Data and Knowledge Modeling and Analysis

Assignment 1

CM1 (Question 1.1)

Data Cleaning on Iris Data:

Step 1: Checking for Missing values in Columns.

Feature	Number of Missing Values
Sepal Length	0
Sepal Width	4
Petal Length	8
Petal Width	0

12 rows had missing values.

Step 2: Checking for Missing values in Rows.

Checking if any row has all or more than one missing values. If any row has more than missing values the rows should have been dropped.

No rows have more than one missing value, thus we can utilise the estimation method to fill for those missing values.

```
In [10]: #checking for null values
         df_iris.isnull().sum()

Out[10]: sepal_length    0
         sepal_width     4
         petal_length    8
         petal_width     0
         species         0
         dtype: int64
```

```
In [11]: #dropping rows that have all values missing
         df_iris.dropna(how='all').shape

Out[11]: (105, 5)
```

```
In [12]: #dropping rows that have both sepal_width and petal_length missing
         df_iris.dropna(subset=['sepal_width', 'petal_length'], how='all').shape

Out[12]: (105, 5)
```

Step 3: Replacing missing Values.

Missing values in sepal width and petal width column have been replaced by their respective class mean values.

```
In [60]: #filling the missing values with the mean values for sepal_width and petal_length for each species
         # df_iris.fillna(df_iris.mean(), inplace = True)
         df_iris_clean = df_iris.copy(deep=True)
         df_iris_clean['sepal_width']= df_iris['sepal_width'].fillna(df_iris.groupby('species')['sepal_width'].transform('mean')
         df_iris_clean['petal_length']= df_iris['petal_length'].fillna(df_iris.groupby('species')['petal_length'].transform('mea
```

```
In [64]: df_iris_clean.isnull().sum()

Out[64]: sepal_length    0
         sepal_width     0
         petal_length    0
         petal_width     0
         species         0
         dtype: int64
```

Normalization on Iris Dataset:

We have used Min-Max normalization technique and Z-Score normalization technique for various features of the Iris dataset.

In [15]:
```
# Creating a copy of Iris dataframe for normalization
nzd_iris_data = df_iris_clean.copy(deep=True)
```

In [16]:
```
nzd_iris_data['MinMaxsl']= (nzd_iris_data.sepal_length - nzd_iris_data.sepal_length.min())/(nzd_iris_data.sepal_length.
nzd_iris_data['MinMaxsw']= (nzd_iris_data.sepal_width - nzd_iris_data.sepal_width.min())/(nzd_iris_data.sepal_width.max
nzd_iris_data['MinMaxpl']= (nzd_iris_data.petal_length - nzd_iris_data.petal_length.min())/(nzd_iris_data.petal_length.
nzd_iris_data['MinMaxpw']= (nzd_iris_data.petal_width - nzd_iris_data.petal_width.min())/(nzd_iris_data.petal_width.max
nzd_iris_data['Zscoresl']= (nzd_iris_data.sepal_length - nzd_iris_data.sepal_length.mean())/nzd_iris_data.sepal_length.
nzd_iris_data['Zscoresw']= (nzd_iris_data.sepal_width - nzd_iris_data.sepal_width.mean())/nzd_iris_data.sepal_width.std
nzd_iris_data['Zscorepl']= (nzd_iris_data.petal_length - nzd_iris_data.petal_length.mean())/nzd_iris_data.petal_length.
nzd_iris_data['Zscorepw']= (nzd_iris_data.petal_length - nzd_iris_data.petal_width.mean())/nzd_iris_data.petal_width.st
nzd_iris_data
```

Out[16]:

	sepal_length	sepal_width	petal_length	petal_width	species	MinMaxsl	MinMaxsw	MinMaxpl	MinMaxpw	Zscoresl	Zscoresw	Zscorepl	Zscorepw
0	5.045070	2.508203	3.018024	1.164924	Iris-versicolor	0.203115	0.228204	0.346084	0.462421	-0.944525	-1.219437	-0.439418	2.309874
1	6.325517	2.115481	4.542052	1.413651	Iris-versicolor	0.574092	0.068791	0.611799	0.555392	0.541536	-2.093435	0.408884	4.245903
2	5.257497	3.814303	1.470660	0.395348	Iris-setosa	0.264660	0.758373	0.076301	0.174764	-0.697987	1.687273	-1.300709	0.344200
3	6.675168	3.201700	5.785461	2.362764	Iris-virginica	0.675395	0.509707	0.828588	0.910157	0.947335	0.323932	1.100988	5.825452
4	5.595237	2.678166	4.077750	1.369266	Iris-versicolor	0.362512	0.297195	0.530848	0.538801	-0.306013	-0.841186	0.150445	3.656083
...	...	...	...	...	...	...	...	...	...	...	...	...	...
100	4.874848	3.217348	1.592887	0.123588	Iris-setosa	0.153798	0.516058	0.097611	0.073184	-1.142082	0.358756	-1.232675	0.499469
101	5.564197	2.771731	3.483588	1.074754	Iris-versicolor	0.353519	0.335175	0.427255	0.428717	-0.342037	-0.632959	-0.180277	2.901297
102	5.548047	4.249211	1.453466	0.214527	Iris-setosa	0.348840	0.934910	0.073303	0.107176	-0.360780	2.655155	-1.310280	0.322357
103	5.510482	2.652867	4.276817	1.298032	Iris-versicolor	0.337957	0.286926	0.565555	0.512175	-0.404377	-0.897489	0.261249	3.908965
104	4.538713	3.056142	1.545136	0.241424	Iris-setosa	0.056411	0.450622	0.089286	0.117229	-1.532194	-0.000006	-1.259254	0.438809

105 rows x 13 columns

Observed Improvements / Results:

- The correlation between petal length and species increased from 0.949 to 0.952 after data cleaning.
- The correlation between sepal width and species increased from -0.324 to -0.327 after data cleaning.
- The correlation between sepal length and species remained unchanged after data cleaning.
- The correlation between petal width and species remained unchanged after data cleaning.

Data Cleaning on Heart Disease Data:

Step 1: Checking for Missing values in Columns.

Feature	Number of Missing Values
age	0
sex	0
cp	0
trestbps	7
Chol	10
Fbs	0
restecg	5
thalach	4
exang	0
oldpeak	12
Slope	2
ca	4
Thal	1
target	0

Step 2: Checking for unique values in 'ca'.

As per the data description, 'CA' values – number of vessels are (0-3). We have observed a value :4 which is invalid.
We identified 4 values with ca = 4 and replaced ca = NaN in the place of ca=4.

In [1025]:
```
#checking for data character errors
df_heart['ca'].unique()
```

Out[1025]:
```
array([0, 1, 2, 3, 4])
```

In [1026]:
```
#now, our description suggests that 'ca' values should be 0-3; thus, checking for rows for 'ca' value as '4'
df_heart[df_heart['ca']==4]
```

Out[1026]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
55	38	1	Non	138.127886	175.063869	0	Normal	172.906918	0	0.185054	flat	4	2.0	1
82	43	1	Asympt.	131.825380	246.955104	1	LV hyper	142.974825	1	0.079109	up	4	3.0	0
103	38	1	Non	138.127030	174.925633	0	Normal	173.028229	0	0.038206	flat	4	2.0	1
128	52	1	Non	137.977233	222.937493	0	Normal	169.090939	0	0.017090	flat	4	2.0	1

In [1027]:
```
#changing the 'ca' value 4 to NaN
df_heart.loc[df_heart['ca']==4, 'ca'] = np.NaN
```

In [1028]:
```
df_heart['ca'].unique()
```

Out[1028]:
```
array([ 0.,  1.,  2.,  3., nan])
```

Step 3: Checking for Missing values in Rows.

Checking if any row has all or more than one missing values. If any row has more than missing values the rows should have been dropped.

None of the rows has more than one missing values. Thus, we decided to go with estimation technique to fill for the missing values.

In [1032]:
```
#dropping rows that have all values missing
df_heart.dropna(how='all').shape
```

Out[1032]:
```
(212, 14)
```

In [1033]:
```
#dropping rows that have all of these values missing: trestbps, chol, restecg, thalach, oldpeak, slope and thal
df_heart.dropna(subset=['trestbps', 'chol','restecg','thalach' , 'oldpeak', 'slope', 'thal' ], how='all').shape
```

Out[1033]:
```
(212, 14)
```

Step 4: Replacing missing Values.

i. Missing values in categorical column ['restecg', 'slope', 'thal', 'ca'] have been replaced by target class mode values.

In [1034]:
```
#filling the missing values for categorical features {'restecg', 'slope', 'thal'}
#Replacing the missing values for these categorical features by the Mode of each feature grouped by the target
df_heart_clean = df_heart.copy(deep=True)
df_heart_clean['restecg'] = df_heart.groupby('target')['restecg'].transform(lambda x: x.fillna(x.mode().iloc[0]))
df_heart_clean['slope'] = df_heart.groupby('target')['slope'].transform(lambda x: x.fillna(x.mode().iloc[0]))
df_heart_clean['ca'] = df_heart.groupby('target')['ca'].transform(lambda x: x.fillna(x.mode().iloc[0]))
df_heart_clean['thal'] = df_heart.groupby('target')['thal'].transform(lambda x: x.fillna(x.mode().iloc[0]))
```

ii. Missing values in numerical column ['trestbps', 'chol', 'thalach', 'oldpeak'] have been replaced by target class mean values.

In [1035]:
```
#filling the missing values for numerical features {'trestbps' ,'chol', 'thalach' , 'oldpeak'}
#Replacing the missing values for these numerical features by the Mean of each feature grouped by the target
df_heart_clean['trestbps'] = df_heart.groupby('target')['trestbps'].transform(lambda x: x.fillna(x.mean()))
df_heart_clean['chol'] = df_heart.groupby('target')['chol'].transform(lambda x: x.fillna(x.mean()))
df_heart_clean['thalach'] = df_heart.groupby('target')['thalach'].transform(lambda x: x.fillna(x.mean()))
df_heart_clean['oldpeak'] = df_heart.groupby('target')['oldpeak'].transform(lambda x: x.fillna(x.mean()))
```

In [1036]:
```
df_heart_clean.isnull().sum()
```

Out[1036]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Step 5: Checking for duplicate rows.

Checking for duplicate rows in the dataset. There were no duplicate entries observed.

In [1037]:
```
#Checking for duplicate rows
duplicated = df_heart_clean.duplicated().sum()
if duplicated:
    print('Duplicates in the Heart Disease dataset are: {}'.format(duplicated))
else:
    print('Dataset does not contain any duplicate values.')
```

```
Dataset does not contain any duplicate values.
```

In [1038]:
```
df_heart_clean.shape
```

Out[1038]:
```
(212, 14)
```

Step 6: Replacing categorical coded values with integers (0,1,2..) for evaluating the correlation matrix

In [1041]:
```
# Replacing categorical coded values with integers (0,1,2..) for evaluating the correlation matrix
df_heart_corr = df_heart_clean.copy(deep=True)
df_heart_corr.cp = df_heart_clean.cp.replace({'Asympt.':0,'Atypical':1,'Non':2, 'Typical':3})
df_heart_corr.restecg = df_heart_clean.restecg.replace({'LV hyper': 0, 'Normal': 1, 'ST-T wave':2})
df_heart_corr.slope = df_heart_clean.slope.replace({'down': 0, 'up':1, 'flat':2})
df_heart_corr.thal = df_heart_clean.thal.replace({'NA':0,'Fixed':1, 'Normal':2, 'Revers.':3})
```

In [1042]:
```
df_heart_corr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 212 entries, 0 to 211
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       212 non-null    int64
 1   sex       212 non-null    int64
 2   cp        212 non-null    int64
 3   trestbps  212 non-null    float64
 4   chol      212 non-null    float64
 5   fbs       212 non-null    int64
 6   restecg   212 non-null    int64
 7   thalach   212 non-null    float64
 8   exang     212 non-null    int64
 9   oldpeak   212 non-null    float64
 10  slope     212 non-null    int64
 11  ca        212 non-null    float64
 12  thal      212 non-null    float64
 13  target    212 non-null    int64
dtypes: float64(6), int64(8)
memory usage: 23.3 KB
```

Normalization on Heart Disease Dataset:

Used One-hot encoding, followed by MinMaxScaler to normalize the heart disease dataset.

One-hot encoding

In [29]:
```
#Segregating categorical and numeric data attributes
CatCol = []
NumCol = []
for i in df_heart_corr.columns:
    if (len(df_heart_corr[i].unique())) > 5:
        NumCol.append(i)
    else:
        CatCol.append(i)
    print('unique values:  {}\t{}'.format(len(df_heart_corr[i].unique()),i))
CatCol.remove('target')
CATdata = df_heart_corr[CatCol]
NumCol.append('target') #adding target to the Numeric column
NUMdata = df_heart_corr[NumCol]

print(NumCol)
print(CatCol)
```

```
unique values:  41      age
unique values:  2       sex
unique values:  4       cp
unique values:  207     trestbps
unique values:  204     chol
unique values:  2       fbs
unique values:  3       restecg
unique values:  210     thalach
unique values:  2       exang
unique values:  202     oldpeak
unique values:  3       slope
unique values:  4       ca
unique values:  3       thal
unique values:  2       target
['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'target']
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']
```

In [30]:
```
#  one hot encoding works on type 'object'
encoded_data = df_heart_corr.copy(deep=True)
for i in CatCol:
    encoded_data[i] = encoded_data[i].astype(object)

df_OHE = encoded_data[CatCol]                #  dataframe with categorical values
df_OHE = pd.get_dummies(df_OHE)    #  one-hot encoding
df_OHE = df_OHE.join(encoded_data[NumCol])  #  add numeric columns

df_OHE.head()
```

Out[30]:

	sex_0	sex_1	cp_0	cp_1	cp_2	cp_3	fbs_0	fbs_1	restecg_0	restecg_1	...	ca_3.0	thal_1.0	thal_2.0	thal_3.0	age	trestbps	chol	thalach
0	1	0	0	0	1	0	1	0	0	0	...	0	0	1	0	76	140.102822	197.105970	115.952071
1	1	0	1	0	0	0	0	1	1	0	...	0	0	0	1	43	132.079599	341.049462	135.970028
2	0	1	0	0	1	0	1	0	0	1	...	0	0	1	0	47	107.899290	242.822816	152.210039
3	0	1	0	0	1	0	1	0	0	1	...	0	0	1	0	51	99.934001	240.825764	143.049207
4	0	1	1	0	0	0	1	0	0	1	...	0	0	0	1	57	110.103508	334.952353	143.099327

5 rows × 29 columns

In [31]:
```
# Normalizing the data

from sklearn.preprocessing import MinMaxScaler
norm = MinMaxScaler().fit_transform(df_OHE)
norm[0:2]
```

Out[31]:
```
array([[1.        , 0.        , 0.        , 0.        , 1.        ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        1.        , 1.        , 0.        , 0.        , 1.        ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 1.        , 0.        , 0.97916667, 0.47064145,
        0.25287858, 0.24468125, 0.23183667, 1.        ],
       [1.        , 0.        , 1.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 1.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 1.        ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 0.29166667, 0.38883528,
        0.76541228, 0.42011512, 0.51966953, 0.        ]])
```

In [32]:
```
#dataframe with the One Hot Encoding & normalized data
df_nzd = pd.DataFrame(norm, index=df_OHE.index, columns=df_OHE.columns)
df_nzd.head()
```

Out[32]:

	sex_0	sex_1	cp_0	cp_1	cp_2	cp_3	fbs_0	fbs_1	restecg_0	restecg_1	...	ca_3.0	thal_1.0	thal_2.0	thal_3.0	age	trestbps	chol	thalach	ol
0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.979167	0.470641	0.252879	0.244681	0.2
1	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	0.0	0.0	1.0	0.291667	0.388835	0.765412	0.420115	0.5
2	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	...	0.0	0.0	1.0	0.0	0.375000	0.142289	0.415661	0.562440	0.0
3	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	...	0.0	0.0	1.0	0.0	0.458333	0.061073	0.408550	0.482156	0.2
4	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.0	0.0	0.0	1.0	0.583333	0.164763	0.743703	0.482595	0.5

5 rows × 29 columns

Also, undertook Z-score normalization of the data as shown below:

In [27]:
```
# Creating a copy of Heart disease dataframe for normalization
nzd_heart_data = df_heart_corr.copy(deep=True)
```

In [28]:
```
#calculating min-max normalized values for each attribute (Numerical attributes)

nzd_heart_data['MinMaxAge'] = (nzd_heart_data.age - nzd_heart_data.age.min())/(nzd_heart_data.age.max() - nzd_heart_dat
nzd_heart_data['MinMaxSex'] = (nzd_heart_data.sex - nzd_heart_data.sex.min())/(nzd_heart_data.sex.max() - nzd_heart_dat
nzd_heart_data['MinMaxTrestbps'] = (nzd_heart_data.trestbps - nzd_heart_data.trestbps.min())/(nzd_heart_data.trestbps.m
nzd_heart_data['MinMaxChol'] = (nzd_heart_data.chol - nzd_heart_data.chol.min())/(nzd_heart_data.chol.max() - nzd_heart
nzd_heart_data['MinMaxThalach'] = (nzd_heart_data.thalach - nzd_heart_data.thalach.min())/(nzd_heart_data.thalach.max()
nzd_heart_data['MinMaxOldpeak'] = (nzd_heart_data.oldpeak - nzd_heart_data.oldpeak.min())/(nzd_heart_data.oldpeak.max()
nzd_heart_data['ZScoreAge'] = (nzd_heart_data.age - nzd_heart_data.age.mean())/nzd_heart_data.age.std()
nzd_heart_data['ZScoreSex'] = (nzd_heart_data.sex - nzd_heart_data.sex.mean())/nzd_heart_data.sex.std()
nzd_heart_data['ZScoreTrestbps'] = (nzd_heart_data.trestbps - nzd_heart_data.trestbps.mean())/nzd_heart_data.trestbps.s
nzd_heart_data['ZScoreChol'] = (nzd_heart_data.chol - nzd_heart_data.chol.mean())/nzd_heart_data.chol.std()
nzd_heart_data['ZScoreThalach'] = (nzd_heart_data.thalach - nzd_heart_data.thalach.mean())/nzd_heart_data.thalach.std()
nzd_heart_data['ZScoreOldpeak'] = (nzd_heart_data.oldpeak - nzd_heart_data.oldpeak.mean())/nzd_heart_data.oldpeak.std()

nzd_heart_data
```

Out[28]:

ldpeak	...	MinMaxTrestbps	MinMaxChol	MinMaxThalach	MinMaxOldpeak	ZScoreAge	ZScoreSex	ZScoreTrestbps	ZScoreChol	ZScoreThalach	ZScoreOldpeak
284822	...	0.470641	0.252879	0.244681	0.231837	2.371556	-1.483808	0.466254	-1.037613	-1.541991	0.151554
110483	...	0.388835	0.765412	0.420115	0.519670	-1.236840	-1.483808	0.014492	2.137335	-0.627917	1.639802
023723	...	0.142289	0.415661	0.562440	0.025532	-0.799459	0.670763	-1.347024	-0.029241	0.113647	-0.915150
195082	...	0.061073	0.408550	0.482156	0.217688	-0.362077	0.670763	-1.795523	-0.073290	-0.304662	0.078400
082052	...	0.164763	0.743703	0.482595	0.515187	0.293995	0.670763	-1.222911	2.002852	-0.302373	1.616626
...	...	...	...	...	...	...	...	...	...	...	...
606726	...	0.409943	0.463255	0.472797	0.124928	0.184649	0.670763	0.131057	0.265589	-0.353424	-0.401218
577227	...	0.347049	0.629880	0.621877	0.120278	0.293995	-1.483808	-0.216264	1.297764	0.423335	-0.425265
715230	...	0.573477	0.377543	0.674333	0.299695	-0.034041	0.670763	1.034148	-0.265366	0.696652	0.502416
992138	...	0.366803	0.313061	0.699519	0.343352	-1.455531	0.670763	-0.107176	-0.664805	0.827878	0.728147
085251	...	0.184873	0.505328	0.735953	0.042713	-1.455531	-1.483808	-1.111860	0.526211	1.017716	-0.826316

Observed Improvements / Results:

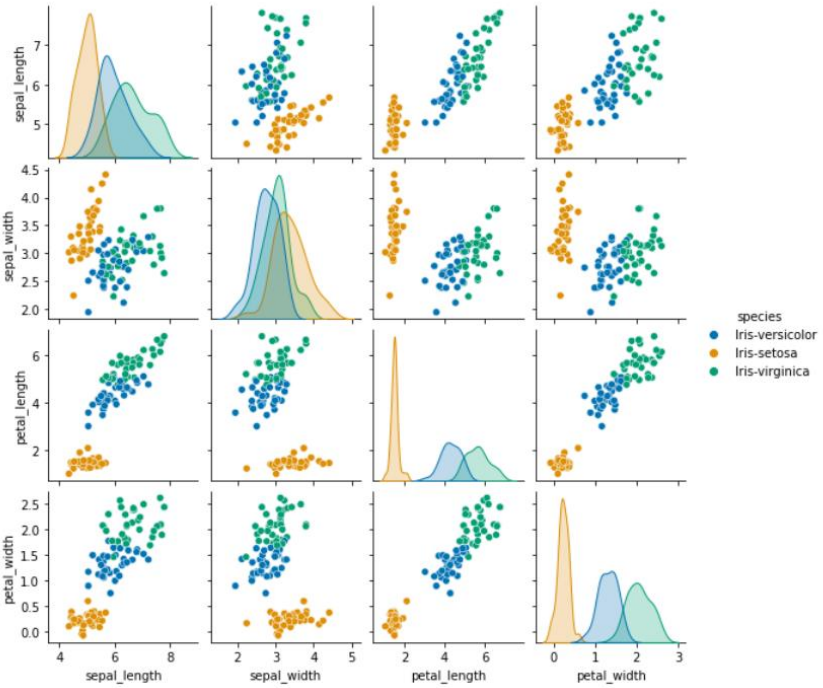
- The correlation between slope and target increased from 0.419 to 0.423 after data cleaning.
- The correlation between ca and target changed from -0.338 to -0.408 after data cleaning.
- The correlation between thalach and target increased from 0.415 to 0.418 after data cleaning.
- The correlation between thal and target changed from -0.367 to -0.383 after data cleaning.
- The correlation between cp and target remained unchanged after data cleaning.
- The correlation between oldpeak and target changed from – 0.454 to -0.464 after data cleaning.
- The correlation between restecg and target increased from 0.087 to 0.105 after data cleaning.

CM2 (Question 1.2)

Pair Plot of Iris Data set:

```
In [32]: #Making scatter plots of all the paired features by using seaborn's pairplot function:
         sns.pairplot(df_iris_clean, hue="species", height = 2, palette = 'colorblind')
```

```
Out[32]: <seaborn.axisgrid.PairGrid at 0x7faa54ebf9d0>
```

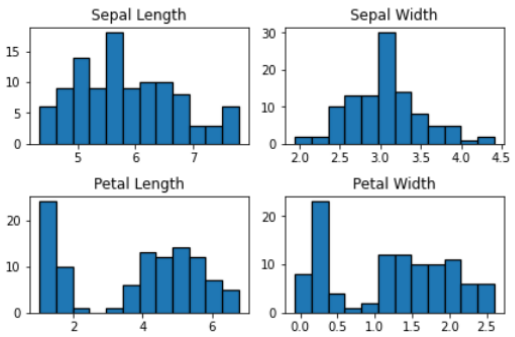


Observations from the Iris dataset pairplot:

- From the above plot, we can deduce that petal width has quite high correlation with petal length.
- Sepal length also has high correlation with petal length.
- Petal width and sepal width are showing low correlation between them.
- There is low correlation observed of all other features and sepal width.
- The diagonal distribution plots depict that of all the attributes, petal width and petal length have higher correlation with species than any other attributes, as a line of separation can be drawn to segregate the distribution curves for each species in these features.
- Sepal width has the maximum overlaps in the distribution curves, which shows that it has the worst correlation with the species.
- From the diagonal distribution plots, it can be deduced that most of the Iris-sentosa flowers have petal length around 2.
- Also, most of the iris sentosa have lower petal width (0-1 units).

Histograms of Iris Dataset:

```
In [56]: #histograms
         n_bins = 12
         fig, axes = plt.subplots(2, 2)
         axes[0,0].hist(df_iris_clean['sepal_length'], bins = n_bins, edgecolor='black', linewidth=1.2)
         axes[0,0].set_title('Sepal Length')
         axes[0,1].hist(df_iris_clean['sepal_width'], bins = n_bins, edgecolor='black', linewidth=1.2)
         axes[0,1].set_title('Sepal Width')
         axes[1,0].hist(df_iris_clean['petal_length'], bins = n_bins, edgecolor='black', linewidth=1.2)
         axes[1,0].set_title('Petal Length')
         axes[1,1].hist(df_iris_clean['petal_width'], bins = n_bins, edgecolor='black', linewidth=1.2)
         axes[1,1].set_title('Petal Width')

         # adding some padding / spacing between subplots
         fig.tight_layout(pad=1.0)
```



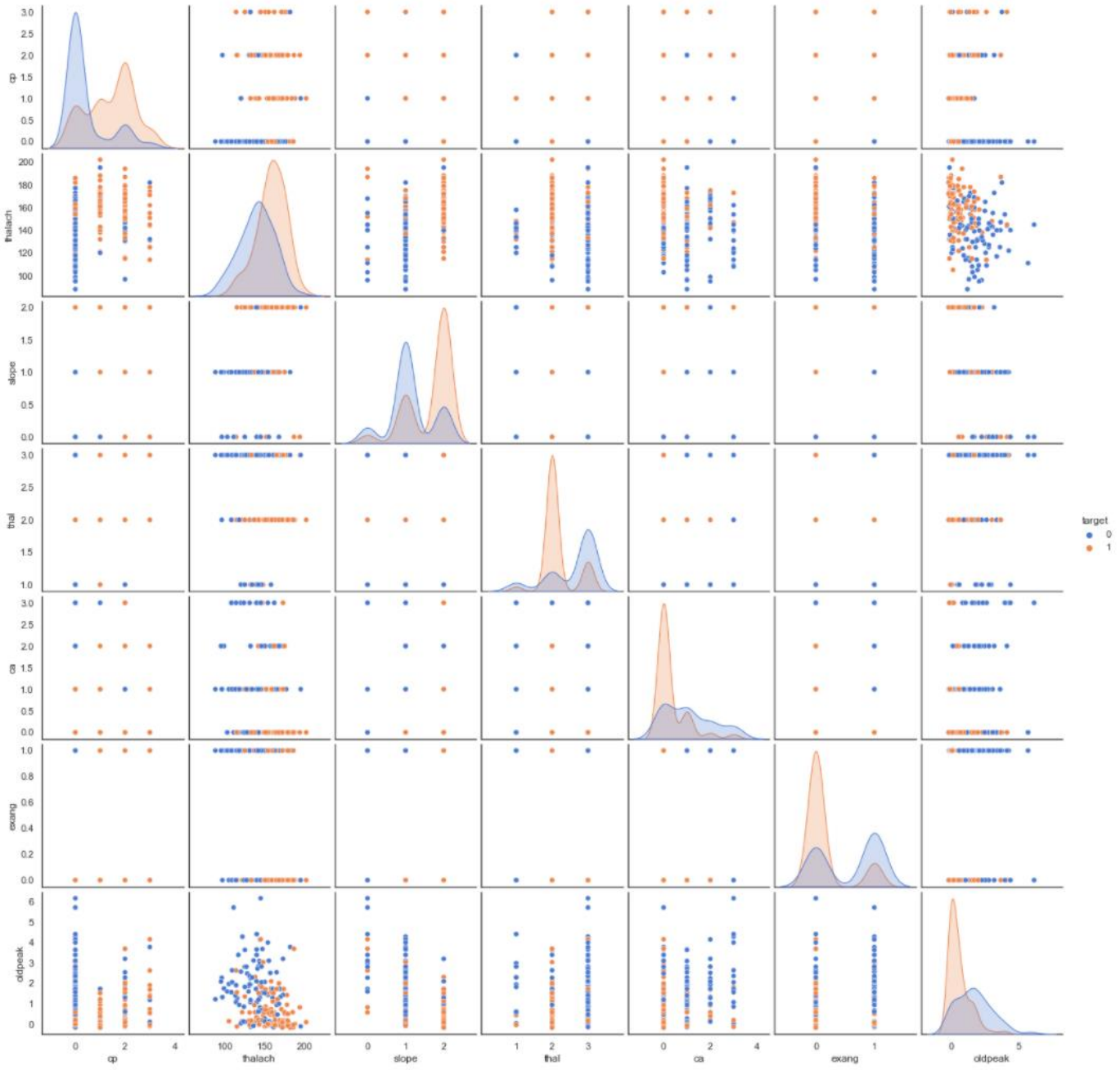
Observations from the histogram:

- There are a few outliers in petal width and petal length.

Pair Plot of Heart Disease Data set:

In [37]:
```
#Making scatter plots of selected paired features by using seaborn's pairplot function:
sns.set(style = 'white', palette="muted")
selected_features_pairplot = ['cp', 'thalach', 'slope', 'thal', 'ca', 'exang', 'oldpeak']
values_subset_pairplot = df_heart_corr[selected_features_pairplot]
pairplot_labels = df_heart_corr['target']

sns.pairplot(values_subset_pairplot.join(pairplot_labels),
             hue='target',
             vars=selected_features_pairplot)
```

Out[37]: <seaborn.axisgrid.PairGrid at 0x7fb0389ec550>



Observations from the Heart disease pairplot:

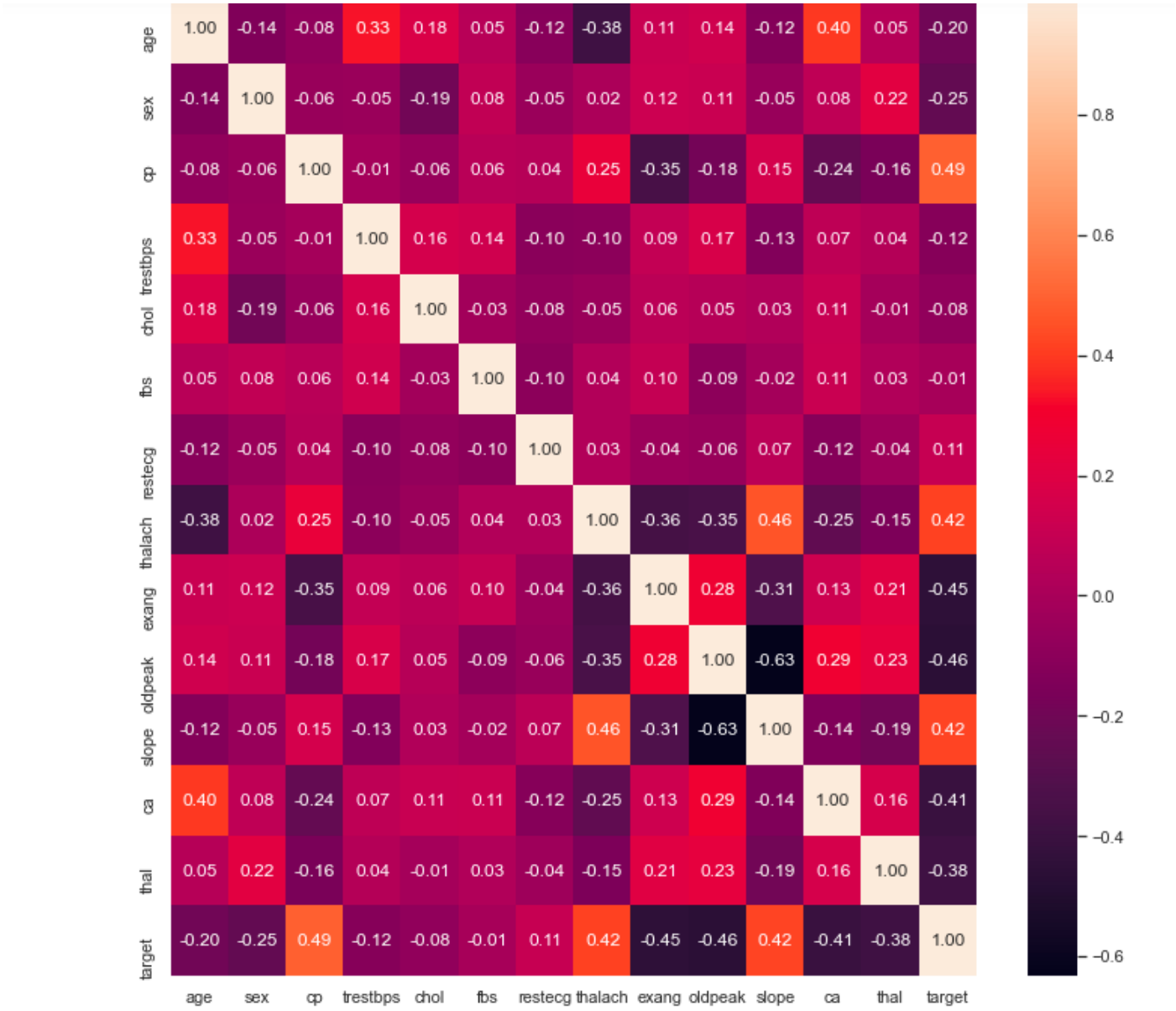
- Deducing from the diagonal distribution plot, most of the people with 'ca' value of 0 have target value '0' i.e. no heart disease.
- Majority of the people with the heart disease have a slope of value '2' i.e. flat slope; and most of those without heart disease have a slope value of '1' i.e. upsloping.
- People with heart disease have attained the peak value of Thallium stress at a value of 1.
- People with heart disease have a peak value of the ca distribution at ca = 0 i.e. number of major vessels 0 coloured by fluoroscopy.
- The peak distribution of people with heart disease is at exang value '0' i.e. exang: Exercise induced angina of value 0.

Reason for choosing these features for Pairplot amongst all the features in the Heart disease dataset:

From the correlation heatmap of all the features of heart disease dataset, we deduced that out of all the features, the following attributes had the highest correlation with the target: cp, thalach, slope, exang, oldpeak, ca and thal. Thus, we chose these attributes (**cp, thalach, slope, exang, oldpeak, ca and thal**) for the pairplot.

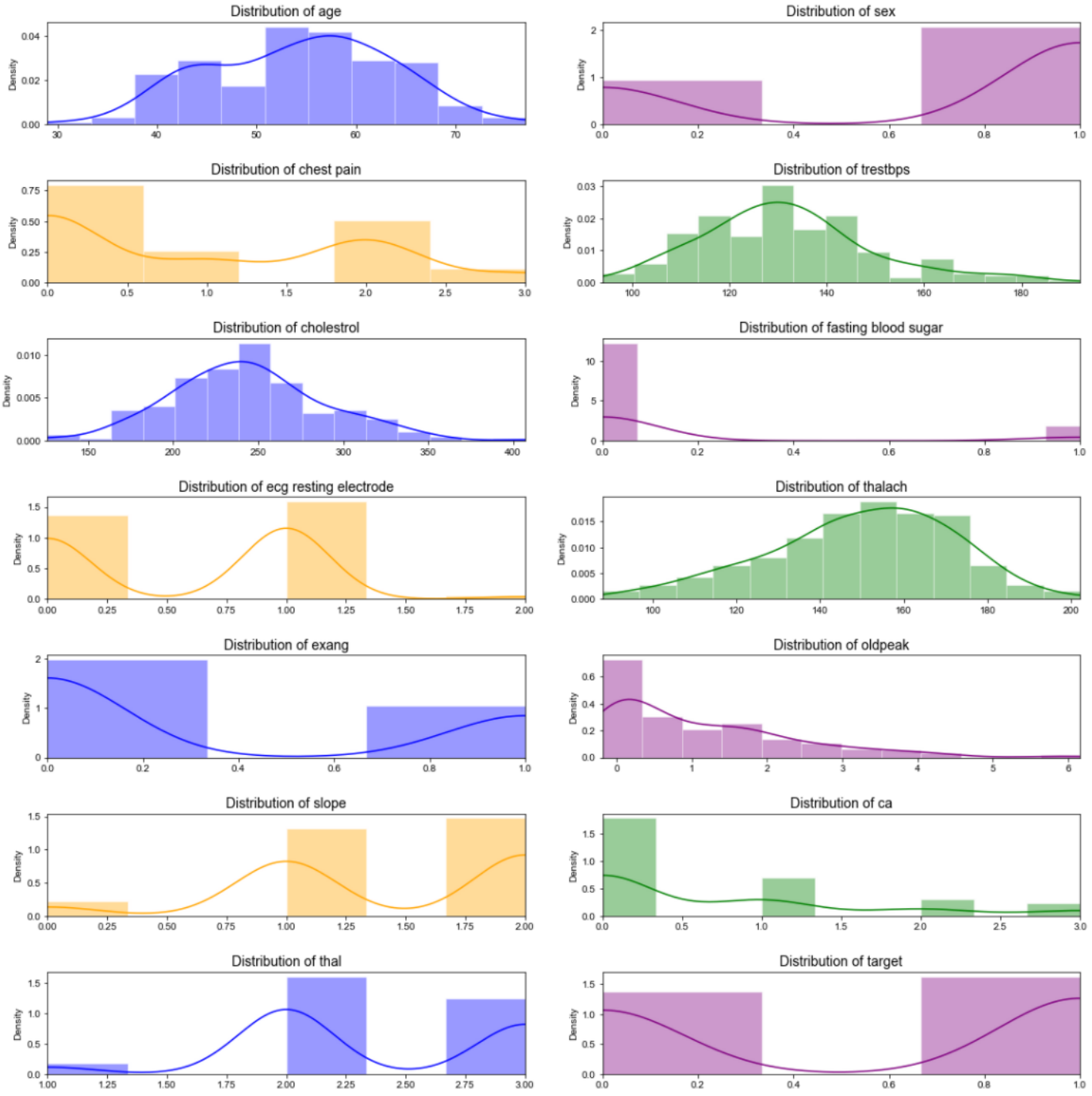
- 'cp' shows the highest correlation of 0.49 with the 'target', thus it has been selected for the pairplot.
- 'thalach' and 'slope' have the second-highest correlation with the 'target' [0.42], thus they have been chosen.
- 'exang' and 'oldpeak' have very high negative correlation with the 'target' [~ -0.46], thus they have been selected.
- 'ca' and 'thal' had very low relative correlation with most of the other features, thus they have been selected for the pairplot.

The correlation plot is shown below for reference:



Histograms of the Heart Disease Dataset:

In [33]:
```
#histograms

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

n_bins = 12

fig, ax = plt.subplots(7, 2, figsize=(16,16))
age = df_heart_corr['age'].values
sex = df_heart_corr['sex'].values
cp = df_heart_corr['cp'].values
trestbps = df_heart_corr['trestbps'].values
chol = df_heart_corr['chol'].values
fbs = df_heart_corr['fbs'].values
restecg = df_heart_corr['restecg'].values
thalach = df_heart_corr['thalach'].values
exang = df_heart_corr['exang'].values
oldpeak = df_heart_corr['oldpeak'].values
slope = df_heart_corr['slope'].values
ca = df_heart_corr['ca'].values
thal = df_heart_corr['thal'].values
target = df_heart_corr['target'].values

sns.set(style="white", palette="PuBuGn_d", color_codes=True)

sns.distplot(age, ax=ax[0,0], color='blue')
ax[0,0].set_title('Distribution of age', fontsize=14)
ax[0,0].set_xlim([min(age), max(age)])
sns.distplot(sex, ax=ax[0,1], color='purple')
ax[0,1].set_title('Distribution of sex', fontsize=14)
ax[0,1].set_xlim([min(sex), max(sex)])

sns.distplot(cp, ax=ax[1,0], color='orange')
ax[1,0].set_title('Distribution of chest pain', fontsize=14)
ax[1,0].set_xlim([min(cp), max(cp)])
sns.distplot(trestbps, ax=ax[1,1], color='green')
ax[1,1].set_title('Distribution of trestbps', fontsize=14)
ax[1,1].set_xlim([min(trestbps), max(trestbps)])

sns.distplot(chol, ax=ax[2,0], color='blue')
ax[2,0].set_title('Distribution of cholestrol', fontsize=14)
ax[2,0].set_xlim([min(chol), max(chol)])
sns.distplot(fbs, ax=ax[2,1], color='purple')
ax[2,1].set_title('Distribution of fasting blood sugar', fontsize=14)
ax[2,1].set_xlim([min(fbs), max(fbs)])

sns.distplot(restecg,ax=ax[3,0], color='orange')
ax[3,0].set_title('Distribution of ecg resting electrode', fontsize=14)
ax[3,0].set_xlim([min(restecg), max(restecg)])
sns.distplot(thalach, ax=ax[3,1], color='green')
ax[3,1].set_title('Distribution of thalach', fontsize=14)
ax[3,1].set_xlim([min(thalach), max(thalach)])

sns.distplot(exang,ax=ax[4,0], color='blue')
ax[4,0].set_title('Distribution of exang', fontsize=14)
ax[4,0].set_xlim([min(exang), max(exang)])
sns.distplot(oldpeak, ax=ax[4,1], color='purple')
ax[4,1].set_title('Distribution of oldpeak', fontsize=14)
ax[4,1].set_xlim([min(oldpeak), max(oldpeak)])

sns.distplot(slope,ax=ax[5,0], color='orange')
ax[5,0].set_title('Distribution of slope', fontsize=14)
ax[5,0].set_xlim([min(slope), max(slope)])
sns.distplot(ca, ax=ax[5,1], color='green')
ax[5,1].set_title('Distribution of ca', fontsize=14)
ax[5,1].set_xlim([min(ca), max(ca)])

sns.distplot(thal,ax=ax[6,0], color='blue')
ax[6,0].set_title('Distribution of thal', fontsize=14)
ax[6,0].set_xlim([min(thal), max(thal)])
sns.distplot(target, ax=ax[6,1], color='purple')
ax[6,1].set_title('Distribution of target', fontsize=14)
ax[6,1].set_xlim([min(target), max(target)])

# adding some padding / spacing between subplots
fig.tight_layout(pad=2.0)

plt.show()
```



Observations from the histogram plots:

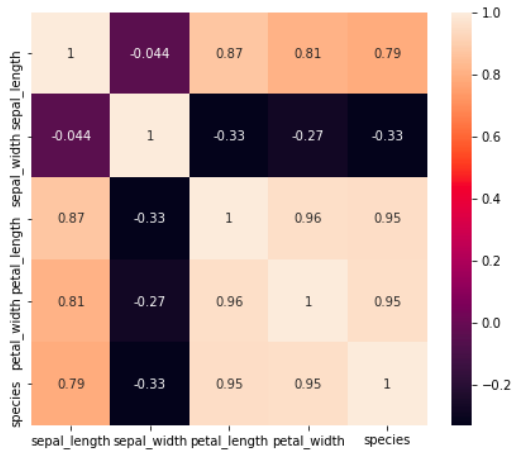
- There is noise in 'ca' values, as it is supposed to be in the categorical range 0-3, but there are a few values with category 4. We have replaced those 'ca' values with 'Nan' and handled these Nan values by transforming them using the class mode as mentioned in CM1.
- There is also some noise in categorical features, such as thal as they are supposed to be categorical, but have been provided continuous values in the original data.

CM3 (Question 1.3)

Correlation Matrix of Iris Data:

In [22]:
```
#Before plotting the correlation matrix (heatmap), label encoding the species column
df_iris_cm2 = df_iris_clean.copy(deep=True)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df_iris_cm2['species'] = le.fit_transform(df_iris_cm2['species'])

corr_iris_clean = df_iris_cm2.corr()
fig, ax = plt.subplots(figsize = (7,6))
sns.heatmap(corr_iris_clean, annot = True, ax=ax)
```

Out[22]: <AxesSubplot:>



Observations:

- Sepal width shows a negative correlation with species whereas all other attributes show a positive correlation.
- Petal width has the highest correlation with species [0.957].
- Sepal width has the lowest correlation with species [-0.327].
- Petal width and petal length have the highest inter-feature correlation of [0.955].
- Sepal width and sepal length have the lowest inter-feature correlation of [-0.044].
- Sepal width shows a negative correlation with all the other attributes.

Calculating Mean, Variance, Skew, Kurtosis for Iris Data set:

Species	Feature	Mean	Variance	Skew	Kurtosis
Iris Setosa (Label Encoded - 0)	Sepal Length	4.987384	0.118106	-0.080329	-0.761361
	Sepal Width	3.383549	0.191624	0.186162	0.432695
	Petal Length	1.484517	0.034137	0.965093	3.168064
	Petal Width	0.229271	0.017524	0.060626	0.693320
Iris Versicolor (Label Encoded – 1)	Sepal Length	5.948732	0.290830	0.532285	-0.306740
	Sepal Width	2.759377	0.101597	-0.412115	-0.192980
	Petal Length	4.283101	0.231754	-0.359083	-0.202708
	Petal Width	1.308528	0.052308	-0.244670	-0.355903
Iris Virginica (Label Encoded – 2)	Sepal Length	6.640611	0.442762	0.250308	-1.033211
	Sepal Width	3.025507	0.122399	0.162343	0.240834
	Petal Length	5.654781	0.306267	0.338572	-0.789542
	Petal Width	2.061326	0.079779	0.092456	-0.758480

Iris Data Statistics

In [24]:
```
rows = []
for column in df_iris_cm2:
    rows.append([column, df_iris_cm2[column].mean(), df_iris_cm2[column].var(), df_iris_cm2[column].skew(), df_iris_cm

df_iris_stats = pd.DataFrame(rows, columns=["column","mean", "variance","skew","kurtosis"])
df_iris_stats
```

Out[24]:

	column	mean	variance	skew	kurtosis
0	sepal_length	5.858909	0.742420	0.401506	-0.544820
1	sepal_width	3.056145	0.201906	0.384964	0.565742
2	petal_length	3.807466	3.227641	-0.258061	-1.414701
3	petal_width	1.199708	0.619672	-0.074751	-1.315451
4	species	1.000000	0.673077	0.000000	-1.514563

In [187]:
```
from scipy.stats import describe

grouper = df_iris_cm2.groupby('species')

variables = df_iris_cm2.columns[0:4]

res = pd.concat([pd.DataFrame(describe(g[x]) for _, g in grouper)\
                  .reset_index().assign(species=x).set_index(['species', 'index']) \
                for x in variables], axis=0)

print(res)
```

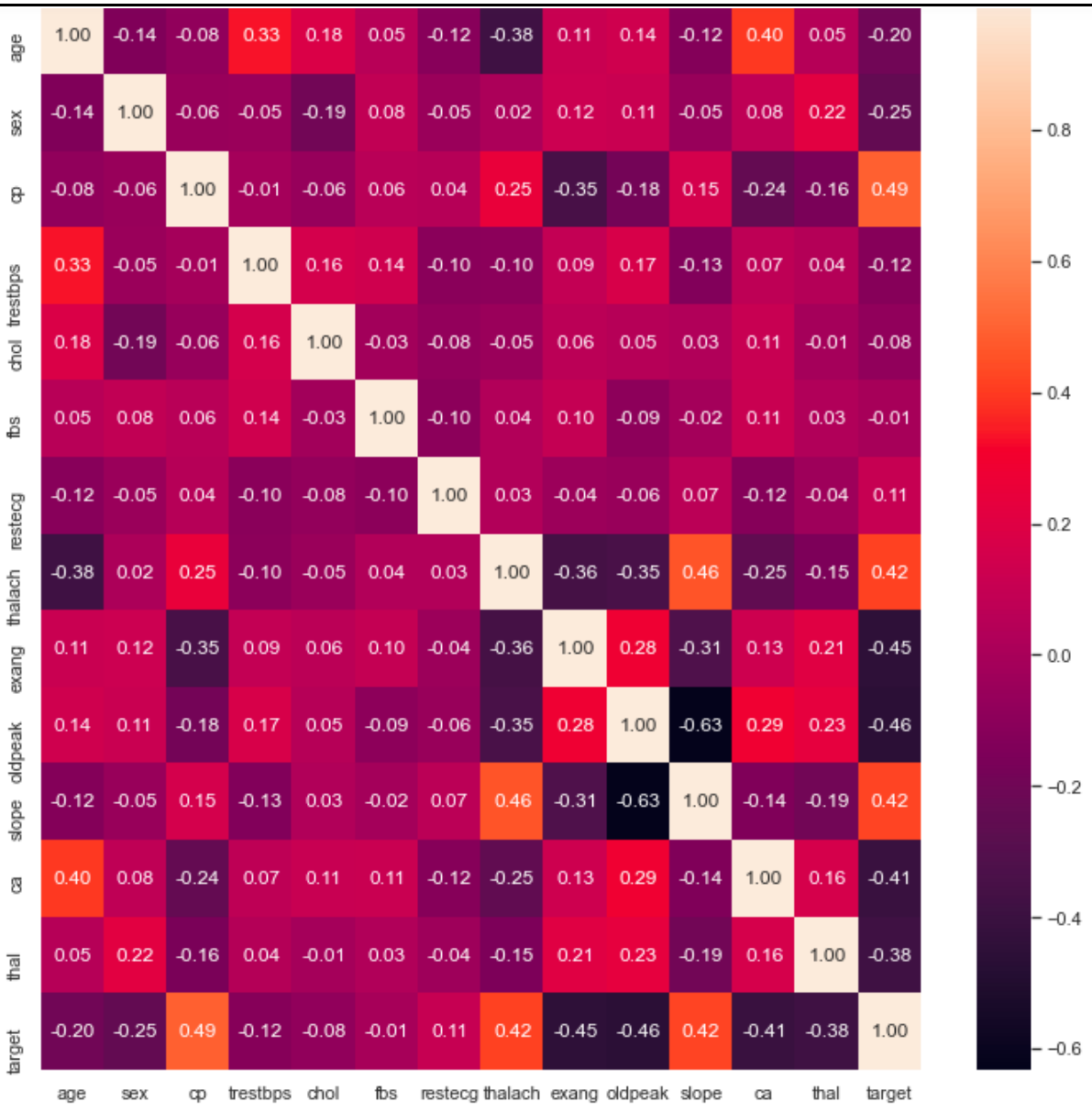
```
                     nobs                                      minmax      mean  \
species      index
sepal_length 0         35     (4.344007208198309, 5.673096305504441)  4.987384
             1         35    (5.0450704418122765, 7.219411090016365)  5.948732
             2         35     (5.567296466231346, 7.795560938119888)  6.640611
sepal_width  0         35     (2.242836984992027, 4.409564795531333)  3.383549
             1         35   (1.9460099525749683, 3.2903177708387377)  2.759377
             2         35    (2.2279179960489275, 3.802923750318586)  3.025507
petal_length 0         35    (1.0330307915806771, 2.103636571764946)  1.484517
             1         35    (3.0180239006876945, 5.094427058100701)  4.283101
             2         35     (4.751795231550932, 6.768610590696335)  5.654781
petal_width  0         35  (-0.0722025871276855, 0.5946143507957459)  0.229271
             1         35   (0.7486814558506012, 1.7654694572091103)  1.308528
             2         35    (1.4539492800831797, 2.603122520446777)  2.061326

                     variance  skewness  kurtosis
species      index
sepal_length 0       0.118106 -0.080329 -0.761361
             1       0.290830  0.532285 -0.306740
             2       0.442762  0.250308 -1.033211
sepal_width  0       0.191624  0.186162  0.432695
             1       0.101597 -0.412115 -0.192980
             2       0.122399  0.162343  0.240834
petal_length 0       0.034137  0.965093  3.168064
             1       0.231754 -0.359083 -0.202708
             2       0.306267  0.338572 -0.789542
petal_width  0       0.017524  0.060626  0.693320
             1       0.052308 -0.244670 -0.355903
             2       0.079779  0.092456 -0.758480
```

Observations:

- Amongst all the species, the highest mean value has been observed for Sepal length [5.858] and the lowest mean value has been observed for Petal width [1.199].
- The flower species Iris-Virginica has overall highest mean values for all the attributes, hence, it can be deduced that Iris-Virginica is larger than the other flower species in general.
- Iris-Virginica has an overall highest variance for Sepal length, followed by Iris-Versicolor.
- Iris-Virginica has an overall highest variance for Petal length, followed by Iris-Versicolor.
- Iris-Setosa shows the least variance for attributes: Sepal length, petal length and petal width, but it shows the highest variance for Sepal width.
- The skewness measures for Iris-Setosa depicts that almost all its features are skewed to right except sepal length.
- For Iris-Versicolor, sepal length is skewed to the right whereas other attributes are skewed to the left.
- For Iris-Virginica, all of its attributes are skewed to the right.
- Sepal length for Iris-Virginica has the highest negative kurtosis value, which depicts that the data is light-tailed and the curve flattens out more at the tails when compared to a normal distribution curve.
- Petal length for Iris-Setosa has the highest positive kurtosis value, which depicts that the range of values is quite small and the curve basically is not flattening out on the tails.

Correlation Matrix of Heart Disease Data:

```
In [35]: #Making a correlation matrix (heatmap) without normalization

corrmat = df_heart_corr.corr()
top_corr_features = corrmat.index
plt.figure(figsize = (12,12))
#plot the heatmap
plt.title('Heart Disease Data (without Normalization) - CORRELATION , Overall', fontsize=14)
p= sns.heatmap(df_heart_corr[top_corr_features].corr(), annot = True,fmt='.2f')
```



Observations:

- 'cp', 'thalach', 'slope', 'restecg' have a positive correlation with the 'target'.
- 'exang', 'oldpeak', 'ca' and 'thal' have a negative correlation with the 'target'.
- 'cp' shows the highest correlation of 0.49 with the 'target'.
- 'thalach' and 'slope' have the second-highest correlation with the 'target' [0.42].
- 'oldpeak' has the highest negative correlation with the 'target' [-0.46].
- Highest inter-feature positive correlation is observed between 'thalach' and 'slope' [0.46].
- Highest inter-feature negative correlation is observed between 'slope' and 'oldpeak' [-0.63].

Calculating Mean, Variance, Skew, Kurtosis for the Heart Disease Data set:

Descriptive Statistics:

Target	Feature	Mean	Mode	Variance	Skew	Kurtosis
0 (No Heart Disease)	cp	0.412371	0	0.703179	1.770177	1.680583
	thalach	139.767679	-	462.113380	-0.165506	-0.302270
	slope	1.134021	1	0.325601	0.011213	-0.071926
	exang	0.577320	0	0.246564	-0.313044	-1.902003
	oldpeak	1.717394	-	1.803246	0.717616	0.494933
	ca	1.072165	0	1.088488	0.574800	-0.870307
1 (Heart Disease)	cp	1.417391	2	0.876888	-0.209612	-0.967300
	thalach	158.116806	-	343.123595	-0.508071	0.181271
	slope	1.660870	2	0.313806	-1.409264	1.012182
	exang	0.147826	-	0.127079	1.984484	1.938175
	oldpeak	0.577227	-	0.666727	1.872144	4.037878
	ca	0.304348	0	0.441648	2.455098	5.912879

In [225]:
```
from scipy.stats import describe

grouper = df_heart_corr.groupby('target')

variables = df_heart_corr.columns[2:12]

res = pd.concat([pd.DataFrame(describe(g[x]) for _, g in grouper)\
                 .reset_index().assign(target=x).set_index(['target', 'index']) \
                for x in variables], axis=0)

print(res)
```

```
                nobs                                      minmax        mean  \
target   index
cp       0        97                                     (0, 3)    0.412371
         1       115                                     (0, 3)    1.417391
trestbps 0        97  (100.03544943034647, 192.02020026184616)  134.149785
         1       115    (93.94418413937092, 180.1188093945384)  129.858981
chol     0        97   (130.99065518844873, 406.9326893091202)  248.087866
         1       115   (126.08581148087978, 359.9306149929762)  240.825764
fbs      0        97                                     (0, 1)    0.134021
         1       115                                     (0, 1)    0.130435
restecg  0        97                                     (0, 2)    0.505155
         1       115                                     (0, 2)    0.617391
thalach  0        97   (88.03261252120137, 195.11892717331648)  139.767679
         1       115   (105.03704770281912, 202.1380406320095)  158.116806
exang    0        97                                     (0, 1)    0.577320
         1       115                                     (0, 1)    0.147826
oldpeak  0        97  (-0.1476576328277588, 6.157113660871983)    1.717394
         1       115  (-0.1856676638126373, 4.150481455773115)    0.577227
slope    0        97                                     (0, 2)    1.134021
         1       115                                     (0, 2)    1.660870
ca       0        97                                  (0.0, 3.0)    1.072165
         1       115                                  (0.0, 3.0)    0.304348

                  variance  skewness  kurtosis
target   index
cp       0        0.703179  1.770177  1.680583
         1        0.876888 -0.209612 -0.967300
trestbps 0      342.438543  0.853828  0.538292
         1      286.922334  0.432610  0.495888
chol     0     2314.575016  0.324790  0.380786
         1     1830.960960  0.296017  0.220441
fbs      0        0.117268  2.148558  2.616300
         1        0.114416  2.194691  2.816667
restecg  0        0.315077  0.512546 -0.786353
         1        0.255835 -0.277941 -1.422108
thalach  0      462.113380 -0.165506 -0.302270
         1      343.123595 -0.508071  0.181271
exang    0        0.246564 -0.313044 -1.902003
         1        0.127079  1.984484  1.938175
oldpeak  0        1.803246  0.717616  0.494933
         1        0.666727  1.872144  4.037878
slope    0        0.325601  0.011213 -0.071926
         1        0.313806 -1.409264  1.012182
ca       0        1.088488  0.574800 -0.870307
         1        0.441648  2.455098  5.912879
```

In [218]:
```
rows = []
for column in df_heart_corr:
     rows.append([column, df_heart_corr[column].mean(), df_heart_corr[column].var(), df_heart_corr[column].skew(), df_h

df_heart_stats = pd.DataFrame(rows, columns=["column","mean", "variance","skew","kurtosis"])
df_heart_stats
```

Out[218]:

	column	mean	variance	skew	kurtosis
0	age	54.311321	83.637217	-0.106027	-0.561563
1	sex	0.688679	0.215416	-0.820789	-1.339028
2	cp	0.957547	1.045583	0.461438	-1.240674
3	trestbps	131.822226	315.412298	0.676997	0.716703
4	chol	244.148518	2055.467876	0.340569	0.413267
5	fbs	0.132075	0.115175	2.188903	2.817791
6	restecg	0.566038	0.284718	0.111397	-1.194129
7	thalach	149.721215	479.596816	-0.405718	-0.171905
8	exang	0.344340	0.226840	0.659880	-1.579550
9	oldpeak	1.098908	1.504838	1.271297	1.578970
10	slope	1.419811	0.386904	-0.586510	-0.579659
11	ca	0.655660	0.880868	1.294338	0.575571
12	thal	2.353774	0.343445	-0.268550	-0.676592
13	target	0.542453	0.249374	-0.171644	-1.989397

Observations:

- From the Heart disease descriptive statistics table, it is evident that people with heart disease (Target = 1) have higher 'thalach' mean value [158.116806] when compared to people with no heart disease (Target = 0) [139.767679].
- Majority of heart disease patients have slope = 2 i.e. flat slope of peak exercise whereas most of those people without heart disease have slope = 0 i.e. upslope of peak exercise.
- Majority of heart disease patients have cp = 2 i.e. non-angina chest pain whereas majority of people without heart disease have cp = 0 i.e. asymptomatic angina chest pain.
- People without heart disease have a higher variance value for all the attributes except chest pain when compared to the variance in the same attributes for people with heart disease.
- People without heart disease have a higher variance in 'thalach' value when compared to the people with heart disease.
- It is evident that 'ca' attribute has the highest skewness and highest kurtosis values for the entire dataset.
- For those without heart disease, attributes: 'cp', 'slope', 'oldpeak' and 'ca' are skewed to right whereas the rest of the features ('thalach', 'exang') are skewed to the left.
- For those with heart disease, attributes: 'cp', 'thalach' and 'slope' are skewed to the left whereas the rest are skewed to the right.
- People without heart disease have negative kurtosis values for 'thalach', 'slope', 'exang' and 'ca' i.e. their distribution curves flatten out more when compared to normal distribution curves.
- People with heart disease have a positive kurtosis value for all features except 'cp' i.e. their distribution curves flatten out lesser on the tails when compared to a normal distribution curve.

CM4 (Question 2.1, 2.2)

Training the KNN Model for Iris Dataset:

The accuracy observed for the KNN model for Iris dataset: **100% for test dataset** and **96% for the training dataset**. The model was trained as shown below:

In [99]:
```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Species'.
df_iris_clean['species']= label_encoder.fit_transform(df_iris_clean['species'])

df_iris_clean['species'].unique()
```

Out[99]:
```
array([1, 0, 2])
```

In [100]:
```
#Splitting the entire dataset for training and testing the model
#train - 0.8; test - 0.2

X = df_iris_clean.drop(['species'], axis = 1)
y = df_iris_clean['species']
```

In [101]:
```
X = preprocessing.StandardScaler().fit_transform(X)
```

In [102]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=98)
```

In [103]:
```
knnmodel=KNeighborsClassifier(n_neighbors=5, metric = "minkowski")
```

In [104]:
```
#model training
knnmodel.fit(X_train,y_train)
```

Out[104]:
```
KNeighborsClassifier()
```

In [105]:
```
y_predict = knnmodel.predict(X_test)
```

In [106]:
```
#print the Accuracy metrics
from sklearn.metrics import accuracy_score
acc = "Accuracy: {:3.4%}".format(accuracy_score(y_test,y_predict))
acc
```

Out[106]:
```
'Accuracy: 100.0000%'
```

In [107]:
```
from sklearn.metrics import f1_score
f1Score = "f1 Score: " ,f1_score(y_test, y_predict, average='weighted')
f1Score
```

Out[107]:
```
('f1 Score: ', 1.0)
```

In [108]:
```
from sklearn.metrics import confusion_matrix
confusion_mat=confusion_matrix(y_test.values,y_predict)
confusion_mat
```

Out[108]:
```
array([[12,  0,  0],
       [ 0,  6,  0],
       [ 0,  0,  3]])
```

In [109]:
```
confusion_mat = pd.DataFrame(data=confusion_mat,index=['setosa','versicolor','virginica'],columns=['setosa','versicolor
confusion_mat
```

Out[109]:

	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	6	0
virginica	0	0	3

In [110]:
```
prediction_output=pd.DataFrame(data=[y_test.values,y_predict],index=['y_test','y_predict'])
```

In [111]:
```
prediction_output.transpose()
```

```
In [111]: prediction_output.transpose()
```

Out[111]:

	y_test	y_predict
0	0	0
1	1	1
2	1	1
3	2	2
4	0	0
5	0	0
6	1	1
7	2	2
8	0	0
9	0	0
10	0	0
11	0	0
12	1	1
13	0	0
14	1	1
15	0	0
16	1	1
17	2	2
18	0	0
19	0	0
20	0	0

```
In [112]: prediction_output.iloc[0,:].value_counts()
```

```
Out[112]: 0    12
          1     6
          2     3
          Name: y_test, dtype: int64
```

```
In [113]: print('The accuracy of the Knn classifier on training data is {:.2f}'.format(knnmodel.score(X_train, y_train)))
          print('The accuracy of the Knn classifier on test data is {:.2f}'.format(knnmodel.score(X_test, y_test)))
```

```
The accuracy of the Knn classifier on training data is 0.96
The accuracy of the Knn classifier on test data is 1.00
```

Training the KNN Model for Heart Disease Dataset:

The accuracy observed for the KNN model for Heart disease dataset: 86.046%.
The scores obtained were as follows:

Accuracy	AUC Score	F1 Score	Confusion Matrix
86.046%	86.4253%	88%	array([[15, 2], [4, 22]])

The model was trained as shown below:

```
In [870]: df_heart_pros = pd.get_dummies(df_heart_corr, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
```

```
In [659]: df_heart_pros
```

Out[659]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_0	slope_1	slope_2	ca_0.0	ca_1.0	ca_2.0	ca_3.0	thal_1
0	76	140.102822	197.105970	115.952071	1.284822	1	1	0	0	0	...	0	1	0	1	0	0	0	
1	43	132.079599	341.049462	135.970028	3.110483	0	1	0	1	0	...	0	1	0	1	0	0	0	
2	47	107.899290	242.822816	152.210039	-0.023723	0	0	1	0	0	...	0	0	1	1	0	0	0	
3	51	99.934001	240.825764	143.049207	1.195082	1	0	1	0	0	...	0	1	0	1	0	0	0	
4	57	110.103508	334.952353	143.099327	3.082052	0	0	1	1	0	...	0	1	0	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
207	56	134.149785	256.189595	141.981335	0.606726	0	0	1	0	0	...	0	1	0	0	1	0	0	
208	57	127.981407	302.985611	158.992132	0.577227	1	1	0	1	0	...	0	0	1	0	1	0	0	
209	54	150.188534	232.117551	164.977674	1.715230	1	0	1	0	0	...	0	0	1	1	0	0	0	
210	41	129.918793	214.008059	167.851493	1.992138	1	0	1	0	0	...	0	1	0	1	0	0	0	
211	41	112.075764	268.005496	172.008896	0.085251	1	1	0	0	0	...	0	0	1	1	0	0	0	

212 rows x 29 columns

In [41]:
```
y = df_heart_pros['target']
X = df_heart_pros.drop(['target'], axis = 1)
```

In [42]:
```
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
X[columns_to_scale] = standardScaler.fit_transform(X[columns_to_scale])
```

In [43]:
```
X.head()
```

Out[43]:

	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	cp_1	cp_2	...	slope_0	slope_1	slope_2	ca_0.0	ca_1.0	ca_2.0	ca_3.0	thal_1.0
0	2.377169	0.467357	-1.040069	-1.545641	0.151913	1	0	0	0	1	...	0	1	0	1	0	0	0	0
1	-1.239767	0.014526	2.142394	-0.629403	1.643683	1	0	1	0	0	...	0	1	0	1	0	0	0	0
2	-0.801351	-1.350212	-0.029310	0.113916	-0.917316	0	1	0	0	1	...	0	0	1	1	0	0	0	0
3	-0.362934	-1.799773	-0.073463	-0.305383	0.078585	0	1	0	0	1	...	0	1	0	1	0	0	0	0
4	0.294690	-1.225806	2.007592	-0.303089	1.620452	0	1	1	0	0	...	0	1	0	0	1	0	0	0

5 rows × 28 columns

In [44]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=98)
```

In [45]:
```
knnmodel=KNeighborsClassifier(n_neighbors = 5)
knnmodel.fit(X_train,y_train)
```

Out[45]:
```
KNeighborsClassifier()
```

In [46]:
```
y_predict = knnmodel.predict(X_test)
```

In [47]:
```
from sklearn.metrics import accuracy_score
acc = "Accuracy: {:3.4%}".format(accuracy_score(y_test,y_predict))
acc
```

Out[47]:
```
'Accuracy: 86.0465%'
```

In [48]:
```
from sklearn.metrics import roc_auc_score
auc = "AUC Score: {:3.4%}".format(roc_auc_score(y_test, y_predict))
auc
```

Out[48]:
```
'AUC Score: 86.4253%'
```

In [49]:
```
from sklearn.metrics import f1_score
f1Score = "f1 Score: {:3.4%}".format(f1_score(y_test, y_predict))
f1Score
```

Out[49]:
```
'f1 Score: 88.0000%'
```

In [50]:
```
from sklearn.metrics import confusion_matrix
conmat=confusion_matrix(y_test.values,y_predict)
conmat
```

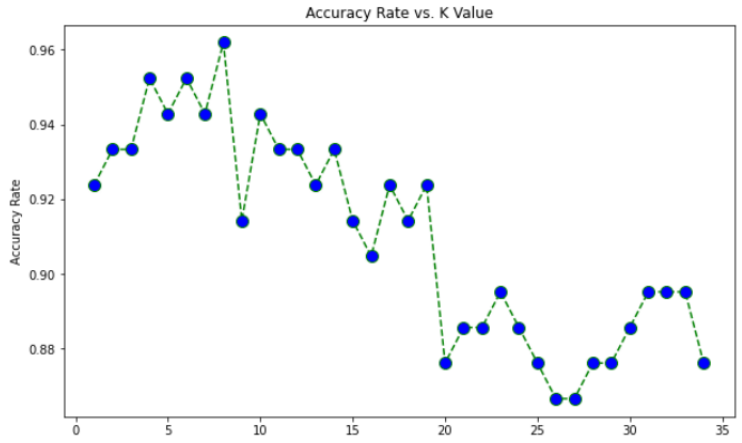
Out[50]:
```
array([[15,  2],
       [ 4, 22]])
```

CM5 (Question 3(a), 3(b) and 3(c))

Question 3(a) Fitting the KNN Model for Iris Dataset using K-Fold Cross validation (k=5):

In [115]:
```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score

accuracy_rate = []

for i in range(1,35):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,X,y,cv=5)
    accuracy_rate.append(score.mean())

plt.figure(figsize=(10,6))
plt.plot(range(1,35),accuracy_rate,color='green', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Accuracy Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy Rate')
```

Out[115]: Text(0, 0.5, 'Accuracy Rate')



In [116]:
```
#print each cv score (accuracy) and average them
print(score)
print("Accuracy:", score.mean()*100 )
```

```
[1.         0.80952381 0.80952381 0.9047619  0.85714286]
Accuracy: 87.6190476190476
```

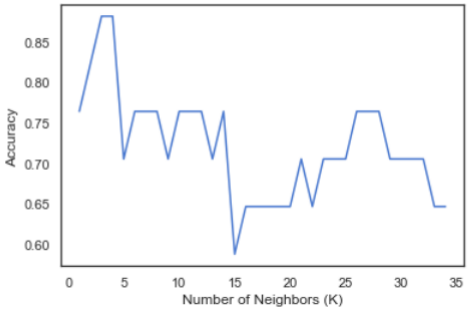
In [117]:
```
print("Variance:", score.var()*100)
```

```
Variance: 0.5079365079365078
```

Fitting the KNN Model for Heart Disease Dataset using Train-validate-test:

In [57]:
```
from sklearn.metrics import accuracy_score

b_index=list(range(1,35))
b=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh1=KNeighborsClassifier(n_neighbors=i).fit(X_train1,y_train1)
    prediction1=neigh1.predict(X_val1)
    b=b.append(pd.Series(accuracy_score(prediction1,y_val1)))
print( "The best accuracy was ", b.max(), "with k=", b.argmax()+1)
plt.plot(b_index, b)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  0.8823529411764706 with k= 3
```



Question 3(b)

Accuracy of K-Fold cross validation for Iris dataset:

We calculated the average accuracies for all folds for Iris dataset and found it to be the best at **K=8**.
The variance was observed as 0.5079%.

In [116]:
```
#print each cv score (accuracy) and average them
print(score)
print("Accuracy:", score.mean()*100 )
```

```
[1.         0.80952381 0.80952381 0.9047619  0.85714286]
Accuracy: 87.6190476190476
```

In [117]:
```
print("Variance:", score.var()*100)
```

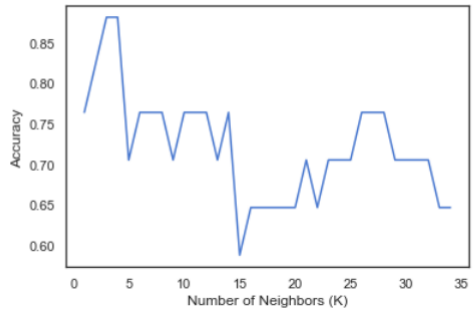
```
Variance: 0.5079365079365078
```

Accuracy of the model for Heart disease dataset:

The accuracy for the train-validate-test model for heart disease dataset was observed to be **~88% for k=3 and k=4.**

In [57]:
```
from sklearn.metrics import accuracy_score

b_index=list(range(1,35))
b=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh1=KNeighborsClassifier(n_neighbors=i).fit(X_train1,y_train1)
    prediction1=neigh1.predict(X_val1)
    b=b.append(pd.Series(accuracy_score(prediction1,y_val1)))
print( "The best accuracy was ", b.max(), "with k=", b.argmax()+1)
plt.plot(b_index, b)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  0.8823529411764706 with k= 3
```



We also implemented K-Fold cross validation on heart disease dataset and observed an **average accuracy of ~82.1484% with a variance of 0.678%.**

In [59]:
```
#print each cv score (accuracy) and average them

from sklearn.model_selection import cross_val_score

accuracy_rate = []

for i in range(1,35):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,X,y,cv=5)
    accuracy_rate.append(score.mean())

print(score)
print("Accuracy: {:3.4%}".format(score.mean() ))
print("Variance:", score.var()*100)
```

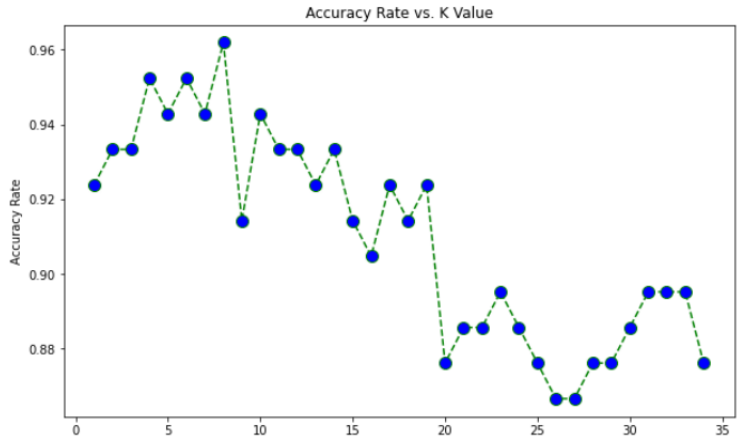
```
[0.74418605 0.74418605 0.88095238 0.78571429 0.95238095]
Accuracy: 82.1484%
Variance: 0.6779984032552987
```

Question 3(c)

Accuracy vs. K curve for Iris Dataset: The best accuracy obtained for Iris dataset with 5-fold cross validation was at **k = 8** as shown in the graph below. The **average accuracy observed was ~87.62%** across all the folds with a **variance of around 0.5079%.**

In [115]:
```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score

accuracy_rate = []

for i in range(1,35):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,X,y,cv=5)
    accuracy_rate.append(score.mean())

plt.figure(figsize=(10,6))
plt.plot(range(1,35),accuracy_rate,color='green', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Accuracy Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy Rate')
```

Out[115]: Text(0, 0.5, 'Accuracy Rate')



In [116]:
```
#print each cv score (accuracy) and average them
print(score)
print("Accuracy:", score.mean()*100 )
```

```
[1.         0.80952381 0.80952381 0.9047619  0.85714286]
Accuracy: 87.6190476190476
```

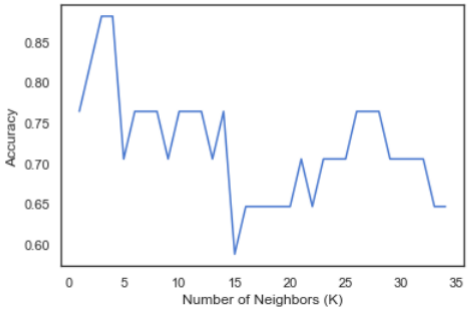
In [117]:
```
print("Variance:", score.var()*100)
```

```
Variance: 0.5079365079365078
```

Accuracy vs. K curve for Heart disease Dataset: The best accuracy obtained for Heart disease dataset with train-validate-test was at **k = 3 and k = 4** as shown in the graph below. The **best accuracy observed was ~88%.**

In [57]:
```
from sklearn.metrics import accuracy_score

b_index=list(range(1,35))
b=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh1=KNeighborsClassifier(n_neighbors=i).fit(X_train1,y_train1)
    prediction1=neigh1.predict(X_val1)
    b=b.append(pd.Series(accuracy_score(prediction1,y_val1)))
print( "The best accuracy was ", b.max(), "with k=", b.argmax()+1)
plt.plot(b_index, b)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  0.8823529411764706 with k= 3
```



Advantage of K-Fold validation over train-validate-test:

We found K-fold Cross validation more effective over train-validate test as K-fold technique involves randomly dividing the dataset into K folds of approximately equal size. The first fold is kept for testing and the model is trained on K-1 folds. This process is repeated K times, and each time different fold of data points are used for validation. Each data point gets to be tested exactly once and is accounted for in training k-1 times. Size of the training set in k-fold is more than that of train validation test. We also observed that the k-fold cross validation uses each sample at least once in training and test dataset, due to which it generally lowers the bias of the final model obtained. We also found that the variance of the resulting estimate is reduced as K increases. Another drawback that we observed with the train-validate-test method is that we need considerable amount of dataset to implement it effectively, otherwise it might lead to overfitting in a very small sample set.

CM6 (Question 3(d))

Fitting the model on the entire training set and predicting the target on test set using the best-found parameter

Iris Dataset:

We observed that the maximum accuracy rate during cross validation for Iris dataset was obtained at **K=8**; thus, fitting our model for the same on the entire training set.

Data	Accuracy	F1 Score	AUC
Training set	95%	0.95	1.0
Test set	100%	1.0	1.0

Confusion matrix:

	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	6	0
virginica	0	0	3

Observations:

- The highest accuracy of 95% was recorded at K = 8, the lowest accuracy of 86% was recorded at K = 27.
- After the optimum value of k, the accuracies diminish as the value of k increases.
- This happened because, In K fold cross-validation, the training data will be randomly split into K parts. K-1 of which will be used for training and 1 will be used for validation. This process repeats again and again until all possible combination using these K splits are used for both training and validation. Finally, we can take the overall accuracy as the mean accuracy of the K-fold cross-validation samples. The accuracy of such a model will always be low because the model is repeatedly tested against unseen data all the time, and the final accuracy score is the mean of these individual test scores which as we mentioned will always be less than training.

Code:

```
In [118]: X = df_iris_clean.drop(['species'], axis = 1)
          y = df_iris_clean['species']

In [119]: X = preprocessing.StandardScaler().fit_transform(X)

In [120]: X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=98)

In [121]: knn2model=KNeighborsClassifier(n_neighbors=8, metric = "minkowski")

In [122]: #model training
          knn2model.fit(X_train,y_train)
Out[122]: KNeighborsClassifier(n_neighbors=8)

In [123]: y_predict = knn2model.predict(X_test)

In [124]: #print the Accuracy metrics
          from sklearn.metrics import accuracy_score
          acc = "Accuracy: {:3.4%}".format(accuracy_score(y_test,y_predict))
          acc
Out[124]: 'Accuracy: 100.0000%'

In [125]: from sklearn.metrics import f1_score
          f1Score = "f1 Score: " ,f1_score(y_test, y_predict, average='weighted')
          f1Score
Out[125]: ('f1 Score: ', 1.0)

In [126]: from sklearn.metrics import confusion_matrix
          confusion_mat=confusion_matrix(y_test.values,y_predict)
          confusion_mat
Out[126]: array([[12,  0,  0],
                 [ 0,  6,  0],
                 [ 0,  0,  3]])
```

In [127]:
```
confusion_mat = pd.DataFrame(data=confusion_mat,index=['setosa','versicolor','virginica'],columns=['setosa','versicolor','virginica'])
confusion_mat
```

Out[127]:

	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	6	0
virginica	0	0	3

In [128]:
```
prediction_output=pd.DataFrame(data=[y_test.values,y_predict],index=['y_test','y_predict'])
```

In [129]:
```
prediction_output.transpose()
```

Out[129]:

	y_test	y_predict
0	0	0
1	1	1
2	1	1
3	2	2
4	0	0
5	0	0
6	1	1
7	2	2
8	0	0
9	0	0
10	0	0
11	0	0
12	1	1
13	0	0
14	1	1
15	0	0
16	1	1
17	2	2
18	0	0

In [130]:
```
prediction_output.iloc[0,:].value_counts()
```

Out[130]:
```
0    12
1     6
2     3
Name: y_test, dtype: int64
```

In [131]:
```
print('The accuracy of the Knn classifier on training data is {:.2f}'.format(knn2model.score(X_train, y_train)))
print('The accuracy of the Knn classifier on test data is {:.2f}'.format(knn2model.score(X_test, y_test)))
```

```
The accuracy of the Knn classifier on training data is 0.95
The accuracy of the Knn classifier on test data is 1.00
```

Heart Disease Dataset:

We observed that the maximum accuracy rate during train-test-validate implementation is obtained at **K=3 and K=4**; thus, fitting our model for the same.

Data	Accuracy	F1 Score	AUC
Test set	90.69%	92.59%	89.25%

Observations:

- The highest accuracy of ~90% was recorded at K = 3 and K=4, the lowest accuracy of ~60% was recorded at K = 15.
- After attaining the maximum accuracy at an optimum K value, the accuracies decrease to its lowest value at K=15, and then increases a little. It finally diminishes after a certain K value. The decrease in the accuracy values as the number of neighbors (K) increases might be because, as K increases, the farther neighbor gets a say and impacts the classification of a data sample which might lead to inaccurate labelling and classification, thus leading to underfitting of the data. A very lower value of K on the other hand, might lead to overfitting of the data as each close neighbor impacts the sample to a much larger extent.

Code:

Fitting the model on the entire training set and predicting the target on test set using the best found parameter

In [60]:
```
y1 = df_heart_pros['target']
X1 = df_heart_pros.drop(['target'], axis = 1)

from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
X1[columns_to_scale] = standardScaler.fit_transform(X1[columns_to_scale])
```

In [61]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split( X1, y1, test_size=0.2, random_state=98)
```

In [62]:
```
knn2model=KNeighborsClassifier(n_neighbors = 3)
knn2model.fit(X_train2,y_train2)
```

Out[62]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [63]:
```
y_predict2 = knn2model.predict(X_test2)
```

In [64]:
```
from sklearn.metrics import accuracy_score
acc = "Accuracy: {:3.4%}".format(accuracy_score(y_test2,y_predict2))
acc
```

Out[64]:
```
'Accuracy: 90.6977%'
```

In [65]:
```
from sklearn.metrics import roc_auc_score
auc = "AUC Score: {:3.4%}".format(roc_auc_score(y_test2, y_predict2))
auc
```

Out[65]:
```
'AUC Score: 89.2534%'
```

In [66]:
```
from sklearn.metrics import f1_score
f1Score = "f1 Score: {:3.4%}".format(f1_score(y_test2, y_predict2))
f1Score
```

Out[66]:
```
'f1 Score: 92.5926%'
```

CM7 (Questions 5, 6 and 7)

Question 5 Weighted KNN Algorithm:

There is a 'weights' parameter for KNN algorithm, that could be 'uniform' or 'distance'. While training the model earlier, it was kept as default i.e. 'uniform'. For weighted KNN algorithm, the 'weights' parameter has been set to 'distance'. This means that the neighbors closer to the test sample to be classified would have a higher weightage when compared to the neighbors that are farther in distance from the test sample.

We have used two distance metrics to train the model: Manhattan distance and Euclidean distance.

Iris Dataset:

Distance Metrics	K values with maximum Accuracy	Optimum K value chosen	Accuracy	F1 Score	AUC
Euclidean	K = {1,2, 5, 6, 7, 8, 9, 10, 11, 12, 14}	**K = 9**	100%	1.0	1.0
Manhattan	K = {5,6,7,8,9,10,11,12,13,14,15,16,17,18,19}	**K = 12**	100%	1.0	1.0

As mentioned above, the highest accuracy occurred at following **K values for Weighted KNN model with Euclidean distance** metrics: {1,2, 5, 6, 7, 8, 9, 10, 11, 12, 14}. We chose **K = 9** as the optimum K value for, because a small value of K might make our model more susceptible to noise. This might further lead to high variations in the performance for unobserved data samples. On the other hand, a very high K value might be computationally more expensive and time taking.

For similar reasons, we have chosen **K=12 as the optimum value for Weighted KNN model with Manhattan distance metrics**, even though we achieved the highest accuracy for these K values: {5,6,7,8,9,10,11,12,13,14,15,16,17,18,19}.

Observations:

- For Iris dataset, both the models performed equally well as it is quite evident from the accuracy, AUC and F1 score values.

Code:

Weighted KNN - Using Euclidean distance and weights = 'distance'

In [133]:
```
#importing all the resources for preprocessing and creating pipeline
from sklearn import preprocessing
```

In [134]:
```
X3 = df_iris_clean.drop(['species'], axis = 1)
y3 = df_iris_clean['species']
y3.head(1)
```

Out[134]:
```
0    1
Name: species, dtype: int64
```

In [135]:
```
X3 = preprocessing.StandardScaler().fit_transform(X3)
```

In [136]:
```
X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3, test_size=0.2, random_state=98)
```

In [137]:
```
knn3model=KNeighborsClassifier(n_neighbors = 5,metric='euclidean', weights='distance')
knn3model.fit(X_train3,y_train3)
```

Out[137]:
```
KNeighborsClassifier(metric='euclidean', weights='distance')
```

In [138]:
```
y_predict3 = knn3model.predict(X_test3)
```

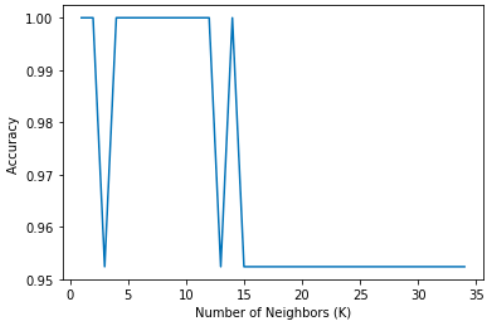
In [139]:
```
from sklearn.metrics import accuracy_score
acc2 = accuracy_score(y_test3,y_predict3)
acc2
```

Out[139]:
```
1.0
```

In [140]:
```
from sklearn.metrics import confusion_matrix
conmat2=confusion_matrix(y_test3.values,y_predict3)
conmat2
```

Out[140]:
```
array([[12,  0,  0],
       [ 0,  6,  0],
       [ 0,  0,  3]])
```

In [141]:
```
from sklearn.metrics import accuracy_score

c_index=list(range(1,35))
c=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh3=KNeighborsClassifier(n_neighbors=i, metric='euclidean', weights='distance').fit(X_train3,y_train3)
    prediction3=neigh3.predict(X_test3)
    c=c.append(pd.Series(accuracy_score(prediction3,y_test3)))
print( "The best accuracy was ", c.max(), "with k=", c.argmax()+1)
plt.plot(c_index, c)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```



In [142]:
```
#Fitting the weighted KNN (Euclidean distance) model with k=9
knn4model=KNeighborsClassifier(n_neighbors = 9,metric='euclidean', weights='distance')
knn4model.fit(X3,y3)
```

Out[142]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=9, weights='distance')
```

In [143]:
```
y_predict_Euclidean = knn4model.predict(X_test3)
```

In [237]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

acc3 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_Euclidean))
f1Score3 = "f1 Score: " ,f1_score(y_test3,y_predict_Euclidean, average='weighted')
auc = roc_auc_score(y_test,knn4model.predict_proba(X_test3),multi_class="ovr")
acc3, f1Score3, auc
```

Out[237]:
```
('Accuracy: 100.0000%', ('f1 Score: ', 1.0), 1.0)
```

Weighted KNN - Using Manhatten distance and weights = 'distance'

In [146]:
```
knn5model=KNeighborsClassifier(n_neighbors = 5,metric='manhattan', weights='distance')
knn5model.fit(X3,y3)
```

Out[146]:
```
KNeighborsClassifier(metric='manhattan', weights='distance')
```

In [147]:
```
y_predict4 = knn5model.predict(X_test3)
```

In [148]:
```
from sklearn.metrics import accuracy_score
acc3 = accuracy_score(y_test3,y_predict4)
acc3
```

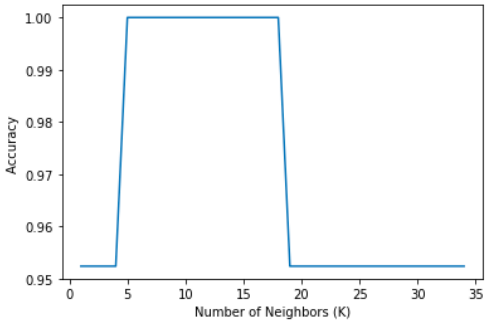
Out[148]:
```
1.0
```

In [150]:
```
from sklearn.metrics import confusion_matrix
conmat3=confusion_matrix(y_test3.values,y_predict4)
conmat3
```

Out[150]:
```
array([[12,  0,  0],
       [ 0,  6,  0],
       [ 0,  0,  3]])
```

In [151]:
```
from sklearn.metrics import accuracy_score

d_index=list(range(1,35))
d=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh4=KNeighborsClassifier(n_neighbors=i, metric='manhattan', weights='distance').fit(X_train3,y_train3)
    prediction4=neigh4.predict(X_test3)
    d=d.append(pd.Series(accuracy_score(prediction4,y_test3)))
print( "The best accuracy was ", d.max(), "with k=", d.argmax()+1)
plt.plot(d_index, d)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  1.0 with k= 5
```



In [153]:
```
#Fitting the weighted KNN (Manhattan distance) model with k=12
knn6model=KNeighborsClassifier(n_neighbors =12,metric='manhattan', weights='distance')
knn6model.fit(X3,y3)
```

Out[153]:
```
KNeighborsClassifier(metric='manhattan', n_neighbors=12, weights='distance')
```

In [154]:
```
y_predict_Manhattan = knn6model.predict(X_test3)
```

In [238]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

acc3 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_Manhattan))
f1Score3 = "f1 Score: " ,f1_score(y_test3,y_predict_Manhattan, average='weighted')
auc = roc_auc_score(y_test,knn6model.predict_proba(X_test3),multi_class="ovr")
acc3, f1Score3, auc
```

Out[238]:
```
('Accuracy: 100.0000%', ('f1 Score: ', 1.0), 1.0)
```

Heart Disease Dataset:

Distance Metrics	K values with maximum Accuracy	Optimum K value chosen	Accuracy	F1 Score	AUC
Euclidean	K = {4,9,14}	**K = 4**	88.37%	90.56%	87.33%
Manhattan	K = {3, 21, 22, 23, 24, 26, 27, 29, 31}	**K = 3**	90.69%	92%	91.28%

As mentioned above, the highest accuracy occurred at following **K values for Weighted KNN model with Euclidean distance** metrics: {4, 9 , 14}. We chose **K = 4** as the optimum K value for, because a very high K value might be computationally more expensive.

For similar reasons, we have chosen **K=3 as the optimum value for Weighted KNN model with Manhattan distance metrics**, even though we achieved the highest accuracy for these K values: {3, 21, 22, 23, 24, 26, 27, 29, 31}.

Observations:

- For Heart disease dataset, the **Weighted KNN model with Manhattan distance metrics performed better** than the model with Euclidean distance metrics as it is quite evident from the higher accuracy, AUC and F1 score values of the Manhattan distance model as compared to the Euclidean distance model.

Code:

Weighted KNN - Using Euclidean distance and weights = 'distance'

In [1106]:
```
#importing all the resources for preprocessing and creating pipeline
from sklearn import preprocessing
```

In [1107]:
```
df_heart_pros1 = pd.get_dummies(df_heart_corr, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'
```

In [1108]:
```
y3 = df_heart_pros1['target']
X3 = df_heart_pros1.drop(['target'], axis = 1)
```

In [1109]:
```
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
X3[columns_to_scale] = standardScaler.fit_transform(X3[columns_to_scale])
```

In [1110]:
```
X_train3, X_test3, y_train3, y_test3 = train_test_split( X3, y3, test_size=0.2, random_state=98)
```

In [1111]:
```
X_train3.head()
```

Out[1111]:

	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	cp_1	cp_2	...	slope_0	slope_1	slope_2	ca_0.0	ca_1.0	ca_2.0	ca_3.0	thal_1
163	0.952315	-0.438565	-1.043354	-0.633654	-0.969532	1	0	1	0	0	...	0	1	0	1	0	0	0	
155	-0.910955	-1.733606	-1.043446	0.283352	-0.904794	0	1	0	1	0	...	0	0	1	1	0	0	0	
52	-0.362934	0.457566	1.217748	1.060194	0.267741	0	1	1	0	0	...	0	0	1	1	0	0	0	
78	-0.472539	-1.226071	-0.073463	0.419944	-0.426271	1	0	1	0	0	...	0	0	1	1	0	0	0	
44	-1.678184	-2.137273	-0.998293	1.341453	-0.768668	1	0	0	0	1	...	0	0	1	1	0	0	0	

5 rows × 28 columns

In [1112]:
```
X_train4, X_val4, y_train4, y_val4 = train_test_split(X_train3, y_train3, test_size=0.10, random_state=98)
```

In [1113]:
```
X_train4.head()
```

Out[1113]:

	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	cp_1	cp_2	...	slope_0	slope_1	slope_2	ca_0.0	ca_1.0	ca_2.0	ca_3.0	thal_1
169	-1.897392	-0.094497	0.127610	1.698545	2.115747	0	1	0	0	1	...	1	0	0	1	0	0	0	
32	-1.787788	-0.669165	-0.287469	1.475874	2.182410	0	1	0	0	0	...	0	1	0	1	0	0	0	
136	1.719544	1.370524	-0.073463	-0.310852	-0.917123	0	1	0	1	0	...	0	0	1	1	0	0	0	
173	0.404294	-0.211034	-0.621630	-0.857091	1.023524	0	1	1	0	0	...	0	1	0	0	0	0	1	
15	0.075482	-0.204654	-0.868149	-0.898189	0.612430	1	0	1	0	0	...	0	1	0	0	1	0	0	

5 rows × 28 columns

In [1114]:
```
knn3model=KNeighborsClassifier(n_neighbors = 5,metric='euclidean', weights='distance')
knn3model.fit(X_train4,y_train4)
```

Out[1114]:
```
KNeighborsClassifier(metric='euclidean', weights='distance')
```

In [1115]:
```
y_predict3 = knn3model.predict(X_val4)
```

In [1116]:
```
from sklearn.metrics import accuracy_score
acc2 = accuracy_score(y_val4,y_predict3)
acc2
```

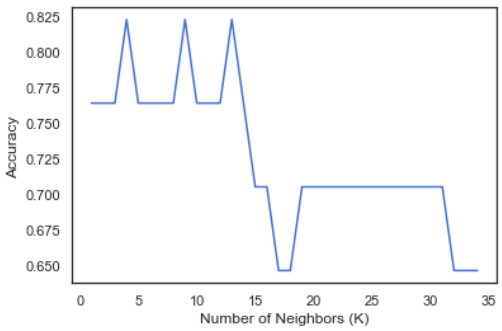
Out[1116]:
```
0.7647058823529411
```

In [1117]:
```
from sklearn.metrics import confusion_matrix
conmat2=confusion_matrix(y_val4.values,y_predict3)
conmat2
```

Out[1117]:
```
array([[7, 2],
       [2, 6]])
```

In [1118]:
```
from sklearn.metrics import accuracy_score

c_index=list(range(1,35))
c=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh3=KNeighborsClassifier(n_neighbors=i, metric='euclidean', weights='distance').fit(X_train4,y_train4)
    prediction3=neigh3.predict(X_val4)
    c=c.append(pd.Series(accuracy_score(prediction3,y_val4)))
print( "The best accuracy was ", c.max(), "with k=", c.argmax()+1)
plt.plot(c_index, c)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  0.8235294117647058 with k= 4
```



In [1119]:
```
#Fitting the weighted KNN (Euclidean distance) model with k=4
knn4model=KNeighborsClassifier(n_neighbors = 4,metric='euclidean', weights='distance')
knn4model.fit(X_train3,y_train3)
```

In [1119]:
```
#Fitting the weighted KNN (Euclidean distance) model with k=4
knn4model=KNeighborsClassifier(n_neighbors = 4,metric='euclidean', weights='distance')
knn4model.fit(X_train3,y_train3)
```

Out[1119]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=4, weights='distance')
```

In [1120]:
```
y_predict_Euclidean = knn4model.predict(X_test3)
```

In [1121]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

acc3 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_Euclidean))
acc3
```

Out[1121]:
```
'Accuracy: 88.3721%'
```

In [1122]:
```
auc = "AUC Score: {:3.4%}".format(roc_auc_score(y_test3, y_predict_Euclidean))
auc
```

Out[1122]:
```
'AUC Score: 87.3303%'
```

In [1123]:
```
f1Score = "f1 Score: {:3.4%}".format(f1_score(y_test3, y_predict_Euclidean))
f1Score
```

Out[1123]:
```
'f1 Score: 90.5660%'
```

Weighted KNN - Using Manhatten distance and weights = 'distance'

In [1124]:
```
knn5model=KNeighborsClassifier(n_neighbors = 5,metric='manhattan', weights='distance')
knn5model.fit(X_train4,y_train4)
```

Out[1124]:
```
KNeighborsClassifier(metric='manhattan', weights='distance')
```

In [1125]:
```
y_predict4 = knn5model.predict(X_val4)
```

In [1126]:
```
from sklearn.metrics import accuracy_score
acc3 = accuracy_score(y_val4,y_predict4)
acc3
```

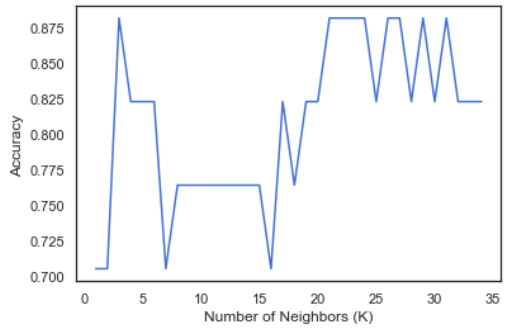
Out[1126]:
```
0.8235294117647058
```

In [1127]:
```
from sklearn.metrics import confusion_matrix
conmat3=confusion_matrix(y_val4.values,y_predict4)
conmat3
```

Out[1127]:
```
array([[7, 2],
       [1, 7]])
```

In [1128]:
```
from sklearn.metrics import accuracy_score

d_index=list(range(1,35))
d=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh4=KNeighborsClassifier(n_neighbors=i, metric='manhattan', weights='distance').fit(X_train4,y_train4)
    prediction4=neigh4.predict(X_val4)
    d=d.append(pd.Series(accuracy_score(prediction4,y_val4)))
print( "The best accuracy was ", d.max(), "with k=", d.argmax()+1)
plt.plot(d_index, d)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  0.8823529411764706 with k= 3
```




```
In [1129]: #Fitting the weighted KNN (Manhattan distance) model with k=3
           knn6model=KNeighborsClassifier(n_neighbors = 3,metric='manhattan', weights='distance')
           knn6model.fit(X_train3,y_train3)

Out[1129]: KNeighborsClassifier(metric='manhattan', n_neighbors=3, weights='distance')

In [1130]: y_predict_Manhattan = knn6model.predict(X_test3)

In [1131]: from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

           acc4 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_Manhattan))
           acc4

Out[1131]: 'Accuracy: 90.6977%'

In [1132]: auc = "AUC Score: {:3.4%}".format(roc_auc_score(y_test3, y_predict_Manhattan))
           auc

Out[1132]: 'AUC Score: 91.2896%'

In [1133]: f1Score = "f1 Score: {:3.4%}".format(f1_score(y_test3, y_predict_Manhattan))
           f1Score

Out[1133]: 'f1 Score: 92.0000%'
```

Question 6 Different NN Algorithms:

There are 3 different NN Algorithms that can be chosen within KNN parameters: Ball Tree Algorithm, KD Tree Algorithm and Brute Force Algorithm. We have used the following two algorithms: **Ball Tree Algorithm, and KD Tree Algorithm.**

Ball tree algorithm is a space partitioning data structure for organizing points in a multi-dimensional space it partitions data points into a nested set of hyperspheres known as "balls". The resulting data structure has characteristics that make it useful for nearest neighbor search.

Kd tree is a space-partitioning data structure for organizing points in a k-dimensional space.

Iris Dataset:

Distance Metrics	K values with maximum Accuracy	Optimum K value chosen	Accuracy	F1 Score	AUC
Ball Tree	K = {1,3,7,9,10}	**K = 10**	100%	1.0	1.0
KD Tree	K = {1,3,7,9,10}	**K = 10**	100%	1.0	1.0

As mentioned above, the highest accuracy occurred at following **K values for the KNN model with Ball tree algorithm**: {1,3,7,9,10}. We chose **K = 10** as the optimum K value for, because a small value of K might make our model more susceptible to noise. This might further lead to high variations in the performance for unobserved data samples. On the other hand, a very high K value might be computationally more expensive and time taking.

For similar reasons, we have chosen **K=10 as the optimum value for KNN model with KD tree algorithm**, even though we achieved the highest accuracy for these K values: {1,3,7,9,10}.

Observations:

- For Iris dataset, both the models performed equally well as it is quite evident from the accuracy, AUC and F1 score values.

Code:

Ball Tree Algorithm for KNN Model

In [156]:
```
#Using Ball Tree algorithm in KNN Model
knn7model=KNeighborsClassifier(n_neighbors = 5,metric='minkowski', algorithm = 'ball_tree', weights = 'uniform', p = 1)
knn7model.fit(X3,y3)
```

Out[156]:
```
KNeighborsClassifier(algorithm='ball_tree', p=1)
```

In [157]:
```
y_predict5 = knn7model.predict(X_test3)
```

In [158]:
```
from sklearn.metrics import accuracy_score
acc5 = accuracy_score(y_test3,y_predict5)
acc5
```

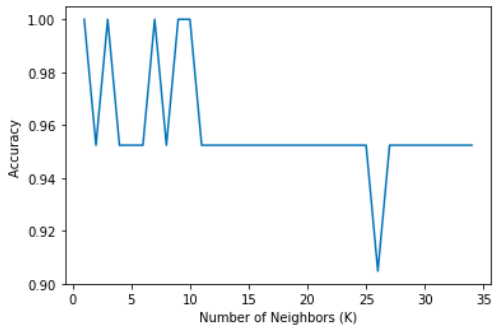
Out[158]:
```
0.9523809523809523
```

In [159]:
```
from sklearn.metrics import confusion_matrix
conmat4=confusion_matrix(y_test3.values,y_predict5)
conmat4
```

Out[159]:
```
array([[12,  0,  0],
       [ 0,  6,  0],
       [ 0,  1,  2]])
```

In [160]:
```
from sklearn.metrics import accuracy_score

e_index=list(range(1,35))
e=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh5=KNeighborsClassifier(n_neighbors=i, metric='minkowski', algorithm = 'ball_tree').fit(X3,y3)
    prediction5=neigh5.predict(X_test3)
    e=e.append(pd.Series(accuracy_score(prediction5,y_test3)))
print( "The best accuracy was ", e.max(), "with k=", e.argmax()+1)
plt.plot(e_index, e)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  1.0 with k= 1
```



In [161]:
```
#Fitting the weighted KNN (Ball Tree algorithm) model with k=10
knn8model=KNeighborsClassifier(n_neighbors = 10,metric='minkowski', algorithm = 'ball_tree')
knn8model.fit(X3,y3)
```

Out[161]:
```
KNeighborsClassifier(algorithm='ball_tree', n_neighbors=10)
```

In [162]:
```
y_predict_BallTree = knn8model.predict(X_test3)
```

In [239]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

acc3 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_BallTree))
f1Score3 = "f1 Score: " ,f1_score(y_test3,y_predict_BallTree, average='weighted')
auc = roc_auc_score(y_test,knn8model.predict_proba(X_test3),multi_class="ovr")
acc3, f1Score3, auc
```

Out[239]:
```
('Accuracy: 100.0000%', ('f1 Score: ', 1.0), 1.0)
```

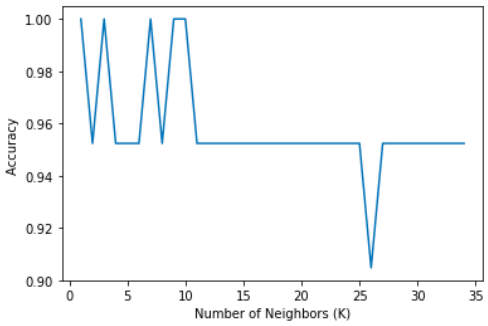
KDTree Algorithm for KNN Model

In [164]:
```
#Using Kd Tree algorithm in KNN Model
knn9model=KNeighborsClassifier(n_neighbors = 5,metric='minkowski', algorithm = 'kd_tree')
knn9model.fit(X3,y3)
```

Out[164]:
```
KNeighborsClassifier(algorithm='kd_tree')
```

In [165]:
```
y_predict6 = knn9model.predict(X_test3)
```

```
In [166]: from sklearn.metrics import accuracy_score
          acc7 = accuracy_score(y_test3,y_predict6)
          acc7

Out[166]: 0.9523809523809523

In [167]: from sklearn.metrics import confusion_matrix
          conmat5=confusion_matrix(y_test3.values,y_predict6)
          conmat5

Out[167]: array([[12,  0,  0],
                 [ 0,  6,  0],
                 [ 0,  1,  2]])

In [168]: from sklearn.metrics import accuracy_score

          f_index=list(range(1,35))
          f=pd.Series(dtype = 'float64')
          x=[0,5,10,15,20,25,30,35]
          for i in list(range(1,35)):
              neigh6=KNeighborsClassifier(n_neighbors=i, metric='minkowski', algorithm = 'kd_tree').fit(X3,y3)
              prediction6=neigh6.predict(X_test3)
              f=f.append(pd.Series(accuracy_score(prediction6,y_test3)))
          print( "The best accuracy was ", f.max(), "with k=", f.argmax()+1)
          plt.plot(f_index, f)
          plt.xticks(x)
          plt.ylabel('Accuracy ')
          plt.xlabel('Number of Neighbors (K)')
          plt.show()
```




```
In [170]: #Fitting the weighted KNN (KD Tree algorithm) model with k=10
          knn_10model=KNeighborsClassifier(n_neighbors = 10,metric='minkowski', algorithm = 'kd_tree')
          knn_10model.fit(X3,y3)

Out[170]: KNeighborsClassifier(algorithm='kd_tree', n_neighbors=10)

In [171]: y_predict_KDTree = knn_10model.predict(X_test3)

In [240]: from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

          acc3 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_KDTree))
          f1Score3 = "f1 Score: " ,f1_score(y_test3,y_predict_KDTree, average='weighted')
          auc = roc_auc_score(y_test,knn_10model.predict_proba(X_test3),multi_class="ovr")
          acc3, f1Score3, auc

Out[240]: ('Accuracy: 100.0000%', ('f1 Score: ', 1.0), 1.0)
```

Heart Disease Dataset:

Distance Metrics	K values with maximum Accuracy	Optimum K value chosen	Accuracy	F1 Score	AUC
Ball Tree	K = {9,13}	**K = 9**	88.37%	90.19%	88.34%
KD Tree	K = {9,13}	**K = 9**	88.37%	90.19%	88.34%

As mentioned above, the highest accuracy occurred at following **K values for the KNN model with Ball tree algorithm**: {9,13}. We chose **K = 9** as the optimum K value for, because a very high K value might be computationally more expensive and time taking.

For similar reasons, we have chosen **K=9 as the optimum value for KNN model with KD tree algorithm**, even though we achieved the highest accuracy for these K values: {9,13}.

Observations:

- For Heart disease dataset, both the models (Ball tree algorithm and KD Tree algorithm) performed equally well as it is quite evident from the accuracy, AUC and F1 score values.

Code:

Ball Tree Algorithm for KNN Model

In [1134]:
```
#Using Ball Tree algorithm in KNN Model
knn7model=KNeighborsClassifier(n_neighbors = 5,metric='minkowski', algorithm = 'ball_tree', weights = 'uniform', p = 1)
knn7model.fit(X_train4,y_train4)
```

Out[1134]:
```
KNeighborsClassifier(algorithm='ball_tree', p=1)
```

In [1135]:
```
y_predict5 = knn7model.predict(X_val4)
```

In [1136]:
```
from sklearn.metrics import accuracy_score
acc5 = accuracy_score(y_val4,y_predict5)
acc5
```

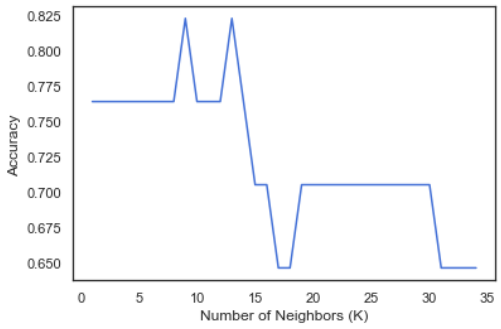
Out[1136]:
```
0.8235294117647058
```

In [1137]:
```
from sklearn.metrics import confusion_matrix
conmat4=confusion_matrix(y_val4.values,y_predict5)
conmat4
```

Out[1137]:
```
array([[7, 2],
       [1, 7]])
```

In [1138]:
```
from sklearn.metrics import accuracy_score

e_index=list(range(1,35))
e=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh5=KNeighborsClassifier(n_neighbors=i, metric='minkowski', algorithm = 'ball_tree').fit(X_train4,y_train4)
    prediction5=neigh5.predict(X_val4)
    e=e.append(pd.Series(accuracy_score(prediction5,y_val4)))
print( "The best accuracy was ", e.max(), "with k=", e.argmax()+1)
plt.plot(e_index, e)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  0.8235294117647058 with k= 9
```



In [1139]:
```
#Fitting the weighted KNN (Ball Tree algorithm) model with k=9
knn8model=KNeighborsClassifier(n_neighbors = 9,metric='minkowski', algorithm = 'ball_tree')
knn8model.fit(X_train3,y_train3)
```

Out[1139]:
```
KNeighborsClassifier(algorithm='ball_tree', n_neighbors=9)
```

In [1140]:
```
y_predict_BallTree = knn8model.predict(X_test3)
```

In [1141]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

acc6 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_BallTree))
acc6
```

Out[1141]:
```
'Accuracy: 88.3721%'
```

In [1142]:
```
auc = "AUC Score: {:3.4%}".format(roc_auc_score(y_test3, y_predict_BallTree))
auc
```

Out[1142]:
```
'AUC Score: 88.3484%'
```

In [1143]:
```
f1Score = "f1 Score: {:3.4%}".format(f1_score(y_test3, y_predict_BallTree))
f1Score
```

Out[1143]:
```
'f1 Score: 90.1961%'
```

KDTree Algorithm for KNN Model

In [1144]:
```
#Using Kd Tree algorithm in KNN Model
knn9model=KNeighborsClassifier(n_neighbors = 5,metric='minkowski', algorithm = 'kd_tree')
knn9model.fit(X_train4,y_train4)
```

Out[1144]:
```
KNeighborsClassifier(algorithm='kd_tree')
```

In [1145]:
```
y_predict6 = knn9model.predict(X_val4)
```

In [1146]:
```
from sklearn.metrics import accuracy_score
acc7 = accuracy_score(y_val4,y_predict6)
acc7
```

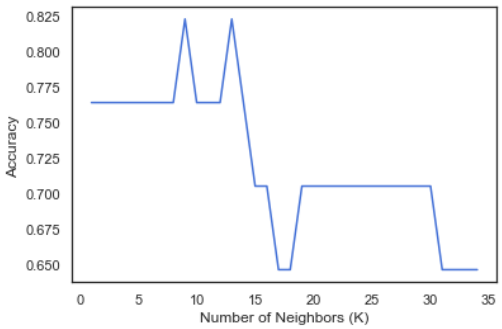
Out[1146]:
```
0.7647058823529411
```

In [1147]:
```
from sklearn.metrics import confusion_matrix
conmat5=confusion_matrix(y_val4.values,y_predict6)
conmat5
```

Out[1147]:
```
array([[7, 2],
       [2, 6]])
```

In [1148]:
```
from sklearn.metrics import accuracy_score

f_index=list(range(1,35))
f=pd.Series(dtype = 'float64')
x=[0,5,10,15,20,25,30,35]
for i in list(range(1,35)):
    neigh6=KNeighborsClassifier(n_neighbors=i, metric='minkowski', algorithm = 'kd_tree').fit(X_train4,y_train4)
    prediction6=neigh6.predict(X_val4)
    f=f.append(pd.Series(accuracy_score(prediction6,y_val4)))
print( "The best accuracy was ", f.max(), "with k=", f.argmax()+1)
plt.plot(f_index, f)
plt.xticks(x)
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.show()
```

```
The best accuracy was  0.8235294117647058 with k= 9
```



In [1149]:
```
#Fitting the weighted KNN (Ball Tree algorithm) model with k=9
knn_10model=KNeighborsClassifier(n_neighbors = 9,metric='minkowski', algorithm = 'kd_tree')
knn_10model.fit(X_train3,y_train3)
```

Out[1149]:
```
KNeighborsClassifier(algorithm='kd_tree', n_neighbors=9)
```

In [1150]:
```
y_predict_KDTree = knn_10model.predict(X_test3)
```

In [1151]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score

acc8 = "Accuracy: {:3.4%}".format(accuracy_score(y_test3,y_predict_KDTree))
acc8
```

Out[1151]:
```
'Accuracy: 88.3721%'
```

In [1152]:
```
auc = "AUC Score: {:3.4%}".format(roc_auc_score(y_test3, y_predict_KDTree))
auc
```

Out[1152]:
```
'AUC Score: 88.3484%'
```

In [1153]:
```
f1Score = "f1 Score: {:3.4%}".format(f1_score(y_test3, y_predict_KDTree))
f1Score
```

Out[1153]:
```
'f1 Score: 90.1961%'
```

Question 7 Accuracy, AUC and F1 scores for various algorithms:

Iris Dataset:

Model Type	Accuracy	F1 Score	AUC
KNN Model (weights = 'uniform')	100%	1.0	1.0
Weighted KNN Model with Euclidean Distance Metrics	100%	1.0	1.0
Weighted KNN Model with Manhattan Distance Metrics	100%	1.0	1.0
KNN Model with Ball Tree algorithm	100%	1.0	1.0
KNN Model with KD Tree algorithm	100%	1.0	1.0

Observations:

- For Iris dataset, all the models performed equally well as it is quite evident from the accuracy, AUC and F1 score values.

Heart Disease Dataset:

Model Type	Accuracy	F1 Score	AUC
KNN Model (weights = 'uniform')	90.69%	92.59%	89.25%
Weighted KNN Model with Euclidean Distance Metrics	88.37%	90.56%	87.33%
Weighted KNN Model with Manhattan Distance Metrics	90.69%	92%	91.28%
KNN Model with Ball Tree algorithm	88.37%	90.19%	88.34%
KNN Model with KD Tree algorithm	88.37%	90.19%	88.34%

Observations:

- The best performance for Heart disease dataset was observed with the weighted KNN model using Manhattan distance metrics as it had an overall Accuracy, AUC score and F1 score.
- The **Weighted KNN model with Manhattan distance metrics performed better** than the model with Euclidean distance metrics as it is quite evident from the higher accuracy, AUC and F1 score values of the Manhattan distance model as compared to the Euclidean distance model.
- Both the models (Ball tree algorithm and KD Tree algorithm) performed equally well as it is quite evident from the accuracy, AUC and F1 score values.