Machine Learning Lab-1

Create a generic segregation of any business scenario data into training and testing part with 70-30% proportions and analyze missing values. Further statistically summarize the data also.

importing the libraries

In [1]:

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

importing the data

In [2]:

```
df = pd.read_csv('/home/mithu/Downloads/archive/housing.csv')
df.head()
```

Out[2]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

In [3]:

```
df.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

Dropping Longitude and Latitude features

In [4]:

```
df.drop(['longitude','latitude'],axis=1,inplace=True)
```

In [5]:

```
df.shape
```

Out[5]:

```
(20640, 8)
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   housing_median_age  20640 non-null  float64
 1   total_rooms         20640 non-null  float64
 2   total_bedrooms      20433 non-null  float64
 3   population          20640 non-null  float64
 4   households          20640 non-null  float64
 5   median_income       20640 non-null  float64
 6   median_house_value  20640 non-null  float64
 7   ocean_proximity     20640 non-null  object 
dtypes: float64(7), object(1)
memory usage: 1.3+ MB
```

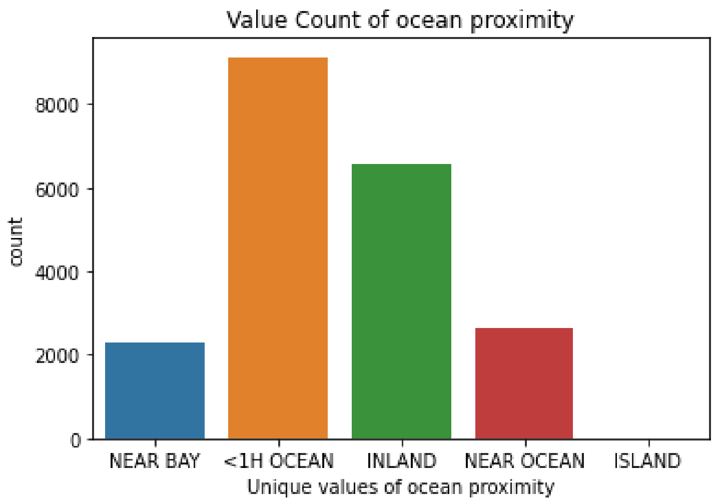
Exploratory Data Analysis

In [7]:

```
# visualization of ocean proximity feature since it had categorical values
sns.countplot(x='ocean_proximity', data=df)
plt.title("Value Count of ocean proximity")
plt.xlabel("Unique values of ocean proximity")
```

Out[7]:

```
Text(0.5, 0, 'Unique values of ocean proximity')
```

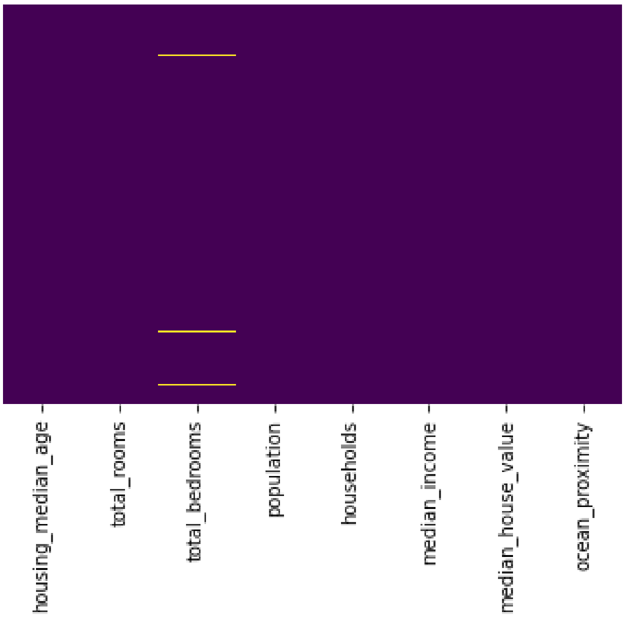


In [8]:

```
# checking for missing values
sns.heatmap(df.isna(), cmap='viridis', yticklabels=False, cbar=False)

# from this we see that there are missing values present in total_bedrooms
```

Out[8]:

```
<AxesSubplot:>
```



In [9]:

```
mean = df['total_bedrooms'].mean()
df['total_bedrooms'].fillna(mean, inplace= True)
```

In [10]:

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

Out[10]:

```
housing_median_age     0
total_rooms            0
total_bedrooms         0
population             0
households             0
median_income          0
median_house_value     0
ocean_proximity        0
dtype: int64
```

Converting categorical values to numerical

In [11]:

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder() # intiating the instance of LabelEncoder()

encoded = encoder.fit_transform(df['ocean_proximity'])
df['Ocean_proximity'] = encoded
```

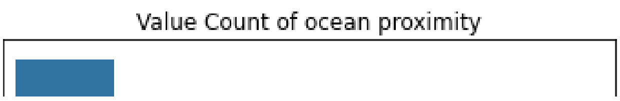
In [13]:

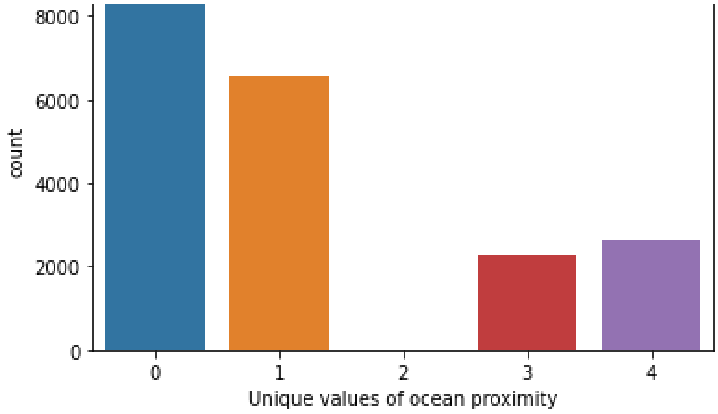
```
# now we can drop the categorical column as there is a numerical column of it
df.drop('ocean_proximity', axis=1, inplace=True)
```

In [18]:

```
sns.countplot(x='Ocean_proximity', data=df)
plt.title("Value Count of ocean proximity")
plt.xlabel("Unique values of ocean proximity")

print(encoder.classes_)  # this is the order of categorical values created
```

```
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```





In [19]:

```
df.head()
```

Out[19]:

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	Ocean_p
0	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	
2	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	
3	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	
4	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	

Splitting the data into features and target variable

In [20]:

```
X = df.drop('median_house_value', axis=1) # features
y = df['median_house_value']              # target variable
```

In [21]:

```
X[:5]
```

Out[21]:

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	Ocean_proximity
0	41.0	880.0	129.0	322.0	126.0	8.3252	3
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	3
2	52.0	1467.0	190.0	496.0	177.0	7.2574	3
3	52.0	1274.0	235.0	558.0	219.0	5.6431	3
4	52.0	1627.0	280.0	565.0	259.0	3.8462	3

In [22]:

```
y[:5]
```

Out[22]:

```
0    452600.0
1    358500.0
2    352100.0
3    341300.0
4    342200.0
Name: median_house_value, dtype: float64
```

Splitting the data into training and testing

In [23]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

Summarizing the train and test data

In [30]:

```
print("Shape of training data of features:          ", X_train.shape)
print("Shape of training data of target variable: ", y_train.shape)
print("Shape of testing data of features:           ", X_test.shape)
print("Shape of testing data of target variable:  ", y_test.shape)
```

```
Shape of training data of features:          (14448, 7)
Shape of training data of target variable: (14448,)
Shape of testing data of features:           (6192, 7)
Shape of testing data of target variable:    (6192,)
```

In [31]:

```
# summary of training data of features
X_train.describe()
```

Out[31]:

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	Ocean_proximity
count	14448.000000	14448.000000	14448.000000	14448.000000	14448.000000	14448.000000	14448.000000
mean	28.575374	2644.939230	539.828281	1427.927326	501.070598	3.876892	1.153931
std	12.613634	2163.054433	419.786747	1140.225190	382.221220	1.904908	1.410235
min	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	0.000000
25%	18.000000	1456.750000	296.000000	791.000000	280.000000	2.567225	0.000000
50%	29.000000	2131.000000	437.000000	1168.000000	411.000000	3.539100	1.000000
75%	37.000000	3169.250000	648.000000	1727.000000	607.000000	4.758075	1.000000
max	52.000000	32627.000000	6445.000000	35682.000000	6082.000000	15.000100	4.000000

In [33]:

```
# summary of training data of target variable
y_train.describe()
```

Out[33]:

```
count     14448.000000
mean     206923.960894
std      115749.242298
min       14999.000000
25%      119300.000000
50%      179300.000000
75%      264600.000000
max      500001.000000
Name: median_house_value, dtype: float64
```

In [34]:

```
# summary of testing data of features
X_test.describe()
```

Out[34]:

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	Ocean_proximity

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	Ocean_proximity
count	6192.000000	6192.000000	6192.000000	6192.000000	6192.000000	6192.000000	6192.000000
mean	28.789083	2614.352067	533.302520	1419.758721	495.967539	3.856156	1.193637
std	12.519541	2224.351073	418.048569	1114.208253	382.589931	1.887973	1.444434
min	1.000000	6.000000	2.000000	8.000000	2.000000	0.499900	0.000000
25%	18.000000	1431.000000	299.000000	778.000000	278.000000	2.552000	0.000000
50%	29.000000	2115.500000	440.000000	1161.000000	406.000000	3.525000	1.000000
75%	37.000000	3094.250000	629.000000	1717.000000	597.250000	4.722200	1.000000
max	52.000000	39320.000000	6210.000000	16305.000000	5358.000000	15.000100	4.000000

In [35]:

```
# summary of testing data of target variables
y_test.describe()
```

Out[35]:

```
count      6192.000000
mean     206696.814276
std      114575.395072
min       14999.000000
25%      120275.000000
50%      181000.000000
75%      265050.000000
max      500001.000000
Name: median_house_value, dtype: float64
```

Machine Learning Lab-2

Explore and implement Linear regression algorithm in a given business scenario and comment on its efficiency and performance.

In [1]:

```
import pandas as pd
data = pd.read_csv("insurance.csv")
```

In [2]:

```
data.head()
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

In [3]:

```
data.columns
data.shape
```

Out[3]:

```
(1338, 7)
```

In [4]:

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype

 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
None
```

In [5]:

```
print(data.nunique())
```

```
age          47
sex           2
bmi         548
```

```
children       6
smoker         2
region         4
charges     1337
dtype: int64
```

In [9]:

```
data1 = {"sex":      {"male": 11, "female": 12},
         "smoker": {"yes": 13, "no": 14},
         "region": {"southwest": 15, "southeast": 16, "northwest": 17, "northeast": 18,}
}
```

In [33]:

```
data1
```

Out[33]:

```
{'sex': {'male': 11, 'female': 12},
 'smoker': {'yes': 13, 'no': 14},
 'region': {'southwest': 15,
  'southeast': 16,
  'northwest': 17,
  'northeast': 18}}
```

In [10]:

```
data2 = data.replace(data1)
```

In [11]:

```
print(data2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   int64
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   int64
 5   region    1338 non-null   int64
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(5)
memory usage: 73.3 KB
None
```

In [14]:

```
from sklearn.model_selection import train_test_split
training, testing =train_test_split(data2, test_size= 0.30, random_state=24)
```

In [15]:

```
training.shape
```

Out[15]:

```
(936, 7)
```

In [16]:

```
testing.shape
```

Out[16]:

```
(402, 7)
```

Simple linear regression

In [17]:

```
X = training['age']
```

In [20]:

```
X.shape
```

Out[20]:

```
(936,)
```

In [28]:

```
import numpy as np
x= np.array(X)
```

In [29]:

```
x = x.reshape(936,1)
```

In [30]:

```
x.shape
```

Out[30]:

```
(936, 1)
```

In [37]:

```
Y = training['charges']
```

In [40]:

```
Y.shape
```

Out[40]:

```
(936,)
```

In [42]:

```
import numpy as np
Y= np.array(Y)
```

In [43]:

```
y = Y.reshape(936,1)
```

In [45]:

```
from sklearn.linear_model import LinearRegression
LR= LinearRegression()
model=LR.fit(x, y)
```

In [59]:

```
print(model)
```

```
LinearRegression()
```

In [65]:

```
print(model.coef_[0][0])  ## Printing the coefficients
print(model.intercept_[0])  ### printing the Intercept term

print("The linear model is: Y = {:.5} + {:.5}X".format(model.intercept_[0], model.coef_[
```

```
0][0])) # Pritning the Simple model
```

```
253.8323324073888
3321.049088797152
The linear model is: Y = 3321.0 + 253.83X
```

Prediction

In [47]:

```
X_test=testing['age']
```

In [48]:

```
X_test.shape
```

Out[48]:

```
(402,)
```

In [51]:

```
X_test= np.array(X_test)
```

In [52]:

```
X_test = X_test.reshape(402,1)
```

In [53]:

```
X_test.shape
```

Out[53]:

```
(402, 1)
```

In [54]:

```
X_tar=testing['charges']
```

In [55]:

```
X_tar.shape
```

Out[55]:

```
(402,)
```

In [56]:

```
X_tar= np.array(X_tar)
X_tar = X_tar.reshape(402,1)
```

In [57]:

```
X_tar.shape
```

Out[57]:

```
(402, 1)
```

Machine Learning Lab-3

Explore and implement logistic regression algorithm in a given business scenario and comment on its efficiency and performance.

In [1]:

```
import pandas as pd
```

In [4]:

```
df = pd.read_csv("C:\\Users\\mithun\\Downloads\\archive\\data.csv")
df[:5]
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	con
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 33 columns

In [5]:

```
df.drop('id',axis=1,inplace = True)
```

In [6]:

```
df.drop('Unnamed: 32', axis=1, inplace = True)
```

In [7]:

```
df.head()
```

Out[7]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.300
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.086
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.197
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.241
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198

5 rows × 31 columns

In [9]:

```
df.columns
```

Out[9]:

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [13]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('diagnosis', axis=1))
scaled_features = scaler.transform(df.drop('diagnosis', axis=1))

df_new = pd.DataFrame(scaled_features, columns=df.columns[1:])

df_new[:5]
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.53
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.54
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.03
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.45
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.42

5 rows × 30 columns

In [14]:

```
X = df_new
y = df['diagnosis']
```

In [15]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state = 4
2)
```

In [16]:

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

Out[16]:

```
LogisticRegression()
```

In [17]:

```
# predicitions
pred = model.predict(X_test)
```

In [18]:

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

```
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[106   2]
 [  1  62]]
              precision    recall  f1-score   support

           B       0.99      0.98      0.99       108
           M       0.97      0.98      0.98        63

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

In [19]:

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

Out[19]:

```
DecisionTreeClassifier()
```

In [20]:

```
pred = dt.predict(X_test)
```

In [21]:

```
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[100   8]
 [  3  60]]
              precision    recall  f1-score   support

           B       0.97      0.93      0.95       108
           M       0.88      0.95      0.92        63

    accuracy                           0.94       171
   macro avg       0.93      0.94      0.93       171
weighted avg       0.94      0.94      0.94       171
```

Machine Learning Lab-4

Explore and implement Linear Regression Using Gradient Descent in a given business scenario and comment on its efficiency and performance.

In [1]:

```
# Making the imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)
from sklearn.linear_model import LinearRegression # To work on Linear Regression
from sklearn.metrics import r2_score # To Calculate Performance matrix
import statsmodels.api as sm # To calculatestats modles
```

Importing Dataset

In [2]:

```
data = pd.read_csv('data.csv')
```

In [3]:

```
data.head()
```

Out[3]:

	mobile	sales
0	32.502345	31.707006
1	53.426804	68.777596
2	61.530358	62.562382
3	47.475640	71.546632
4	59.813208	87.230925

In [4]:

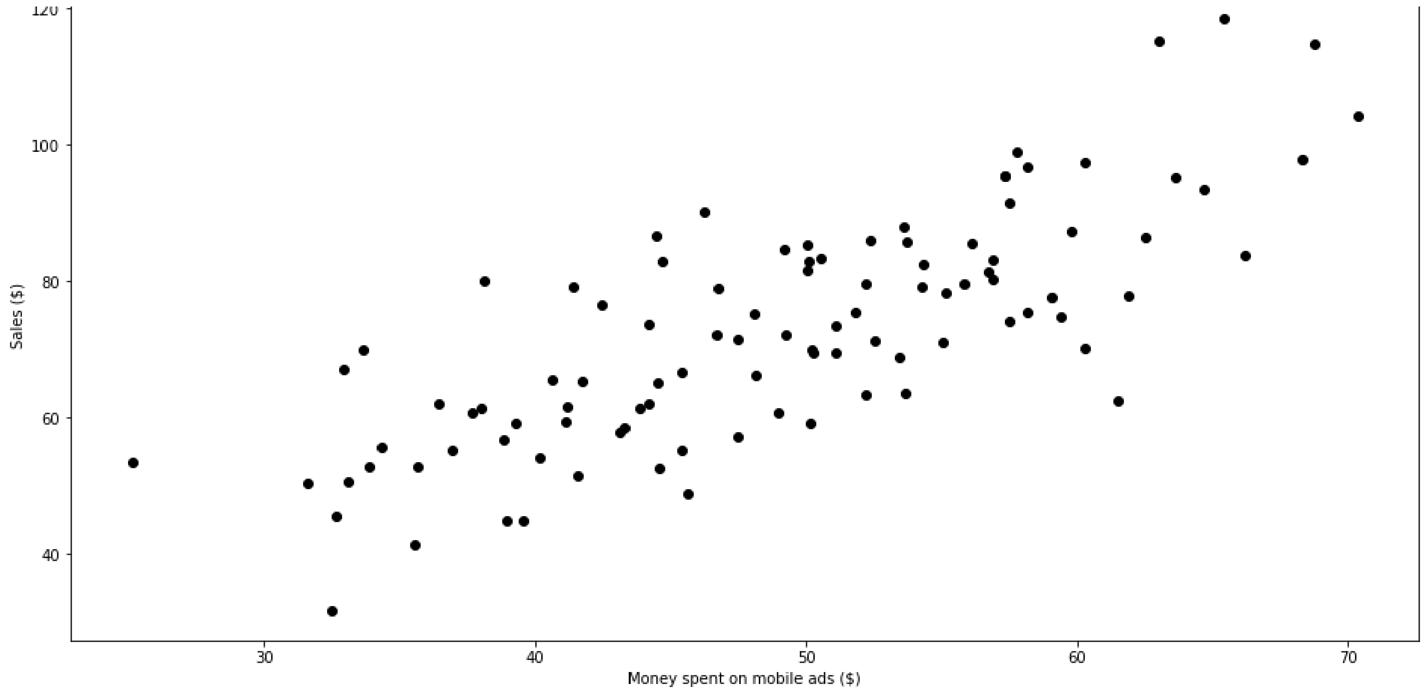
```
# Looking the shape of the data
data.shape
```

Out[4]:

```
(100, 2)
```

In [5]:

```
plt.figure(figsize=(16, 8)) ## Plotting TV ad vs Sales
plt.scatter(
    data['mobile'],
    data['sales'],
    c='black'
)
plt.xlabel("Money spent on mobile ads ($)")
plt.ylabel("Sales ($)")
plt.show()
```



In [6]:

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
training, testing =train_test_split(data, test_size= 0.30, random_state=24)
```

As you can see, there is a clear positive relationship between the amount spent on mobile ads and sales

In [7]:

```
training
```

Out[7]:

	mobile	sales
49	64.707139	93.576119
9	52.550014	71.300880
69	35.678094	52.721735
23	41.575643	51.391744
74	70.346076	104.257102
...	...	...
17	60.297327	97.379897
87	50.282836	69.510503
64	33.644706	69.899682
3	47.475640	71.546632
34	57.504448	74.084130

70 rows × 2 columns

Implementing Simple linear regression using Gradient Descent

In [8]:

```
# Building the model
m = 0
c = 0
```

```
L = 0.01  # The learning Rate
epochs = 5  # The number of iterations to perform gradient descent

n = float(len(data['mobile'])) # Number of elements in X

# Performing Gradient Descent
for i in range(epochs):
    Y_pred = m*(data['mobile']) + c  # The current predicted value of Y
    D_m = (-2/n) * sum(data['mobile'] * (data['sales'] - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(data['sales'] - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c

print (m, c)
```

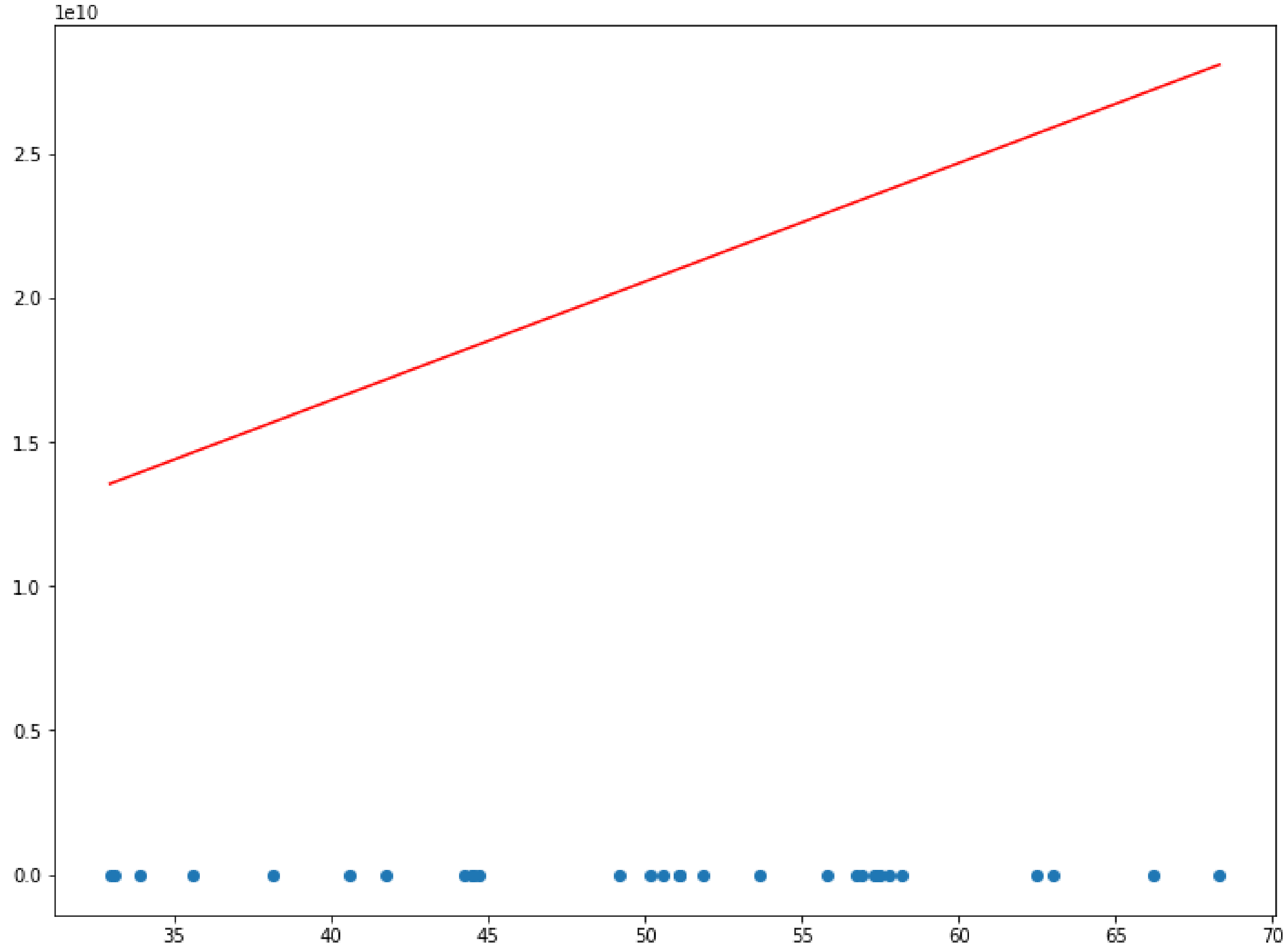
```
410917957.7002709 8076467.286673318
```

Prediction

Let's visualize how the line(prediction line) fits the data

In [9]:

```
# Making predictions
Y_pred = m*(testing['mobile']) + c

plt.scatter(testing['mobile'],testing['sales'])
plt.plot([min(testing['mobile']), max(testing['mobile'])], [min(Y_pred), max(Y_pred)], co
lor='red') # predicted
plt.show()
```



From the above graph using Gradient DescentPrediction line is the bewst fit line for L=0.0001 and epochs=1000

Assessing the efficiency and Performance of the model

To see if the model is any good, we need to look at the R² value and the p-value from each coefficient.

In [11]:

```
X = testing['mobile'] ## Assign TV ad value to X
y = testing['sales'] ## assign sales values to y

X2 = sm.add_constant(X)# Assign stat model  constant to X2
est = sm.OLS(y, X2) # Build Ordinary least square
est2 = est.fit() #Fitting OLS Regression
print(est2.summary()) # Printing OLS Results
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  sales   R-squared:                       0.544
Model:                            OLS   Adj. R-squared:                  0.528
Method:                 Least Squares   F-statistic:                     33.45
Date:                Mon, 23 Aug 2021   Prob (F-statistic):           3.28e-06
Time:                        10:27:46   Log-Likelihood:                -113.61
No. Observations:                  30   AIC:                             231.2
Df Residuals:                      28   BIC:                             234.0
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         16.8433     10.655      1.581      0.125      -4.983      38.670
mobile         1.2030      0.208      5.784      0.000       0.777       1.629
==============================================================================
Omnibus:                        0.950   Durbin-Watson:                   2.478
Prob(Omnibus):                  0.622   Jarque-Bera (JB):                0.863
Skew:                           0.177   Prob(JB):                        0.650
Kurtosis:                       2.249   Cond. No.                         271.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
```

Looking at both coefficients, we have a p-value that is very low (although it is probably not exactly 0). This means that there is a correlation between these coefficients and the target (Sales)

Then, looking at the R² value, we have 0.599. Therefore, about 60% of the variability of sales is explained by the amount spent on by advertising mobile. This is okay, but definitely not the best we can to accurately predict the sales.

Machine Learning Lab-5

Explore and implement Logistic Regression by Stochastic Gradient Descent in a given business scenario and comment on its efficiency and performance.

In [1]:

```
import pandas as pd
```

In [2]:

```
df = pd.read_csv("C:\\Users\\mithun\\Downloads\\archive\\data.csv")
df[:5]
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	con
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 33 columns

In [3]:

```
df.drop('id',axis=1,inplace = True)
```

In [4]:

```
df.drop('Unnamed: 32', axis=1, inplace = True)
```

In [5]:

```
df.head()
```

Out[5]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.300
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.086
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.197
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.241
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198

5 rows × 31 columns

In [6]:

```
df.columns
```

Out[6]:

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [7]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('diagnosis', axis=1))
scaled_features = scaler.transform(df.drop('diagnosis', axis=1))

df_new = pd.DataFrame(scaled_features, columns=df.columns[1:])

df_new[:5]
```

Out[7]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.53
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.54
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.03
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.45
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.42

5 rows × 30 columns

In [8]:

```
X = df_new
y = df['diagnosis']
```

In [9]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state = 4
2)
```

Implementing Stochastic Gradient Descent Algorithm

In [10]:

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

Out[10]:

```
SGDClassifier()
```

In [11]:

```
# predicitions
pred = model.predict(X_test)
```

In [12]:

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

```
[[105   3]
 [  3  60]]
              precision    recall  f1-score   support

           B       0.97      0.97      0.97       108
           M       0.95      0.95      0.95        63

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

Machine Learning Lab-6

Implement Decision Tree algorithm in a given business environment and comment on its efficiency and performance.

In [1]:

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

In [3]:

```
df= pd.read_csv("data.csv")
```

In [4]:

```
df.head()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	con
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

Attribute Information:

1) ID number

2) Diagnosis (M = malignant, B = benign)

3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)

b) texture (standard deviation of gray-scale values)

c) perimeter

d) area

e) smoothness (local variation in radius lengths)

f) compactness (perimeter^2 / area - 1.0)

g) concavity (severity of concave portions of the contour)

h) concave points (number of concave portions of the contour)

i) symmetry

j) fractal dimension ("coastline approximation" - 1)

In [5]:

```
df.columns
```

Out[5]:

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [6]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [7]:

```
scaler.fit(df[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']])
```

Out[7]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [8]:

```
scaler.transform(df[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']])
```

Out[8]:

```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

In [10]:

```
scaled_features = pd.DataFrame(df[['radius_mean', 'texture_mean', 'perimeter_mean',
        'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
        'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
        'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
        'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
        'fractal_dimension_se', 'radius_worst', 'texture_worst',
        'perimeter_worst', 'area_worst', 'smoothness_worst',
        'compactness_worst', 'concavity_worst', 'concave points_worst',
        'symmetry_worst', 'fractal_dimension_worst']], columns=['radius_mean', 'texture_m
ean', 'perimeter_mean',
        'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
        'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
        'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
        'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
        'fractal_dimension_se', 'radius_worst', 'texture_worst',
        'perimeter_worst', 'area_worst', 'smoothness_worst',
        'compactness_worst', 'concavity_worst', 'concave points_worst',
        'symmetry_worst', 'fractal_dimension_worst'] )
```

In [11]:

```
scaled_features[:5]
```

Out[11]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1

In [19]:

```
df.shape
```

Out[19]:

```
(569, 33)
```

In [13]:

```
X = scaled_features
X[:5]
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1

In [15]:

```
y=df['diagnosis']
y[:5]
```

Out[15]:

```
0    M
1    M
2    M
3    M
4    M
Name: diagnosis, dtype: object
```

In [16]:

```
y.value_counts()
# B is non cancerous
# M is cancerous cells
```

Out[16]:

```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [17]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size= 0.3, random_state = 4
2)
```

In [25]:

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
```

In [26]:

```
dtree.fit(X_train, y_train)
```

Out[26]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [27]:

```
pred_tree = dtree.predict(X_test)
pred_tree[:5]
```

Out[27]:

```
array(['B', 'M', 'M', 'B', 'B'], dtype=object)
```

In [28]:

```
print(confusion_matrix(y_test, pred_tree))
print(classification_report(y_test, pred_tree))
```

```
[[100   8]
 [  4  59]]
              precision    recall  f1-score   support

           B       0.96      0.93      0.94       108
           M       0.88      0.94      0.91        63

    accuracy                           0.93       171
   macro avg       0.92      0.93      0.93       171
weighted avg       0.93      0.93      0.93       171
```

Machine Learning Lab-7

Implement Naïve Bayes algorithm in a given business environment and comment on its efficiency and performance.

In [1]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('spam.csv')
df.head()
```

Out[1]:

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [2]:

```
df.groupby('Category').describe()
```

Out[2]:

	Message			
	count	unique	top	freq
Category				
ham	4825	4516	Sorry, I'll call later	30
spam	747	641	Please call our customer service representativ...	4

In [12]:

```
df['spam'] = df['Category'].apply(lambda x:1 if x=='spam' else 0)
df.head()
```

Out[12]:

	Category	Message	spam
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

In [20]:

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(df.Message,df.spam,test_size=0.2, ra
ndom_state= 100)
```

In [15]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

clf = Pipeline(
    [('vectorizer', CountVectorizer()),
    ("nb", MultinomialNB())]
    )
```

In [16]:

```
clf.fit(X_train,y_train)
```

Out[16]:

```
Pipeline(steps=[('vectorizer', CountVectorizer()), ('nb', MultinomialNB())])
```

In [18]:

```
clf.predict(X_test)
```

Out[18]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [21]:

```
clf.score(X_test,y_test)
```

Out[21]:

```
0.9928251121076234
```

In [29]:

```
items = ['Confirm Funds sent to you ','Please confirm receipt of $500.00']

clf.predict(items)
```

Out[29]:

```
array([0, 1], dtype=int64)
```

Machine Learning Lab-8

Implement K Nearest Neighbors algorithm in a given business environment and comment on its efficiency and performance.

In [1]:

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

In [3]:

```
df= sns.load_dataset('iris')
df.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

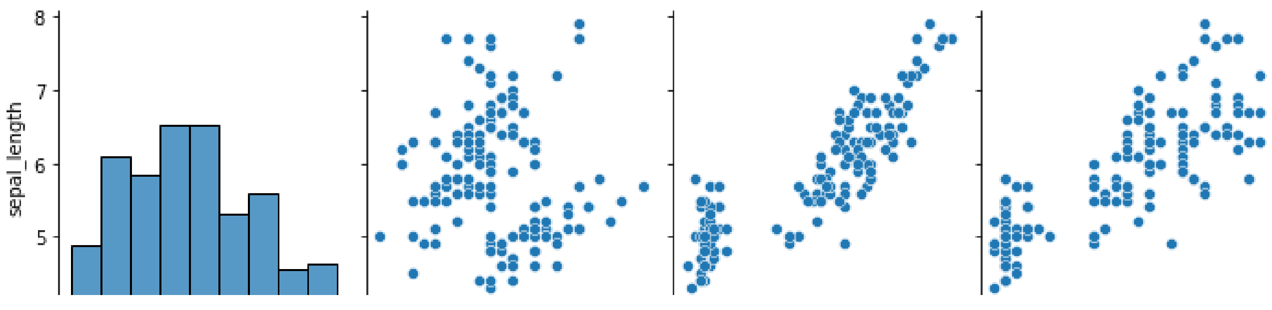
In [4]:

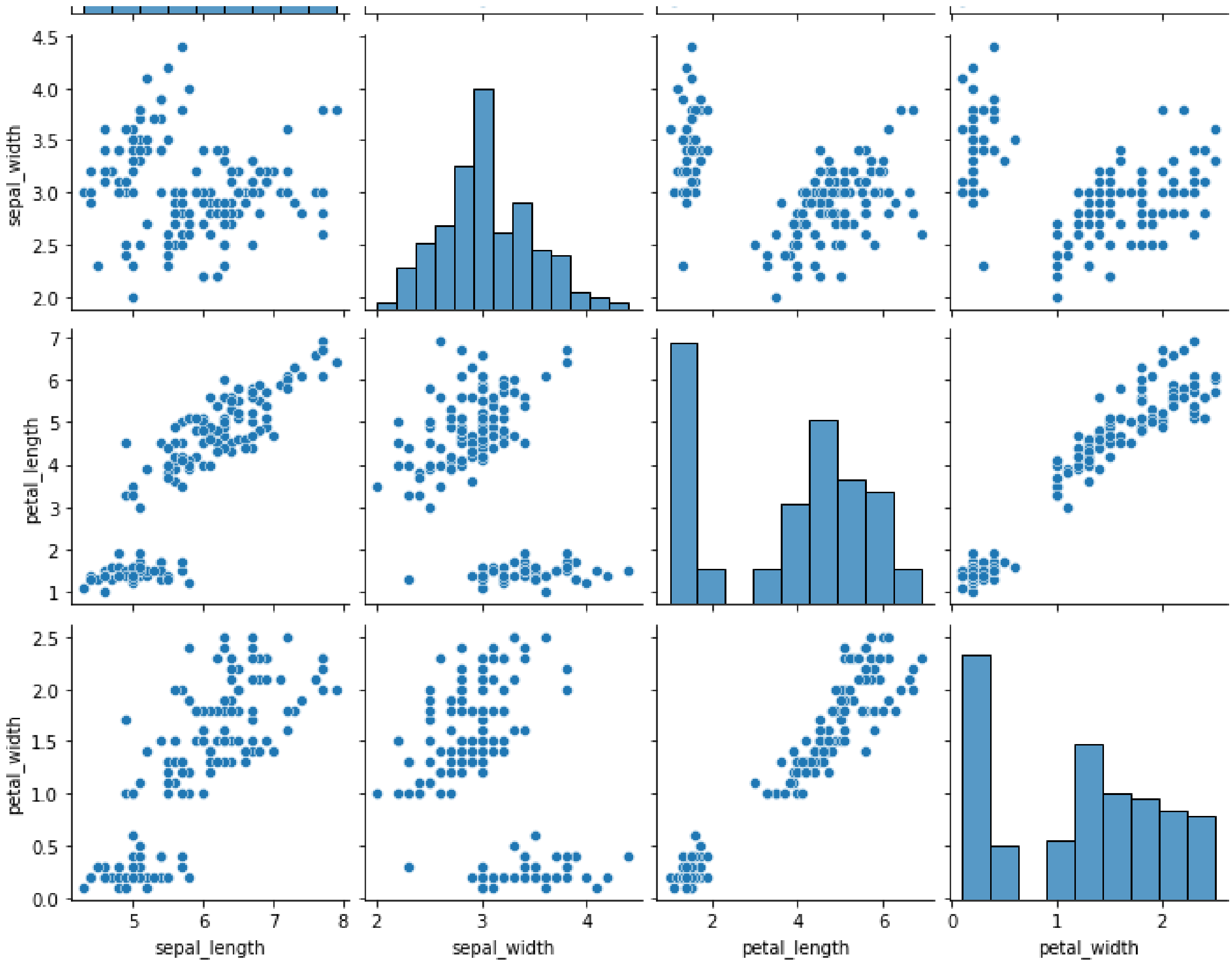
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [7]:

```
sns.pairplot(df)
sns.set_theme(style= 'whitegrid', palette= 'seismic')
```





In [8]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [9]:

```
df.columns
```

Out[9]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [11]:

```
scaler.fit(df[['sepal_length','sepal_width','petal_length','petal_width']])
```

Out[11]:

```
StandardScaler()
```

In [14]:

```
scaled_features = scaler.transform(df[['sepal_length','sepal_width','petal_length','petal
_width']])
scaled_features[:5]
```

Out[14]:

```
array([[-0.90068117,  1.01900435, -1.34022653, -1.3154443 ],
       [-1.14301691, -0.13197948, -1.34022653, -1.3154443 ],
       [-1.38535265,  0.32841405, -1.39706395, -1.3154443 ],
       [-1.50652052,  0.09821729, -1.2833891 , -1.3154443 ],
       [-1.02184904,  1.24920112, -1.34022653, -1.3154443 ]])
```

In [16]:

```
df_new = pd.DataFrame(scaled_features, columns= df.columns[:-1])
df_new[:5]
```

Out[16]:

	sepal_length	sepal_width	petal_length	petal_width
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

In [19]:

```
X= df_new
X[:5]
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

In [23]:

```
y= df['species']
y[:5]
```

Out[23]:

```
0    setosa
1    setosa
2    setosa
3    setosa
4    setosa
Name: species, dtype: object
```

In [24]:

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
```

In [25]:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=1)
```

In [26]:

```
model.fit(X_train, y_train)
```

Out[26]:

```
KNeighborsClassifier(n_neighbors=1)
```

In [65]:

```
prediction = model.predict(X_test)
prediction[17:25]
```

Out[65]:

```
array(['versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
       'setosa', 'virginica', 'virginica'], dtype=object)
```

In [64]:

```
y_test[17:25]
```

Out[64]:

```
69     versicolor
55     versicolor
132     virginica
29         setosa
127     virginica
26         setosa
128     virginica
131     virginica
Name: species, dtype: object
```

In [30]:

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

```
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      0.92      0.96        13
   virginica       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

Here we already have 98% accuracy, but still we will try to maximise the accuracy as much as possible

In [31]:

```
error_rate = []
```

In [33]:

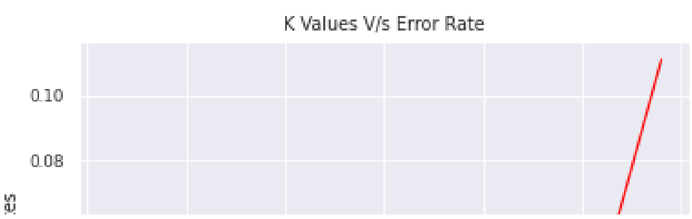
```
for i in range(1,30):
    model_final = KNeighborsClassifier(n_neighbors=i)
    model_final.fit(X_train, y_train)
    prediction_final = model_final.predict(X_test)
    error_rate.append(np.mean(prediction_final != y_test))
```

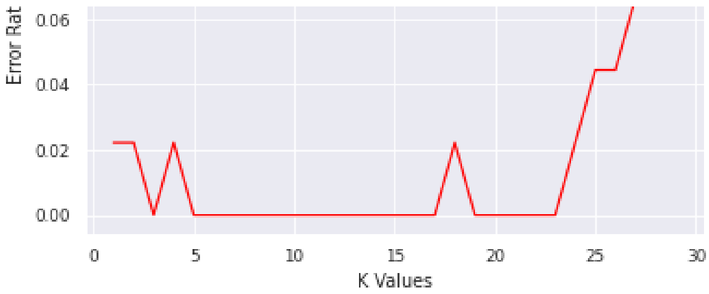
In [53]:

```
sns.lineplot(x= range(1,30), y= error_rate, color= 'red')
sns.set_theme(context= 'paper', style = 'darkgrid')
plt.title("K Values V/s Error Rate")
plt.xlabel("K Values")
plt.ylabel("Error Rates")
```

Out[53]:

```
Text(0, 0.5, 'Error Rates')
```





In [54]:

```
# We see that for the K values of range 5-17 has the least amount of error rate

knn= KNeighborsClassifier(n_neighbors=9)
```

In [55]:

```
knn.fit(X_train,y_train)
```

Out[55]:

```
KNeighborsClassifier(n_neighbors=9)
```

In [62]:

```
knn_prediction = knn.predict(X_test)
knn_prediction[17:25]
```

Out[62]:

```
array(['versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
       'setosa', 'virginica', 'virginica'], dtype=object)
```

In [63]:

```
y_test[17:25]
```

Out[63]:

```
69     versicolor
55     versicolor
132     virginica
29         setosa
127     virginica
26         setosa
128     virginica
131     virginica
Name: species, dtype: object
```

In [66]:

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

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Here, we can see that we have achieved 100% accuracy

Machine Learning Lab-9

Implement Support Vector Machine algorithm for classification in a given business environment and comment on its efficiency and performance.

In [1]:

```
# Importing the libraries
import numpy as  np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
# Now let's import the data from sklearn library
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()    # we intialise the cancer model to load the dataset
```

In [3]:

```
# we can check the contents of the cancer datasets loaded from sklearn
cancer.keys()
```

Out[3]:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename
', 'data_module'])
```

In [4]:

```
# we can check the description of the cancer dataset provided from sklearn
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.
```

```
        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass. They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree. Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].
```

```
This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

In [5]:

```
print(cancer['target_names'])
# we see that these are the two labels to be classified
```

```
['malignant' 'benign']
```

In [6]:

```
# Now, let's load this data to a DataFrame
df= pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
```

In [7]:

```
df[:5]  # similar to head function
```

Out[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	wors textur
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.3
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.4
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.5
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.5
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.6

5 rows × 30 columns

In [8]:

```
# let's check the info()
df.info()
# we see that there are 569 observations and 30 features
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
```

```
7   mean concave points      569 non-null    float64
```

```
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [9]:

```python
# Let's look at the summary of the dataframe
df.describe()
```

Out[9]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.

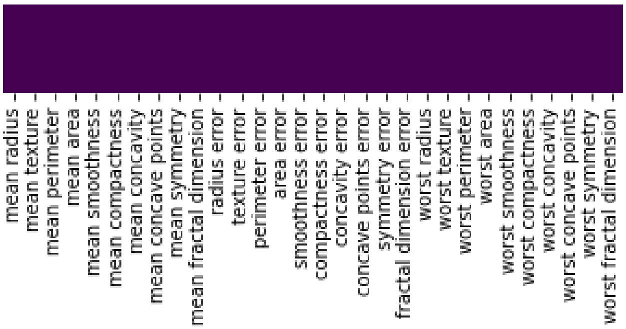
8 rows × 30 columns

In [10]:

```python
# Let's check whether there are any missing values present in the data or not
sns.heatmap(df.isna(), yticklabels= False, cbar= False, cmap= 'viridis')
```

Out[10]:

```
<AxesSubplot:>
```



We see that there are no missing values present in the data and the data is clean

In [11]:

```
df.columns
```

Out[11]:

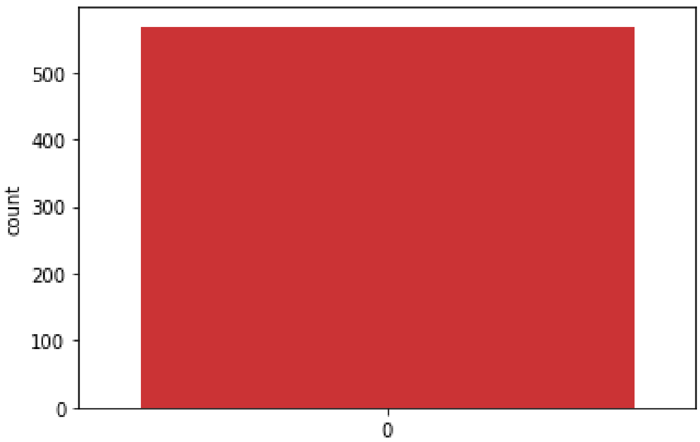
```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension'],
      dtype='object')
```

In [12]:

```
sns.countplot(data=cancer['target'], palette= 'Set1')
# we can see there are almost 200 patients with class label 0
# and around 350 patients with class label 1
```

Out[12]:

<AxesSubplot:ylabel='count'>

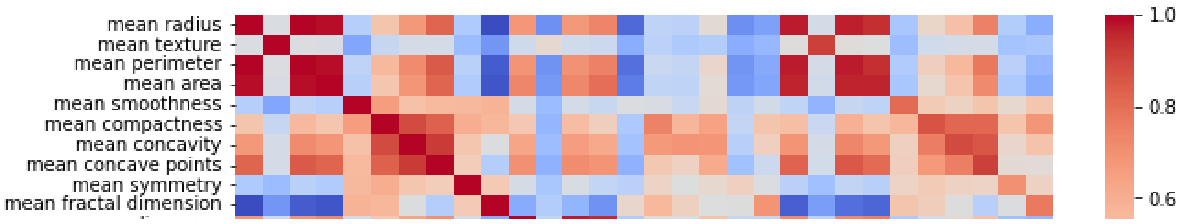


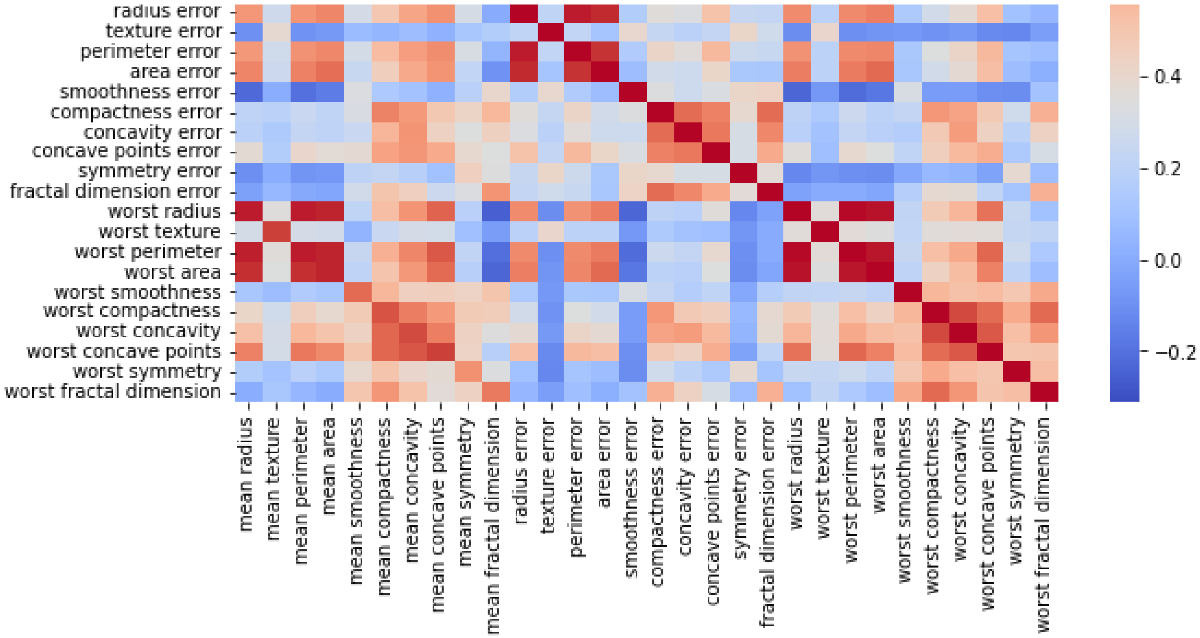
In [13]:

```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), cmap= 'coolwarm')
```

Out[13]:

<AxesSubplot:>





In [14]:

```
# now let's create feature matrix and target array
X = df
y = cancer['target']
```

In [15]:

```
# Now let's split the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state =
101)
```

In [16]:

```
# Let's build our SVM model now
from sklearn.svm import SVC
model_1 = SVC()    # let's intialise the model

# default kernel is RBF kernel
```

In [17]:

```
# Now we train the model
model_1.fit(X_train, y_train)
```

Out[17]:

```
SVC()
```

In [18]:

```
pred = model_1.predict(X_test)
```

In [19]:

```
pred[:5]
```

Out[19]:

```
array([1, 1, 1, 0, 1])
```

In [20]:

```
y_test[:5]
```

Out[20]:

```
array([1, 1, 1, 0, 1])
```

In [21]:

```
# Model Evaluation Process
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[ 56  10]
 [  3 102]]
              precision    recall  f1-score   support

           0       0.95      0.85      0.90        66
           1       0.91      0.97      0.94       105

    accuracy                           0.92       171
   macro avg       0.93      0.91      0.92       171
weighted avg       0.93      0.92      0.92       171
```

In [22]:

```
model_2 = SVC(kernel = 'linear')
model_2.fit(X_train, y_train)
```

Out[22]:

```
SVC(kernel='linear')
```

In [23]:

```
pred_2 = model_2.predict(X_test)
```

In [24]:

```
print(confusion_matrix(y_test, pred_2))
print(classification_report(y_test, pred_2))
```

```
[[ 60   6]
 [  3 102]]
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        66
           1       0.94      0.97      0.96       105

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

In [25]:

```
model_3 = SVC(kernel = 'poly')
model_3.fit(X_train, y_train)
```

Out[25]:

```
SVC(kernel='poly')
```

In [26]:

```
pred_3 = model_3.predict(X_test)
```

In [27]:

```
print(confusion_matrix(y_test, pred_3))
print(classification_report(y_test, pred_3))
```

```
[[ 53  13]
 [  3 102]]
              precision    recall  f1-score   support

           0       0.95      0.80      0.87        66
           1       0.89      0.97      0.93       105
```

```
    accuracy                           0.91       171
   macro avg       0.92      0.89      0.90       171
weighted avg       0.91      0.91      0.90       171
```

In [28]:

```
model_4 = SVC(kernel = 'sigmoid')
model_4.fit(X_train, y_train)
```

Out[28]:

```
SVC(kernel='sigmoid')
```

In [29]:

```
pred_4 = model_4.predict(X_test)
```

In [30]:

```
print(confusion_matrix(y_test, pred_4))
print(classification_report(y_test, pred_4))
```

```
[[10 56]
 [44 61]]
              precision    recall  f1-score   support

           0       0.19      0.15      0.17        66
           1       0.52      0.58      0.55       105

    accuracy                           0.42       171
   macro avg       0.35      0.37      0.36       171
weighted avg       0.39      0.42      0.40       171
```

In [31]:

```
print("Accuracy achieved in each kernel:- ")
print()
print("for RBF kernel: 92%")
print("for linear kernel: 95%")
print("for polynomial kernel: 91%")
print("for sigmoidal kernel: 42%")
```

```
Accuracy achieved in each kernel:-

for RBF kernel: 92%
for linear kernel: 95%
for polynomial kernel: 91%
for sigmoidal kernel: 42%
```

Machine Learning Lab-10

Implement Principal Component Analysis for dimensionality reduction in a given business environment and comment on its efficiency and performance.

In [1]:

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

In [48]:

```
df = sns.load_dataset('iris')
df.head()
```

Out[48]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [49]:

```
df.shape
```

Out[49]:

```
(150, 5)
```

In [50]:

```
df2 = df.copy()
```

In [51]:

```
df.drop(['petal_length','petal_width'], inplace= True, axis = 1)
```

In [52]:

```
df.head()
```

Out[52]:

	sepal_length	sepal_width	species
0	5.1	3.5	setosa
1	4.9	3.0	setosa
2	4.7	3.2	setosa
3	4.6	3.1	setosa
4	5.0	3.6	setosa

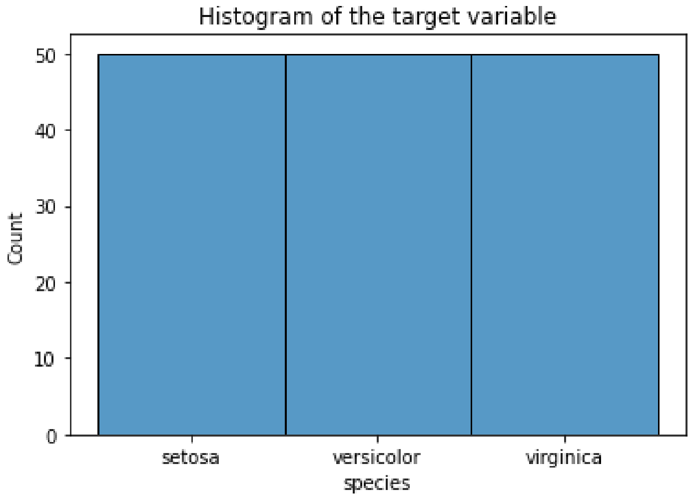
Exploratory Data Analysis

In [53]:

```
sns.histplot(df['species'])
plt.title('Histogram of the target variable')
```

Out[53]:

```
Text(0.5, 1.0, 'Histogram of the target variable')
```

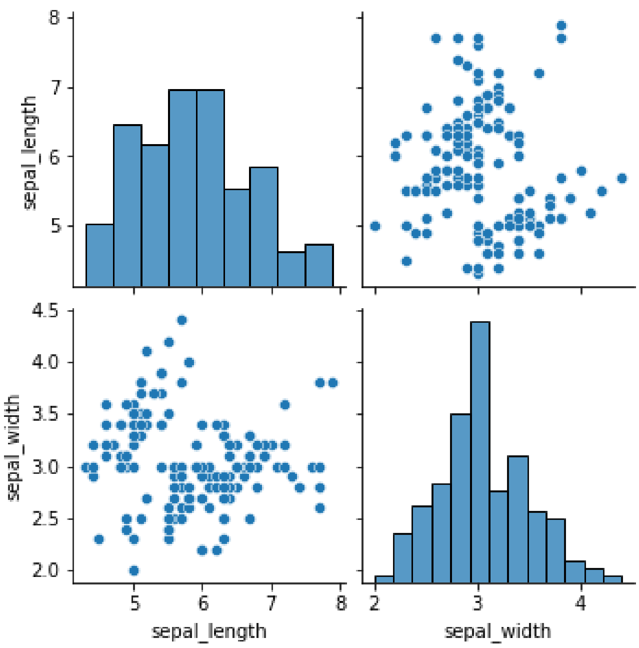


In [54]:

```
plt.figure(figsize=(6,6))
sns.pairplot(data=df)
```

Out[54]:

```
<seaborn.axisgrid.PairGrid at 0x1152008d820>

<Figure size 432x432 with 0 Axes>
```



- **Here, we can see that the plot is almost normally distributed for both the features**

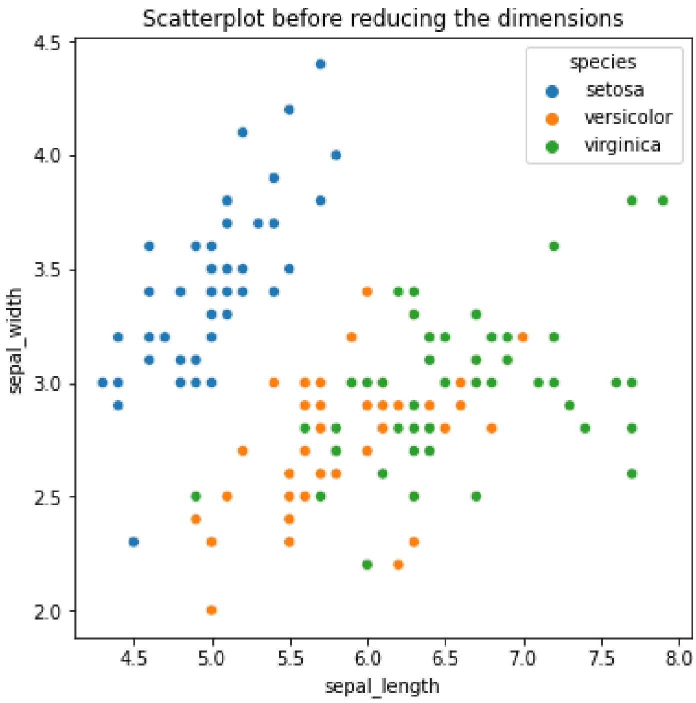
PCA FOR REDUCING 2D to 1D

In [55]:

```
plt.figure(figsize=(6,6))
sns.scatterplot(x='sepal_length',y='sepal_width', data=df, hue = 'species')
plt.title("Scatterplot before reducing the dimensions")
```

Out[55]:

```
Text(0.5, 1.0, 'Scatterplot before reducing the dimensions')
```



In [56]:

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoded = encoder.fit_transform(df['species'])
df['Species'] = encoded
```

In [57]:

```
df.drop(['species'],axis=1,inplace=True)
```

In [58]:

```
df.head()
```

Out[58]:

	sepal_length	sepal_width	Species
0	5.1	3.5	0
1	4.9	3.0	0
2	4.7	3.2	0
3	4.6	3.1	0
4	5.0	3.6	0

In [59]:

```
from sklearn.decomposition import PCA
pca_model = PCA(n_components=1)
```

In [60]:

```
x_pca = pca_model.fit_transform(df)
```

In [61]:

```
component_df = pd.DataFrame(data=x_pca, columns=['component_value'])
component_df[:5]
```

Out[61]:

	component_value
0	-1.280815
1	-1.354709
2	-1.520858
3	-1.577574
4	-1.363889

In [62]:

```
component_df.shape
```

Out[62]:

```
(150, 1)
```

In [64]:

```
reduced_df = pd.concat([component_df,df['Species']],axis=1)
reduced_df[:5]
```

Out[64]:

	component_value	Species
0	-1.280815	0
1	-1.354709	0
2	-1.520858	0
3	-1.577574	0
4	-1.363889	0

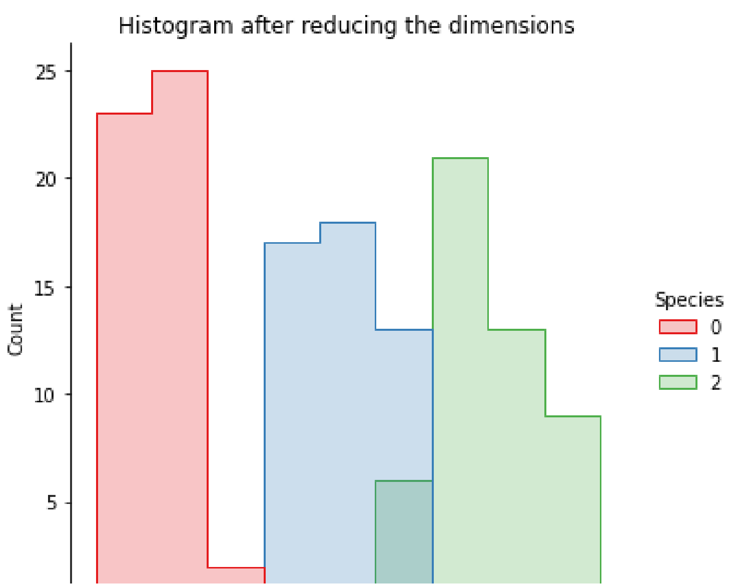
In [65]:

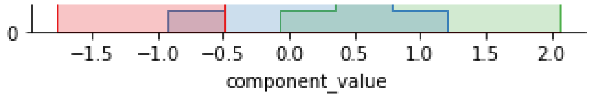
```
plt.figure(figsize=(5,5))
sns.displot(reduced_df, x="component_value", hue="Species", element="step", palette='Set
1')
plt.title('Histogram after reducing the dimensions')
```

Out[65]:

```
Text(0.5, 1.0, 'Histogram after reducing the dimensions')
```

```
<Figure size 360x360 with 0 Axes>
```



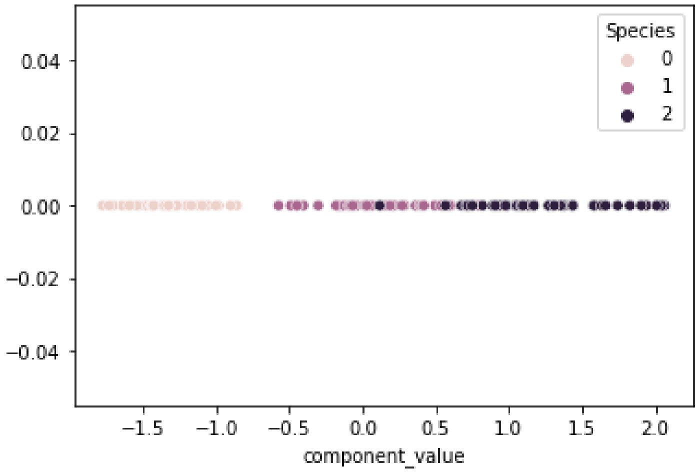


In [67]:

```
sns.scatterplot(x=reduced_df['component_value'], y=0,hue=reduced_df['Species'])
```

Out[67]:

<AxesSubplot:xlabel='component_value'>



PCA FOR REDUCING 3D to 2D

In [69]:

```
df2.head()
```

Out[69]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- **Will be considering sepal_length, sepal_width, petal_length as features**

In [70]:

```
df2.drop(['petal_width'],inplace=True,axis=1)
```

In [71]:

```
df2.head()
```

Out[71]:

	sepal_length	sepal_width	petal_length	species
0	5.1	3.5	1.4	setosa
1	4.9	3.0	1.4	setosa
2	4.7	3.2	1.3	setosa
3	4.6	3.1	1.5	setosa
4	5.0	3.6	1.4	setosa

In [72]:

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoded = encoder.fit_transform(df2['species'])
df2['Species'] = encoded

df2.drop(['species'],axis=1,inplace=True)
```

In [73]:

```
df2.head()
```

Out[73]:

	sepal_length	sepal_width	petal_length	Species
0	5.1	3.5	1.4	0
1	4.9	3.0	1.4	0
2	4.7	3.2	1.3	0
3	4.6	3.1	1.5	0
4	5.0	3.6	1.4	0

In [74]:

```
# scaling the values present in the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df2)
```

In [75]:

```
scaled_df = pd.DataFrame(scaled_features, columns = df2.columns)
```

In [76]:

```
scaled_df[:5]
```

Out[76]:

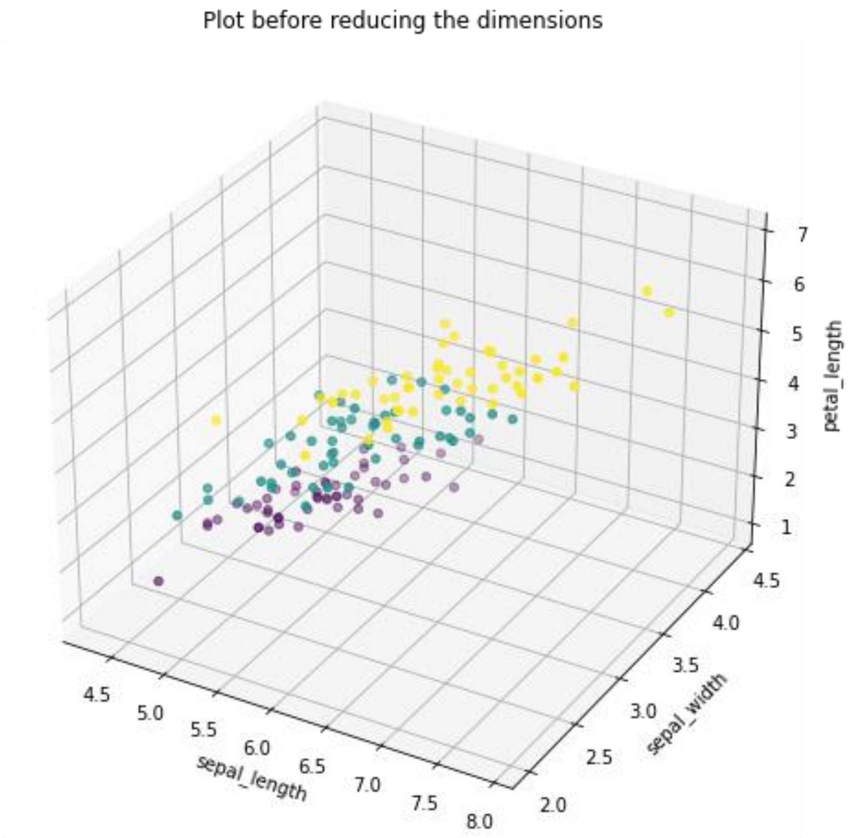
	sepal_length	sepal_width	petal_length	Species
0	-0.900681	1.019004	-1.340227	-1.224745
1	-1.143017	-0.131979	-1.340227	-1.224745
2	-1.385353	0.328414	-1.397064	-1.224745
3	-1.506521	0.098217	-1.283389	-1.224745
4	-1.021849	1.249201	-1.340227	-1.224745

In [77]:

```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection = '3d')

x = df2['sepal_length']
y = df2['sepal_width']
z = df2['petal_length']

ax.set_xlabel("sepal_length")
ax.set_ylabel("sepal_width")
ax.set_zlabel("petal_length")

ax.scatter(x, y, z, c=df2['Species'])

plt.title("Plot before reducing the dimensions")
```

Out[77]:

```
Text(0.5, 0.92, 'Plot before reducing the dimensions')
```



In [28]:

```
pca_model = PCA(n_components=2)
x_pca= pca_model.fit_transform(df2)
component_df = pd.DataFrame(data=x_pca, columns=['component_value 1','component_value 2'
])
component_df[:5]
```

Out[28]:

	component_value 1	component_value 2
0	-2.685487	0.298473
1	-2.713845	-0.185127
2	-2.887395	-0.168736
3	-2.744465	-0.329862
4	-2.729760	0.300692

In [29]:

```
component_df.shape
```

Out[29]:

```
(150, 2)
```

In [30]:

```
reduced_df = pd.concat([component_df,df2['Species']],axis=1)
reduced_df[:5]
```

Out[30]:

	component_value 1	component_value 2	Species
0	-2.685487	0.298473	0

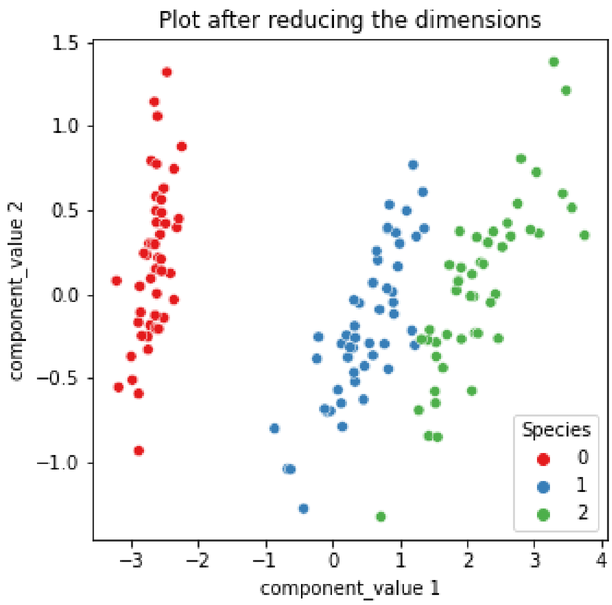
	component_value 1	component_value 2	Species
1	-2.713845	-0.185127	0
2	-2.887395	-0.168736	0
3	-2.744465	-0.329862	0
4	-2.729760	0.300692	0

In [31]:

```
plt.figure(figsize=(5,5))
sns.scatterplot(data = reduced_df, x="component_value 1", y="component_value 2", hue="Spe
cies", palette='Set1')
plt.title('Plot after reducing the dimensions')
```

Out[31]:

```
Text(0.5, 1.0, 'Plot after reducing the dimensions')
```



ANALYSIS

- Let's create a ML model and train with original data and reduced data
- Then we compare with the amount of time the model takes to train and also the accuracy

In [32]:

```
# Original data with 3 dimensions
df2.head()
```

Out[32]:

	sepal_length	sepal_width	petal_length	Species
0	5.1	3.5	1.4	0
1	4.9	3.0	1.4	0
2	4.7	3.2	1.3	0
3	4.6	3.1	1.5	0
4	5.0	3.6	1.4	0

In [33]:

```
# reduced data with 2 dimensions
reduced_df.head()
```

Out[33]:

	component_value 1	component_value 2	Species
0	-2.685487	0.298473	0
1	-2.713845	-0.185127	0
2	-2.887395	-0.168736	0
3	-2.744465	-0.329862	0
4	-2.729760	0.300692	0

Analysis for Original Data

In [34]:

```
X = df2[['sepal_length','sepal_width','petal_length']]
y = df2['Species']
```

In [35]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=42)
```

In [36]:

```
# let's consider a decision tree model
from sklearn.tree import DecisionTreeClassifier
original_model = DecisionTreeClassifier().fit(X_train, y_train)
original_pred = original_model.predict(X_test)
```

In [37]:

```
# model evaluation
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,original_pred))
print(confusion_matrix(y_test, original_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.87      1.00      0.93        13
           2       1.00      0.85      0.92        13

    accuracy                           0.96        45
   macro avg       0.96      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45

[[19  0  0]
 [ 0 13  0]
 [ 0  2 11]]
```

Analysis for Reduced Data

In [38]:

```
X = reduced_df[['component_value 1','component_value 2']]
y = reduced_df['Species']
```

In [39]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=42)
```

In [40]:

```
# let's consider a decision tree model
from sklearn.tree import DecisionTreeClassifier
reduced_model = DecisionTreeClassifier().fit(X_train, y_train)
reduced_pred = reduced_model.predict(X_test)
```

In [41]:

```
# model evaluation
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,reduced_pred))
print(confusion_matrix(y_test, reduced_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

- **we see that the reduced data is performing well**

Machine Learning Lab-11

Perform Time Series Analysis in a given business environment exploring Horizontal Pattern, Trend Pattern, Seasonal Pattern, and moving averages and comment on Forecasting accuracy.

What is a time series problem

In the field for machine learning and data science, most of the real-life problems are based upon the prediction of future which is totally oblivious to us such as stock market prediction, future sales prediction and so on.Time series problem is basically the prediction of such problems using various machine learning tools.Time series problem is tackled efficiently when first it is analyzed properly (Time Series Analysis) and according to that observation suitable algorithm is used (Time Series Forecasting).

In [1]:

```
# Load required Libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #to plot some parameters in seaborn
from sklearn.linear_model import LinearRegression # To work on Linear Regression
from sklearn.metrics import r2_score # To Calculate Performance matrix
import statsmodels.api as sm # To calculatestats modle
import seaborn as sns
```

Importing Dataset

In [2]:

```
# Reading the data
df = pd.read_csv("portland-oregon-average-monthly.csv")
```

In [3]:

```
# A glance on the data
df.head()
```

Out[3]:

	Month	Portland Oregon average monthly bus ridership (/100) January 1973 through June 1982, n=114
0	1960-01	648
1	1960-02	646
2	1960-03	639
3	1960-04	654
4	1960-05	630

In [4]:

```
# getting some information about dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115 entries, 0 to 114
```

```
Data columns (total 2 columns):
 #   Column
Non-Null Count  Dtype
---  ------
-----------------------------------  ------
 0   Month
115 non-null    object
 1   Portland Oregon average monthly bus ridership (/100) January 1973 through June 1982,
n=114  115 non-null    object
dtypes: object(2)
memory usage: 1.9+ KB
```

From this you can infer two necessary things:

1. **You really need to change change columns name**
2. **Both the columns have object datatype**

In [5]:

```
# further Analysis
df.describe()
```

Out[5]:

	Month	Portland Oregon average monthly bus ridership (/100) January 1973 through June 1982, n=114
count	115	115
unique	115	112
top	1965-12	1424
freq	1	2

In [6]:

```
df.columns = ["month", "average_monthly_ridership"]
df.head()
```

Out[6]:

	month	average_monthly_ridership
0	1960-01	648
1	1960-02	646
2	1960-03	639
3	1960-04	654
4	1960-05	630

In [7]:

```
df.dtypes
```

Out[7]:

```
month                        object
average_monthly_ridership    object
dtype: object
```

In [8]:

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

Out[8]:

```
array(['648', '646', '639', '654', '630', '622', '617', '613', '661',
       '695', '690', '707', '817', '839', '810', '789', '760', '724',
       '704', '691', '745', '803', '780', '761', '857', '907', '873',
```

```
       '910', '900', '880', '867', '854', '928', '1064', '1103', '1026',
       '1102', '1080', '1034', '1083', '1078', '1020', '984', '952',
       '1033', '1114', '1160', '1058', '1209', '1200', '1130', '1182',
       '1152', '1116', '1098', '1044', '1142', '1222', '1234', '1155',
       '1286', '1281', '1224', '1280', '1228', '1181', '1156', '1124',
       '1205', '1260', '1188', '1212', '1269', '1246', '1299', '1284',
       '1345', '1341', '1308', '1448', '1454', '1467', '1431', '1510',
       '1558', '1536', '1523', '1492', '1437', '1365', '1310', '1441',
       '1450', '1424', '1360', '1429', '1440', '1414', '1408', '1337',
       '1258', '1214', '1326', '1417', '1329', '1461', '1425', '1419',
       '1432', '1394', '1327', ' n=114'], dtype=object)
```

We can see here that this series consist an anamolous data which is the last one.

In [9]:

```
df = df.drop(df.index[df['average_monthly_ridership'] == ' n=114'])
```

In [10]:

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

Out[10]:

```
array(['648', '646', '639', '654', '630', '622', '617', '613', '661',
       '695', '690', '707', '817', '839', '810', '789', '760', '724',
       '704', '691', '745', '803', '780', '761', '857', '907', '873',
       '910', '900', '880', '867', '854', '928', '1064', '1103', '1026',
       '1102', '1080', '1034', '1083', '1078', '1020', '984', '952',
       '1033', '1114', '1160', '1058', '1209', '1200', '1130', '1182',
       '1152', '1116', '1098', '1044', '1142', '1222', '1234', '1155',
       '1286', '1281', '1224', '1280', '1228', '1181', '1156', '1124',
       '1205', '1260', '1188', '1212', '1269', '1246', '1299', '1284',
       '1345', '1341', '1308', '1448', '1454', '1467', '1431', '1510',
       '1558', '1536', '1523', '1492', '1437', '1365', '1310', '1441',
       '1450', '1424', '1360', '1429', '1440', '1414', '1408', '1337',
       '1258', '1214', '1326', '1417', '1329', '1461', '1425', '1419',
       '1432', '1394', '1327'], dtype=object)
```

Now our data is clean !!!

Changing data type of both the column

- **Assign int to `monthly_ridership_data` column**
- **Assign datetime to `month` column**

In [11]:

```
df['average_monthly_ridership'] = df['average_monthly_ridership'].astype(np.int32)
```

In [12]:

```
df['month'] = pd.to_datetime(df['month'], format = '%Y-%m')
```

In [13]:

```
df.dtypes
```

Out[13]:

```
month                        datetime64[ns]
average_monthly_ridership             int32
dtype: object
```

Time Series Analysis

Horizental Pattern :- Horizontal pattern exists when data values fluctuate around a constant mean. This is the simplest pattern and the easiest to predict. An example is sales of a product that do not increase or decrease

over time. This type of pattern is common for products in the mature stage of their life cycle, in which demand is steady and predictable.

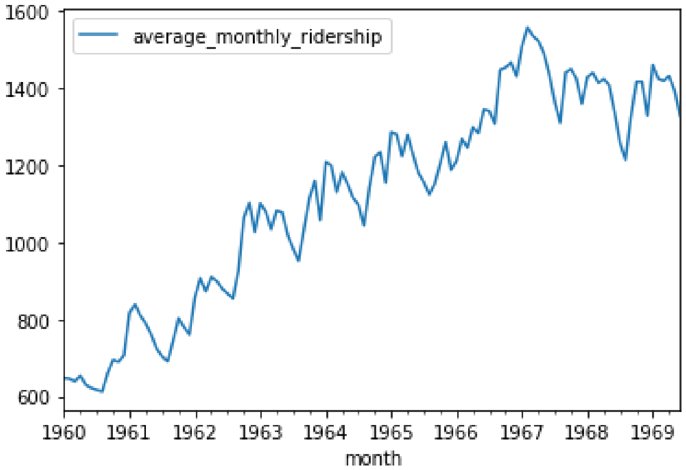
Trend Pattern:- As the name suggests trend depicts the variation in the output as time increases.It is often non-linear. Sometimes we will refer to trend as “changing direction” when it might go from an increasing trend to a decreasing trend.

Seasonal Pattern:- As its name depicts it shows the repeated pattern over time. In layman terms, it shows the seasonal variation of data over time.

Moving Average:-As the name suggests moving average is a technique to get an overall idea of the trends in a data set; it is an average of any subset of numbers. The moving average is extremely useful for forecasting long-term trends

In [14]:

```
# Normal line plot so that we can see data variation
# We can observe that average number of riders is increasing most of the time
# We'll later see decomposed analysis of that curve
df.plot.line(x = 'month', y = 'average_monthly_ridership')
plt.show()
```



Ploting monthly variation of dataset

It gives us idea about seasonal variation of our data set

In [15]:

```
to_plot_monthly_variation = df
```

In [16]:

```
# only storing month for each index
mon = df['month']
```

In [17]:

```
# decompose yyyy-mm data-type
temp= pd.DatetimeIndex(mon)
```

In [18]:

```
# assign month part of that data to ```month``` variable
month = pd.Series(temp.month)
```

In [19]:

```
# dropping month from to_plot_monthly_variation
to_plot_monthly_variation = to_plot_monthly_variation.drop(['month'], axis = 1)
```

In [20]:

```
# join months so we can get month to average monthly rider mapping
to_plot_monthly_variation = to_plot_monthly_variation.join(month)
```

In [21]:

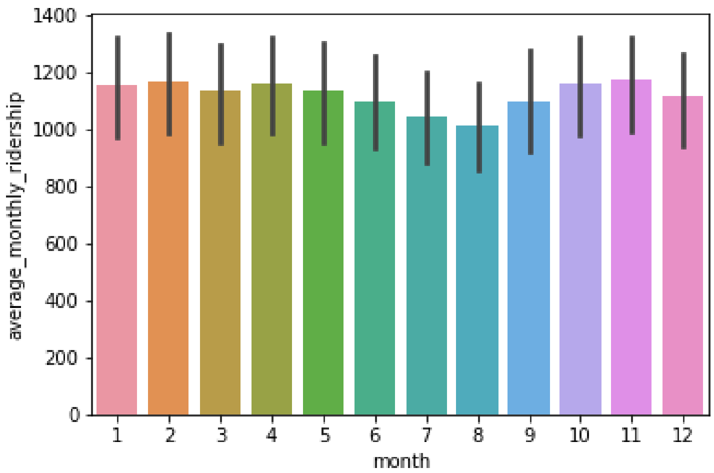
```
# A quick glance
to_plot_monthly_variation.head()
```

Out[21]:

	average_monthly_ridership	month
0	648	1
1	646	2
2	639	3
3	654	4
4	630	5

In [22]:

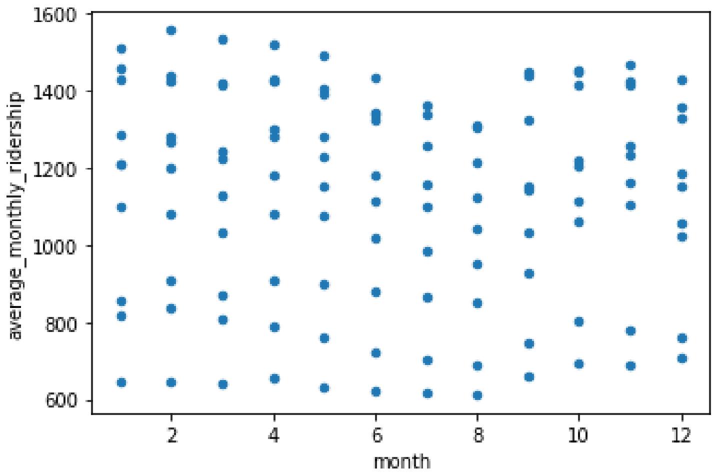
```
# Plotting bar plot for each month
sns.barplot(x = 'month', y = 'average_monthly_ridership', data = to_plot_monthly_variati
on)
plt.show()
```



Well this looks tough to decode. Not a typical box plot. One can infer that data is too sparse for this graph to represent any pattern. Hence it cannot represents monthly variation effectively.In such a scenerio we can use our traditional scatter plot to understand pattern in dataset

In [23]:

```
to_plot_monthly_variation.plot.scatter(x = 'month', y = 'average_monthly_ridership')
plt.show()
```



We can see here the yearly variation of data in this plot. To understand this curve more effectively first look at the every row from bottom to top and see each year's variation.To understand yearly variation take a look at each column representing a month.

Another tool to visualize the data is the seasonal_decompose function in statsmodel. With this, the trend and seasonality become even more obvious.

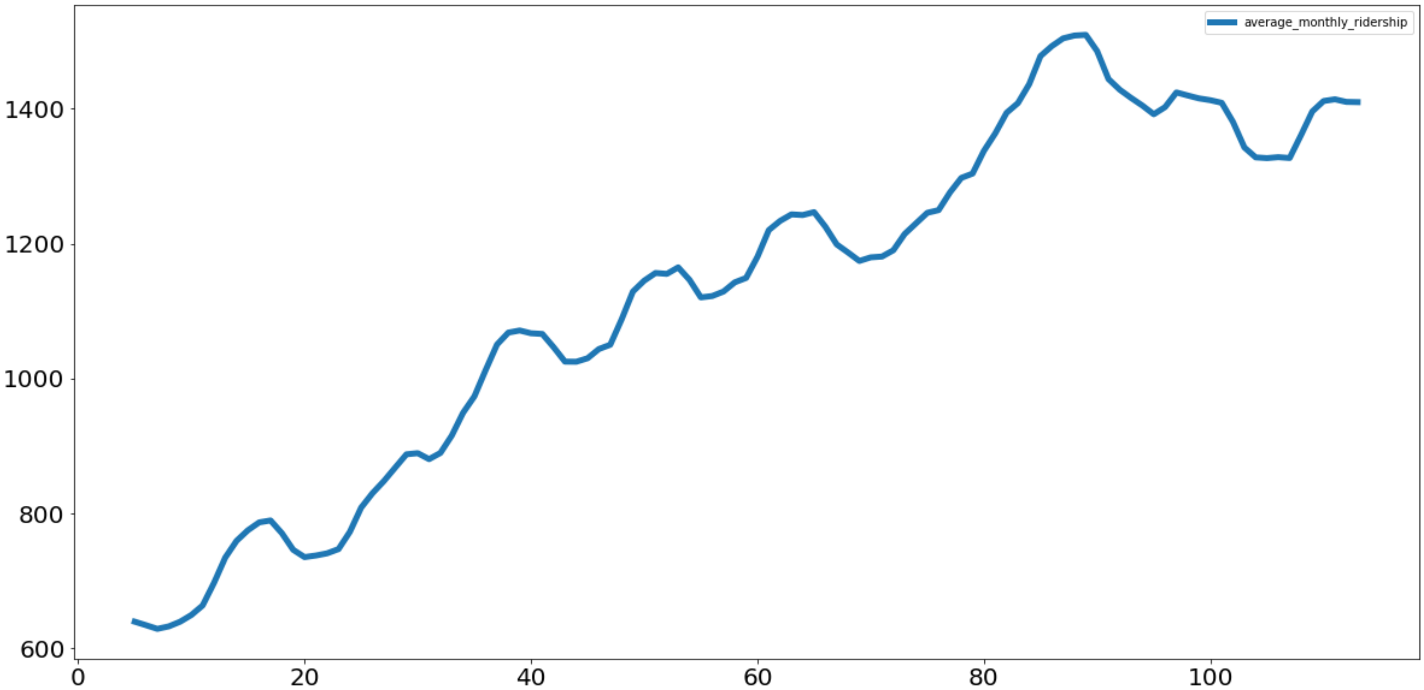
In [24]:

```
rider = df[['average_monthly_ridership']]
```

Trend Analysis

In [25]:

```
rider.rolling(6).mean().plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.show()
```



For trend analysis, we use smoothing techniques. In statistics smoothing a data set means to create an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal.We implement smoothing by taking moving averages. [Exponential moving average] is frequently used to compute smoothed function.Here we used the rolling method which is inbuilt in pandas and frequently used for smoothing.

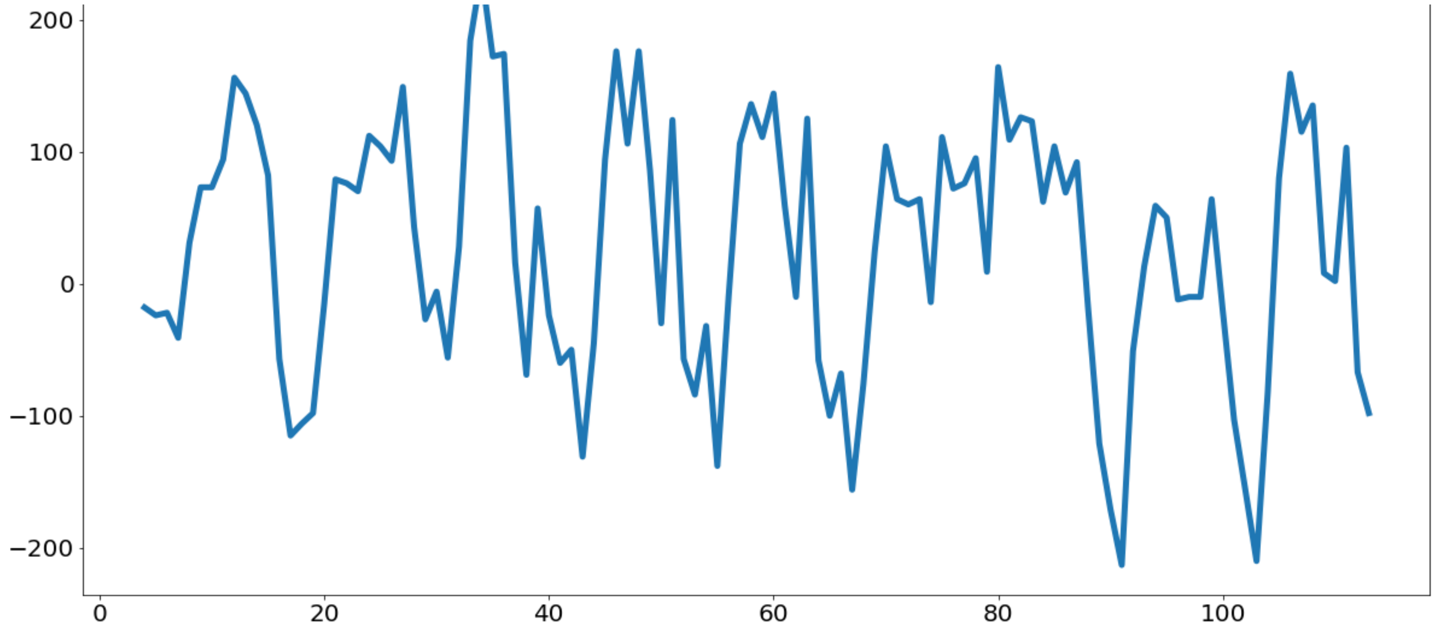
Seasonability Analysis

Two most famous seasonability analysis algorithms are:-

[Using 1st discrete difference of object]

In [26]:

```
rider.diff(periods=4).plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.show()
```



The above figure represents difference between average rider of a month and 4 months before that month i.e

> $$d[month] = a[month] - a[month - periods.]$$

This gives us idea about variation of data for a period of time.

In [27]:

```
df = df.set_index('month')
```

In [28]:

```
# Applying Seasonal ARIMA model to forcast the data
mod = sm.tsa.SARIMAX(df['average_monthly_ridership'], trend='n', order=(0,1,0), seasonal
_order=(1,1,1,12))
results = mod.fit()
print(results.summary())
```

```
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarnin
g: No frequency information was provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarnin
g: No frequency information was provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
```

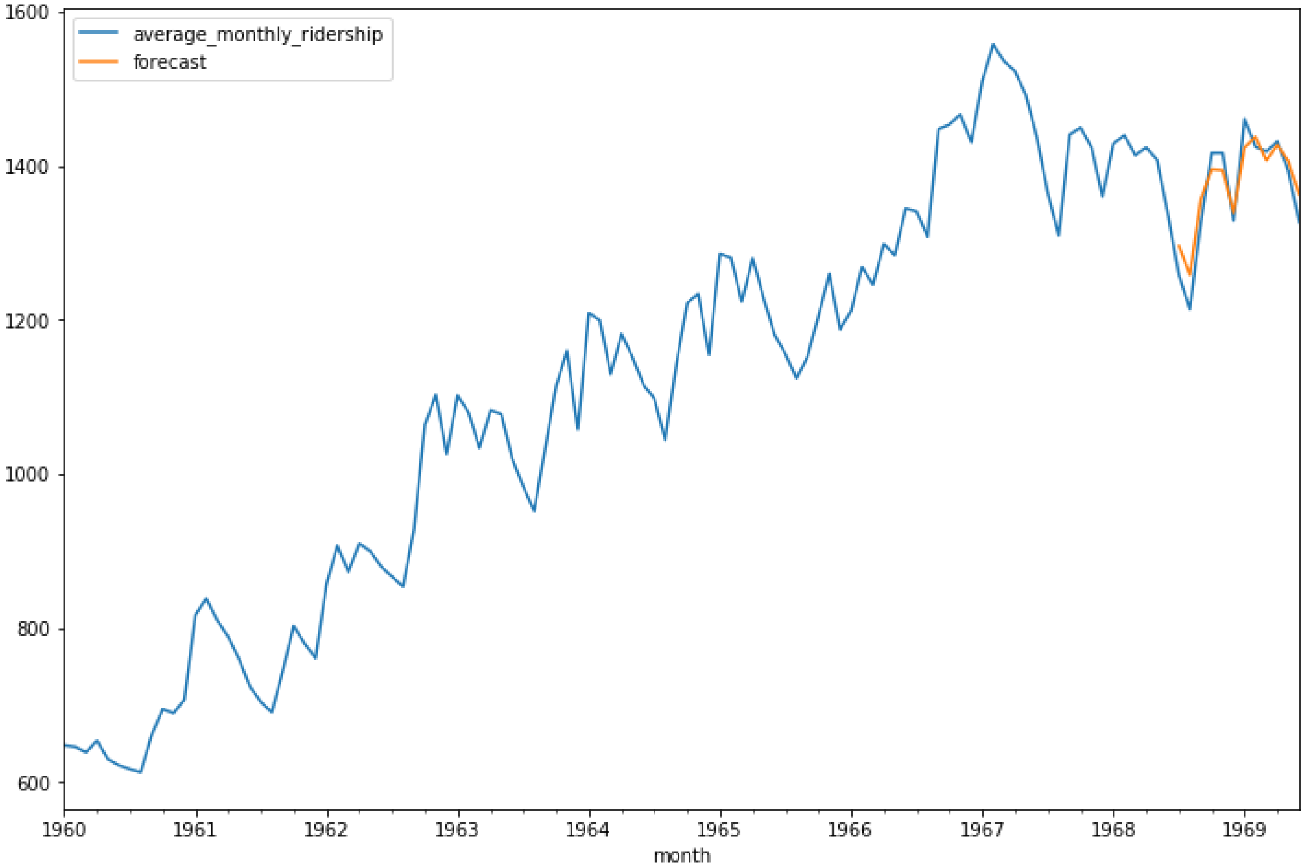
```
                                      SARIMAX Results                                     
==========================================================================================
===
Dep. Variable:              average_monthly_ridership   No. Observations:                
114
Model:             SARIMAX(0, 1, 0)x(1, 1, [1], 12)   Log Likelihood                 -501
.340
Date:                              Sun, 09 Aug 2020   AIC                            1008
.680
Time:                                      12:10:34   BIC                            101
6.526
Sample:                                  01-01-1960   HQIC                           1011
.856
                                       - 06-01-1969                                      

Covariance Type:                                opg                                      
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12       0.3236      0.186      1.739      0.082      -0.041       0.688
ma.S.L12      -0.9990     41.957     -0.024      0.981     -83.232      81.234
sigma2       984.8231   4.12e+04      0.024      0.981   -7.98e+04    8.17e+04
```

```
===================================================================================
Ljung-Box (Q):                       36.56   Jarque-Bera (JB):                  4.81
Prob(Q):                              0.63   Prob(JB):                          0.09
Heteroskedasticity (H):               1.48   Skew:                              0.38
Prob(H) (two-sided):                  0.26   Kurtosis:                          3.75
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Forecast

In [29]:

```
df['forecast'] = results.predict(start = 102, end= 120, dynamic= True)
df[['average_monthly_ridership', 'forecast']].plot(figsize=(12, 8))
plt.show()
```



Forecast Accuracy

In [30]:

```
expected=df['average_monthly_ridership'].tail(12)
predictions=df['forecast'].tail(12)
```

In [31]:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
mse = mean_squared_error(expected, predictions)
rmse = sqrt(mse)
print('Root MeanSquared Error: %f' % rmse)
```

```
Root MeanSquared Error: 26.772801
```

The RMSE error values are in the same units as the predictions. As with the mean squared error, an RMSE of zero indicates no error

Machine Learning Lab-12

Explore Holt's Linear Exponential Smoothing, Nonlinear Trend Regression, and Seasonality for the Time Series Analysis in a given business environment.

In [29]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
```

Data Preparation

In [30]:

```
data = [446.6565,  454.4733,  455.663 ,  423.6322,  456.2713,  440.5881, 425.3325,  485.
1494,  506.0482,  526.792 ,  514.2689,  494.211 ]
index= pd.date_range(start='1996', end='2008', freq='A')
oildata = pd.Series(data, index)

data = [17.5534,  21.86  ,  23.8866,  26.9293,  26.8885,  28.8314, 30.0751,  30.9535,  3
0.1857,  31.5797,  32.5776,  33.4774, 39.0216,  41.3864,  41.5966]
index= pd.date_range(start='1990', end='2005', freq='A')
air = pd.Series(data, index)

data = [263.9177,  268.3072,  260.6626,  266.6394,  277.5158,  283.834 , 290.309 ,  292.
4742,  300.8307,  309.2867,  318.3311,  329.3724, 338.884 ,  339.2441,  328.6006,  314.2
554,  314.4597,  321.4138, 329.7893,  346.3852,  352.2979,  348.3705,  417.5629,  417.123
6, 417.7495,  412.2339,  411.9468,  394.6971,  401.4993,  408.2705, 414.2428]
index= pd.date_range(start='1970', end='2001', freq='A')
livestock2 = pd.Series(data, index)

data = [407.9979 ,  403.4608,  413.8249,  428.105 ,  445.3387,  452.9942, 455.7402]
index= pd.date_range(start='2001', end='2008', freq='A')
livestock3 = pd.Series(data, index)

data = [41.7275,  24.0418,  32.3281,  37.3287,  46.2132,  29.3463, 36.4829,  42.9777,  4
8.9015,  31.1802,  37.7179,  40.4202, 51.2069,  31.8872,  40.9783,  43.7725,  55.5586,
33.8509, 42.0764,  45.6423,  59.7668,  35.1919,  44.3197,  47.9137]
index= pd.date_range(start='2005', end='2010-Q4', freq='QS-OCT')
aust = pd.Series(data, index)
```

Simple Exponential Smoothing

In [31]:

```
ax=oildata.plot()
ax.set_xlabel("Year")
ax.set_ylabel("Oil (millions of tonnes)")
plt.show()
print("Figure 7.1: Oil production in Saudi Arabia from 1996 to 2007.")
```

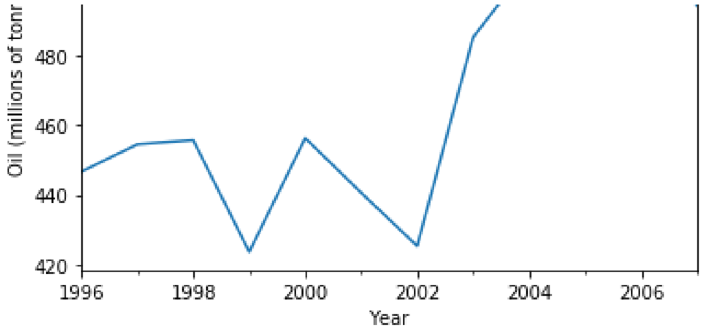


Figure 7.1: Oil production in Saudi Arabia from 1996 to 2007.

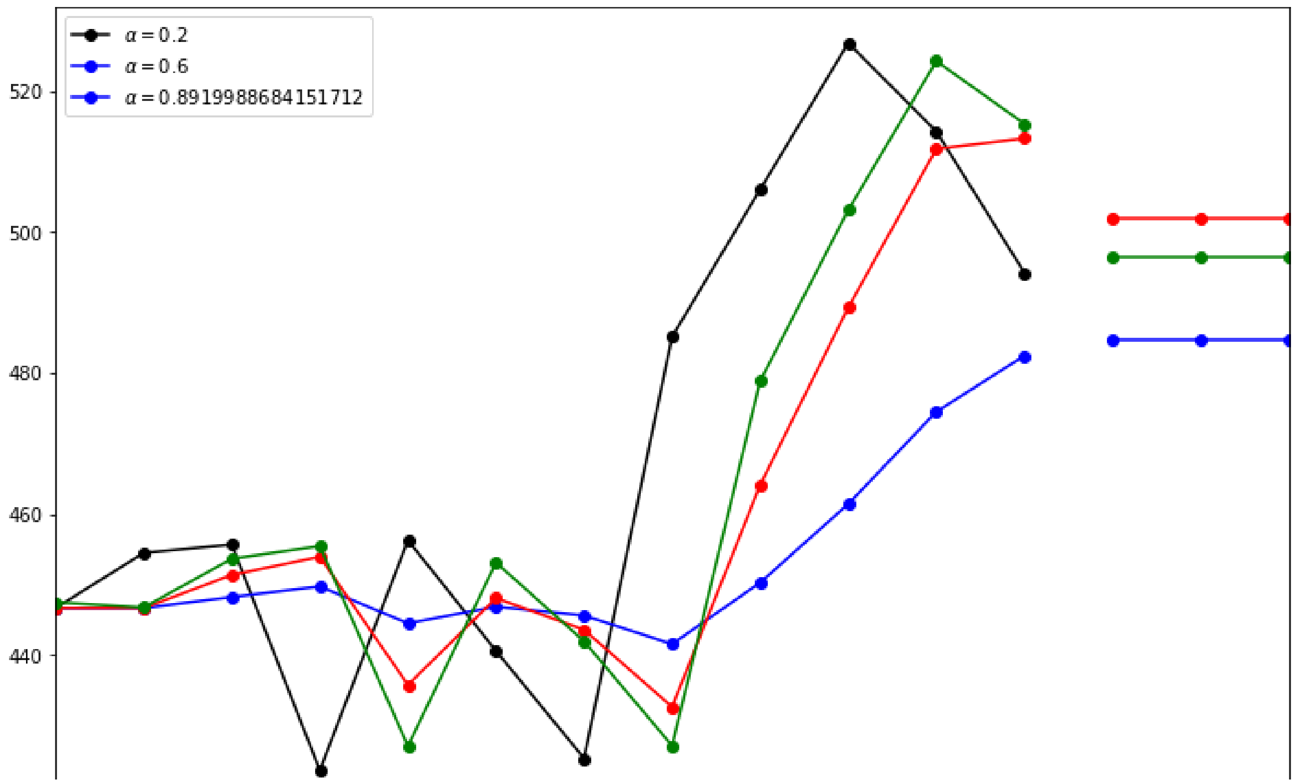
Here we run three variants of simple exponential smoothing:

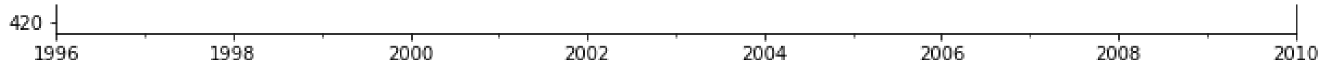
1. **In fit1 we do not use the auto optimization but instead choose to explicitly provide the model with the *α*=0.2 parameter**
2. **In fit2 as above we choose an *α*=0.6**
3. **In fit3 we allow statsmodels to automatically find an optimized *α* value for us. This is the recommended approach.**

In [32]:

```
fit1 = SimpleExpSmoothing(oildata).fit(smoothing_level=0.2,optimized=False)
fcast1 = fit1.forecast(3).rename(r'$\alpha=0.2$')
fit2 = SimpleExpSmoothing(oildata).fit(smoothing_level=0.6,optimized=False)
fcast2 = fit2.forecast(3).rename(r'$\alpha=0.6$')
fit3 = SimpleExpSmoothing(oildata).fit()
fcast3 = fit3.forecast(3).rename(r'$\alpha=%s$'%fit3.model.params['smoothing_level'])

ax = oildata.plot(marker='o', color='black', figsize=(12,8))
fcast1.plot(marker='o', ax=ax, color='blue', legend=True)
fit1.fittedvalues.plot(marker='o', ax=ax, color='blue')
fcast2.plot(marker='o', ax=ax, color='red', legend=True)

fit2.fittedvalues.plot(marker='o', ax=ax, color='red')
fcast3.plot(marker='o', ax=ax, color='green', legend=True)
fit3.fittedvalues.plot(marker='o', ax=ax, color='green')
plt.show()
```

```
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:731: RuntimeWarning
: invalid value encountered in greater_equal
  loc = initial_p >= ub
```





Holt’s Method

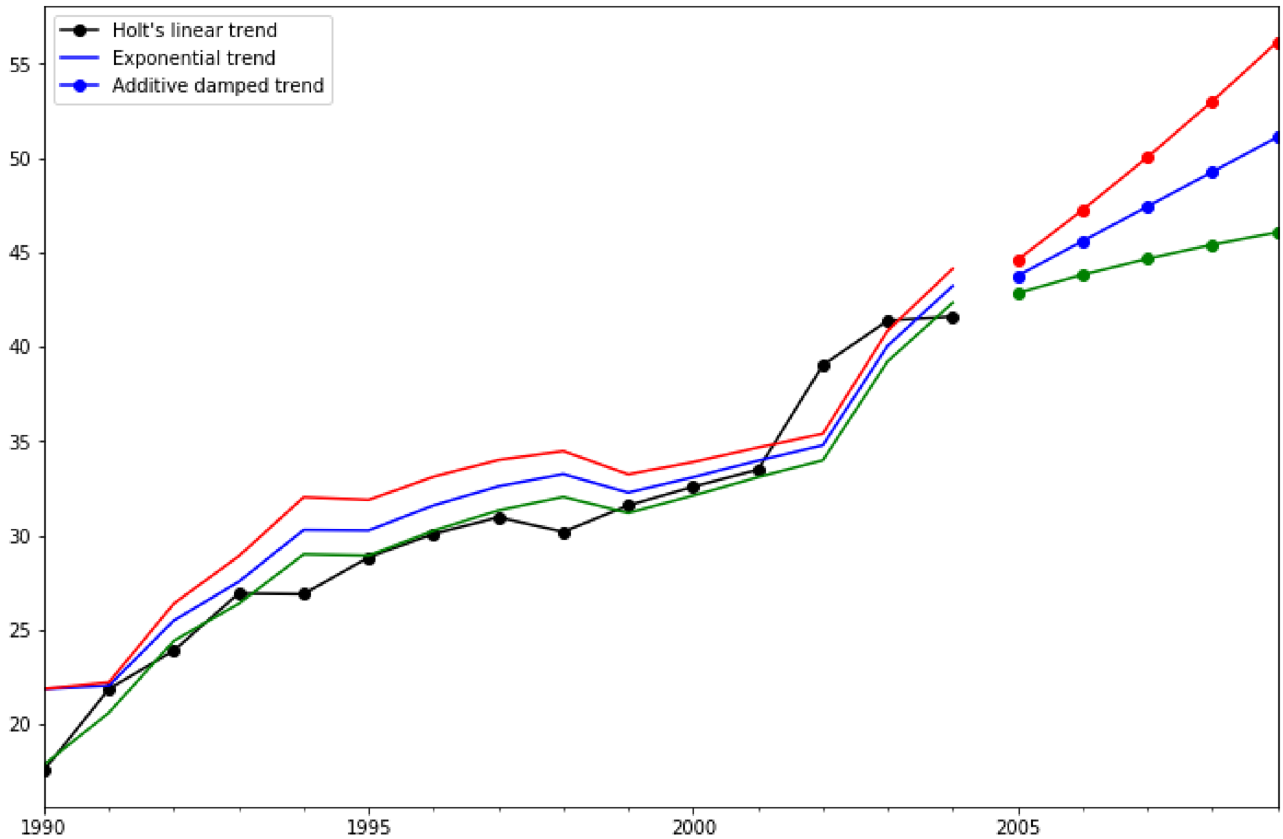
This time we use air pollution data and the Holt’s Method. We will fit three examples again.

1. In fit1 we again choose not to use the optimizer and provide explicit values for α=0.8 and β=0.2
2. In fit2 we do the same as in fit1 but choose to use an exponential model rather than a Holt’s additive model.
3. In fit3 we used a damped versions of the Holt’s additive model but allow the dampening parameter ϕ to be optimized while fixing the values for α=0.8 and β=0.2

In [33]:

```
fit1 = Holt(air).fit(smoothing_level=0.8, smoothing_slope=0.2, optimized=False)
fcast1 = fit1.forecast(5).rename("Holt's linear trend")
fit2 = Holt(air, exponential=True).fit(smoothing_level=0.8, smoothing_slope=0.2, optimize
d=False)
fcast2 = fit2.forecast(5).rename("Exponential trend")
fit3 = Holt(air, damped=True).fit(smoothing_level=0.8, smoothing_slope=0.2)
fcast3 = fit3.forecast(5).rename("Additive damped trend")

ax = air.plot(color="black", marker="o", figsize=(12,8))
fit1.fittedvalues.plot(ax=ax, color='blue')
fcast1.plot(ax=ax, color='blue', marker="o", legend=True)
fit2.fittedvalues.plot(ax=ax, color='red')
fcast2.plot(ax=ax, color='red', marker="o", legend=True)
fit3.fittedvalues.plot(ax=ax, color='green')
fcast3.plot(ax=ax, color='green', marker="o", legend=True)

plt.show()
```

```
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:731: RuntimeWarning
: invalid value encountered in greater_equal
  loc = initial_p >= ub
```



Seasonally adjusted data

Lets look at some seasonally adjusted livestock data. We fit five Holt's models. The below table allows us to compare results when we use exponential versus additive and damped versus non-damped.

Note: fit4 does not allow the parameter ϕ to be optimized by providing a fixed value of ϕ=0.98

In [34]:

```
fit1 = SimpleExpSmoothing(livestock2).fit()
fit2 = Holt(livestock2).fit()
fit3 = Holt(livestock2,exponential=True).fit()
fit4 = Holt(livestock2,damped=True).fit(damping_slope=0.98)
fit5 = Holt(livestock2,exponential=True,damped=True).fit()
params = ['smoothing_level', 'smoothing_slope', 'damping_slope', 'initial_level', 'initi
al_slope']
results=pd.DataFrame(index=[r"$\alpha$",r"$\beta$",r"$\phi$",r"$l_0$","$b_0$","SSE"] ,co
lumns=['SES', "Holt's","Exponential", "Additive", "Multiplicative"])
results["SES"] =             [fit1.params[p] for p in params] + [fit1.sse]
results["Holt's"]  =  [fit2.params[p]  for  p  in  params]  +  [fit2.sse]
results["Exponential"]  = [fit3.params[p] for  p  in  params]  +  [fit3.sse]
results["Additive"]  =  [fit4.params[p]  for  p  in  params]  +  [fit4.sse]
results["Multiplicative"] = [fit5.params[p] for p in params] + [fit5.sse]
results
```

```
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:731: RuntimeWarning
: invalid value encountered in greater_equal
  loc = initial_p >= ub
```

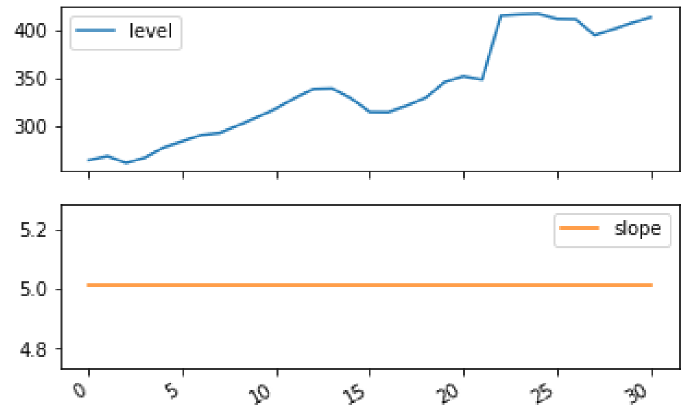
Out[34]:

	SES	Holt's	Exponential	Additive	Multiplicative
α	1.000000	0.974306	0.977634	0.978826	0.974909
β	NaN	0.000000	0.000000	0.000000	0.000000
ϕ	NaN	NaN	NaN	0.980000	0.981647
l_0	263.918414	258.882635	260.341624	257.355245	258.951921
b_0	NaN	5.010775	1.013780	6.644290	1.038144
SSE	6761.350218	6004.138200	6104.194747	6036.555016	6081.995045

Plots of Seasonally Adjusted Data

In [35]:

```
for fit in [fit2,fit4]:
    pd.DataFrame(np.c_[fit.level,fit.slope]).rename(
        columns={0:'level',1:'slope'}).plot(subplots=True)
plt.show()
print('Figure 7.4: Level and slope components for Holt's linear trend method and the addi
tive damped trend method.')
```



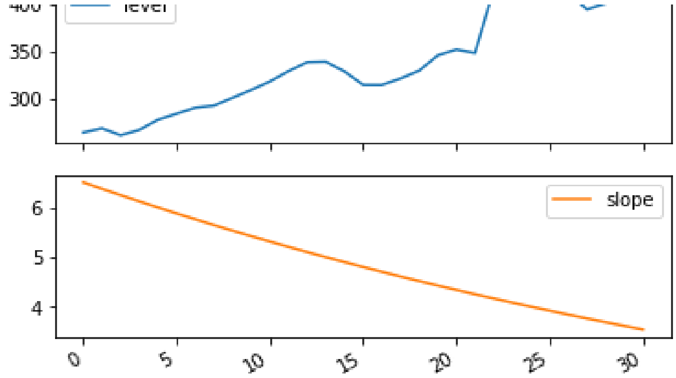


Figure 7.4: Level and slope components for Holt’s linear trend method and the additive da
mped trend method.

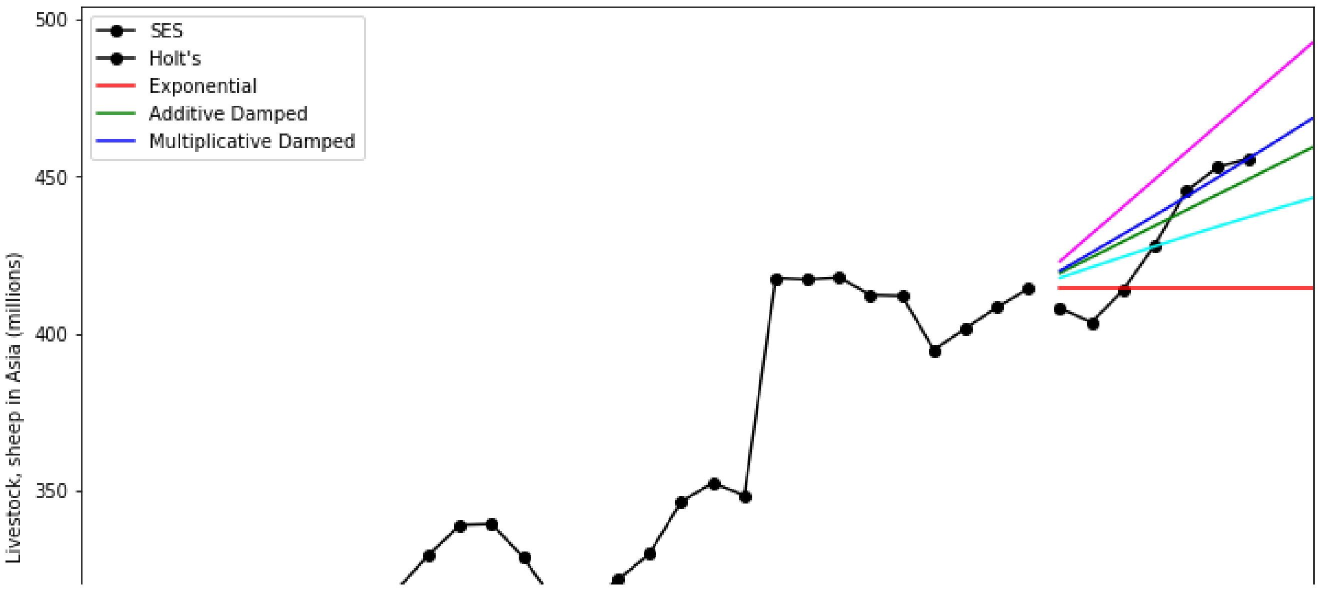
Comparison

Here we plot a comparison Simple Exponential Smoothing and Holt’s Methods for various additive, exponential and damped combinations. All of the models parameters will be optimized by statsmodels.

In [36]:

```
fit1 = SimpleExpSmoothing(livestock2).fit()
fcast1 = fit1.forecast(9).rename("SES")
fit2 = Holt(livestock2).fit()
fcast2  =  fit2.forecast(9).rename("Holt's")
fit3 = Holt(livestock2, exponential=True).fit()
fcast3 = fit3.forecast(9).rename("Exponential")
fit4 = Holt(livestock2, damped=True).fit(damping_slope=0.98)
fcast4 = fit4.forecast(9).rename("Additive Damped")
fit5 = Holt(livestock2, exponential=True, damped=True).fit()
fcast5 = fit5.forecast(9).rename("Multiplicative Damped")

ax = livestock2.plot(color="black", marker="o", figsize=(12,8))
livestock3.plot(ax=ax, color="black", marker="o", legend=False)
fcast1.plot(ax=ax, color='red', legend=True)
fcast2.plot(ax=ax, color='green', legend=True)
fcast3.plot(ax=ax, color='blue', legend=True)
fcast4.plot(ax=ax, color='cyan', legend=True)
fcast5.plot(ax=ax, color='magenta', legend=True)
ax.set_ylabel('Livestock, sheep in Asia (millions)')
plt.show()
print('Figure 7.5: Forecasting livestock, sheep in Asia: comparing forecasting performanc
e of non-seasonal methods.')
```

```
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:731: RuntimeWarning
: invalid value encountered in greater_equal
  loc = initial_p >= ub
```



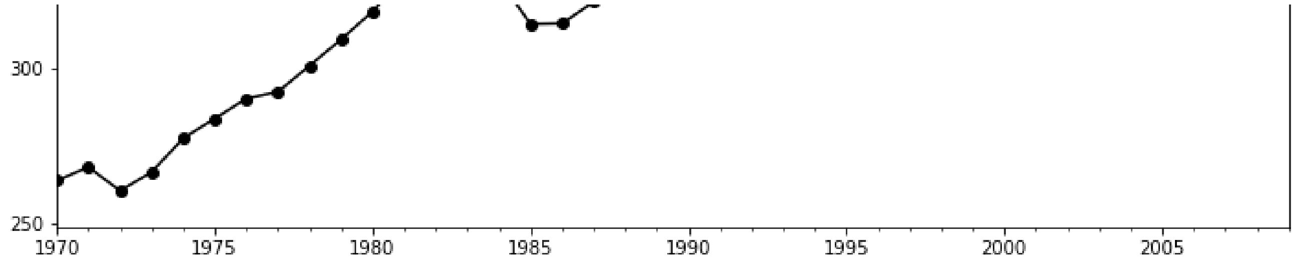


Figure 7.5: Forecasting livestock, sheep in Asia: comparing forecasting performance of non-seasonal methods.

Holt's Winters Seasonal

Finally we are able to run full Holt's Winters Seasonal Exponential Smoothing including a trend component and a seasonal component. statsmodels allows for all the combinations including as shown in the examples below:

1. **fit1 additive trend, additive seasonal of period season_length=4 and the use of a Box-Cox transformation.**
2. **fit2 additive trend, multiplicative seasonal of period season_length=4 and the use of a Box-Cox transformation..**
3. **fit3 additive damped trend, additive seasonal of period season_length=4 and the use of a Box-Cox transformation.**
4. **fit4 additive damped trend, multiplicative seasonal of period season_length=4 and the use of a Box-Cox transformation.**

The plot shows the results and forecast for fit1 and fit2. The table allows us to compare the results and parameterizations.

In [37]:

```
fit1 = ExponentialSmoothing(aust, seasonal_periods=4, trend='add', seasonal='add').fit(use_boxcox=True)
fit2 = ExponentialSmoothing(aust, seasonal_periods=4, trend='add', seasonal='mul').fit(use_boxcox=True)
fit3 = ExponentialSmoothing(aust, seasonal_periods=4, trend='add', seasonal='add', damped=True).fit(use_boxcox=True)
fit4 = ExponentialSmoothing(aust, seasonal_periods=4, trend='add', seasonal='mul', damped=True).fit(use_boxcox=True)
results=pd.DataFrame(index=[r"$\alpha$",r"$\beta$",r"$\phi$",r"$\gamma$",r"$l_0$","$b_0$","SSE"])
params = ['smoothing_level', 'smoothing_slope', 'damping_slope', 'smoothing_seasonal', 'initial_level', 'initial_slope']
results["Additive"]  =  [fit1.params[p]  for  p  in  params]  +  [fit1.sse]
results["Multiplicative"] = [fit2.params[p] for p in params] + [fit2.sse]
results["Additive  Dam"]  =  [fit3.params[p]  for  p  in  params]  +  [fit3.sse]
results["Multiplica Dam"] = [fit4.params[p] for p in params] + [fit4.sse]

ax = aust.plot(figsize=(10,6), marker='o', color='black', title="Forecasts from Holt-Winters' multiplicative method" )
ax.set_ylabel("International visitor night in Australia (millions)")
ax.set_xlabel("Year")
fit1.fittedvalues.plot(ax=ax, style='--', color='red')
fit2.fittedvalues.plot(ax=ax, style='--', color='green')

fit1.forecast(8).rename('Holt-Winters (add-add-seasonal)').plot(ax=ax, style='--', marker='o', color='red', legend=True)
fit2.forecast(8).rename('Holt-Winters (add-mul-seasonal)').plot(ax=ax, style='--', marker='o', color='green', legend=True)

plt.show()
print("Figure 7.6: Forecasting international visitor nights in Australia using Holt-Winters method with both additive and multiplicative seasonality.")

results
```

```
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:725: RuntimeWarning
```

```
: invalid value encountered in less_equal
  loc = initial_p <= lb
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:731: RuntimeWarning
: invalid value encountered in greater_equal
  loc = initial_p >= ub
C:\Users\P\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:744: ConvergenceWar
ning: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
```

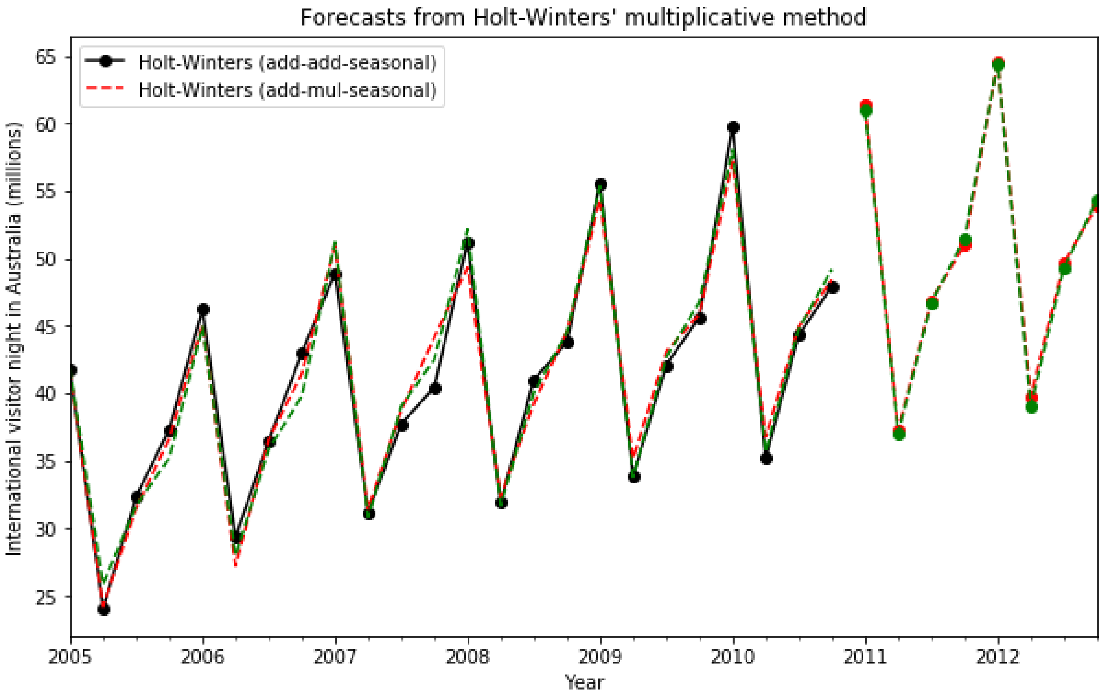


Figure 7.6: Forecasting international visitor nights in Australia using Holt-Winters method with both additive and multiplicative seasonality.

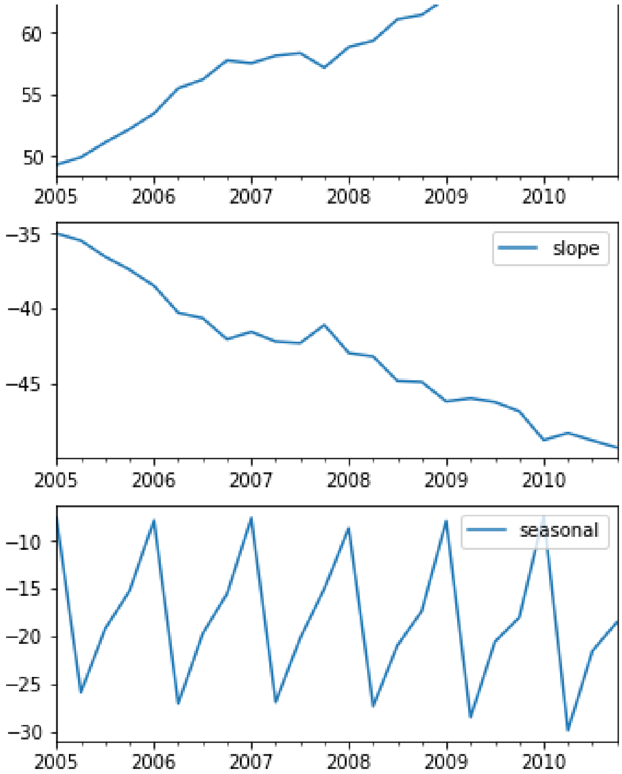
Out[37]:

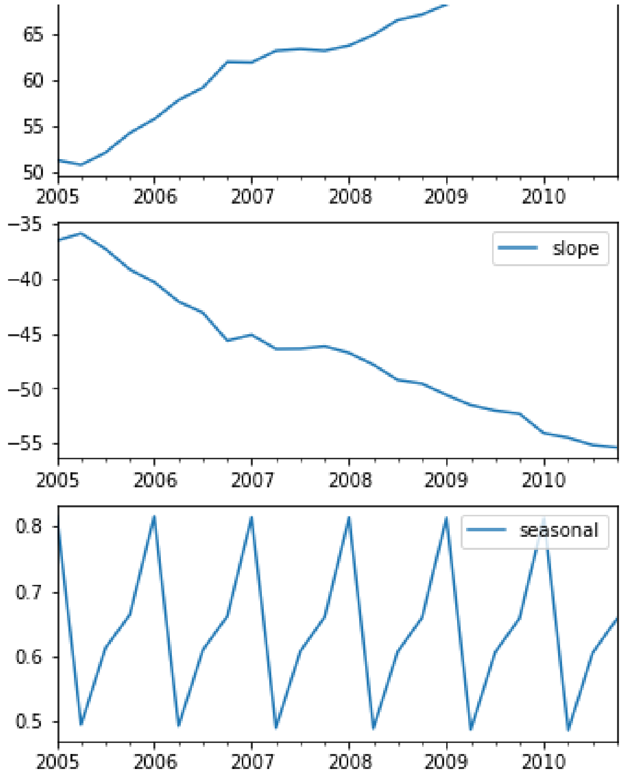
	Additive	Multiplicative	Additive Dam	Multiplica Dam
α	4.546647e-01	3.664383e-01	8.739510e-09	0.000188
β	1.558825e-08	4.118538e-24	8.207203e-70	0.000188
ϕ	NaN	NaN	9.428597e-01	0.913530
γ	5.243636e-01	5.957200e-18	2.306364e-07	0.000000
l_0	1.421755e+01	1.454998e+01	1.415675e+01	14.534974
b_0	1.307571e-01	1.661634e-01	2.461681e-01	0.483888
SSE	5.001713e+01	4.307388e+01	3.528424e+01	39.678743

Finally lets look at the levels, slopes/trends and seasonal components of the models

In [38]:

```
states1 = pd.DataFrame(np.c_[fit1.level, fit1.slope, fit1.season], columns=['level','slope','seasonal'], index=aust.index)
states2 = pd.DataFrame(np.c_[fit2.level, fit2.slope, fit2.season], columns=['level','slope','seasonal'], index=aust.index)
fig, [[ax1, ax4],[ax2, ax5], [ax3, ax6]] = plt.subplots(3, 2, figsize=(12,8))
states1[['level']].plot(ax=ax1)
states1[['slope']].plot(ax=ax2)
states1[['seasonal']].plot(ax=ax3)
states2[['level']].plot(ax=ax4)
states2[['slope']].plot(ax=ax5)
states2[['seasonal']].plot(ax=ax6)
plt.show()
```





Nonlinear Trend Regression

NonlinearTrend Regression is a form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables.

Librarys

Importing Librarys

In [1]:

```
import numpy.polynomial.polynomial as poly
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
```

Data Preparation

In [5]:

```
df = pd.DataFrame([[i for i in range(2000,2018)],
[23.0,27.5,46.0,56.0,64.8,71.2,80.2,98.0,113.0,155.8,414.0,2297.8,3628.4,16187.8,25197.8
,42987.8,77555.5,130631.9]])
df = df.T
df.columns = ['Year', 'Values']
df['Year'] = df['Year'].astype(int)
df['Values'] = df['Values'].astype(int)
```

Forecasting for next 5 years

In [4]:

```
no_of_predictions = 5
X = np.array(df.Year, dtype = float)
```

```
y = np.array(df.Values, dtype = float)
Z = [2019,2020,2021,2022]
coefs = poly.polyfit(X, y, 4)
X_new = np.linspace(X[0], X[-1]+no_of_predictions, num=len(X)+no_of_predictions)
ffit = poly.polyval(X_new, coefs)
pred = poly.polyval(Z, coefs)
predictions = pd.DataFrame(Z,pred)
print (predictions)
plt.plot(X, y, 'ro', label="Original data")
plt.plot(X_new, ffit, label = "Fitted data")
plt.legend(loc='upper left')
plt.show()
```

```
                 0
271916.90625  2019
377429.68750  2020
510010.28125  2021
673797.15625  2022
```