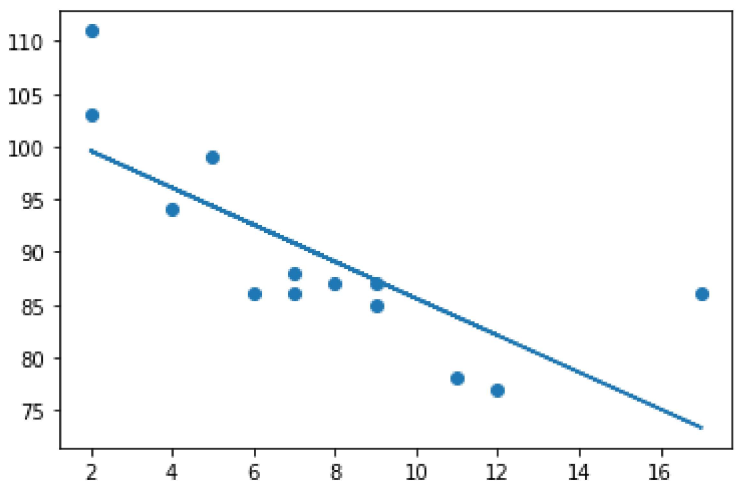
BASIC MACHINE LEARNING

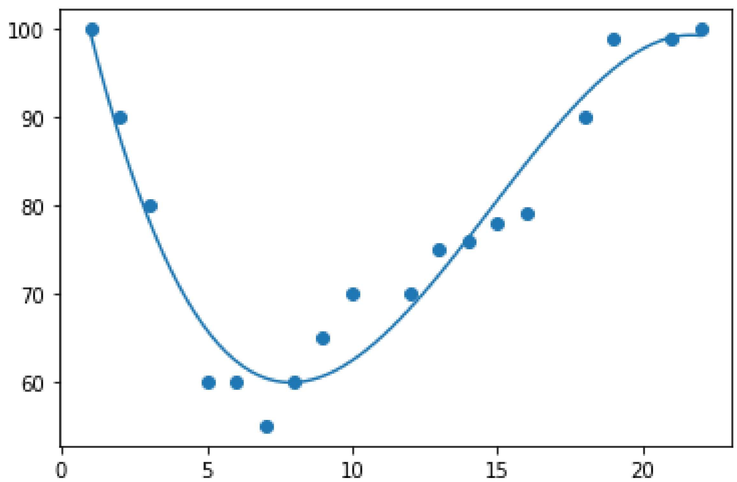
linear regression

In [11]:
```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



polynomial regression

In [12]:
```
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



statistics:-

MEAN , MEDIAN , MODE

In [20]:
```
import numpy
from scipy import stats
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = numpy.mean(speed)
print(x)
```

```
89.76923076923077
```

In [18]:
```
x = stats.mode(speed)
print(x)
```

```
ModeResult(mode=array([86]), count=array([3]))
```

In [19]:
```
x = numpy.median(speed)
print(x)
```

```
87.0
```

STANDARD DEVIATION & VARIANCE

In [21]:
```
x = numpy.std(speed)
print(x)
```

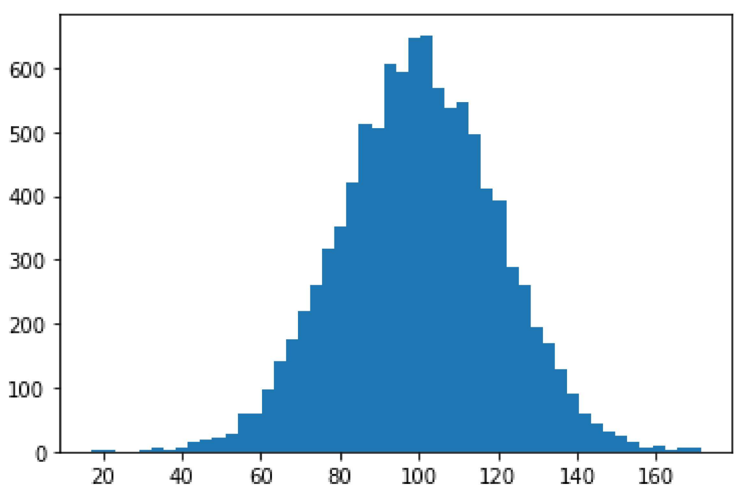
```
9.258292301032677
```

In [22]:
```
x = numpy.var(speed)
print(x)
```

```
85.71597633136093
```

implement all in graph

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt
a=np.random.normal(100,20,10000)
plt.hist(a,50)
plt.show()
```



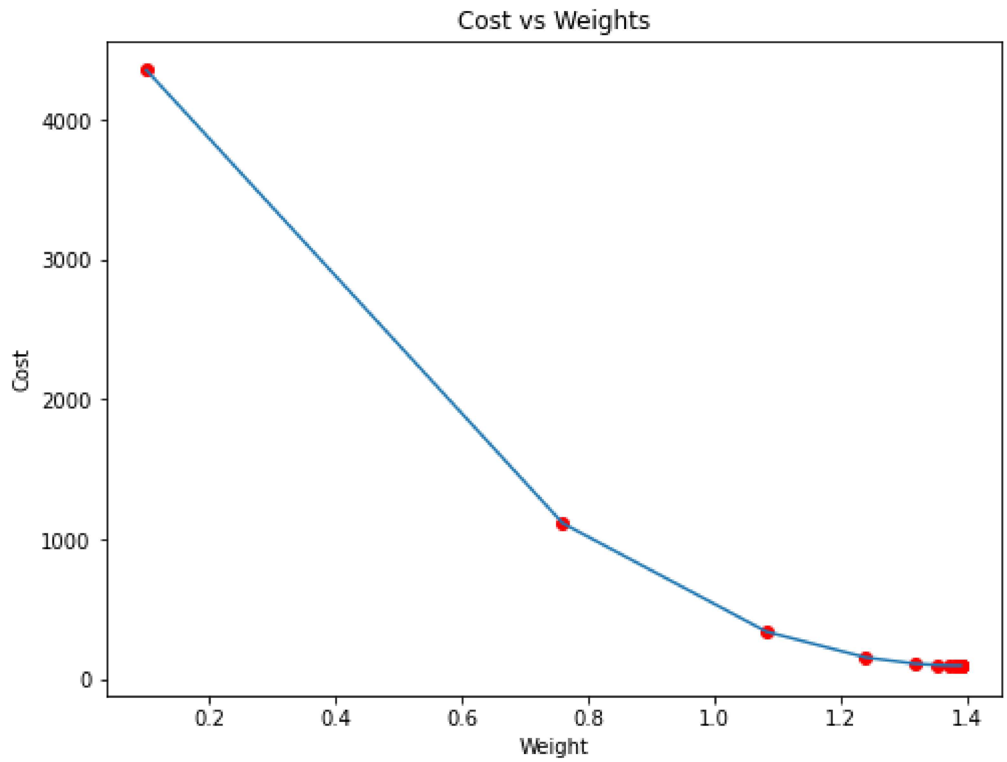
Gradient descent

In [27]:

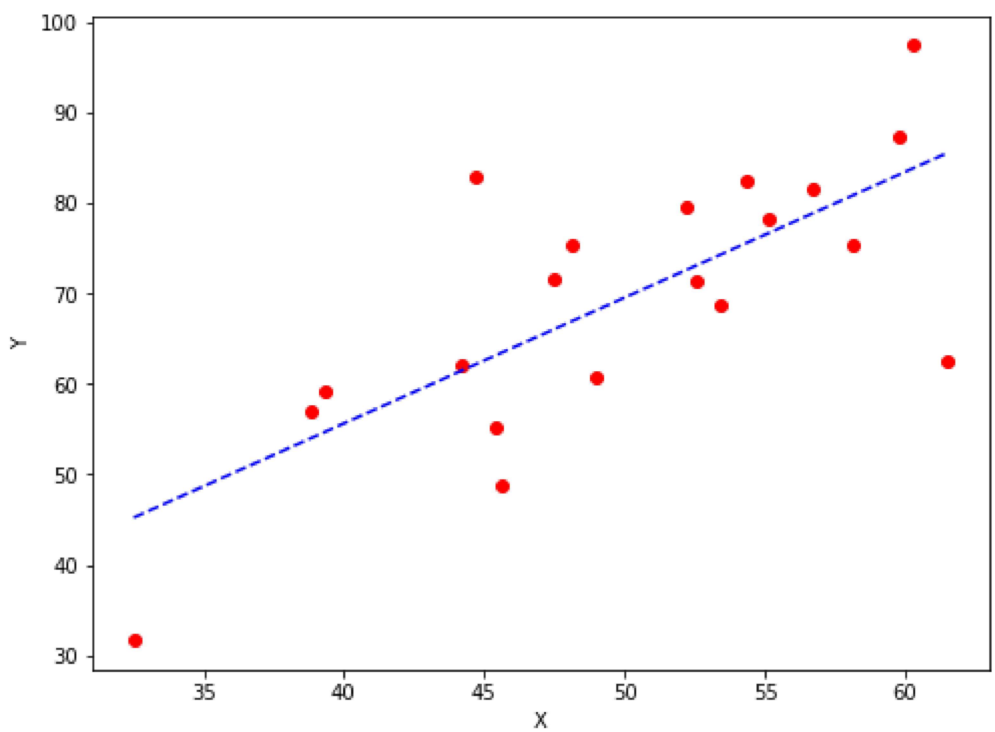
```python
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

def mean_squared_error(y_true, y_predicted):

    # Calculating the loss or cost
    cost = np.sum((y_true-y_predicted)**2) / len(y_true)
    return cost

# Gradient Descent Function
# Here iterations, learning_rate, stopping_threshold
# are hyperparameters that can be tuned
def gradient_descent(x, y, iterations = 1000, learning_rate = 0.0001,
                     stopping_threshold = 1e-6):

    # Initializing weight, bias, learning rate and iterations
    current_weight = 0.1
    current_bias = 0.01
    iterations = iterations
    learning_rate = learning_rate
    n = float(len(x))

    costs = []
    weights = []
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(iterations):

        # Making predictions
        y_predicted = (current_weight * x) + current_bias

        # Calculating the current cost
        current_cost = mean_squared_error(y, y_predicted)

        # If the change in cost is less than or equal to
        # stopping_threshold we stop the gradient descent
        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshol
            break

        previous_cost = current_cost

        costs.append(current_cost)
        weights.append(current_weight)

        # Calculating the gradients
        weight_derivative = -(2/n) * sum(x * (y-y_predicted))
        bias_derivative = -(2/n) * sum(y-y_predicted)

        # Updating weights and bias
        current_weight = current_weight - (learning_rate * weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)

        # Printing the parameters for each 1000th iteration
        print(f"Iteration {i+1}: Cost {current_cost}, Weight \
        {current_weight}, Bias {current_bias}")
```

```
    # Visualizing the weights and cost at for all iterations
    plt.figure(figsize = (8,6))
    plt.plot(weights, costs)
    plt.scatter(weights, costs, marker='o', color='red')
    plt.title("Cost vs Weights")
    plt.ylabel("Cost")
    plt.xlabel("Weight")
    plt.show()

    return current_weight, current_bias


def main():

    # Data
    X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.813207
           55.14218841, 52.21179669, 39.29956669, 48.10504169, 52.55001444,
           45.41973014, 54.35163488, 44.1640495 , 58.16847072, 56.72720806,
           48.95588857, 44.68719623, 60.29732685, 45.61864377, 38.81681754])
    Y = np.array([31.70700585, 68.77759598, 62.5623823 , 71.54663223, 87.230925
           78.21151827, 79.64197305, 59.17148932, 75.3312423 , 71.30087989,
           55.16567715, 82.47884676, 62.00892325, 75.39287043, 81.43619216,
           60.72360244, 82.89250373, 97.37989686, 48.84715332, 56.87721319])

    # Estimating weight and bias using gradient descent
    estimated_weight, estimated_bias = gradient_descent(X, Y, iterations=2000)
    print(f"Estimated Weight: {estimated_weight}\nEstimated Bias: {estimated_bi

    # Making predictions using estimated parameters
    Y_pred = estimated_weight*X + estimated_bias

    # Plotting the regression line
    plt.figure(figsize = (8,6))
    plt.scatter(X, Y, marker='o', color='red')
    plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='blue',markerf
             markersize=10,linestyle='dashed')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()


if __name__=="__main__":
    main()
```

```
Iteration 1: Cost 4352.088931274409, Weight        0.7593291142562117, Bias
0.02288558130709
Iteration 2: Cost 1114.8561474350017, Weight        1.081602958862324, Bias
0.029180147485695 13
Iteration 3: Cost 341.42912086804455, Weight        1.2391274084945083, Bias
0.03225308846928192
Iteration 4: Cost 156.64495290904443, Weight        1.3161239281746984, Bias
0.03375132986012604
Iteration 5: Cost 112.49704004742098, Weight        1.3537591652024805, Bias
0.034479873154934775
Iteration 6: Cost 101.9493925395456, Weight        1.3721549833978113, Bias
0.03483219539286850 5
Iteration 7: Cost 99.4293893333546, Weight        1.3811467575154601, Bias
0.03500062439068245
Iteration 8: Cost 98.82731958262897, Weight        1.3855419247507244, Bias
0.03507916814736111
Iteration 9: Cost 98.68347500997261, Weight        1.3876903144657764, Bias
0.035113776874486774
Iteration 10: Cost 98.64910780902792, Weight        1.3887405007983562, Bias
0.035126910596389935
Iteration 11: Cost 98.64089651459352, Weight        1.389253895811451, Bias
0.03512954755833985
Iteration 12: Cost 98.63893428729509, Weight        1.38950491235671, Bias
0.035127053821718185
Iteration 13: Cost 98.63846506273883, Weight        1.3896276808137857, Bias
0.035122052266051224
Iteration 14: Cost 98.63835254057648, Weight        1.38968776283053, Bias
0.03511582492978764
Iteration 15: Cost 98.63832524036214, Weight        1.3897172043139192, Bias
0.03510899846107016
Iteration 16: Cost 98.63831830104695, Weight        1.389731668997059, Bias
0.035101879159522745
Iteration 17: Cost 98.63831622628217, Weight        1.389738813163012, Bias
0.03509461674147458
```



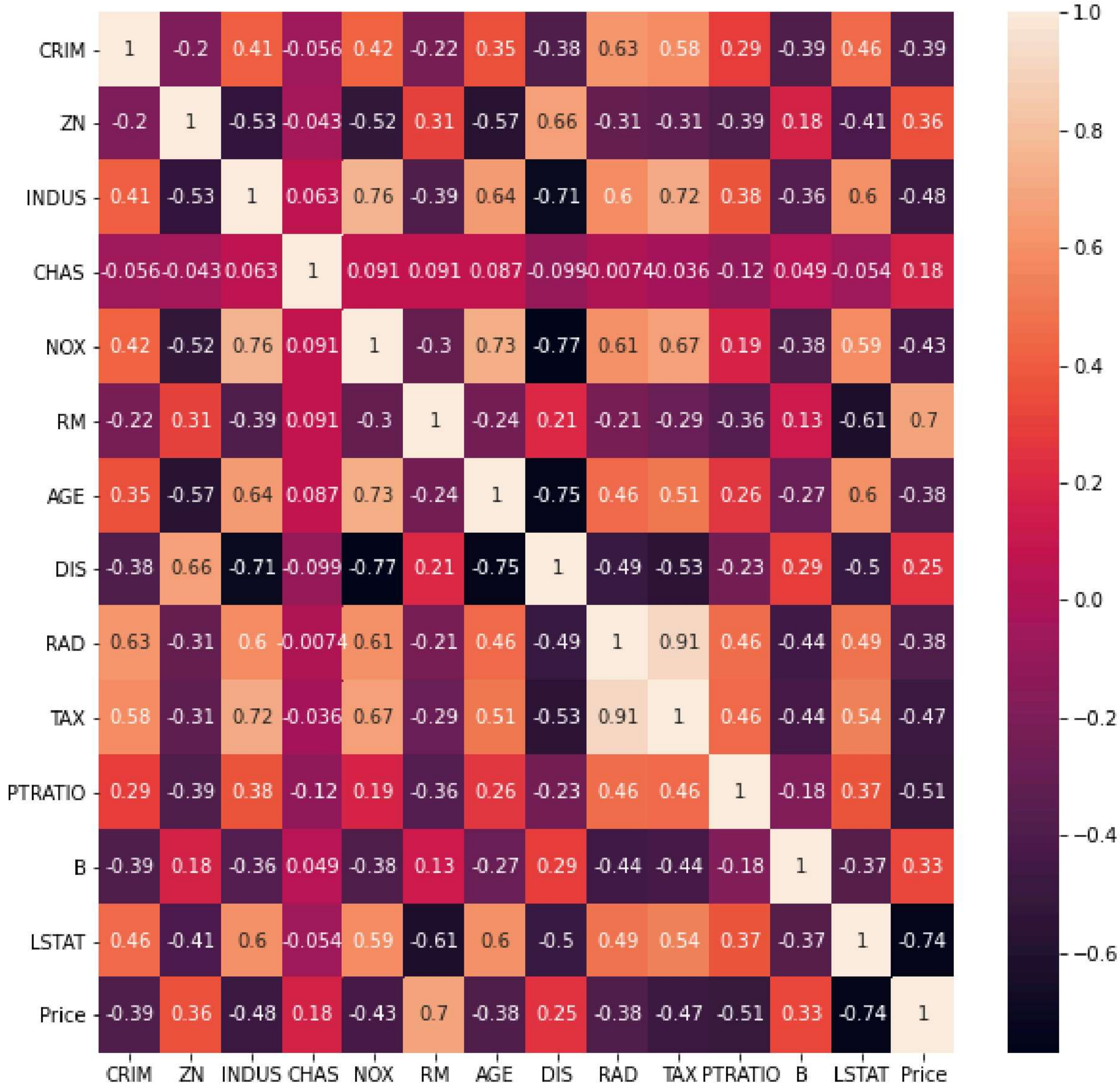

```
Estimated Weight: 1.389738813163012
Estimated Bias: 0.03509461674147458
```



lasso regression

In [31]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston

#data
boston = load_boston()
boston_df=pd.DataFrame(boston.data,columns=boston.feature_names)
#target variable
boston_df['Price']=boston.target
#preview
boston_df.head()

#Exploration
plt.figure(figsize = (10, 10))
sns.heatmap(boston_df.corr(), annot = True)
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x9af24f0>
```


CRIM 1 -0.2 0.41 -0.056 0.42 -0.22 0.35 -0.38 0.63 0.58 0.29 -0.39 0.46 -0.39
ZN -0.2 1 -0.53 -0.043 -0.52 0.31 -0.57 0.66 -0.31 -0.31 -0.39 0.18 -0.41 0.36
INDUS 0.41 -0.53 1 0.063 0.76 -0.39 0.64 -0.71 0.6 0.72 0.38 -0.36 0.6 -0.48
CHAS -0.056 -0.043 0.063 1 0.091 0.091 0.087 -0.099 -0.0074 -0.036 -0.12 0.049 -0.054 0.18
NOX 0.42 -0.52 0.76 0.091 1 -0.3 0.73 -0.77 0.61 0.67 0.19 -0.38 0.59 -0.43
RM -0.22 0.31 -0.39 0.091 -0.3 1 -0.24 0.21 -0.21 -0.29 -0.36 0.13 -0.61 0.7
AGE 0.35 -0.57 0.64 0.087 0.73 -0.24 1 -0.75 0.46 0.51 0.26 -0.27 0.6 -0.38
DIS -0.38 0.66 -0.71 -0.099 -0.77 0.21 -0.75 1 -0.49 -0.53 -0.23 0.29 -0.5 0.25
RAD 0.63 -0.31 0.6 -0.0074 0.61 -0.21 0.46 -0.49 1 0.91 0.46 -0.44 0.49 -0.38
TAX 0.58 -0.31 0.72 -0.036 0.67 -0.29 0.51 -0.53 0.91 1 0.46 -0.44 0.54 -0.47
PTRATIO 0.29 -0.39 0.38 -0.12 0.19 -0.36 0.26 -0.23 0.46 0.46 1 -0.18 0.37 -0.51
B -0.39 0.18 -0.36 0.049 -0.38 0.13 -0.27 0.29 -0.44 -0.44 -0.18 1 -0.37 0.33
LSTAT 0.46 -0.41 0.6 -0.054 0.59 -0.61 0.6 -0.5 0.49 0.54 0.37 -0.37 1 -0.74
Price -0.39 0.36 -0.48 0.18 -0.43 0.7 -0.38 0.25 -0.38 -0.47 -0.51 0.33 -0.74 1
CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT Price
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

```
In [ ]:
```

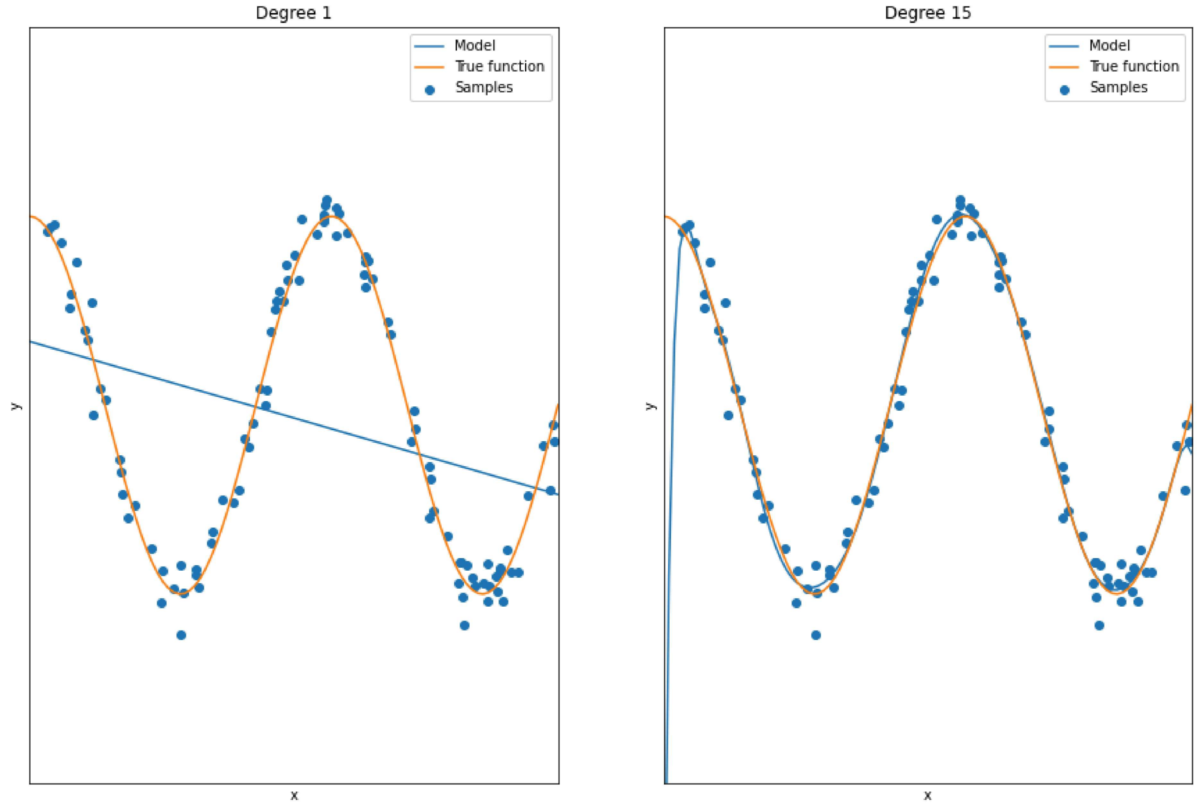
In [34]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings('ignore')
# Load the Titanic dataset
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titani
titanic_data = pd.read_csv(url)

# Drop rows with missing target values
titanic_data = titanic_data.dropna(subset=['Survived'])

# Select relevant features and target variable
X = titanic_data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
y = titanic_data['Survived']

# Convert categorical variable 'Sex' to numerical using .loc
X.loc[:, 'Sex'] = X['Sex'].map({'female': 0, 'male': 1})

# Handle missing values in the 'Age' column using .loc
X.loc[:, 'Age'].fillna(X['Age'].median(), inplace=True)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_rep)
```

```
Accuracy: 0.80

Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.85      0.83       105
           1       0.77      0.73      0.75        74

    accuracy                           0.80       179
   macro avg       0.79      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179
```

underfitting & overfitting

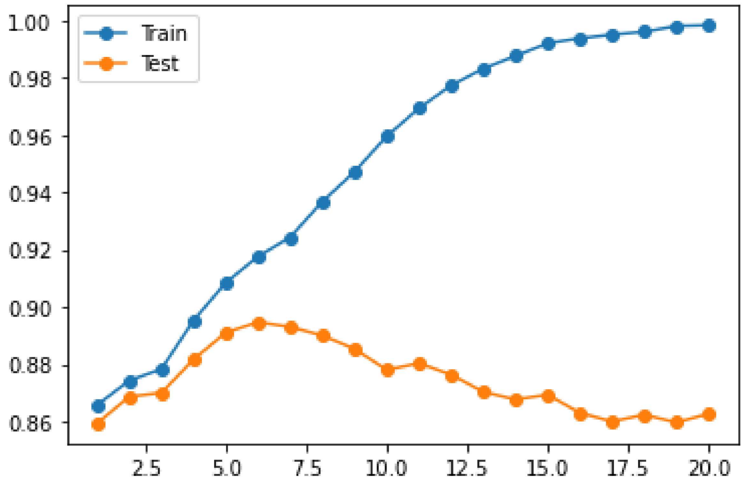
In [35]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
#this allows us to create a random dataset
X = np.sort(np.random.rand(100))
#Lets create a true function
true_f = lambda X: np.cos(3.5 * np.pi * X)
y = true_f(X) + np.random.randn(100) * 0.1
degrees = [1,15]
plt.figure(figsize=(15, 10))
for i in range(len(degrees)):
   ax = plt.subplot(1, len(degrees), i+1)
   plt.setp(ax, xticks=(), yticks=())
   polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=Fal
   linear_regression = LinearRegression()
   #creating a structure for operation
   pipeline = Pipeline([("polynomial_features", polynomial_features),("linear_r
   pipeline.fit(X[:, np.newaxis], y)
   #Testing
   X_test = np.linspace(0, 1, 100)
   yhat = pipeline.predict(X_test[:, np.newaxis])
   plt.plot(X_test, yhat,label="Model")
   plt.plot(X_test, true_f(X_test), label="True function")
   plt.scatter(X, y, label="Samples")
   plt.xlabel("x")
   plt.ylabel("y")
   plt.xlim((0, 1))
   plt.ylim((-2, 2))
   plt.legend(loc="best")
   plt.title("Degree %d" % degrees[i])
plt.show()
```


Degree 1
Model
True function
Samples
y
x
Degree 15
Model
True function
Samples
y
x

In [36]:

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot
# create dataset
X, y = make_classification(n_samples=10000, n_features=20, random_state=4)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# define lists to collect scores
train_scores, test_scores = list(), list()
# define the tree depths to evaluate
values = [i for i in range(1, 21)]
# evaluate a decision tree for each depth
for i in values:
    # configure the model
    model = DecisionTreeClassifier(max_depth=i)
    # fit model on the training dataset
    model.fit(X_train, y_train)
    # evaluate on the train dataset
    train_yhat = model.predict(X_train)
    train_acc = accuracy_score(y_train, train_yhat)
    train_scores.append(train_acc)
    # evaluate on the test dataset
    test_yhat = model.predict(X_test)
    test_acc = accuracy_score(y_test, test_yhat)
    test_scores.append(test_acc)
    # summarize progress
    print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))
# plot of train and test scores vs tree depth
pyplot.plot(values, train_scores, '-o', label='Train')
pyplot.plot(values, test_scores, '-o', label='Test')
pyplot.legend()
pyplot.show()
```

```
>1, train: 0.866, test: 0.859
>2, train: 0.874, test: 0.869
>3, train: 0.878, test: 0.870
>4, train: 0.895, test: 0.882
>5, train: 0.908, test: 0.891
>6, train: 0.918, test: 0.895
>7, train: 0.924, test: 0.893
>8, train: 0.937, test: 0.890
>9, train: 0.947, test: 0.885
>10, train: 0.960, test: 0.878
>11, train: 0.969, test: 0.880
>12, train: 0.977, test: 0.876
>13, train: 0.983, test: 0.870
>14, train: 0.988, test: 0.868
>15, train: 0.992, test: 0.869
>16, train: 0.994, test: 0.863
>17, train: 0.995, test: 0.860
>18, train: 0.996, test: 0.862
>19, train: 0.998, test: 0.860
>20, train: 0.998, test: 0.863
```



Train
Test
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0