Syed Karar
ML & DL Week 8-9

Supervised learning part 1

- ## Supervised vs. Unsupervised learning

Supervised:

Defined by its use of labeled datasets, which are designed to train or “supervise” algorithms into classifying data or predicting outcomes accurately. Using labeled inputs and outputs, the model can measure its accuracy and learn over time.

- Uses labeled input and output data.
- "Learns" from the training dataset by iteratively making predictions on the data and adjusting for the correct answer.
- Require upfront human intervention to label the data appropriately.
- **Goals:** Predict outcomes for new data. You know up front the type of results to expect.
- **Applications:** Spam detection, sentiment analysis, weather forecasting and pricing predictions.
- **Complexity:** Simple method, Typically calculated through the use of programs like R or Python
- **Types:** Classification & Regression

Unsupervised:

Uses machine learning algorithms to analyze and cluster unlabeled data sets. These algorithms discover hidden patterns in data without the need for human intervention.

- Does not needs labeled data.
- Work on their own to discover the inherent structure of unlabeled data.
- Still require some human intervention for validating output variables.
- **Goals:** Get insights from large volumes of new data. Ttself determines what is different or interesting from the dataset.
- **Applications:** Anomaly detection, recommendation engines, customer personas and medical imaging.
- **Complexity:** Need powerful tools for working with large amounts of unclassified data, Models are computationally complex because they need a large training set to produce intended outcomes.
- Used for three main **tasks:** Clustering, Association and Dimensionality reduction:

Semi-supervised learning:

- Where you use a training dataset with both labeled and unlabeled data.
- Particularly useful when it’s difficult to extract relevant features from data.
- Ideal for medical images, Where a small amount of training data can lead to a significant improvement in accuracy.

- # Difference b/w Classification & Regression

Regression:

- Regression is the task of predicting a continuous quantity.
- Method of calculation: By measurement of root mean square error.
- Find the best fit line, which can predict the output more accurately.
- Solve the regression problems such as Weather Prediction, House price prediction, etc.
- Algorithms: Regression tree (Random forest), Linear regression, Supports Vector Regression, etc.

Classification:

- Classification is the task of predicting a discrete class label.
- Method of calculation: by measuring accuracy
- Find the decision boundary, which can divide the dataset into different classes.
- Solve classification problems: Identification of spam emails, Speech Recognition, Identification of cancer cells, etc.
- Algorithms: Naive Bayes, decision trees, K-Nearest Neighbours, logistic regression, etc.

- # Types of Supervised learning

- ## Regression:
  - **Univariate linear regression**
    - Predicting a response(Y) using a single fearure(X) where, **X** is independent & **Y** is dependent variable
    - Assuming 2 variables are linearly realted.
    - Find linear functin that predicts (y), Gives Straight Line that best fits Data points.
    - **Equation:** y = mx +b
    - **How to find best fit line?**
      - Minimize errors in prediction by finding 'line of best fit'.
      - We try to minimize lenght b/w observed(Yi) & predicted value(Yp)

- **Code::**

Simple Linear Regression

$$y = b_0 + b_1 x_1$$

Multiple Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

Polynomial Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

In [1]:

```
# Import the Libraries
import pandas as pd; import numpy as np
import matplotlib.pyplot as plt; from scipy import stats
from scipy import stats
```

In [2]:

```
# Import DataSet
df = pd.read_csv('data_set.csv')
display(df.head(4))

# Check for Missing Data
print('Nan/null check:\n',df.isnull().any())
df.dropna(axis=0, inplace=True)
print(df.isnull().any()) # False indicates no NaN/null values

# Deleting 3rd column named "Sleep"
display(df.drop('Sleep', axis=1, inplace=True))
```

	Hours	Scores	Sleep
0	2.5	21.0	7.1
1	5.1	47.0	6.5
2	3.2	27.0	5.5
3	8.5	75.0	7.1

```
Nan/null check:
 Hours      True
Scores     True
Sleep      True
dtype: bool
Hours      False
Scores     False
Sleep      False
dtype: bool

None
```

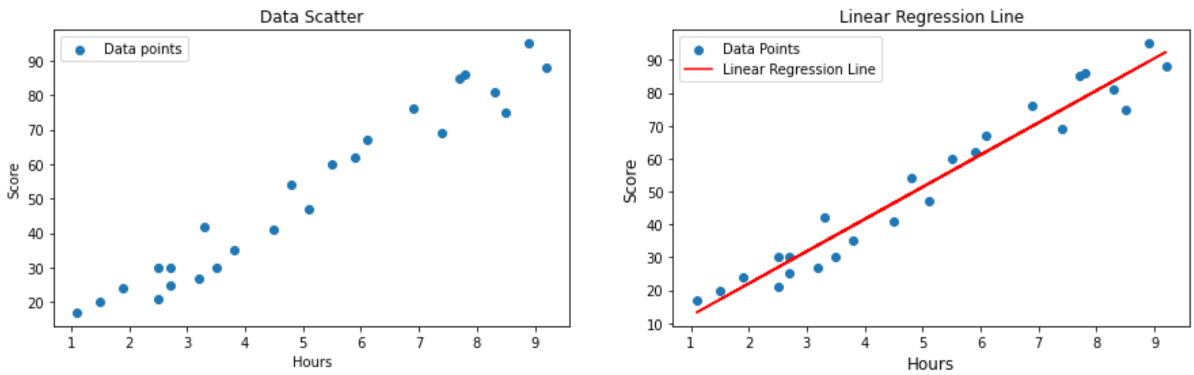
In [3]:

```
# Creating X & Yi
x,y = df['Hours'].values, df['Scores'].values

# Getting Slope(m), Intercept(c)
slope, intercept, r, p, std_error = stats.linregress(x,y)

# Function to predict Yp
def linearModel(x):
    return slope*x + intercept
predictions =list(map(linearModel, x))

print('Relationship Cofficient is: ',r)
print('Data set is a got fit for Linear Regression as the accuracy is:',r*
```

```
Relationship Cofficient is:  0.9761906560220887
Data set is a got fit for Linear Regression as the accuracy is: 97.61906560220
886
```

In [4]:

```
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(15, 4))

# Visulization
ax1.scatter(df.Hours,df.Scores, label='Data points'); ax1.legend(loc='best
ax1.set_title('Data Scatter'); ax1.set_xlabel('Hours');ax1.set_ylabel('Sco

# Visulize results/Regression line
ax2.scatter(x,y, label = 'Data Points')
ax2.plot(x, predictions, label = 'Linear Regression Line', c='red')
ax2.set_title('Linear Regression Line');ax2.legend(loc='best')
ax2.set_xlabel('Hours', fontsize=12);ax2.set_ylabel('Score', fontsize=12)
plt.show()
```



User can predict here:

In [5]:

```
user_inp = int(input('Enter study hours to predict score: '))
print('Score for', user_inp,'hours is: ', linearModel(user_inp))
```

```
Enter study hours to predict score: 8
Score for 8 hours is:  80.69010053167297
```

- ## Regression:
  - **Multiple linear regression**
    - Predicting a response(Y) using 2 or more fearure(X) by fitting linear equation.
    - Finds out which factor has highest impact on predicted output & how variables relate to each other.
    - **Assumptions:**
      - **Linearity:** Relationship b/w dependent & independent variable should be Linear.
      - **Homoscedasticity:** Equal or similar variances in different groups being compared.
      - **Multivariate Normality** Residuals are normally distributed.
      - **Lack of Multicollinearity** Little or no correlation in data.
    - **Equation:** y = bo + b1x1 + b2x2 + … + bnxn

- **Code::**

In [6]:
```
import pandas as pd; import numpy as np; import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
```

In [7]:
```
# Importing data
df = pd.read_csv('cars2.csv')
display(df.head(3),df.describe())
print('Nan/null check:\n',df.isna().any())
```

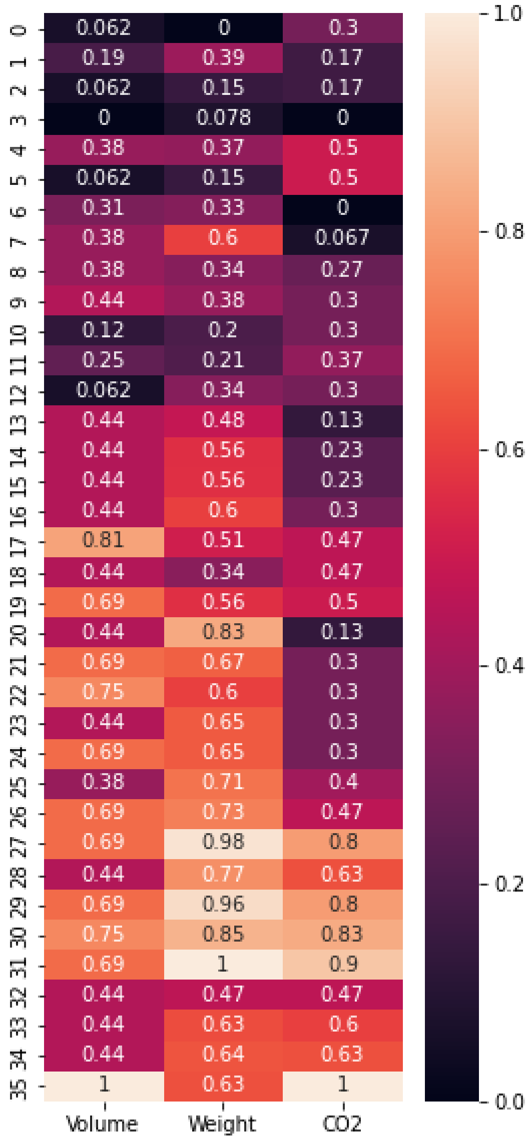
	Car	Model	Volume	Weight	CO2
0	Toyoty	Aygo	1.0	790	99
1	Mitsubishi	Space Star	1.2	1160	95
2	Skoda	Citigo	1.0	929	95

	Volume	Weight	CO2
count	36.000000	36.000000	36.000000
mean	1.611111	1292.277778	102.027778
std	0.388975	242.123889	7.454571
min	0.900000	790.000000	90.000000
25%	1.475000	1117.250000	97.750000
50%	1.600000	1329.000000	99.000000
75%	2.000000	1418.250000	105.000000
max	2.500000	1746.000000	120.000000

```
Nan/null check:
 Car         False
Model       False
Volume      False
Weight      False
CO2         False
dtype: bool
```

In [8]:

```python
# Scaling for headmap visualization
scalarX = MinMaxScaler(); scalarY = MinMaxScaler()
cx = scalarX.fit_transform(df[['Volume', 'Weight',]].values)
cy = scalarY.fit_transform((df['CO2'].values).reshape(-1,1))

# Creating DataFrame for scaled data
c = pd.DataFrame(cx, columns=['Volume','Weight']); c['CO2'] = cy

# Head-map visualization
plt.figure(figsize=(4,10))
sns.heatmap(data = c, annot=True, xticklabels=['Volume', 'Weight','CO2']);
```



In [9]:

```python
# Fitting model to train
x,y = df[['Volume','Weight']].values, df['CO2'].values
multiRegression = LinearRegression()
multiRegression.fit(x, y)
print('Value of R:',multiRegression.score(x,y))
```

```
Value of R: 0.37655640436199855
```

User can predict here:

In [10]:

```python
inp1 = float(input('---------------------------\nCO2 prediction system\
                   \n---------------------------\nEnter Volume of Car: '))
inp2 = float(input('Enter Weight of Car: '))
temp_inp = [[inp1, inp2]]
temp_pred = multiRegression.predict(temp_inp)
print('Predicted value of CO2: ',temp_pred)
```

```
---------------------------
CO2 prediction system
---------------------------
Enter Volume of Car: 1.2
Enter Weight of Car: 1100
Predicted value of CO2:  [97.36707032]
```

- ## Regression:
  - **Polynomial Regression**
    - Relationship between the independent variable **x** and the dependent variable **y** is described as an **nth degree** polynomial in x.
    - Fits **nonlinear relationship** between the value of x and the corresponding conditional mean of y, denoted **E(y |x)**
    - Data points are arranged in a non-linear fashion.
    - Higher degree (10 or 20) makes it pass throught all data points & reduce error **BUT** captures noise of data which is overfitting.
    - Two techniques:
      - **Forward Selection:** Increasing degree until it is significant enouth to define model.
      - **Backward Selection:** Decreasing degree until it is significant enouth to define model.
    - **Uses:**
      - Growth rate of tissues.
      - Disease epidemics
      - Distribution of carbon isotopes in lake sediments
    - **Equation:** y = a + b1x + b2x^2 +....+ bnx^n

- **Code::**

In [11]:

```python
from sklearn.metrics import r2_score
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as sp
```

In [12]:

```python
# Importing Dataset
df = pd.read_csv('data_inc.csv')
display(df.head(3))

# Depleting NaN/null rows
print('Length of DF before droping: ',len(df))
df.dropna(inplace=True)
print('Length of DF before droping: ',len(df))

# Change value form 'Salary' located at rown 12 to 83000
print("Value at 12 in 'Salary':", df.loc[19, 'Salary'])
df.loc[19, 'Salary'] = 83000

# Correcting  14 to 14.5 at 29 in 'Year of Exp'
print("Value at 29 in 'Year of Exp':",df.loc[27, 'Year_of_Experience'])
df.loc[27, 'Year_of_Experience'] = 14.5

# Duplicate check
print('Duplicates:',df.duplicated().any()) # False indicates that no duplic
```

	Year_of_Experience	Salary
0	1.0	25000.0
1	1.5	28000.0
2	2.0	30000.0

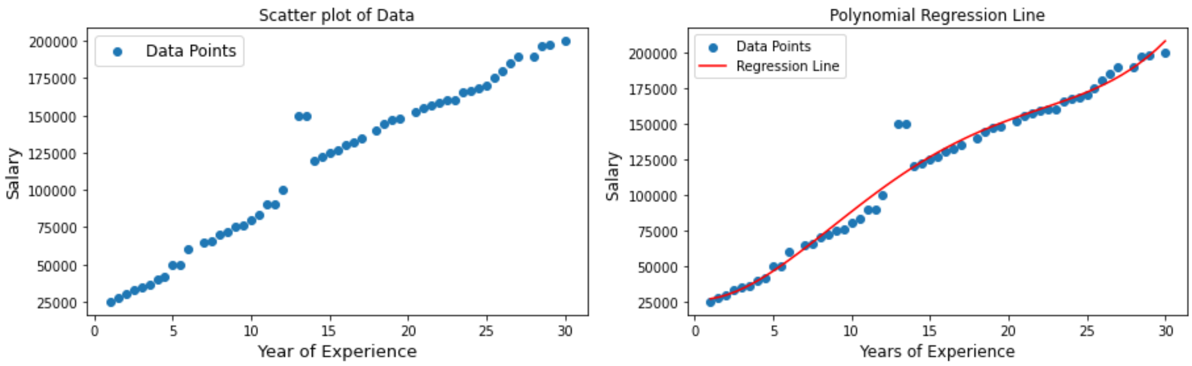
```
Length of DF before droping:  58
Length of DF before droping:  53
Value at 12 in 'Salary': 85000.0
Value at 29 in 'Year of Exp': 14.0
Duplicates: False
```

In [13]:

```python
x,y = df['Year_of_Experience'].values, df['Salary'].values

# Fitting model to train
polyModle = np.poly1d(np.polyfit(x,y,4))
```

In [14]:

```python
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(15, 4))

# Visualizing
ax1.scatter(x,y, label = 'Data Points')
ax1.set_title('Scatter plot of Data', fontsize = 12)
ax1.set_ylabel('Salary', fontsize=13);ax1.set_xlabel('Year of Experience',
ax1.legend(loc='best',fontsize=12)

# Creating x-axis values to plot prediction plot of model
myline = np.linspace(1,30,53)

# Visulize results/Regression Line
ax2.scatter(x,y,label='Data Points')
ax2.plot(myline, polyModle(myline), label='Regression Line', c='red')
ax2.set_xlabel('Years of Experience', fontsize=12); ax2.set_ylabel('Salary
ax2.set_title('Polynomial Regression Line'); ax2.legend(loc='best'); plt.sl
```



User can predict here:

In [15]:

```python
def_val = 20
chk1 = input('Enter "D" to predict using Default values\t OR\
             \tEnter "U" for custom values: ')
if chk1.upper() == 'D':
    print('\n{} is predicted salary against {}\
years of experience'.format(round(polyModle(def_val)),def_val))
elif chk1.upper() == 'U':
    temYear = float(input('Enter "Years of Experience"  to predict Salary:
    print('\n{} is predicted salary against {} \
years of experience'.format(round(polyModle(temYear)),temYear))
```

```
Enter "D" to predict using Default values        OR                       Enter
"U" for custom values: u
Enter "Years of Experience"  to predict Salary: 20

152309 is predicted salary against 20.0 years of experience
```

- **Train-Test split & Evalidation**

  Procedure of dividing dataset into two subsets. First subset is used to fit the model and it refers to as training dataset. Only input element form second subset is provided to mode, then predictions are made and compared to expected values.

  - **Train Dataset**: Used to fit the machine learning model.
  - **Test Dataset**: Used to evaluate the fit machine learning model.
  - When **dataset** is **large**, a costly model to train, or require a good estimate of model performance quickly.

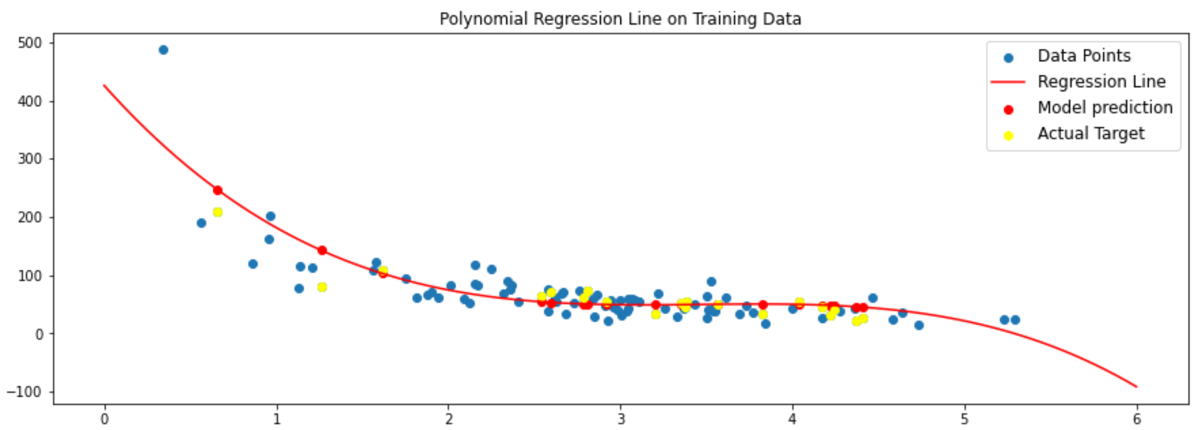
- Train-test split can be done throught **scikit-learn** machine learning library.
- We can **evaluate** ML algorithms for classification and regression on Test dataset.

- **Code::**

In [16]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score; np.random.seed(2)
```

In [17]:
```
# Generating data
x = np.random.normal(3, 1, 100); y = np.random.normal(150, 40, 100) / x

''' Manually Splitting data into Training & Testing 80-20 ratio
    Using slicing technique'''
train_x = x[:80]; train_y = y[:80]
test_x  = x[80:]; test_y  = y[80:]

'''Using sklearn library for Train-Test split'''
X_train, X_test, y_train, y_test = train_test_split(x,y, train_size=0.8, ra
```

In [18]:
```
# Fitting model to train
polyModle = np.poly1d(np.polyfit(X_train,y_train,3))

# Creating x-axis values to plot prediction plot of model
myline = np.linspace(0,6,100)

# Visulize results/Regression line
plt.figure(figsize=(15,5))
plt.scatter(x,y,label='Data Points')
plt.plot(myline, polyModle(myline), label='Regression Line', c='red')
plt.scatter(test_x, polyModle(test_x), c='red', label='Model prediction')
plt.scatter(test_x, test_y, c='yellow', label='Actual Target')
plt.title('Polynomial Regression Line on Training Data')
plt.legend(loc='best',fontsize=12); plt.show()
print('Value of R-square: ',r2_score(y_test, polyModle(X_test)))
```




```
Value of R-square:  0.43522237127396457
```

Supervised learning part 2

- ## Classification:
  - **Logistic Regression**:

    Measures the relationship between dependent variable (label,what we want to predict) and one or more independent variable (feature), by astimating probabilities using it's underlying logistic function.
    - It is used for **classification problem** when the dependent variable(target) is categorical.
    - Gives you a discrete binary outcome b/w 0 & 1.
    - **Example**:
      - A person will vote or not in upcoming elections.
      - predict whether an email is spam (1) or (0).
      - Whether the tumor is malignant (1) or not (0).
    - **Making Predictions**:
      - Probabilities are transformed into binary values to make prediction using Logistic function.
      - Sigmod function transforms them into 1 or 0 using a threshold classifier.
    - **Sigmoid Function**:
      - S-shape curve that can take any real-value number.
      - Maps it into a value b/w range of 0 & 1, BUT never exactly at those limits.

Note:- **Logistic regression** gives you a discrete outcome BUT **Linear regression** gives a continuous outcome.

- **Code::**

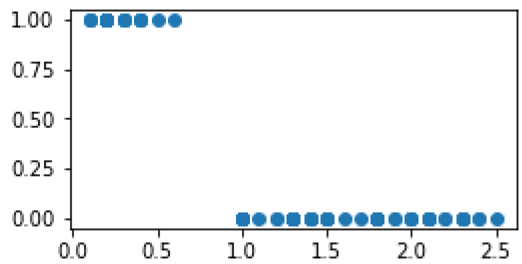
In [19]:

```python
import pandas as pd; import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [20]:

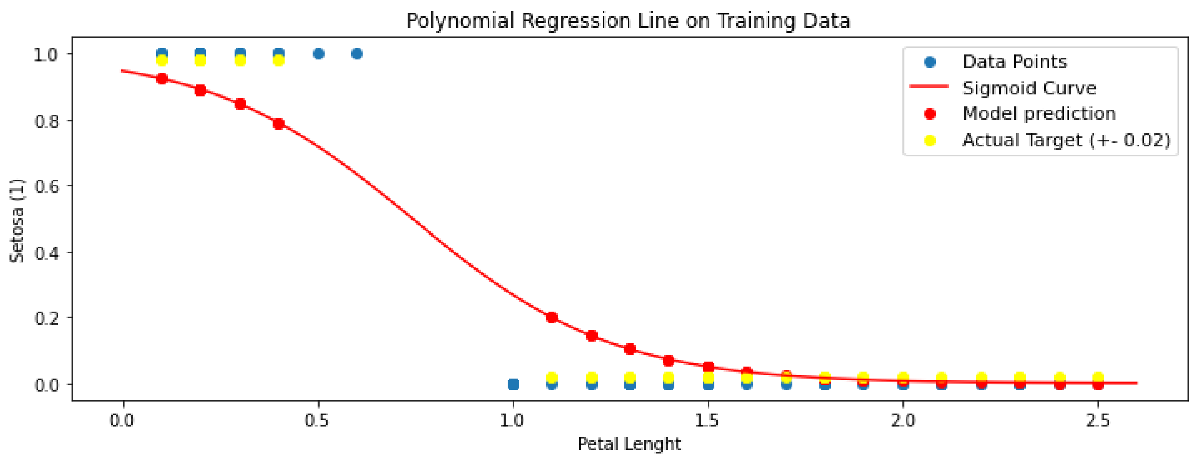
```
# Importing Data
df = pd.read_csv('iris.csv')
df.iloc[[1,50,100],:]

x = df.petal_width.values.reshape(-1,1)
y = (df.variety.values == 'Setosa').astype('int8')
print('Value of X:',x[0],'& Value of Y:',y[0])

# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(x,y, train_size=90, ra
y_test =np.where(y_test==1, 0.98, 0.02)
plt.figure(figsize=(4,2))
plt.scatter(X_train,y_train);plt.show()
```

```
Value of X: [0.2] & Value of Y: 1
```



In [21]:

```
# Fitting model to train
LogisModel = LogisticRegression()
LogisModel.fit(X_train,y_train)

# Creating x-axis values to plot prediction plot of model
myline = np.linspace(0,2.6,80).reshape(-1,1)
pre = LogisModel.predict_proba(myline)

# Visulize results/Regression line
plt.figure(figsize=(12,4))
plt.scatter(X_train,y_train, label='Data Points')
plt.plot(myline, pre[:,1], c='red', label='Sigmoid Curve')
plt.title('Polynomial Regression Line on Training Data')
plt.scatter(X_test, LogisModel.predict_proba(X_test.reshape(-1,1))[:,1], c
plt.scatter(X_test, y_test, c='yellow', label='Actual Target (+- 0.02)')
plt.xlabel('Petal Lenght'); plt.ylabel('Setosa (1)')
plt.legend(loc='best',fontsize=11); plt.show()
```



User can predict here:

In [22]:

```
def_val = 1.2
chk1 = input('Enter "D" to predict using Default values\t OR\
             \tEnter "U" for custom values: ')

def numTOname():
    global pre1
    if temp_pred == 0:
        pre1 = 'Not Setosa'
    elif temp_pred==1:
        pre1 = 'Setosa'

if chk1.upper() == 'D':
    temp_pred = LogisModel.predict([[def_val]])
    numTOname()
    print('\n{} is predicted against\
petal length {}'.format(pre1,def_val))
elif chk1.upper() == 'U':
    temYear = float(input('Enter petal length to predict Iris flower: '))
    temp_pred = LogisModel.predict([[temYear]])
    numTOname()
    print('\n{} is predicted against\
 petal length {}'.format(pre1,temYear))
else:
    print('Try again')
```

```
Enter "D" to predict using Default values        OR                      Enter
"U" for custom values: u
Enter petal length to predict Iris flower: 1

Not Setosa is predicted against petal length 1.0
```

- ## Classification:
  - **K-Nearest Neighbours**:

    KNN, which uses **proximity** to make predictions about the grouping of an individual data point. Working off the assumption that similar points can be found near one another.
    - KNN is simple yet most used classification algorithm. It can also be used for regression.
    - It is non-parametric (doesn't make any assumptions on underlying data.
    - Instance-based (algorithm doesn't learn a model. Instead it memorize training insurances).
    - **Lazy learning**, only stores a training dataset vs. undergoing a training stage, Memory hungry.
    - Doesn’t perform well with high-dimensional data inputs.
    - **Making Predections**:
      - Distance of object to labeled objects is computed, it's KNN are identified.
      - Majority of nearest neighbor is used to determine class label of object.
    - **The Distance**:
      - Euclidean distance as square root of sum of squared differences b/w new & existing point across all attributes.
    - **Value of k**:
      - Small k means noise, high variance, but low bias.
      - Large K make it computationally expensive & high bias and lower variance.

- **For classification problems**:
  - Class label (discrete values) is assigned on the basis of a majority vote.
  - Majority of greater than 50%, when there are only two categories. 4 categories can be classified with aa vote of greater than 25% and so on.
- **For Regression problems**:
  - Average KNN is taken to make prediction (continuous value) about classification.

- **Code::**

In [23]:
```
import pandas as pd; import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

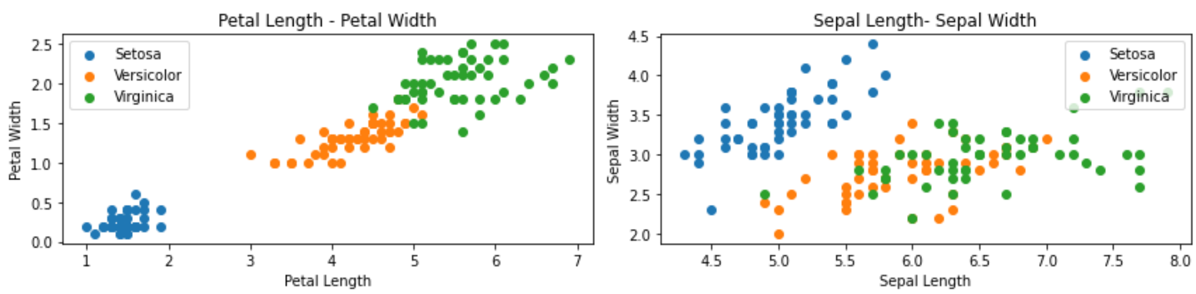
In [24]:
```
# Importing Data
iris = load_iris()
print(list(iris.keys()))

# Creating DataFrame
df = pd. DataFrame(data=iris['data'],
                   columns=["Sepal_length", "Sepal_width", "petal_length",
df['Target'] = iris['target']
display(df.iloc[[1,50,100],:]) # 0:setosa 1:versicolor 2:virginica
```

```
['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filenam
e', 'data_module']
```

	Sepal_length	Sepal_width	petal_length	petal_width	Target
1	4.9	3.0	1.4	0.2	0
50	7.0	3.2	4.7	1.4	1
100	6.3	3.3	6.0	2.5	2

In [25]:

```python
# Visulizing
fig,a =  plt.subplots(1,2,figsize=(12,3))
a[0].scatter(df.petal_length[df.Target == 0],
             df.petal_width[df.Target == 0],
             label='Setosa')
a[0].scatter(df.petal_length[df.Target == 1],
             df.petal_width[df.Target == 1],
             label='Versicolor')
a[0].scatter(df.petal_length[df.Target == 2],
             df.petal_width[df.Target == 2],
             label='Virginica')
a[0].set_title('Petal Length - Petal Width')
a[0].set_xlabel('Petal Length'); a[0].set_ylabel('Petal Width'); a[0].leger
# Visulizing
a[1].scatter(df.Sepal_length[df.Target == 0],
             df.Sepal_width[df.Target == 0],
             label='Setosa')
a[1].scatter(df.Sepal_length[df.Target == 1],
             df.Sepal_width[df.Target == 1],
             label='Versicolor')
a[1].scatter(df.Sepal_length[df.Target == 2],
             df.Sepal_width[df.Target == 2],
             label='Virginica')
a[1].set_title('Sepal Length- Sepal Width')
a[1].set_xlabel('Sepal Length'); a[1].set_ylabel('Sepal Width'); a[1].leger
plt.tight_layout();plt.show()
```



In [26]:

```python
x,y = iris['data'] , iris['target']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=.3, ra
print('There are {} samples in the training set and {} samples in the \
test set'.format(X_train.shape[0], X_test.shape[0]))

# Fitting model
knnmodel = KNeighborsClassifier()
knnmodel.fit(X_train,y_train)

# Checking accuracy
print('The accuracy of the knn classifier is {:.2f} out of 1 on \
training data'.format(knnmodel.score(X_train, y_train)))
print('The accuracy of the knn classifier is {:.2f} out of 1 on \
test data'.format(knnmodel.score(X_test, y_test)))
```

```
There are 105 samples in the training set and 45 samples in the test set
The accuracy of the knn classifier is 0.97 out of 1 on training data
The accuracy of the knn classifier is 0.98 out of 1 on test data
```

User can predict here:

In [27]:

```python
pre_val = [[3.1,1,1,1],[5.1,3.5,1.4,6.4],[9.1,9.5,0.2,0.2]]

first_chk = input('Enter "D" for default prediction values: \nOR\
 \nEnter "U" for custom user values: ')

def numTOname():
    global pre1
    if temp_pred == 0:
        pre1 = 'Setosa'
    elif temp_pred==1:
        pre1 = 'Versicolor'
    elif temp_pred==2:
        pre1 ='Virginica'

if first_chk.upper() == 'D':
    for i in pre_val:
        temp_pred = knnmodel.predict([i])
        numTOname()
        print('Result against {} is {}'.format(i, pre1))
elif first_chk.upper() == 'U':
    sl = float(input('Enter Sepal_length :'))
    sw = float(input('Enter Sepal_width :'))
    pl = float(input('Enter Petal_length :'))
    pw = float(input('Enter Petal_witdth :'))
    i = [[sl,sw,pl,pw]]
    temp_pred = knnmodel.predict(i)
    numTOname()
    print('Result against {} is {}'.format(i, pre1))
else:
    print('Try again')
```

```
Enter "D" for default prediction values:
OR
Enter "U" for custom user values: u
Enter Sepal_length :0.5
Enter Sepal_width :0.1
Enter Petal_length :1
Enter Petal_witdth :0.3
Result against [[0.5, 0.1, 1.0, 0.3]] is Setosa
```

- ## Classification:
  - **Naïve Bayes Classifier :**
    - A supervised learning algorithm, Based on Bayes theorem.
    - Mainly used in text classification that includes a complex/high-dimensional training dataset.
    - One of the simple and most effective Classification algorithm which can make quick predicitnos.
    - Require a relatively small number of training data samples to perform classification efficiently.
    - It's a **probabilistic classifier**, which means it predicts on the basis of the probability of an object.

- **Assumes** that there's no correlation between features in a dataset used to train the model.
- Examples: Spam filtration, Sentimental analysis, Credit Scoring, Medical data classification and classifying articles.
- **Bayes Theorem**:
  - Describes probability of a feature, based on prior knowledge of situations related to that feature.
  - Naive implies that every pair of features in the dataset is independent of each other
- **Zero frequency problem**:
  - No occurrences of a class label & a certain attribute value together then the frequency-based probability estimate will be zero. **&** If a instance in test data set has a category that was not present during training then it will assign it "Zero" probability and won't be able to make prediction.
- **Steps to Apply Bayes Theorem**:
  - Collect "raw" data.
  - Convert long data to a frequency table.
  - Row and column sums to get probabilities.
  - Apply probabilities from frequency table to Bayes theorem.

```
- Member attendance to Globo Gym given the weather:
```

Step 1

	weather	attended
0	sunny	yes
1	rainy	no
2	snowy	no
3	cloudy	yes

Step 2

	attended	
weather	no	yes
cloudy	1	3
rainy	2	1
snowy	3	1
sunny	1	3

Step 3

	weather probabilities	
cloudy	4/15	0.267
rainy	3/15	0.20
snowy	4/15	0.267
sunny	4/15	0.267
	attendance probabilities	
no	7/15	0.467
yes	8/15	0.533

Step 4

	Likelihood	
P(sunny yes)	3/8	0.375
	Class Prior Probability	
P(yes)	8/15	0.533
	Predictor Prior Probability	
P(sunny)	4/15	0.267
	Bayes Theorem values	
P(yes sunny)	(0.375 . 0.533) / 0.267	0.533

Note:- **Probability**: How likely an event X is to happen considering the total of potential outcomes.

- **Code::**

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

A, B = events

$P(A|B)$ = probability of A given B is true

$P(B|A)$ = probability of B given A is true

$P(A), P(B)$ = the independent probabilities of A and B

In [43]:

```
import pandas as pd; import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

In [44]:

```
df = pd.read_csv('titanic.csv')
df.drop(['PassengerId', 'Name','SibSp','Parch','Ticket','Cabin','Embarked']

# Replacing 'male' & 'female' with 1 & 0 respectively
df.Sex = np.where(df.Sex.values == 'male', 1, 0)
display(df.head())

# Filling Nan/null values with mean of the column
# if True it means this column has Nan/null values
print("Any null/Nan:",df.Age.isna().any())
df.Age = df.Age.fillna(df.Age.mean())
print("Any null/Nan:",df.Age.isna().any())
```

	Survived	Pclass	Sex	Age	Fare
0	0	3	1	22.0	7.2500
1	1	1	0	38.0	71.2833
2	1	3	0	26.0	7.9250
3	1	1	0	35.0	53.1000
4	0	3	1	35.0	8.0500

```
Any null/Nan: True
Any null/Nan: False
```

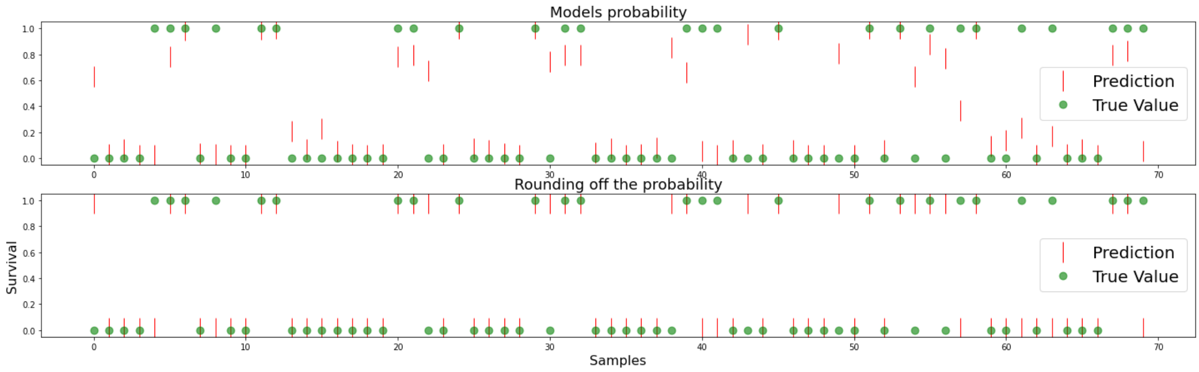
In [45]:

```
# Separation of labels form data
inputs1 = np.array(df[['Pclass', 'Sex', 'Age', 'Fare']])
target = np.array(df['Survived'])

# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(inputs1, target, test_s

# Fitting model
NBmodel = GaussianNB()
NBmodel.fit(X_train, y_train)

# Checking accuracy
scr = NBmodel.score(X_test, y_test)
print('Score of our model is {} OR {}%'.format(round(scr,3), round(scr*100)
```

```
Score of our model is 0.726 OR 73%
```

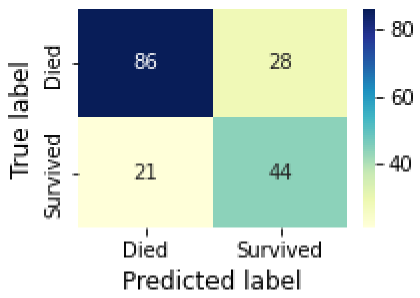
In [47]:

```
# Visualizing the results/probability
fig, (ax1, ax2) = plt.subplots(2, 1,figsize=(25, 7))

pred = NBmodel.predict_proba(X_test)[:70,1]
ax1.plot(pred, '|', color='red', label='Prediction ',markersize=25)
ax1.plot(y_test[:70],'o',color='green',label='True Value', alpha=0.6,marker
ax1.legend(fontsize =20); ax1.set_title('Models probability',fontsize =18)

pred = np.where(NBmodel.predict(X_test)[:70] == 1, 0.98, 0.02)
ax2.plot(pred, '|', color='red', label='Prediction ',markersize=25)
ax2.plot(y_test[:70],'o',color='green',label='True Value', alpha=0.6,marker
ax2.legend(fontsize =20); ax2.set_title('Rounding off the probability',font
ax2.set_ylabel('Survival',fontsize =16); ax2.set_xlabel('Samples',fontsize

plt.show()

#ax2.xlabel('Samples',fontsize=15); plt.ylabel('Survival probability',fonts
```



In [46]:

```python
# Confusion metric
confMdf = pd.DataFrame(confusion_matrix(y_test, NBmodel.predict(X_test)))
plt.figure(figsize = (3,2))
sns.heatmap(confMdf, xticklabels=['Died','Survived'],
            yticklabels=['Died','Survived'], annot=True,cmap="YlGnBu")
plt.xlabel('Predicted label', fontsize=12); plt.ylabel('True label', fonts
```



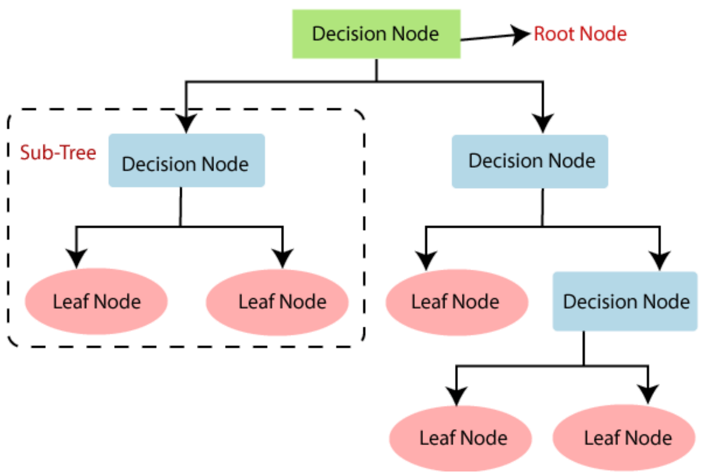
User can predict here:

In [32]:

```python
def numTOname():
    global pre1
    if temp_pred == 0:
        pre1 = 'Died'
    elif temp_pred==1:
        pre1 = 'Survived'

pre_val = [[3,1,18,7]]
first_chk = input('Enter "D" for default prediction values OR\
 Enter "U" for custom user values: ')

if first_chk.upper() == 'D':
    temp_pred = NBmodel.predict(pre_val); numTOname()
    print('Result against {} is {}'.format(i, pre1))
elif first_chk.upper() == 'U':
    sl = float(input('Enter Pclass (1,2 or 3):'))
    sw = float(input('Enter Sex (male:1 & female:0):'))
    pl = float(input('Enter Age :'))
    pw = float(input('Enter Fare (max 515):'))
    i = [[sl,sw,pl,pw]]; temp_pred = NBmodel.predict(i)
    numTOname()
    print('\nResult against {} is {}'.format(i, pre1))
else:
    print('Try again')
```

```
Enter "D" for default prediction values OR Enter "U" for custom user values: u
Enter Pclass (1,2 or 3):2
Enter Sex (male:1 & female:0):0
Enter Age :26
Enter Fare (max 515):7.9

Result against [[2.0, 0.0, 26.0, 7.9]] is Survived
```

- **Classification:**
  - **Decision Tree :**
    - Supervised learning technique, Used for both classification and Regression problems, but mostly used for Classification problems.
    - Called decision tree because constructs a tree-like structure, starts with root node, which expands on further branches.
    - **Internal nodes** represent features of a dataset, **branches** represent decision rules and each **leaf node** represents outcome.
    - Hole training set is considered as the root.
    - **Working**:
      - Starts from the root node of the tree.
      - Compares the values of root attribute with the record (real dataset) attribute.
      - Jumps to the next node based on the comparison.
      - Again compares the attribute value with the other sub-nodes and move further.
      - Continues the process until it reaches the leaf node of the tree.
      - Purity of the node increases with respect to the target variable as it splits node into 2 or more sub-nodes.
    - **Advantages**:
      - Useful for solving decision-related problems & easy to understand.
      - Helps to think about all the possible outcomes for a problem.
      - Less requirement of data cleaning compared to other algorithms.
    - **Disadvantage**:
      - Assumes that all features are independent/unrelated, so it cannot learn relationship between features.
    - **When to stop splitting**? Set maximum depth: length of the longest path from a root to a leaf.
    - **Pruning**: Removing the branches that make use of features having low importance, which increases performance.



In [33]:

```
import pandas as pd; import numpy as np
import matplotlib.pyplot as plt; from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
```

In [34]:

```python
df = pd.read_csv('salaries.csv')
display(df.head())
```

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0

In [35]:

```python
# Creating target and input data
inputs1 = df.drop('salary_more_then_100k', axis=1)
target = df['salary_more_then_100k']

# Encoding levels of categorical features into numeric values
le_comp = LabelEncoder(); le_job = LabelEncoder(); le_degree = LabelEncode

inputs1['company_n'] = le_comp.fit_transform(inputs1['company'])
inputs1['job_n'] = le_job.fit_transform(inputs1['job'])
inputs1['degree_n'] = le_degree.fit_transform(inputs1['degree'])

inputs1.drop(['company','job','degree'], axis=1, inplace= True)
display(inputs1.head(3), target[:3])
```

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0

```
0    0
1    0
2    1
Name: salary_more_then_100k, dtype: int64
```

In [36]:

```python
# Fitting model
DTreemodel = DecisionTreeClassifier()
DTreemodel.fit(inputs1.values, target.values)

# Checking accuracy
print('Score of our model is', DTreemodel.score(inputs1.values,target.value
```

```
Score of our model is 1.0
```

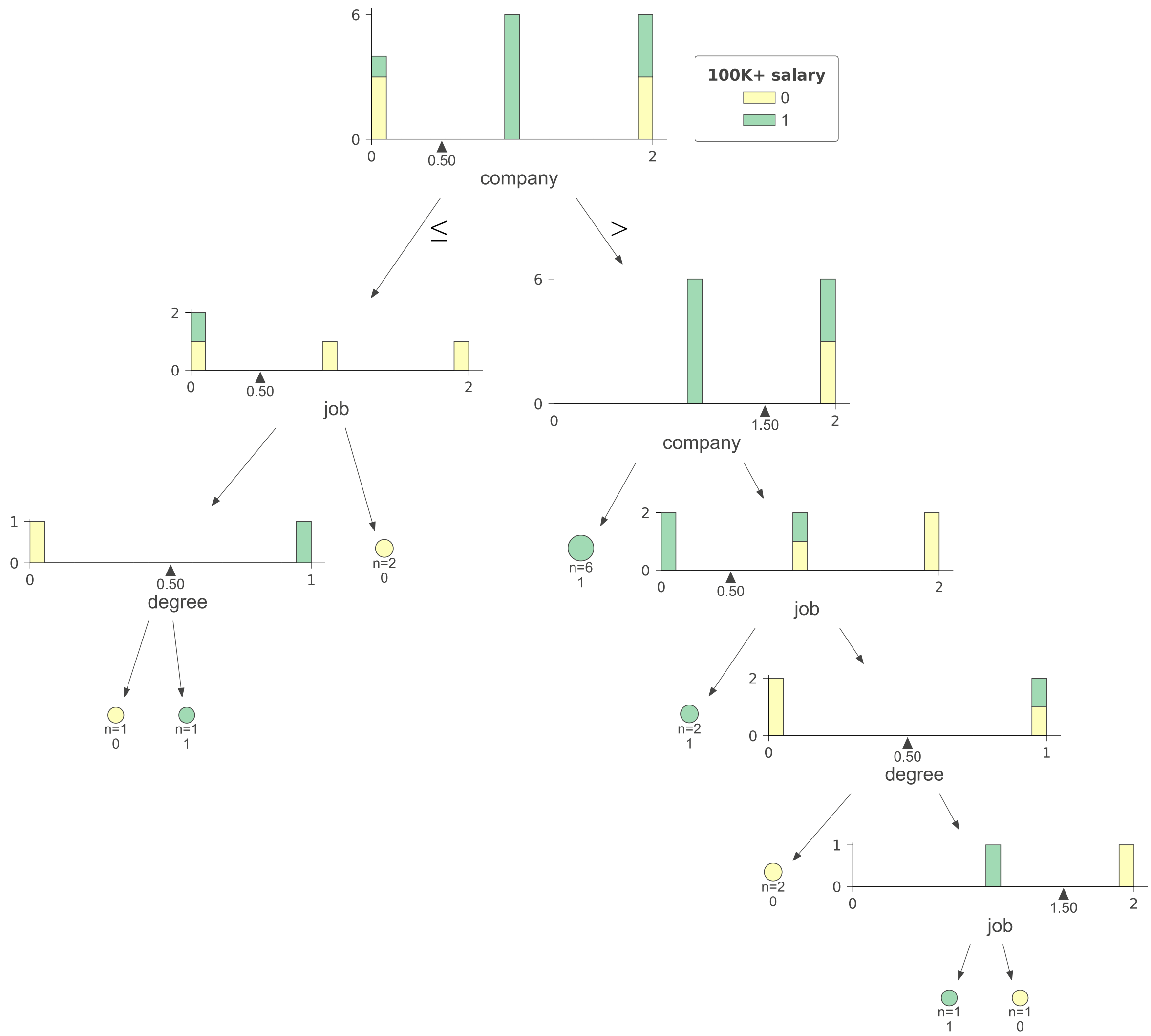
In [40]:

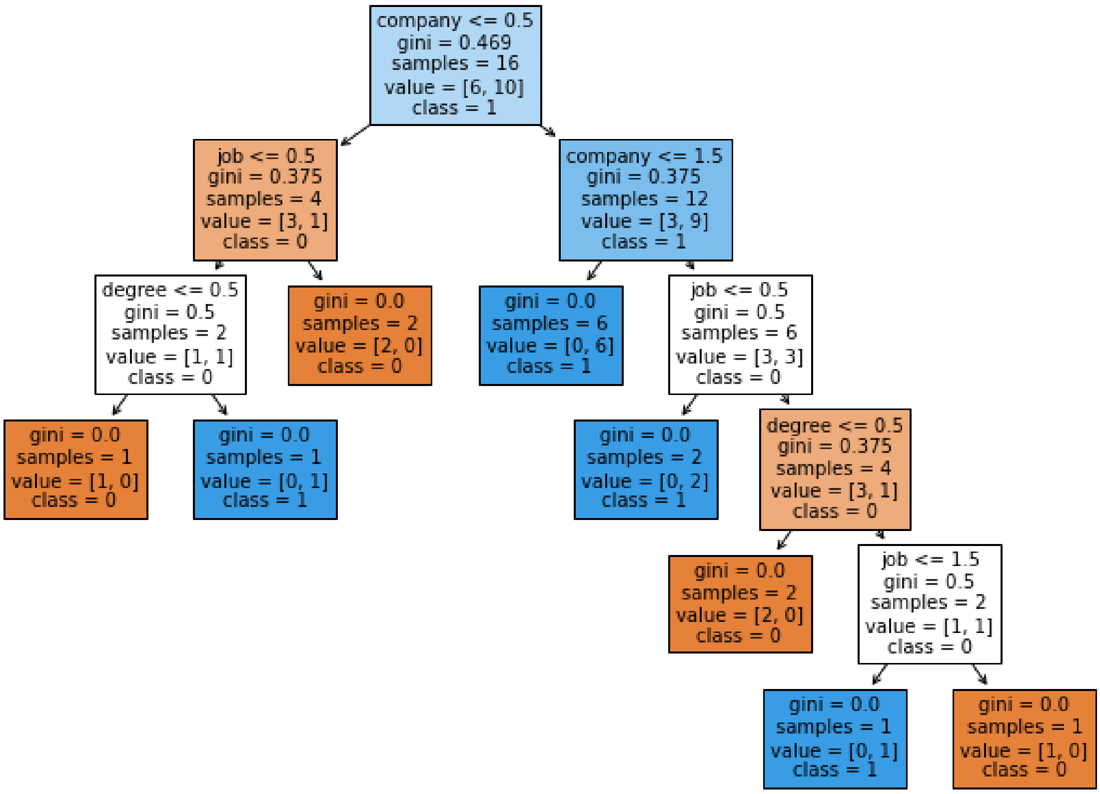
```python
from dtreeviz.trees import dtreeviz # remember to set in user path
import graphviz; from sklearn import tree
```

In [41]:

```
# Visualizing Tree; sklear.tree 2nd option
plt.figure(figsize=(11,8))
ax1 = tree.plot_tree(DTreemodel,
                     feature_names=['company', 'job', 'degree'],
                     class_names=['0','1'], filled=True, fontsize=10)
ax2 = dtreeviz(
    DTreemodel,inputs1.values,target.values,target_name='100K+ salary',
    feature_names = ['company', 'job', 'degree'],scale=1.1,
    class_names=['0','1'],ticks_fontsize= 9,)
ax2
```

Out[41]:





User can predict here:

In [42]:

```
def numTOname():
    global pre1
    if temp_pred == 0:
        pre1 = 'Salary less then 100K'
    elif temp_pred==1:
        pre1 = 'Salary More then 100K'

pre_val = [[2,0,0]]
first_chk = input('Enter "D" for default prediction values OR\
 Enter "U" for custom user values: ')

if first_chk.upper() == 'D':
    temp_pred = DTreemodel.predict(pre_val); numTOname()
    print('Result against {} is: {}'.format(i, pre1))
elif first_chk.upper() == 'U':
    sl = float(input('Enter Company  [0,1 or 2]:'))
    sw = float(input('Enter Job [0,1 or 2]):'))
    pl = float(input('Enter Degress [0 or 1]:'))
    i = [[sl,sw,pl]]; temp_pred = DTreemodel.predict(i)
    numTOname()
    print('\nResult against {} is {}'.format(i, pre1))
else:
    print('Try again')
```

```
Enter "D" for default prediction values OR Enter "U" for custom user values: u
Enter Company  [0,1 or 2]:2
Enter Job [0,1 or 2]):0
Enter Degress [0 or 1]:0

Result against [[2.0, 0.0, 0.0]] is Salary More then 100K
```