```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib as mpl
        %matplotlib inline
        import matplotlib.pyplot as plt
        plt.rcParams['font.family'] = 'sans-serif'
        plt.rcParams['font.sans-serif'] = 'Helvetica'
        plt.rcParams['axes.unicode_minus'] = False
        import seaborn as sns
        sns.set_style('whitegrid')
        %matplotlib inline
        %load_ext rpy2.ipython

In [2]: data = pd.read_csv('data/bank.csv')
        data.shape

Out[2]: (4521, 17)

        data.head()

Out[3]:   age  job  marital  education  default  balance  housing  loan  \
        0   30   unemployed  married    primary    no     1787    no    no
        1   33   services    married    secondary  no     4789    yes   yes
        2   35   management  single     tertiary   no     1350    yes   no
        3   30   management  married    tertiary   no     1476    yes   yes
        4   59   blue-collar married    secondary  no     0       yes   no

           contact  day  month  duration  campaign  pdays  previous  poutcome  \
        0  cellular  19   oct    79        1         -1     0         unknown
        1  cellular  11   may    220       1         339    4         failure
        2  cellular  16   apr    185       1         330    1         failure
        3  unknown   5    may    199       4         -1     0         unknown
        4  unknown   5    may    226       1         -1     0         unknown

           y
        0  no
        1  no
        2  no
        3  no
        4  no

In [4]: data.dtypes

Out[4]: age            int64
        job           object
        marital       object
        education     object
        default       object
        balance        int64
        housing       object
        loan          object
        contact       object
        day            int64
        month         object
        duration       int64
        campaign       int64
        pdays          int64
        previous       int64
        poutcome      object
        y             object
        dtype: int64

In [5]: data.describe()

Out[5]:          age      balance  day     duration  campaign  pdays  \
        count  4521.000000  4521.000000  4521.000000  4521.000000  4521.000000  4521.000000
        mean   41.170095  1422.782519  15.915284  263.961292  2.793630  39.766645
        std    10.576211  3009.638150  8.247667  259.856633  3.109992  100.121124
        min    19.000000  -3313.000000  1.000000  4.000000  1.000000  -1.000000
        25%    33.000000  69.000000  9.000000  104.000000  1.000000  -1.000000
        50%    39.000000  444.000000  16.000000  185.000000  2.000000  -1.000000
        75%    49.000000  1480.000000  21.000000  329.000000  3.000000  33.000000
        max    87.000000  71188.000000  31.000000  3025.000000  50.000000  871.000000

                previous
        count  4521.000000
        mean   0.542579
        std    1.693562
        min    0.000000
        25%    0.000000
        50%    0.000000
        75%    1.000000
        max    25.000000

In [6]: data.isnull()

Out[6]:      age  job  marital  education  default  balance  housing  loan  \
        0  False  False  False  False  False  False  False  False
        1  False  False  False  False  False  False  False  False
        2  False  False  False  False  False  False  False  False
        3  False  False  False  False  False  False  False  False
        4  False  False  False  False  False  False  False  False

           contact  day  month  duration  campaign  pdays  previous  poutcome  \
        0  False  False  False  False  False  False  False  False
        1  False  False  False  False  False  False  False  False
        2  False  False  False  False  False  False  False  False
        3  False  False  False  False  False  False  False  False
        4  False  False  False  False  False  False  False  False

           y
        0  False
        1  False
        2  False
        3  False
        4  False
```

Checking Null Values

```
In [7]: [sum(data.isnull().iloc[:, i]) for i in range(data.shape[-1])]

Out[7]: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

        [sum(data.isnull().iloc[:, i]) / data.shape[0] for i in range(data.shape[-1])]

Out[8]: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

        [sum(data.isnull().iloc[:, i]) for i in range(data.shape[-1])]

Out[9]: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

        [sum(data.isnull().iloc[:, i]) / data.shape[0] for i in range(data.shape[-1])]

Out[10]: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

No data is missing in our data field values. Hence, we will skip this part of data cleaning.

Finding Outliers' Univariate Analysis

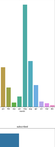
```
In [11]: # Find the column dataframes for job, edu, marital, pdays, campaign, balance and previous variables to see how
         # many values are there
         job = data[['job', 'y']].groupby('job').count.reset_index().sort_values('y', ascending=False)
         edu = data[['education', 'y']].groupby('education').count.reset_index().sort_values('y', ascending=False)
         marital = data[['marital', 'y']].groupby('marital').count.reset_index().sort_values('y', ascending=False)
         pdays = data[['pdays', 'y']].groupby('pdays').count.reset_index().sort_values('y', ascending=False)
         campaign = data[['campaign', 'y']].groupby('campaign').count.reset_index().sort_values('y', ascending=False)
         balance = data[['balance', 'y']].groupby('balance').count.reset_index().sort_values('y', ascending=False)
         previous = data[['previous', 'y']].groupby('previous').count.reset_index().sort_values('y', ascending=False)

In [12]: plt.figure(figsize=(8,6))
         sns.boxplot(x='y',
                     y='age',
                     data=data,
                     palette='Set2')
         plt.title('age')
         plt.ylabel('age')
         plt.xlabel('y')
         plt.xticks(rotation=0)
         plt.show()
         plt.figure(figsize=(8,6))
         sns.boxplot(x='y',
                     y='balance',
                     data=data,
                     palette='Set2')
         plt.title('balance')
         plt.ylabel('balance')
         plt.xlabel('y')
         plt.xticks(rotation=0)
         plt.show()
```

So we can see that we have outlier present in balance. Age has campaign,pdays and previous. We will not be removing any Outlier as these are outlier and not error but legitimate data points.

```
In [13]: plt.figure(figsize=(8,6))
         ax = plt.subplot(1,2,1)
         sns.countplot(x='y',
                       data=data,
                       palette='Set2')
         plt.title('y')
         plt.ylabel('count')
         plt.xlabel('y')
         plt.xticks(rotation=0)
         ax = plt.subplot(1,2,2)
         sns.countplot(x='job',
                       data=data,
                       palette='Set2')
         plt.title('job')
         plt.ylabel('count')
         plt.xlabel('job')
         plt.xticks(rotation=90)
         plt.show()
```

```
In [14]: plt.figure(figsize=(8,6))
         ax = plt.subplot(1,2,1)
         sns.countplot(x='education',
                       data=data,
                       palette='Set2')
         plt.title('education')
         plt.ylabel('count')
         plt.xlabel('education')
         plt.xticks(rotation=0)
         ax = plt.subplot(1,2,2)
         sns.countplot(x='default',
                       data=data,
                       palette='Set2')
         plt.title('default')
         plt.ylabel('count')
         plt.xlabel('default')
         plt.xticks(rotation=0)
         plt.show()
```




```
In [15]: plt.figure(figsize=(8,6))
         ax = plt.subplot(1,2,1)
         sns.countplot(x='housing',
                       data=data,
                       palette='Set2')
         plt.title('housing')
         plt.ylabel('count')
         plt.xlabel('housing')
         plt.xticks(rotation=0)
         ax = plt.subplot(1,2,2)
         sns.countplot(x='loan',
                       data=data,
                       palette='Set2')
         plt.title('loan')
         plt.ylabel('count')
         plt.xlabel('loan')
         plt.xticks(rotation=0)
         plt.show()
```

```
In [16]: plt.figure(figsize=(8,6))
         ax = plt.subplot(1,2,1)
         sns.countplot(x='contact',
                       data=data,
                       palette='Set2')
         plt.title('contact')
         plt.ylabel('count')
         plt.xlabel('contact')
         plt.xticks(rotation=0)
         ax = plt.subplot(1,2,2)
         sns.countplot(x='month',
                       data=data,
                       palette='Set2')
         plt.title('month')
         plt.ylabel('count')
         plt.xlabel('month')
         plt.xticks(rotation=0)
         plt.show()
```

```
In [17]: plt.figure(figsize=(8,6))
         ax = plt.subplot(1,2,1)
         sns.countplot(x='poutcome',
                       data=data,
                       palette='Set2')
         plt.title('poutcome')
         plt.ylabel('count')
         plt.xlabel('poutcome')
         plt.xticks(rotation=0)
         ax = plt.subplot(1,2,2)
         sns.countplot(x='education',
                       data=data,
                       palette='Set2')
         plt.title('education')
         plt.ylabel('count')
         plt.xlabel('education')
         plt.xticks(rotation=0)
         plt.show()
```


1. Most of the Clients are married and have a job in the people having Blue collar, Management and Technician jobs with students are the least.
2. We have a very few Clients that are defaulters.
3. Most of the Clients have housing loan.
4. Most of the clients were made on the month of May.
5. Most of the Clients are in the Age group of 30-40 years.
6. Most of the Clients have salary lies in the range of 0-2000.
7. Most of the Clients have salary lies in a term deposit.

We have a problem for the age as we need to convert age into categories. We will convert the age into categories in bucks and can't take mean.

```
In [18]: data['job'].unique()

Out[18]: array(['unemployed', 'services', 'management', 'blue-collar', 'self-employed',
                 'technician', 'entrepreneur', 'retired', 'admin.', 'student',
                 'housemaid', 'unknown'], dtype=object)

         data['job'].value_counts()

Out[19]: management    969
         blue-collar   946
         technician    768
         admin.        478
         services      417
         retired       230
         self-employed 183
         entrepreneur  168
         unemployed    128
         housemaid     112
         student       84
         unknown       38
         Name: job, dtype: int64

         data['education'].unique()

Out[20]: array(['primary', 'secondary', 'tertiary', 'unknown'], dtype=object)

         data['education'].value_counts()

Out[21]: secondary    2306
         tertiary     1350
         primary      678
         unknown      187
         Name: education, dtype: int64

         data['marital'].unique()

Out[22]: array(['married', 'single', 'divorced'], dtype=object)

         data['marital'].value_counts()

Out[23]: married    2797
         single     1196
         divorced   528
         Name: marital, dtype: int64

         data['pdays'].unique()

Out[24]: array([ -1, 339, 330, 176, 239, 307, 148, 341, 351, 182, 172, 304,
                 183, 328, 171, 300, 294, 244, 349, 342, 299, 180, 297, 295,
                 178, 301, 354, 308, 187, 293, 353, 174, 302, 296, 179, 177,
                 298, 303, 175, 173, 181, 305, 306, 340, 338, 337, 336, 335,
                 334, 333, 332, 331, 329, 327, 326, 325, 324, 323, 322, 321,
                 320, 319, 318, 317, 316, 315, 314, 313, 312, 311, 310, 309,
                 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366,
                 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378,
                 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390,
                 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,
                 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414,
                 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426,
                 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438,
                 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450,
                 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462,
                 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474,
                 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486,
                 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498,
                 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510,
                 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522,
                 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534,
                 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546,
                 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,
                 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570,
                 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582,
                 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594,
                 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606,
                 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618,
                 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630,
                 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642,
                 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654,
                 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666,
                 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678,
                 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690,
                 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702,
                 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714,
                 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726,
                 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738,
                 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750,
                 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762,
                 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774,
                 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786,
                 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798,
                 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810,
                 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822,
                 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834,
                 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846,
                 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858,
                 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870,
                 871])

         data['pdays'].value_counts()

Out[25]: -1    3705
         182   35
         184   33
         91    29
         92    28
         183   28
         181   27
         370   25
         188   25
         370   25
         92    25
         183   25
         181   25
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
```

```
label_encoder = LabelEncoder()
df['job'] = label_encoder.fit_transform(df['job'])
df['marital'] = label_encoder.fit_transform(df['marital'])
df['education'] = label_encoder.fit_transform(df['education'])
df['default'] = label_encoder.fit_transform(df['default'])
df['housing'] = label_encoder.fit_transform(df['housing'])
df['loan'] = label_encoder.fit_transform(df['loan'])
df['contact'] = label_encoder.fit_transform(df['contact'])
df['month'] = label_encoder.fit_transform(df['month'])
df['poutcome'] = label_encoder.fit_transform(df['poutcome'])
df['y'] = label_encoder.fit_transform(df['y'])
```

```
X = df.drop(['y'], axis=1)
y = df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train.shape, y_train.shape
```

```
((36168, 16), (36168,))
```

```
X_train.head()
```

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
31	32	7	2	2	0	0	0	2	20	0	97	1	-1	0	3
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3

```
y_train.shape
```

```
(36168, )
```

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
31	32	7	2	2	0	0	0	2	20	0	97	1	-1	0	3
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3

```
y_train.head(5)
```

```
34811    0
31       0
34811    0
40461    0
40461    0
Name: y, dtype: int64
```

```
y_train.shape, y_test.shape
```

```
((36168, ), (9042, ))
```

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
31	32	7	2	2	0	0	0	2	20	0	97	1	-1	0	3
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3

```
y_train.head(5), y_test.head(5)
```

```
(34811    0
31       0
34811    0
40461    0
40461    0
Name: y, dtype: int64,
28000    0
7000     0
7000     0
7000     0
7000     0
Name: y, dtype: int64)
```

```
y_train = y_train.astype('int')
y_test = y_test.astype('int')
```

```
y_train.shape, y_test.shape
```

```
((36168, ), (9042, ))
```

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
31	32	7	2	2	0	0	0	2	20	0	97	1	-1	0	3
34811	31	4	1	1	0	1	0	0	14	6	142	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3
40461	35	4	1	2	0	1	0	0	20	7	407	1	-1	0	3

```
y_train.head(5), y_test.head(5)
```

```
(34811    0
31       0
34811    0
40461    0
40461    0
Name: y, dtype: int64,
28000    0
7000     0
7000     0
7000     0
7000     0
Name: y, dtype: int64)
```

```
y_train.shape, y_test.shape
```

```
((36168, ), (9042, ))
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
y_pred = model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[7952 1000]
 [ 690 400]]
```

```
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
0.8800486618004866
```

```
# Calculate the precision and recall
precision = cm[1,1] / (cm[0,1] + cm[1,1])
recall = cm[1,1] / (cm[1,0] + cm[1,1])

# Print the results
print("Precision: ", precision)
print("Recall: ", recall)
```

```
Precision:  0.7142857142857143
Recall:  0.3670033670033670
```

```
import seaborn as sns
sns.heatmap(cm, annot=True, fmt='d')
```


Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
X_train = DecisionTreeClassifier()
X_train.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# Plot the decision tree
plot_tree(X_train, filled=True, feature_names=X_train.columns, class_names=['0', '1'])
```

```
[Text(0.5, 0.5, 'age <= 60.5\ngini = 0.233\nsamples = 36168\nvalue = [31952, 4216]'),
 Text(0.2, 0.5, 'job <= 9.5\ngini = 0.227\nsamples = 35900\nvalue = [31952, 3948]'),
 Text(0.1, 0.5, 'marital <= 0.5\ngini = 0.224\nsamples = 35863\nvalue = [31952, 3911]'),
 Text(0.05, 0.5, 'education <= 2.5\ngini = 0.209\nsamples = 8\nvalue = [6, 2]'),
 Text(0.04, 0.5, 'poutcome <= 2.5\ngini = 0.167\nsamples = 7\nvalue = [6, 1]'),
 Text(0.03, 0.5, 'day <= 15.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.02, 0.5, 'month <= 4.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.01, 0.5, 'contact <= 0.5\ngini = 0.667\nsamples = 2\nvalue = [1, 1]'),
 Text(0.0, 0.5, 'pdays <= 0.5\ngini = 0.000\nsamples = 1\nvalue = [1, 0]'),
 Text(0.0, 0.5, '0'),
 Text(0.0, 0.5, '1'),
 Text(0.0, 0.5, '0'),
 Text(0.04, 0.5, '0'),
 Text(0.06, 0.5, '0'),
 Text(0.07, 0.5, 'campaign <= 1.5\ngini = 0.000\nsamples = 1\nvalue = [0, 1]'),
 Text(0.06, 0.5, '1'),
 Text(0.07, 0.5, '0'),
 Text(0.08, 0.5, '0'),
 Text(0.09, 0.5, '0'),
 Text(0.11, 0.5, 'job <= 7.5\ngini = 0.224\nsamples = 35855\nvalue = [31952, 3903]'),
 Text(0.09, 0.5, 'marital <= 1.5\ngini = 0.221\nsamples = 35811\nvalue = [31952, 3859]'),
 Text(0.08, 0.5, 'education <= 1.5\ngini = 0.219\nsamples = 26811\nvalue = [24000, 2811]'),
 Text(0.07, 0.5, 'poutcome <= 2.5\ngini = 0.215\nsamples = 18811\nvalue = [17000, 1811]'),
 Text(0.06, 0.5, 'day <= 27.5\ngini = 0.209\nsamples = 17811\nvalue = [16000, 1811]'),
 Text(0.05, 0.5, 'month <= 8.5\ngini = 0.206\nsamples = 17800\nvalue = [16000, 1800]'),
 Text(0.04, 0.5, 'contact <= 1.5\ngini = 0.203\nsamples = 17789\nvalue = [16000, 1789]'),
 Text(0.03, 0.5, 'pdays <= 0.5\ngini = 0.201\nsamples = 17778\nvalue = [16000, 1778]'),
 Text(0.02, 0.5, 'job <= 4.5\ngini = 0.198\nsamples = 17767\nvalue = [16000, 1767]'),
 Text(0.01, 0.5, 'education <= 0.5\ngini = 0.195\nsamples = 17756\nvalue = [16000, 1756]'),
 Text(0.0, 0.5, 'age <= 59.5\ngini = 0.192\nsamples = 17745\nvalue = [16000, 1745]'),
 Text(0.0, 0.5, '0'),
 Text(0.0, 0.5, '0'),
 Text(0.0, 0.5, '0
```

Table 2: [illegible]

| | | |

```
Train:[00:00000000000000] loss: 0.5[illegible]  Valid: loss: 0.[illegible]  Time: 0.[illegible]s, 0.[illegible]s/it, 0.[illegible]s
```

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)
print("Decision Tree Accuracy:", clf.score(X_test, y_test))
```

```
Decision Tree Accuracy: 0.9333333333333333
```

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figure=(20,10))
plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.target_names)
plt.title("Decision Tree Visualization (max_depth=3)")
plt.show()
```

```
Decision Tree Visualization (max_depth=3)
```

```
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(clf,
          filled=True,
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          rounded=True,
          fontsize=12)
plt.title("Decision Tree Visualization (max_depth=3)")
plt.show
```

```
Decision Tree Visualization (max_depth=3)
```



```
from sklearn.tree import DecisionTreeClassifier, plot_tree
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)
plt.figure(figsize=(20,10))
plot_tree(clf,
          filled=True,
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          rounded=True,
          fontsize=12)
plt.title("Decision Tree Visualization (max_depth=3)")
plt.show()
```

```
Decision Tree Visualization (max_depth=3)
```



```
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)

print("Training Set Acc:", train_acc)
```

```
Training Set Acc: 0.9666666666666667
```

```
cm = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:\n", cm)

plot_cm = plot_confusion_matrix(clf, X_test, y_test, cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```

```
Confusion Matrix:
 [[10 0 0]
 [ 0 9 1]
 [ 0 0 10]]
```



```
cross_val_scores = cross_val_score(clf, X, y, cv=5)
print("Cross-Validation Scores:", cross_val_scores)
print("Mean CV Accuracy:", cross_val_scores.mean())
```

```
Cross-Validation Scores: [1. 0.9 0.9 0.9 1.]
Mean CV Accuracy: 0.9666666666666667
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# Define parameter grid
param_grid = {
    'max_depth': [2, 3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')

# Fit the model
grid_search.fit(X_train, y_train)

# Print the best parameters and best cross-validation score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
```

```
Best Parameters: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 2}
Best Cross-Validation Accuracy: 0.9666666666666667
```