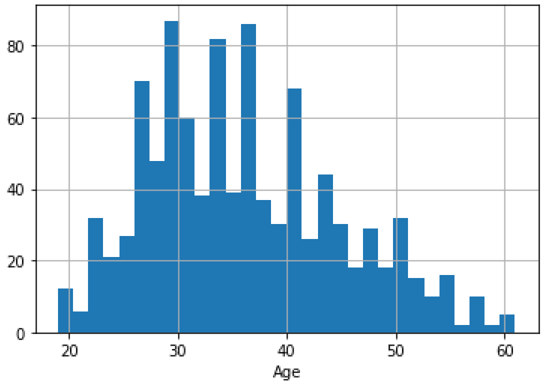
	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000	19.000000	13996.500000	104.780000	0.000000	0.00000
25%	51.360000	29.000000	47031.802500	138.830000	0.000000	0.00000
50%	68.215000	35.000000	57012.300000	183.130000	0.000000	0.50000
75%	78.547500	42.000000	65470.635000	218.792500	1.000000	1.00000
max	91.430000	61.000000	79484.800000	269.960000	1.000000	1.00000

Exploratory Data Analysis

Create a histogram of the Age

In [6]:
```
ad_data['Age'].hist(bins=30)
plt.xlabel('Age')
```

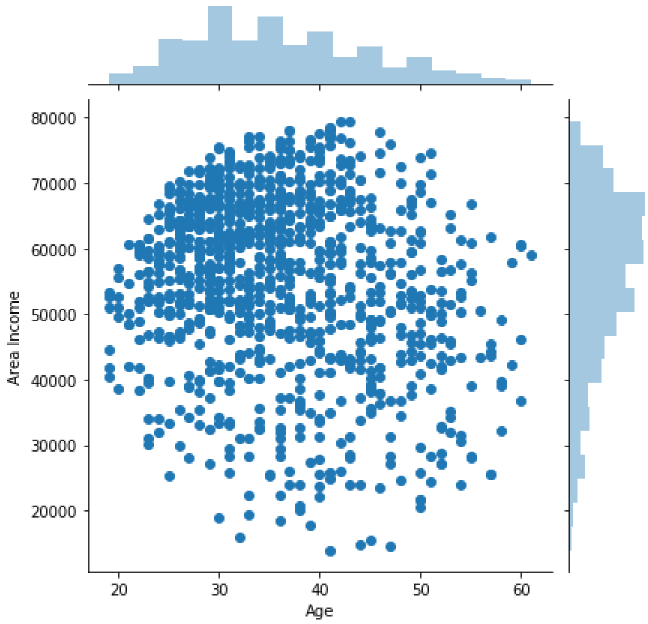
Out[6]: Text(0.5, 0, 'Age')



Create a jointplot showing Area Income versus Age.

In [7]:
```
sns.jointplot(x='Age', y='Area Income', data=ad_data)
```

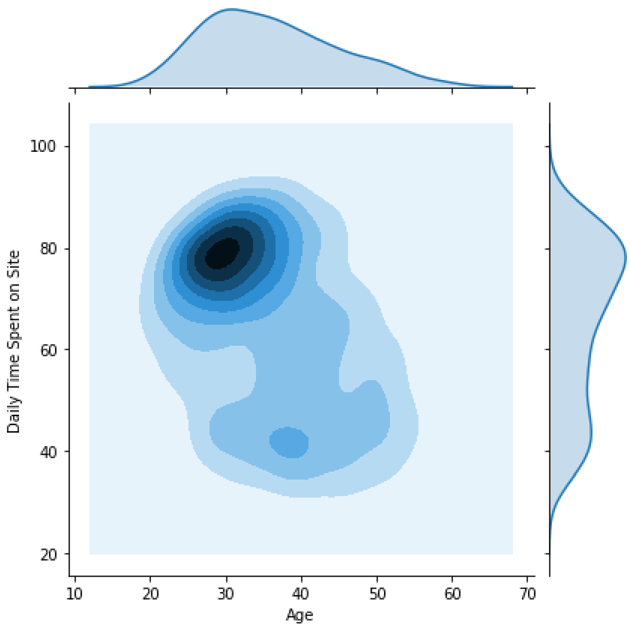
Out[7]: <seaborn.axisgrid.JointGrid at 0x1a250ec210>



Create a jointplot showing the kde distributions of Daily Time spent on site vs. Age.

In [8]:
```
sns.jointplot(x='Age', y='Daily Time Spent on Site', data=ad_data, kind='kde')
```

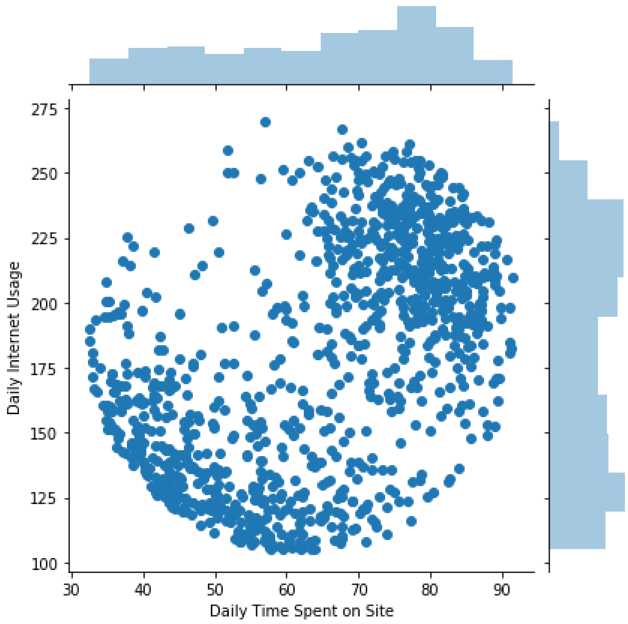
Out[8]: <seaborn.axisgrid.JointGrid at 0x1a253cef50>



Create a jointplot of `Daily Time Spent on Site` vs. `Daily Internet Usage`

In [9]:
```
sns.jointplot(x='Daily Time Spent on Site', y='Daily Internet Usage', data=ad_data)
```

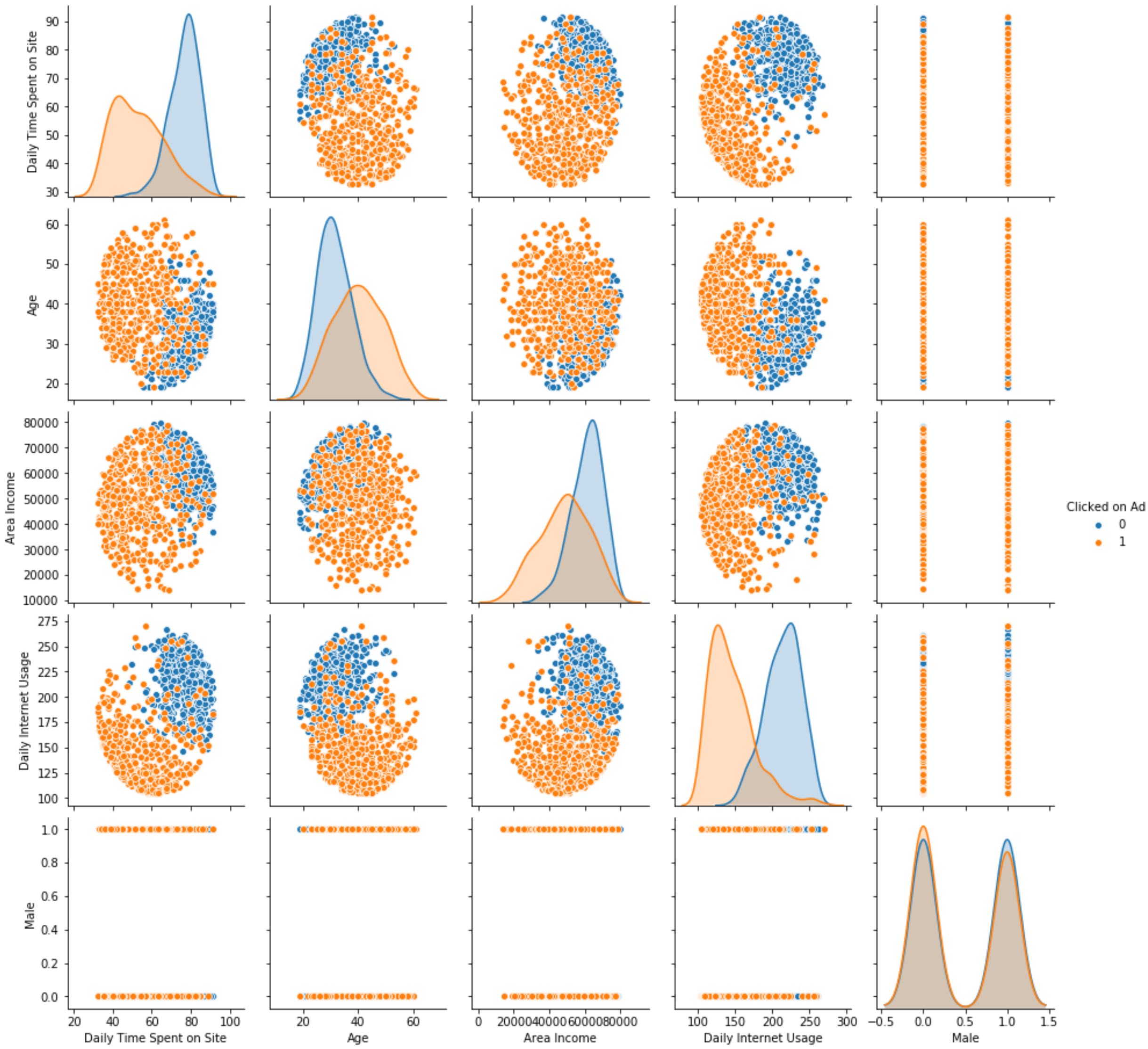
Out[9]: <seaborn.axisgrid.JointGrid at 0x1a25685250>



Finally, create a pairplot with the hue defined by the 'Clicked on Ad' column feature.

In [10]:
```
sns.pairplot(ad_data, hue='Clicked on Ad')
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x1a25992250>



Logistic Regression

Split the data into training set and testing set

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
X = ad_data[['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'Male']]
y = ad_data['Clicked on Ad']
```

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Train and fit a logistic regression model on the training set.

In [15]:
```
from sklearn.linear_model import LogisticRegression
```

In [16]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)
```

Out[16]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Predictions and Evaluations

Now predict values for the testing data.

In [17]:
```
predictions = logmodel.predict(X_test)
```

Create a classification report for the model.

In [18]:
```
from sklearn.metrics import classification_report
```

In [19]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91       162
           1       0.96      0.85      0.90       168

    accuracy                           0.91       330
   macro avg       0.91      0.91      0.91       330
weighted avg       0.91      0.91      0.91       330
```

K Nearest Neighbors

- It's simple (tries to cluster with the k closest points)
- It costs a lot for large datasets (it has to compute the distance to all the other points per each iteration).

Import Libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Get the Data

Set `index_col=0` to use the first column as the index.

In [2]:
```
df = pd.read_csv("Classified Data", index_col=0)
```

In [3]:
```
df.head()
```

Out[3]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

Standardize the Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters, we need to normalize them.

In [4]:
```
from sklearn.preprocessing import StandardScaler
```

In [5]:
```
scaler = StandardScaler()
```

In [6]:
```
features = df.drop('TARGET CLASS', axis=1)
scaler.fit(features)
```

Out[6]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [7]:
```
scaled_features = scaler.transform(features)
```

In [8]:
```
df_feat = pd.DataFrame(scaled_features, columns=features.columns)
df_feat.head()
```

Out[8]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

Train Test Split

In [9]:
```
from sklearn.model_selection import train_test_split
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(scaled_features,
                                                    df['TARGET CLASS'],
                                                    test_size=0.30)
```

Using KNN

We are trying to come up with a model to predict TARGET CLASS. We'll start with `k=1` .

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [12]:
```
knn = KNeighborsClassifier(n_neighbors=1)
```

In [13]:
```
knn.fit(X_train, y_train)
```

Out[13]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [14]:
```
pred = knn.predict(X_test)
```

Predictions and Evaluations

Let's evaluate our KNN model!

In [15]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [16]:
```
print(confusion_matrix(y_test, pred))
```

```
[[135  13]
 [ 17 135]]
```

In [17]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.91      0.90       148
           1       0.91      0.89      0.90       152

    accuracy                           0.90       300
   macro avg       0.90      0.90      0.90       300
weighted avg       0.90      0.90      0.90       300
```

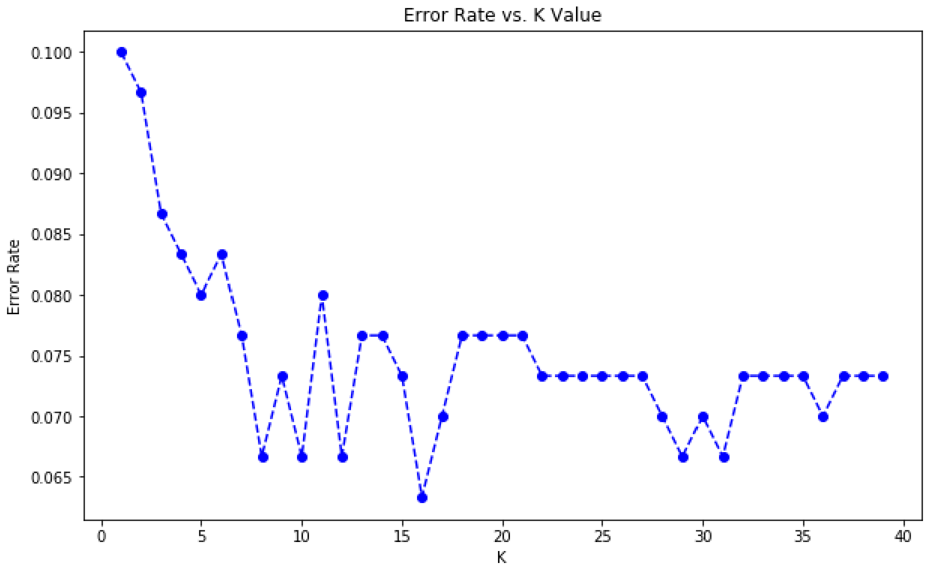
Choosing a K Value

Let's go ahead and use the **elbow method** to pick a good K Value:

In [18]:
```
error_rate = []

# Will take some time
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [19]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),
         error_rate,
         'bo--')

plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[19]:
```
Text(0, 0.5, 'Error Rate')
```



Here we can see that that after arouns K>20 the error rate lowers

In [20]:
```
# NOW WITH K=21
knn = KNeighborsClassifier(n_neighbors=21)

knn.fit(X_train, y_train)
pred = knn.predict(X_test)

print('WITH K=21')
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
WITH K=21
[[134  14]
 [  9 143]]
              precision    recall  f1-score   support

           0       0.94      0.91      0.92       148
           1       0.91      0.94      0.93       152

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300
```

Decision Trees and Random Forests

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

In [2]:
```
df = pd.read_csv('kyphosis.csv')
```

In [3]:
```
# the age in months
df.head()
```

Out[3]:

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15

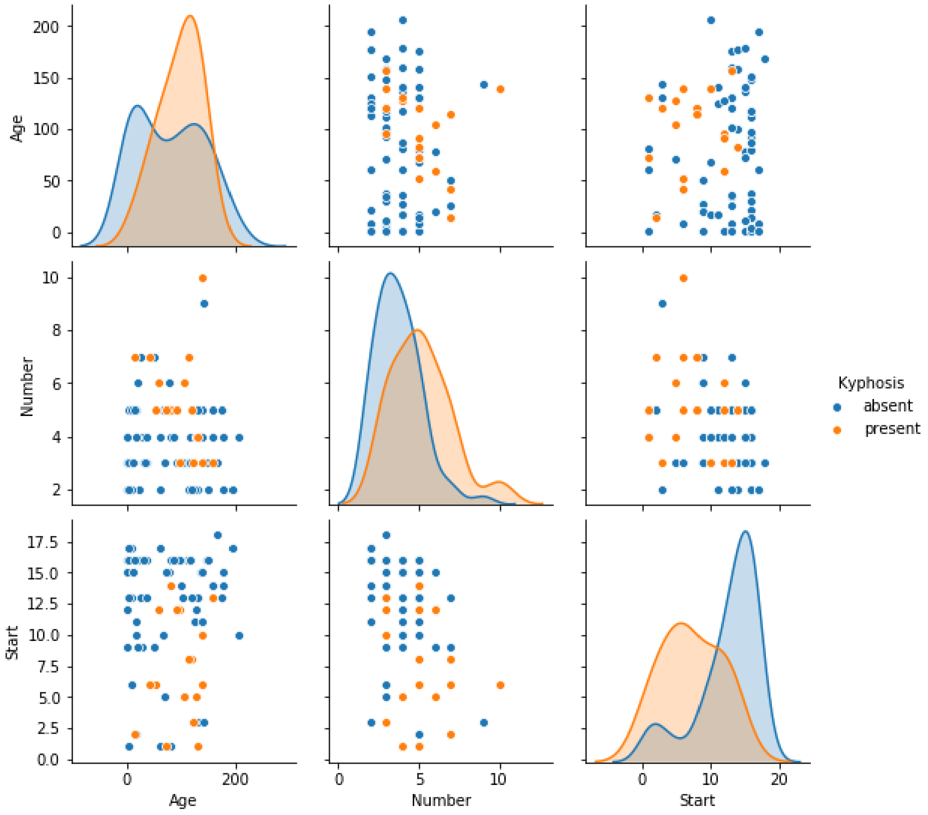
In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Kyphosis  81 non-null     object
 1   Age       81 non-null     int64 
 2   Number    81 non-null     int64 
 3   Start     81 non-null     int64 
dtypes: int64(3), object(1)
memory usage: 2.7+ KB
```

Data Analysis

```
In [5]: sns.pairplot(df, hue='Kyphosis')
```

```
Out[5]: <seaborn.axisgrid.PairGrid at 0x1a1b485a90>
```



Train Test Split

```
In [6]: from sklearn.model_selection import train_test_split
```

```
In [7]: X = df.drop('Kyphosis', axis=1)
        y = df['Kyphosis']
```

```
In [8]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

Decision Trees

```
In [9]: from sklearn.tree import DecisionTreeClassifier
```

```
In [10]: dtree = DecisionTreeClassifier()
```

```
In [11]: fit = dtree.fit(X_train, y_train)
```

Prediction and Evaluation

```
In [12]: predictions = dtree.predict(X_test)
```

```
In [13]: from sklearn.metrics import classification_report, confusion_matrix
```

In [14]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

      absent       0.90      0.86      0.88        21
     present       0.40      0.50      0.44         4

    accuracy                           0.80        25
   macro avg       0.65      0.68      0.66        25
weighted avg       0.82      0.80      0.81        25
```

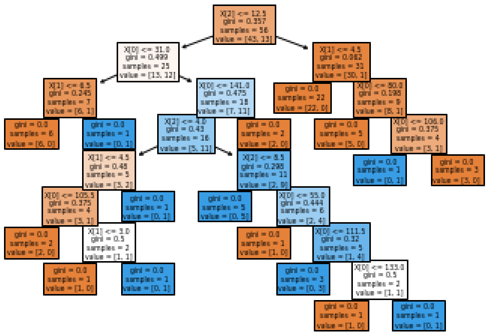
In [15]:
```
print(confusion_matrix(y_test, predictions))
```

```
[[18  3]
 [ 2  2]]
```

Tree Visualization

Scikit learn actually has some built-in visualization capabilities for decision trees.

In [16]:
```
from sklearn import tree
tree.plot_tree(dtree, filled=True);
```



Random Forests

Now let's compare the decision tree model to a random forest.

In [17]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
```

Out[17]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [18]:
```
rfc_pred = rfc.predict(X_test)
```

In [19]:
```
print(confusion_matrix(y_test, rfc_pred))
```

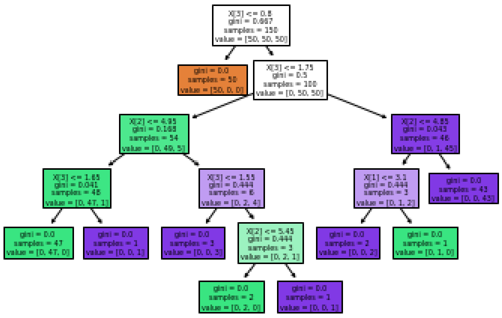
```
[[20  1]
 [ 3  1]]
```

In [20]:
```
print(classification_report(y_test, rfc_pred))
```

```
              precision    recall  f1-score   support

      absent       0.87      0.95      0.91        21
     present       0.50      0.25      0.33         4

    accuracy                           0.84        25
   macro avg       0.68      0.60      0.62        25
weighted avg       0.81      0.84      0.82        25
```

Other example using IRIS dataset

In [21]:
```python
from sklearn.datasets import load_iris
iris = load_iris()
iris_tree = DecisionTreeClassifier(random_state=0)
iris_tree.fit(iris.data, iris.target)
tree.plot_tree(iris_tree, filled=True);
```



Support Vector Machines

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

We'll use the built in breast cancer dataset from Scikit Learn. We can get with the load function:

In [2]:
```
from sklearn.datasets import load_breast_cancer
```

In [3]:
```
cancer = load_breast_cancer()
```

The data set is presented in a dictionary form:

In [4]:
```
cancer.keys()
```

Out[4]:
```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

We can grab information and arrays out of this dictionary to set up our data frame and understanding of the features:

In [5]:
```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2
```

```
Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

In [6]:
```
cancer['feature_names']
```

Out[6]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Set up DataFrame

In [7]:
```
df_feat = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
df_feat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [8]:
```
cancer['target']
```

Out[8]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

In [9]:
```
df_target = pd.DataFrame(cancer['target'], columns=['Cancer'])
```

In [10]:
```python
df_feat.head()
```

Out[10]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	pe
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	

5 rows × 30 columns

Train Test Split

In [11]:
```python
from sklearn.model_selection import train_test_split
```

In [12]:
```python
X_train, X_test, y_train, y_test = train_test_split(df_feat, np.ravel(df_target), test_size=0.30,
random_state=101)
```

Train the Support Vector Classifier

In [13]:
```python
from sklearn.svm import SVC
```

In [14]:
```python
model = SVC()
```

In [15]:
```python
model.fit(X_train, y_train)
```

Out[15]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Predictions and Evaluations

In [16]:
```python
predictions = model.predict(X_test)
```

In [17]:
```python
from sklearn.metrics import classification_report, confusion_matrix
```

In [18]:
```python
print(confusion_matrix(y_test, predictions))
```

```
[[ 56  10]
 [  3 102]]
```

In [19]:
```python
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.85      0.90        66
           1       0.91      0.97      0.94       105

    accuracy                           0.92       171
   macro avg       0.93      0.91      0.92       171
weighted avg       0.93      0.92      0.92       171
```

Support Vector Machines Project - Solutions

The Data

For this exercise, we will be using the famous Iris flower data set (http://en.wikipedia.org/wiki/Iris_flower_data_set).

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor), so 150 total samples. Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

Here's a picture of the three different Iris types:

Get the data

Use seaborn to get the iris data by using: `iris = sns.load_dataset('iris')`

In [1]:
```
import seaborn as sns
iris = sns.load_dataset('iris')
```

Exploratory Data Analysis ¶

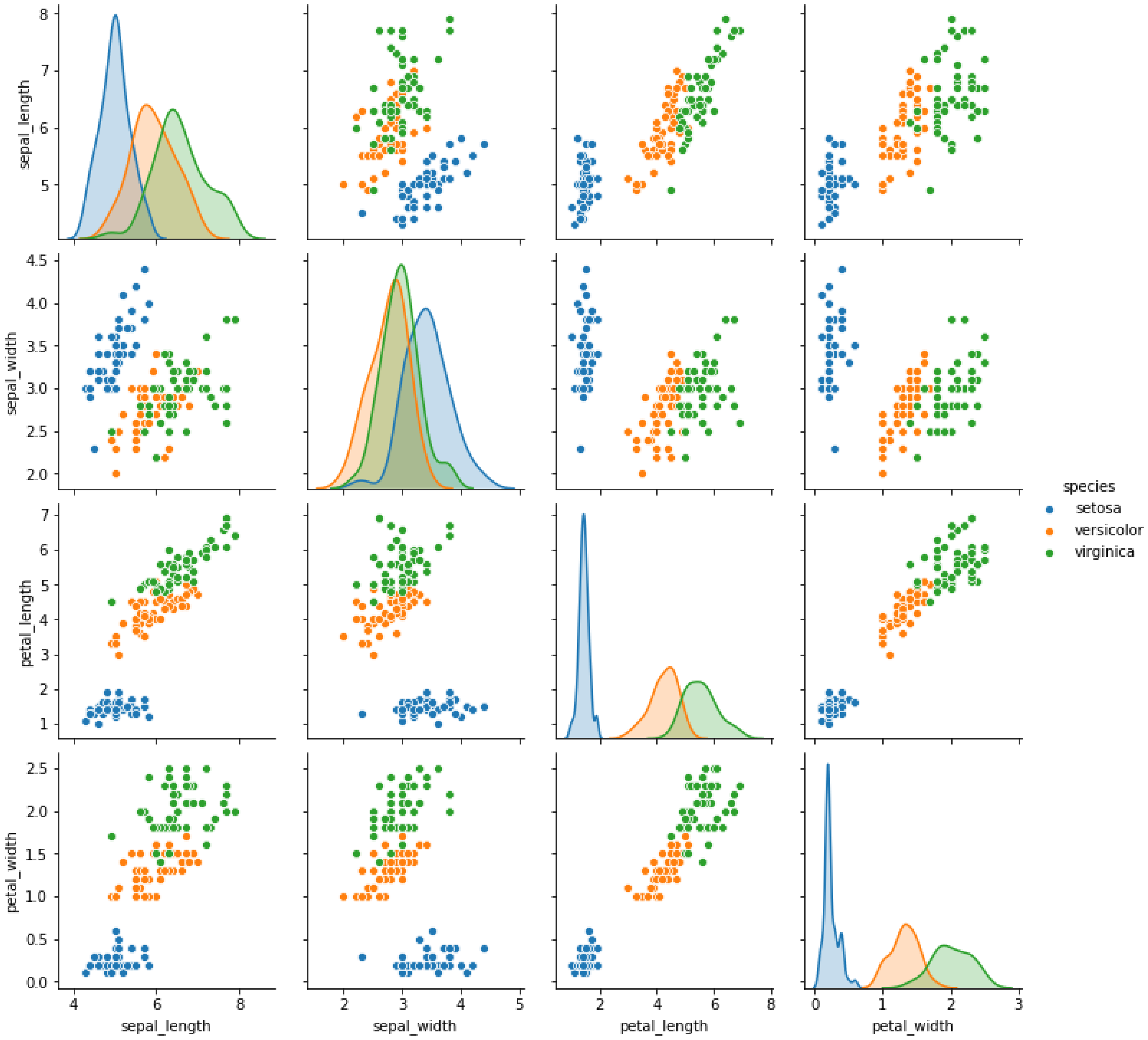
Import some libraries you think you'll need.

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Create a pairplot of the data set. Which flower species seems to be the most separable?

In [3]:
```python
# Setosa is the most separable.
sns.pairplot(iris, hue='species')
```

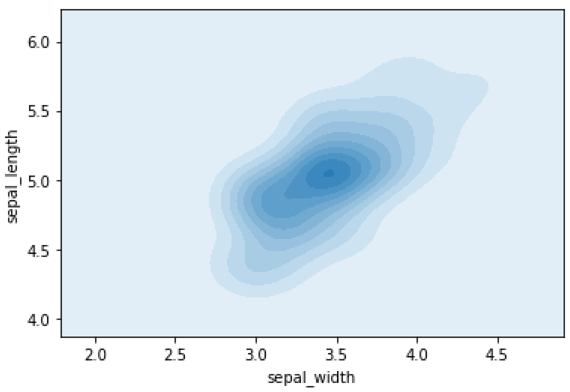
Out[3]: <seaborn.axisgrid.PairGrid at 0x1a1c593dd0>



Create a kde plot of sepal_length versus sepal width for setosa species of flower.

In [4]:
```python
setosa = iris[iris['species']=='setosa']
sns.kdeplot(setosa['sepal_width'],
            setosa['sepal_length'],
            shade=True)
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c593fd0>



Train Test Split

Split your data into a training set and a testing set.

In [5]:
```
from sklearn.model_selection import train_test_split
```

In [6]:
```
X = iris.drop('species', axis=1)
y = iris['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

Train a Model

Call the SVC() model from sklearn and fit the model to the training data.

In [7]:
```
from sklearn.svm import SVC
```

In [8]:
```
svc_model = SVC()
```

In [9]:
```
svc_model.fit(X_train, y_train)
```

Out[9]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Model Evaluation

Now get predictions from the model and create a confusion matrix and a classification report.

In [10]:
```
predictions = svc_model.predict(X_test)
```

In [11]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [12]:
```
print(confusion_matrix(y_test, predictions))
```

```
[[13  0  0]
 [ 0 14  2]
 [ 0  0 16]]
```

In [13]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      0.88      0.93        16
   virginica       0.89      1.00      0.94        16

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```