In [1]:

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
df = pd.read_csv('uber.csv')
df.head()
```

Out[2]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1.0
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1.0
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1.0
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3.0
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5.0

In [3]:

```
df.shape
```

Out[3]:

```
(26782, 9)
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26782 entries, 0 to 26781
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         26782 non-null  int64  
 1   key                26782 non-null  object 
 2   fare_amount        26782 non-null  float64
 3   pickup_datetime    26782 non-null  object 
 4   pickup_longitude   26782 non-null  float64
 5   pickup_latitude    26782 non-null  float64
 6   dropoff_longitude  26782 non-null  float64
 7   dropoff_latitude   26782 non-null  float64
 8   passenger_count    26781 non-null  float64
dtypes: float64(6), int64(1), object(2)
memory usage: 1.8+ MB
```

In [5]:

```
#find any null value present
df.isnull().sum()
```

Out[5]:

```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      1
dtype: int64
```

In [6]:

```
#drop null rows
df.dropna(axis=0,inplace=True)
df.isnull().sum()
```

Out[6]:

```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [7]:

```
#Calculatin the distance between the pickup and drop co-ordinates
#using the Haversine formual for accuracy.
def haversine (lon_1, lon_2, lat_1, lat_2):
    lon_1, lon_2, lat_1, lat_2 = map(np.radians, [lon_1, lon_2, lat_1, lat_2])  #Degrees to Radians

    diff_lon = lon_2 - lon_1
    diff_lat = lat_2 - lat_1

    km = 2 * 6371 * np.arcsin(np.sqrt(np.sin(diff_lat/2.0)**2 + np.cos(lat_1) * np.cos(lat_2) * np.sin(diff_lon/2.0)
**2))

    return km
```

In [8]:

```
#find distance travelled per ride
df['Distance']= haversine(df['pickup_longitude'],df['dropoff_longitude'], df['pickup_latitude'],df['dropoff_latitude
'])
```

In [9]:

```
#round it to 2 decimal points
df['Distance'] = df['Distance'].astype(float).round(2)
df.head()
```

Out[9]:

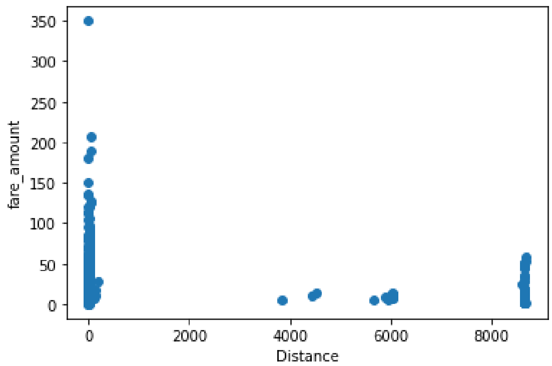
	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	Distance
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1.0	1.68
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1.0	2.46
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1.0	5.04
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3.0	1.66
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5.0	4.48

In [10]:

```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare amount")
```

Out[10]:

```
Text(0, 0.5, 'fare_amount')
```



In [11]:

```
#Outliers
#We can get rid of the trips with very large distances that are outliers as well as trips with 0 distance.
df.drop(df[df['Distance'] > 60].index, inplace = True)
df.drop(df[df['Distance'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] < 0].index, inplace = True)
df.shape
```

Out[11]:

```
(25942, 10)
```

In [12]:

```
# removing rows with non-plausible fare amounts and distance travelled
df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index, inplace = True )
df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index, inplace = True )
df.shape
```

Out[12]:

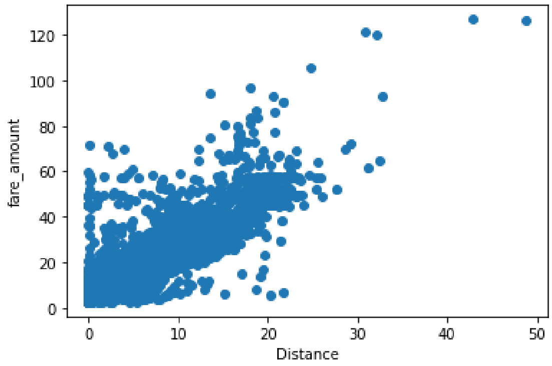
```
(25939, 10)
```

In [13]:

```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare amount")
```

Out[13]:

```
Text(0, 0.5, 'fare_amount')
```



In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25939 entries, 0 to 26780
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         25939 non-null  int64
 1   key                25939 non-null  object
 2   fare_amount        25939 non-null  float64
 3   pickup_datetime    25939 non-null  object
 4   pickup_longitude   25939 non-null  float64
 5   pickup_latitude    25939 non-null  float64
 6   dropoff_longitude  25939 non-null  float64
 7   dropoff_latitude   25939 non-null  float64
 8   passenger_count    25939 non-null  float64
 9   Distance           25939 non-null  float64
dtypes: float64(7), int64(1), object(2)
memory usage: 3.2+ MB
```

In [15]:

```
# Create New DataFrame of Specific column
df2 = pd.DataFrame().assign(fare=df['fare_amount'], Distance=df['Distance'])
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25939 entries, 0 to 26780
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   fare      25939 non-null  float64
 1   Distance  25939 non-null  float64
dtypes: float64(2)
memory usage: 1.6 MB
```

In [16]:

```
df2.shape
```

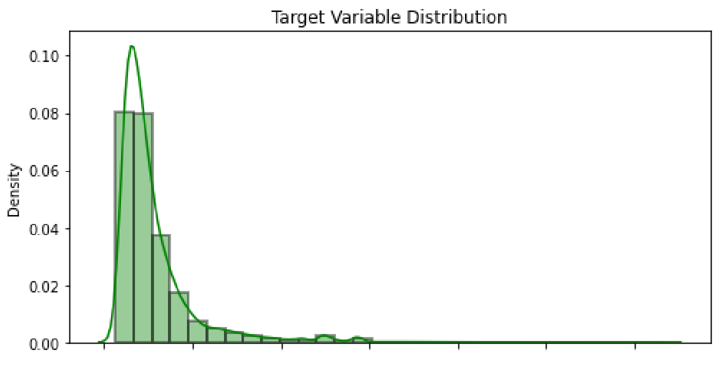
Out[16]:

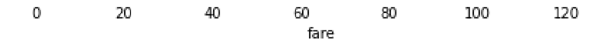
```
(25939, 2)
```

In [17]:

```
# plot target fare distribution
plt.figure(figsize=[8,4])
sns.distplot(df2['fare'], color='g',hist_kws=dict(edgecolor="black", linewidth=2), bins=30)
plt.title('Target Variable Distribution')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated func
tion and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level functio
n with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



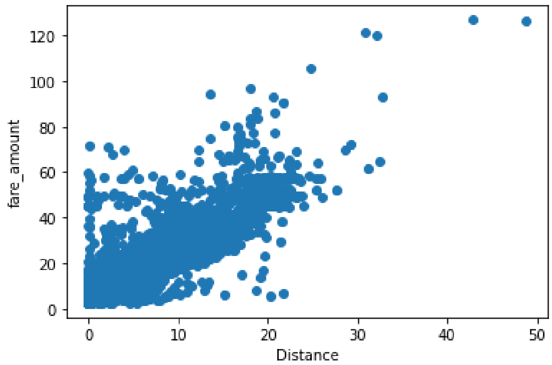


In [18]:

```
#plots
plt.scatter(df2['Distance'], df2['fare'])
plt.xlabel("Distance")
plt.ylabel("fare amount")
```

Out[18]:

```
Text(0, 0.5, 'fare_amount')
```



In [19]:

```
x=df2['fare']
y=df2['Distance']
```

In [20]:

```
#independant variable
X = df2['Distance'].values.reshape(-1, 1)
```

In [21]:

```
#dependant variable
Y= df2['fare'].values.reshape(-1, 1)
```

In [22]:

```
# scale by standardscalar
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(Y)
x std = std.fit transform(X)
```

In [23]:

```
#split in test-train
from sklearn.model_selection import train_test_split
X train, X test, y train, y test = train test split(x std, y std, test size=0.2, random state=0)
```

In [24]:

```
#simple linear regression
from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l reg.fit(X train, y train)
```

Out[24]:

```
LinearRegression()
```

In [25]:

```
#predict test values
y pred = l reg.predict(X test)
```

In [26]:

```
#find the error
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean squared error(y test, y pred)))
```

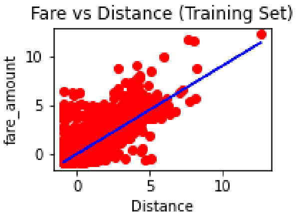
```
Mean Absolute Error: 0.2385901323763794
Mean Squared Error: 0.18371277215345383
Root Mean Squared Error: 0.4286172793454014
```

In [27]:

```
#final plot
plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
```

Out[27]:

```
Text(0.5, 0, 'Distance')
```

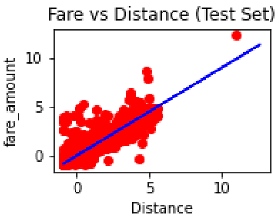


In [28]:

```
plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")
```

Out[28]:

```
Text(0.5, 1.0, 'Fare vs Distance (Test Set)')
```



In [55]:

```
import string
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re
```

In [56]:

```
df = pd.read_csv('spam_ham_dataset.csv')
```

In [57]:

```
df.head()
```

Out[57]:

	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0

In [58]:

```
df = df.drop(['Unnamed: 0'], axis=1)
df.head()
```

Out[58]:

	label	text	label_num
0	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	spam	Subject: photoshop , windows , office . cheap ...	1
4	ham	Subject: re : indian springs\r\nthis deal is t...	0

In [59]:

```
print('Total %s data email'% len(df))
```

```
Total 5171 data email
```

In [60]:

```
#total class memebers
df['label'].value_counts()
```

Out[60]:

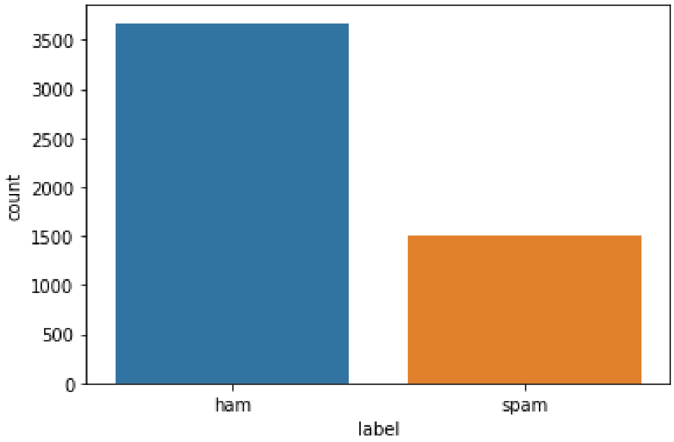
```
ham     3672
spam    1499
Name: label, dtype: int64
```

In [61]:

```
#show graph
df_label = sns.countplot(df['label'])
df_label.set_xticklabels(df['label'].unique())
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the
```

```
following variable as a keyword arg: x. From version 0.12, the only valid positional argument wil
l be `data`, and passing other arguments without an explicit keyword will result in an error or m
isinterpretation.
  FutureWarning
```



In [62]:

```
#data preprocessing
#data text cleaning
# punchuations
punct = []
for char in string.punctuation:
    punct.append(char)
```

In [63]:

```
def cleaning(txt):
    # case folding
    text = txt.lower()

    # remove multiple space, tabs, dan newlines
    text = re.sub('\s+',' ',text)
    # remove links
    text = text.replace("http://", " ").replace("https://", " ")
    # remove special characters
    text = text.encode('ascii', 'replace').decode('ascii')
    text = ' '.join(re.sub("([@#][A-Za-z0-9]+)|(\w+:\/\/\S+)"," ", text).split())
    # remove punctuation
    text = ''.join([word for word in text if word not in punct])
    #remove single character
    text = re.sub(r"\b[a-zA-Z]\b", "", text)
    #remove numbers
    text = re.sub(r"\d+", "", text)
    #remove multiple spaces (again)
    text = re.sub('\s+',' ',text)
    return text
```

In [64]:

```
# call function for cleaning
# apply fungsi cleaning ke setiap text
df['text_cleaned'] = df['text'].apply(lambda x: cleaning(x))
df = df[['text', 'text_cleaned', 'label']]
df.head()
```

Out[64]:

	text	text_cleaned	label
0	Subject: enron methanol ; meter # : 988291\r\n...	subject enron methanol meter this is follow up...	ham
1	Subject: hpl nom for january 9 , 2001\r\n(see...	subject hpl nom for january see attached file ...	ham
2	Subject: neon retreat\r\nho ho ho , we ' re ar...	subject neon retreat ho ho ho we re around to ...	ham
3	Subject: photoshop , windows , office . cheap ...	subject photoshop windows office cheap main tr...	spam
4	Subject: re : indian springs\r\nthis deal is t...	subject re indian springs this deal is to book...	ham

In [65]:

```
#compare
print(df['text'][0])
print(df['text_cleaned'][0])
```

```
Subject: enron methanol ; meter # : 988291
this is a follow up to the note i gave you on monday , 4 / 3 / 00 { preliminary
flow data provided by daren } .
please override pop ' s daily volume { presently zero } to reflect daily
activity you can obtain from gas control .
this change is needed asap for economics purposes .
subject enron methanol meter this is follow up to the note gave you on monday preliminary flow da
ta provided by daren please override pop daily volume presently zero to reflect daily activity yo
u can obtain from gas control this change is needed asap for economics purposes
```

In [66]:

```
# to remove stop words
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
from nltk.corpus import stopwords
stop = stopwords.words('english')
df['text_cleaned'] = df['text_cleaned'].apply(lambda x: ' '.join([word for word in x.split() if wo
rd not in stop]))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [67]:

```
#lemmatization
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
lemmatizer = WordNetLemmatizer()
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

def do_lemma(string):
    lemmatized = ' '.join([lemmatizer.lemmatize(word, get_wordnet_pos(word)) for word in nltk.word
_tokenize(string)])
    return lemmatized
```

In [68]:

```
import nltk
nltk.download('omw-1.4')
df['text_cleaned'] = df['text_cleaned'].apply(lambda x: do_lemma(x))
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [69]:

```
df = df.drop(['text'], axis=1)
df = df.rename(columns = {'text_cleaned' : 'text'})
df.columns
```

Out[69]:

```
Index(['text', 'label'], dtype='object')
```

In [70]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(df['text'])
y = df['label']
```

In [71]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [72]:

```
from time import time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# defining the classifier
clf = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
```

In [73]:

```
#predicting the time of train and testing
t0 = time()
clf.fit(X_train, y_train)
print("\nTraining time:", round(time()-t0, 3), "s\n")
```

```
Training time: 0.01 s
```

In [74]:

```
#predicting the time of  testing
t1 = time()
pred = clf.predict(X_test)
print("Predicting time:", round(time()-t1, 3), "s\n")
```

```
Predicting time: 0.305 s
```

In [75]:

```
#calculating and printing the accuracy of the algorithm
print("Accuracy of KNN Algorithm: ", accuracy_score(pred,y_test))
```

```
Accuracy of KNN Algorithm:  0.9623188405797102
```

In [1]:

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
df = pd.read_csv('Churn_Modelling.csv')
df.shape
```

Out[2]:

```
(10000, 14)
```

In [3]:

```
df.drop(['CustomerId','RowNumber','Surname'], axis = 'columns', inplace =True)
```

In [4]:

```
df.isna().sum()
```

Out[4]:

```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [5]:

```
df.dtypes
```

Out[5]:

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [6]:

```
df['Geography'].unique()
```

Out[6]:

```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [7]:

```
#one hot encoding
df = pd.get_dummies(data = df, columns=['Geography'])
df.dtypes
```

Out[7]:

```
CreditScore            int64
Gender                object
Age                    int64
Tenure                 int64
Balance              float64
NumOfProducts          int64
HasCrCard              int64
```

```
IsActiveMember          int64
EstimatedSalary       float64
Exited                  int64
Geography_France        uint8
Geography_Germany       uint8
Geography_Spain         uint8
dtype: object
```

In [8]:

```
df['Gender'].unique()
```

Out[8]:

```
array(['Female', 'Male'], dtype=object)
```

In [9]:

```
df['Gender'].replace(['Male', 'Female'],[1, 0], inplace= True)
```

In [10]:

```
df['Exited'].value_counts()
```

Out[10]:

```
0    7963
1    2037
Name: Exited, dtype: int64
```

In [11]:

```
#separate outcome or target col
X = df.drop(['Exited'], axis=1)
y = df['Exited']
```

In [12]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

In [13]:

```
from sklearn.preprocessing import StandardScaler

# feature scaling

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [14]:

```
import tensorflow as tf
from tensorflow import keras
```

In [15]:

```
model = keras.Sequential([
    keras.layers.Dense(12, input_shape=(12,),activation='relu'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

In [16]:

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [17]:

```
model.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
250/250 [==============================] - 2s 2ms/step - loss: 0.5976 - accuracy: 0.6628
Epoch 2/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4396 - accuracy: 0.8119
Epoch 3/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4111 - accuracy: 0.8267
Epoch 4/100
```

```
250/250 [==============================] - 1s 2ms/step - loss: 0.3898 - accuracy: 0.8382
Epoch 5/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3703 - accuracy: 0.8470
Epoch 6/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3564 - accuracy: 0.8537
Epoch 7/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3481 - accuracy: 0.8586
Epoch 8/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3434 - accuracy: 0.8587
Epoch 9/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3396 - accuracy: 0.8610
Epoch 10/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3372 - accuracy: 0.8625
Epoch 11/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3361 - accuracy: 0.8624
Epoch 12/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3341 - accuracy: 0.8637
Epoch 13/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3332 - accuracy: 0.8625
Epoch 14/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3322 - accuracy: 0.8641
Epoch 15/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3315 - accuracy: 0.8614
Epoch 16/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3305 - accuracy: 0.8631
Epoch 17/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3302 - accuracy: 0.8646
Epoch 18/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3295 - accuracy: 0.8634
Epoch 19/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3292 - accuracy: 0.8635
Epoch 20/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3282 - accuracy: 0.8648
Epoch 21/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3276 - accuracy: 0.8643
Epoch 22/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3279 - accuracy: 0.8634
Epoch 23/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3271 - accuracy: 0.8676
Epoch 24/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3275 - accuracy: 0.8649
Epoch 25/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3267 - accuracy: 0.8644
Epoch 26/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3261 - accuracy: 0.8656
Epoch 27/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3253 - accuracy: 0.8660
Epoch 28/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3257 - accuracy: 0.8673
Epoch 29/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3248 - accuracy: 0.8648
Epoch 30/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3253 - accuracy: 0.8658
Epoch 31/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3247 - accuracy: 0.8658
Epoch 32/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3242 - accuracy: 0.8668
Epoch 33/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3242 - accuracy: 0.8666
Epoch 34/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3240 - accuracy: 0.8652
Epoch 35/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3239 - accuracy: 0.8655
Epoch 36/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3239 - accuracy: 0.8679
Epoch 37/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3231 - accuracy: 0.8670
Epoch 38/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3230 - accuracy: 0.8671
Epoch 39/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3225 - accuracy: 0.8674
Epoch 40/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3223 - accuracy: 0.8668
Epoch 41/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3225 - accuracy: 0.8684
Epoch 42/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3217 - accuracy: 0.8656
Epoch 43/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3220 - accuracy: 0.8679
Epoch 44/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3221 - accuracy: 0.8680
Epoch 45/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3221 - accuracy: 0.8665
```

```
Epoch 46/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3219 - accuracy: 0.8679
Epoch 47/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3220 - accuracy: 0.8700
Epoch 48/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3211 - accuracy: 0.8692
Epoch 49/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3209 - accuracy: 0.8690
Epoch 50/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3211 - accuracy: 0.8695
Epoch 51/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3205 - accuracy: 0.8694
Epoch 52/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3212 - accuracy: 0.8658
Epoch 53/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3210 - accuracy: 0.8684
Epoch 54/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3205 - accuracy: 0.8695
Epoch 55/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3206 - accuracy: 0.8687
Epoch 56/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3206 - accuracy: 0.8674
Epoch 57/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3198 - accuracy: 0.8694
Epoch 58/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3199 - accuracy: 0.8686
Epoch 59/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3200 - accuracy: 0.8677
Epoch 60/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3193 - accuracy: 0.8695
Epoch 61/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3199 - accuracy: 0.8673
Epoch 62/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3193 - accuracy: 0.8696
Epoch 63/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3194 - accuracy: 0.8685
Epoch 64/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3194 - accuracy: 0.8683
Epoch 65/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3193 - accuracy: 0.8675
Epoch 66/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3191 - accuracy: 0.8696
Epoch 67/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3193 - accuracy: 0.8685
Epoch 68/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3188 - accuracy: 0.8676
Epoch 69/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3186 - accuracy: 0.8696
Epoch 70/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3190 - accuracy: 0.8681
Epoch 71/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3182 - accuracy: 0.8698
Epoch 72/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3187 - accuracy: 0.8690
Epoch 73/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3187 - accuracy: 0.8683
Epoch 74/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3184 - accuracy: 0.8698
Epoch 75/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3183 - accuracy: 0.8686
Epoch 76/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3184 - accuracy: 0.8687
Epoch 77/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3181 - accuracy: 0.8701
Epoch 78/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3185 - accuracy: 0.8695
Epoch 79/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3179 - accuracy: 0.8684
Epoch 80/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3183 - accuracy: 0.8711
Epoch 81/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3180 - accuracy: 0.8705
Epoch 82/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3179 - accuracy: 0.8702
Epoch 83/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3177 - accuracy: 0.8701
Epoch 84/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3175 - accuracy: 0.8692
Epoch 85/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3177 - accuracy: 0.8710
Epoch 86/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3172 - accuracy: 0.8696
Epoch 87/100
```

```
250/250 [==============================] - 0s 2ms/step - loss: 0.3173 - accuracy: 0.8699
Epoch 88/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3175 - accuracy: 0.8687
Epoch 89/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3169 - accuracy: 0.8701
Epoch 90/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3173 - accuracy: 0.8699
Epoch 91/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3166 - accuracy: 0.8706
Epoch 92/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3166 - accuracy: 0.8705
Epoch 93/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3169 - accuracy: 0.8696
Epoch 94/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3165 - accuracy: 0.8704
Epoch 95/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3159 - accuracy: 0.8684
Epoch 96/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3162 - accuracy: 0.8704
Epoch 97/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3154 - accuracy: 0.8706
Epoch 98/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3168 - accuracy: 0.8704
Epoch 99/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3161 - accuracy: 0.8691
Epoch 100/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3169 - accuracy: 0.8709
```

Out[17]:

```
<keras.callbacks.History at 0x7fab6ed52d50>
```

In [18]:

```
model.evaluate(X_test, y_test)
```

```
63/63 [==============================] - 1s 4ms/step - loss: 0.3311 - accuracy: 0.8625
```

Out[18]:

```
[0.3311230540275574, 0.862500011920929]
```

In [19]:

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

```
63/63 [==============================] - 0s 1ms/step
```

In [23]:

```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [24]:

```
from sklearn.metrics import confusion_matrix , classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      1595
           1       0.72      0.52      0.61       405

    accuracy                           0.86      2000
   macro avg       0.80      0.74      0.76      2000
weighted avg       0.85      0.86      0.85      2000
```

In [25]:

```
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_pred)
```

In [26]:

```
cm
```

Out[26]:

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[1514,   81],
```

```
       [ 194,   211]], dtype=int32)>
```

In [27]:

```
tf.math.confusion_matrix(labels=y_test,predictions=y_pred)
```

Out[27]:

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[1514,   81],
       [ 194,   211]], dtype=int32)>
```

In [1]:

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
df = pd.read_csv('diabetes.csv')
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [3]:

```
df.drop(['Pregnancies', 'BloodPressure', 'SkinThickness'], axis=1, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Glucose   768 non-null    int64  
 1   Insulin   768 non-null    int64  
 2   BMI       768 non-null    float64
 3   Pedigree  768 non-null    float64
 4   Age       768 non-null    int64  
 5   Outcome   768 non-null    int64  
dtypes: float64(2), int64(4)
memory usage: 36.1 KB
```

In [4]:

```
df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	max
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
Pedigree	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

In [5]:

```
#aiming to impute nan values for the columns in accordance
#with their distribution
df[['Glucose','Insulin','BMI']].replace(0,np.NaN)
```

Out[5]:

	Glucose	Insulin	BMI
0	148.0	NaN	33.6
1	85.0	NaN	26.6
2	183.0	NaN	23.3
3	89.0	94.0	28.1
4	137.0	168.0	43.1
...	...	...	...
763	101.0	180.0	32.9
764	122.0	NaN	36.8
765	121.0	112.0	26.2
766	126.0	NaN	30.1
767	93.0	NaN	30.4

768 rows × 3 columns

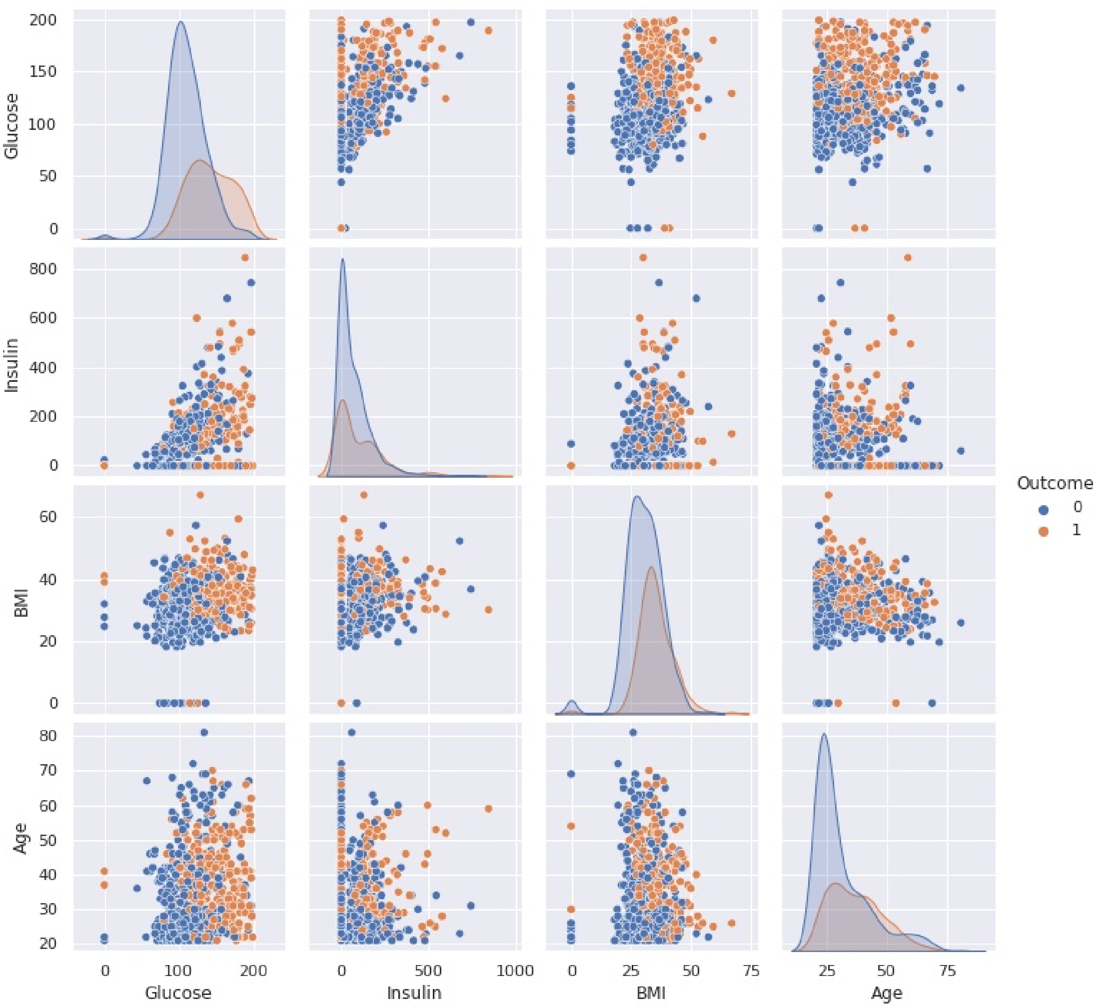
In [6]:

```
columns = ['Glucose','Insulin','BMI']
for col in columns:
    val = df[col].mean()
    df[col].replace(0, val)
```

In [7]:

```
#plot graph
graph = ['Glucose','Insulin','BMI','Age','Outcome']
sns.set()
print(sns.pairplot(df[graph],hue='Outcome', diag kind='kde'))
```

```
<seaborn.axisgrid.PairGrid object at 0x7ff895ce6390>
```



In [8]:

```
#separate outcome or target col
X = df.drop(['Outcome'], axis=1)
y = df['Outcome']
```

In [9]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy score
```

In [10]:

```
X train,X test,y train,y test = train test split(X,y,test size=0.2,random state=0)
```

In [11]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy score
```

In [12]:

```
# feature scaling

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X test = scaler.transform(X test)
```

In [13]:

```
classifier = KNeighborsClassifier(n_neighbors=11,p=2,metric='euclidean')

classifier.fit(X_train,y_train)
```

```
y pred = classifier.predict(X test)
```

In [16]:

```
# evaluating model

conf_matrix = confusion_matrix(y_test,y_pred)
print(conf matrix)
```

```
[[93 14]
 [18 29]]
```

In [17]:

```
print(f1 score(y test,y pred))
```

```
0.6444444444444444
```

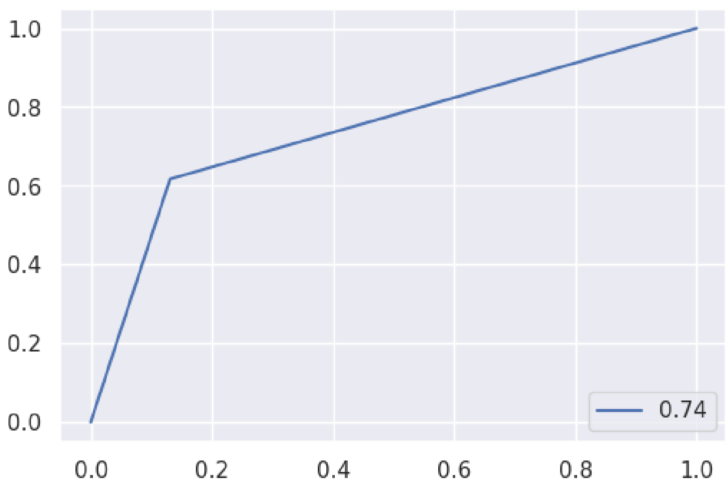
In [15]:

```
# accuracy
print(accuracy score(y test,y pred))
```

```
0.7922077922077922
```

In [18]:

```
# roc curve
from sklearn.metrics import roc_curve
plt.figure(dpi=100)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
from sklearn.metrics import roc_auc_score

temp=roc_auc_score(y_test,y_pred)

plt.plot(fpr,tpr,label = "%.2f" %temp)
plt.legend(loc = 'lower right')
plt.grid(True)
```



In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
df = pd.read_csv('Mall_Customers.csv')
df.shape
```

Out[2]:

```
(200, 5)
```

In [3]:

```
df.head()
```

Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [4]:

```
df["A"]= df[["Annual Income (k$)"]]
df["B"]=df[["Spending Score (1-100)"]]
```

In [5]:

```
X=df[["A","B"]]
X.head()
```

Out[5]:

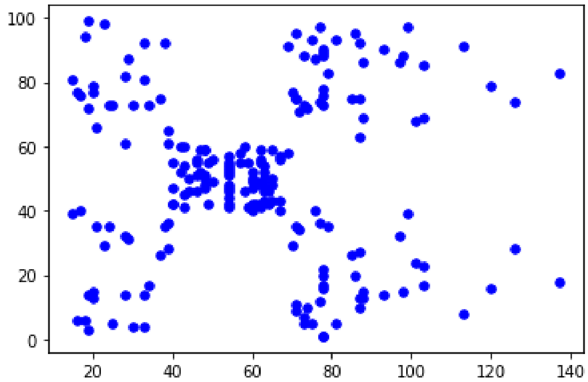
	A	B
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40

In [6]:

```
# Commented out IPython magic to ensure Python compatibility.
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# %matplotlib inline
```

In [7]:

```
plt.scatter(X["A"], X["B"], s = 30, c = 'b')
plt.show()
```



In [8]:

```
Kmean = KMeans(n_clusters=5)
Kmean.fit(X)
```

Out[8]:

```
KMeans(n_clusters=5)
```

In [9]:

```
centers=Kmean.cluster_centers_
print(Kmean.cluster_centers_)
```

```
[[88.2        17.11428571]
 [55.2962963  49.51851852]
 [26.30434783 20.91304348]
 [86.53846154 82.12820513]
 [25.72727273 79.36363636]]
```

In [10]:

```
clusters = Kmean.fit_predict(X)
df["label"] = clusters
df.head(100)
```

Out[10]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)	A	B	label
0	1	Male	19	15	39	15	39	3
1	2	Male	21	15	81	15	81	4
2	3	Female	20	16	6	16	6	3
3	4	Female	23	16	77	16	77	4
4	5	Female	31	17	40	17	40	3
...	...	...	...	...	...	...	...	...
95	96	Male	24	60	52	60	52	0
96	97	Female	47	60	47	60	47	0
97	98	Female	27	60	50	60	50	0
98	99	Male	48	61	42	61	42	0
99	100	Male	20	61	49	61	49	0

100 rows × 8 columns

In [11]:

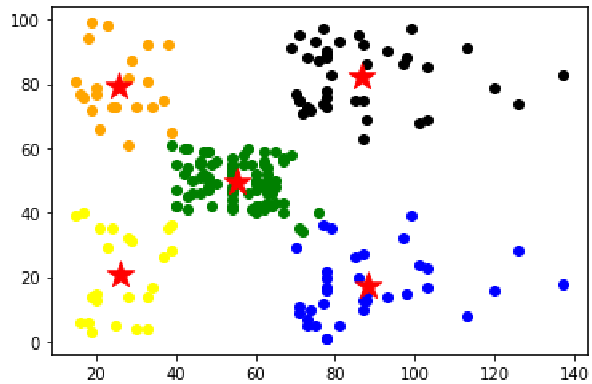
```
col=['green','blue','black','yellow','orange',]
```

In [12]:

```
for i in range(5):
    a=col[i]
    # print(a)
    plt.scatter(df.A[df.label==i], df.B[df.label == i], c=a, label='cluster 1')
plt.scatter(centers[:, 0], centers[:, 1], marker='*', s=300, c='r', label='centroid')
```

Out[12]:

```
<matplotlib.collections.PathCollection at 0x7f47c1c47910>
```

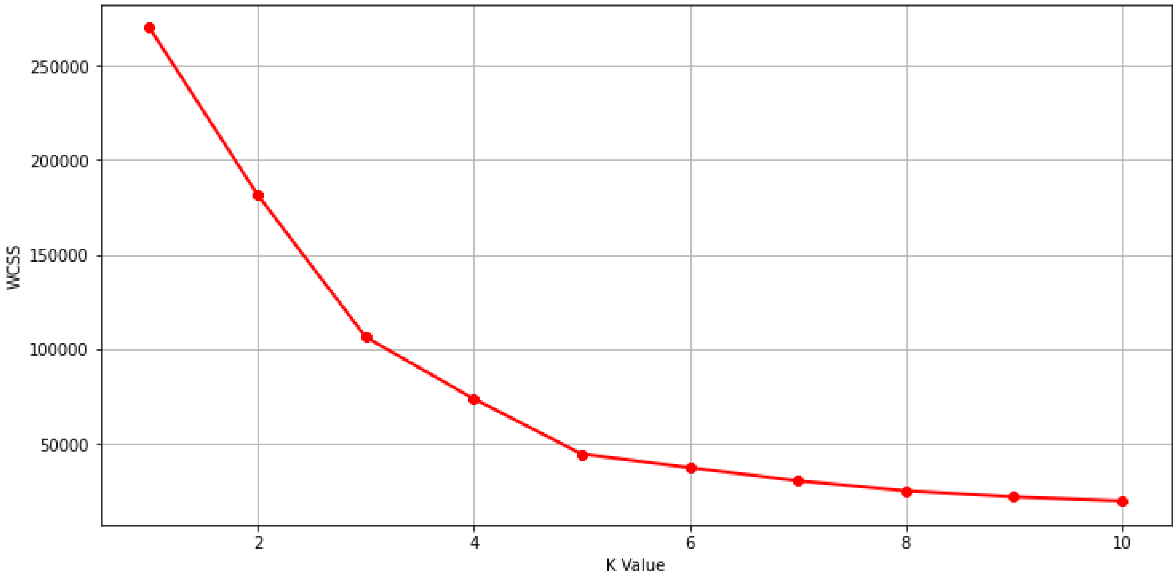


In [13]:

```
X1 = X.loc[:,["A","B"]].values
```

In [14]:

```
wcss=[]
for k in range(1,11):
    kmeans = KMeans(n_clusters = k, init = "k-means++")
    kmeans.fit(X1)
    wcss.append(kmeans.inertia_)
plt.figure(figsize =( 12,6))
plt.grid()
plt.plot(range(1,11),wcss,linewidth=2,color="red",marker="8")
plt.xlabel("K Value")
plt.ylabel("WCSS")
plt.show()
```



In [39]:

```python
# import

from math import sqrt

import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, plot_confus
ion_matrix
from scipy.spatial import distance
```

In [40]:

```python
# load dataset
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

Data Exploration

In [41]:

```python
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [42]:

```python
df_train.head()
```

Out[42]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

PassengerId	Survived	Pclass	Henry Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

In [43]:

```
df_train.describe()
```

Out[43]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

- **Fare ranges between 0 - 512.3**
- **Pclass could be modeled as a categorical feature**

In [44]:

```
print(f"Pclass n of unique values: {df_train['Pclass'].nunique()}")
print(f"Pclass unique values:      {df_train['Pclass'].unique()}")
```

```
Pclass n of unique values: 3
Pclass unique values:      [3 1 2]
```

In [45]:

```
df_train['Pclass'] = df_train['Pclass'].map({1:'Upper', 2:'Middle', 3:'Lower'})
df_test['Pclass'] = df_test['Pclass'].map({1:'Upper', 2:'Middle', 3:'Lower'})
```

- **Pclass has 3 unique values (1: Upper Class, 2: Middle Class, 3: Lower Class)**
- **Translate it into a categorical feature**

In [46]:

```
df_train.select_dtypes(include = 'object').nunique()
```

Out[46]:

```
Pclass        3
Name        891
Sex           2
Ticket      681
Cabin       147
Embarked      3
dtype: int64
```

In [47]:

```
df_train['Embarked'] = df_train['Embarked'].map({'C':'Cherbourg', 'Q':'Queenstown', 'S':'Southam
pton'})
df_test['Embarked'] = df_test['Embarked'].map({'C':'Cherbourg', 'Q':'Queenstown', 'S':'Southampt
on'})
```

- **Sex is defined in the data documentary as female, male**
- **Embarked can be mapped according to data documentation (C = Cherbourg, Q = Queenstown, S = Southampton) for better readablity**

In [48]:

```
df_train['Name'].head()
```

Out[48]:

```
0                                Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                 Heikkinen, Miss. Laina
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                               Allen, Mr. William Henry
Name: Name, dtype: object
```

In [49]:

```
df_train['Name'].duplicated().any()
```

Out[49]:

```
False
```

- **Name is mostly unstructured text**
- **There are no duplicates in the train set**
- **The title (Mr., Ms., etc) is contained in the name**

In [53]:

```
df_train.isnull().sum().sort_values(ascending = False)
```

Out[53]:

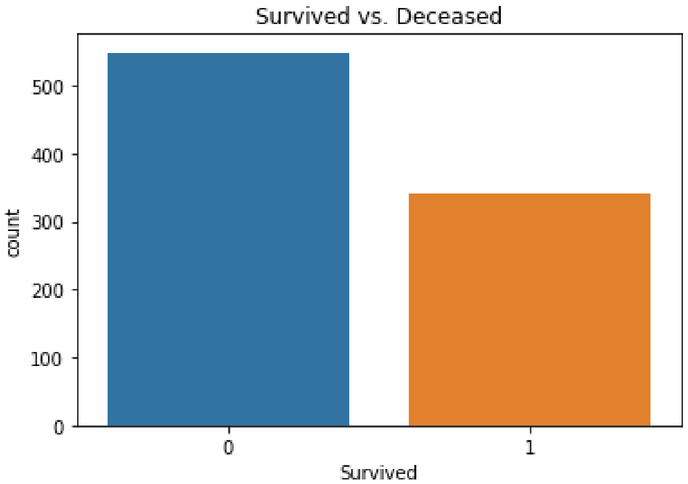
```
Cabin          687
Age            177
Embarked         2
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
SibSp            0
Parch            0
Ticket           0
Fare             0
dtype: int64
```

- **Cabin and Age have maximum NaN count**
- **Cabin should be removed in Pre Processing**
- **Age should be cleared in Pre Processing, as it important for model building**
- **Embarked should be cleared in Pre Processing**

Data Visualization

In [54]:

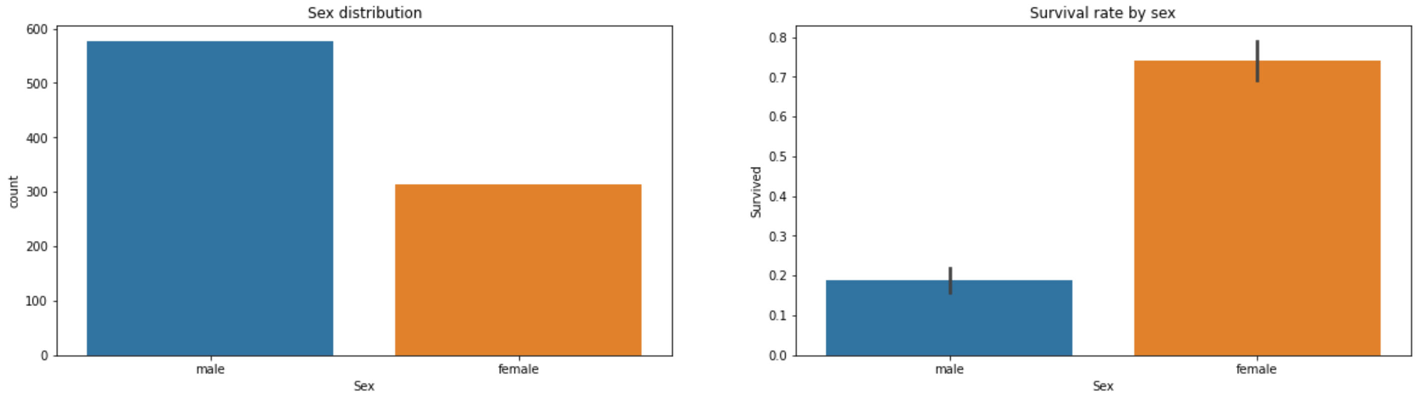
```
# survived
sns.countplot(x = df_train['Survived']).set_title('Survived vs. Deceased');
```



Over a third of the people survied

In [55]:

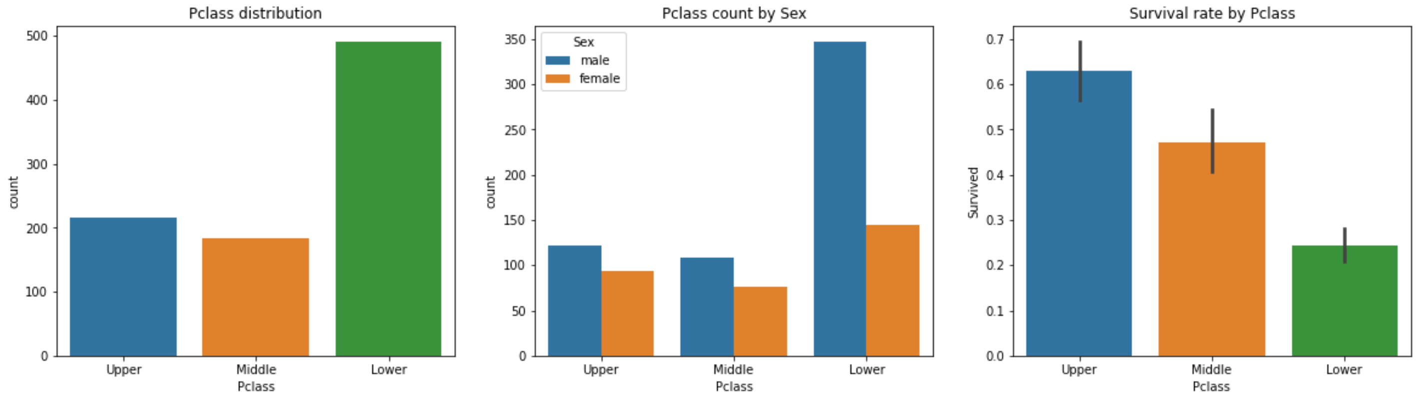
```
# Sex
fig, axes = plt.subplots(1, 2, figsize=(20, 5))

sns.countplot(ax = axes[0], x = df_train['Sex']).set_title('Sex distribution')
sns.barplot(ax = axes[1], data = df_train, x = "Sex", y = "Survived").set_title('Survival rate b
y sex');
```



- **Most travelers were male**
- **The survival rate for females is higher**

In [56]:

```
fig, axes = plt.subplots(1, 3, figsize=(20, 5))

pclass_order = ["Upper", "Middle", "Lower"]

sns.countplot(ax = axes[0], x = df_train['Pclass'], order = pclass_order).set_title('Pclass dist
ribution')
sns.countplot(ax = axes[1], data = df_train, x = 'Pclass', order = pclass_order, hue = 'Sex').se
t_title('Pclass count by Sex')
sns.barplot(ax = axes[2], data = df_train, x = "Pclass", y = "Survived", order = pclass_order).s
et_title('Survival rate by Pclass');
```

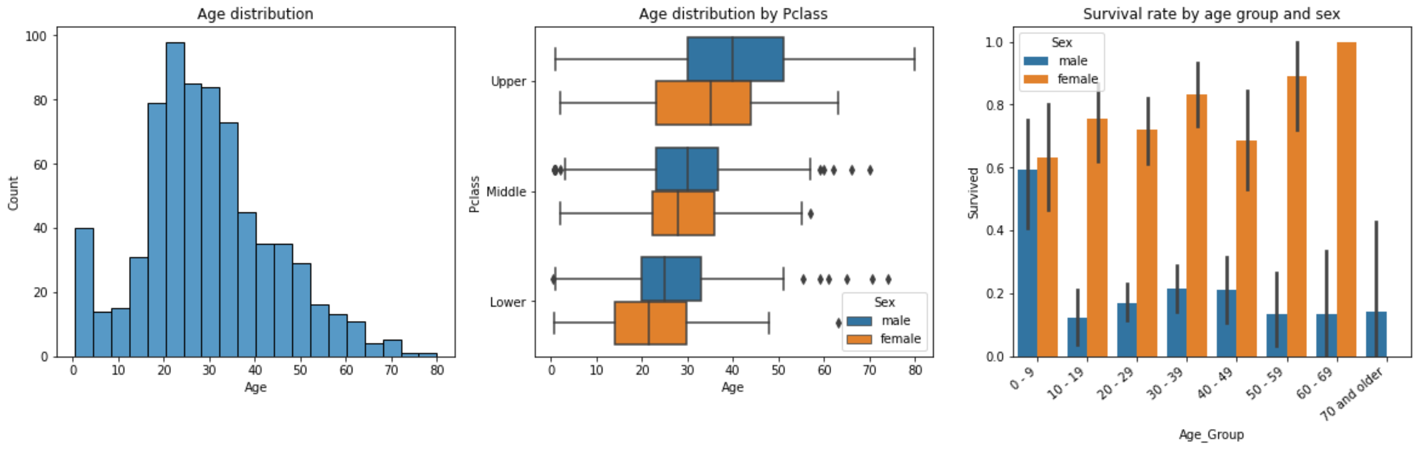


- **Most travelers were in the lower class**
- **Chance of survival grows with class**

In [57]:

```
# Age
fig, axes = plt.subplots(1, 3, figsize=(20, 5))

sns.histplot(ax = axes[0], data=df_train, x="Age").set_title('Age distribution')
sns.boxplot(ax = axes[1], data=df_train, x="Age", y="Pclass", hue='Sex', order=["Upper", "Middle
", "Lower"]).set_title('Age distribution by Pclass')

#Plot by age group

#Define age limit for groups and their labels
age_groups_thresholds = [0, 9, 19, 29, 39, 49, 59, 69, np.inf]
age_groups = ["0 - 9", "10 - 19", "20 - 29", "30 - 39", "40 - 49", "50 - 59", "60 - 69", "70 and
older"]

#Cut Age Series by thresholds and load into new feature
df_train["Age_Group"] = pd.cut(df_train['Age'], age_groups_thresholds, labels=age_groups)
```

```
sns.barplot(ax = axes[2], data=df_train, x="Age_Group", y="Survived", hue="Sex").set_title('Surv
ival rate by age group and sex')
axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation = 40, ha="right");
```



- **The Age is normally distributed with a positive skew**
- **For males, children from 0 - 9 had the highest chance of surival**
- **Women in all age groups had high survival rate**
- **Women of old age had a higher survival rate than girls**

Data Pre-processing

- **Fill the missing Age values with their mean**
- **Fill the missing Embarked values with backward fill (gets the last avaiable value)**

In [60]:

```
#Train data
age_mean = df_train['Age'].mean()
df_train['Age'].fillna(round(age_mean),inplace=True)
df_train['Embarked'].fillna(method = 'bfill', inplace = True)
```

In [61]:

```
#Test data
age_mean_test = df_test['Age'].mean()
df_test['Age'].fillna(round(age_mean_test),inplace=True)
df_test['Embarked'].fillna(method = 'bfill', inplace = True)
```

Data normalization

- **Numeric data - standardize it by normalizing the min max to be inbetween 0 and 1**
- **Categorical data - One Hot Encoding**

In [62]:

```
#Scale all numeric features to 0 - 1
def scale(num_features):
    min_max_scaler = MinMaxScaler()
    num_features = min_max_scaler.fit_transform(num_features)
    return pd.DataFrame(num_features)

#One hot encode categorical features
def one_hot_encode(cat_features):
    one_hot_enc = OneHotEncoder(handle_unknown = 'ignore', sparse = False)
    cat_features_one_hot = pd.DataFrame(one_hot_enc.fit_transform(cat_features))
    return pd.DataFrame(cat_features_one_hot)

#Normalize data according to data type
def normalize_data(df):
    cat_features = df.select_dtypes(include = 'object')
    num_features = df.select_dtypes(exclude = 'object')

    cat_features = one_hot_encode(cat_features)
    num_features = scale(num_features)
```

```
    df = pd.concat([num_features, cat_features], axis = 1)
    return df.to_numpy()
```

Splitting the data into training and testing dataset

In [63]:

```
#Training
X = df_train[['Age', 'Fare', 'SibSp', 'Parch', 'Sex', 'Pclass', 'Embarked']]
X = normalize_data(X)

y = df_train['Survived'].to_numpy()

X_train, X_dev, y_train, y_dev = train_test_split(X, y, train_size = 0.8, test_size = 0.2, rando
m_state = 0)
```

In [64]:

```
#Testing
X_test = df_test[['Age', 'Fare', 'SibSp', 'Parch', 'Sex', 'Pclass', 'Embarked']]
X_test = normalize_data(X_test)
```

Model building

In [65]:

```
class KNearestNeighbourEstimatorVect():
    def __init__(self, k_):
        self.k_ = k_

    def fit(self, X, y):
        self.X_ = X
        self.y_ = y

    def predict(self, X):
        distances = distance.cdist(X, self.X_, 'euclidean')
        i_k_smallest = np.argpartition(distances, self.k_)[:,:self.k_]
        values = self.y_[i_k_smallest]
        predictions = np.average(values, axis=1) > 0.5
        return 1*predictions
```

In [66]:

```
m = len(y_train)

best_accuracy =  float('-inf')
best_k = -1
for k in range(1, m):
    knn_estimator = KNearestNeighbourEstimatorVect(k_ = k)
    knn_estimator.fit(X_train, y_train)
    y_pred = knn_estimator.predict(X_dev)
    accuracy = accuracy_score(y_dev, y_pred)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
        best_y_pred = y_pred

print(f'Best k: {best_k}')
print(f'Score:  {round(best_accuracy, 2)}')
```

```
Best k: 76
Score:  0.83
```

In [67]:

```
confusion_m = confusion_matrix(y_dev, best_y_pred)
confusion_m
```

Out[67]:

```
array([[108,   2],
       [ 28,  41]])
```

In [69]:

```
labels = ['True Negative','False Positive','False Negative','True Positive']
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(confusion_m, annot = labels, fmt = '', cmap = 'Blues');
```