Experiment 1

Perform exploratory data analysis and visualization after importing a .CSV file.

Handle missing data by detecting and dropping/ filling missing values.

Transform data using different methods. Detect and filter outliers.

Perform Vectorized String operations on Pandas Series.

Visualize data using Line Plots, Bar Plots, Histograms, Density Plots and Scatter Plots

Importing libraries and reading csv file

```
import pandas as pd
from pandas import DataFrame,Series
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from scipy import stats
from scipy.stats import skew
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
ds=pd.read_csv("PlacementinCampusRecruitment.csv")
ds.head()
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No

Handling missing data with forward fill method

```
ds.isnull().sum()
```

```
sl_no             0
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
hsc_b             0
hsc_s             0
degree_p          0
degree_t          0
workex            0
etest_p           0
specialisation    0
mba_p             0
status            0
salary           67
dtype: int64
```

```
update_ds=ds.fillna(method="ffill")
update_ds
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	worke
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	N
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Ye
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	N
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	N
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	N
...	...	...	...	...	...	...	...	...	...	
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	N
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	N
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Ye
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	N
214	215	M	62.00	Central	58.00	Others	Science	53.00	Comm&Mgmt	N

```
update_ds.isnull().sum()
```

```
sl_no             0
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
hsc_b             0
hsc_s             0
```

```
degree_p          0
degree_t          0
workex            0
etest_p           0
specialisation    0
mba_p             0
status            0
salary            0
dtype: int64
```

Replacing NA values with mean vale

```
mean_ds=ds.interpolate()
mean_ds
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	worke
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	N
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Ye
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	N
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	N
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	N
...	...	...	...	...	...	...	...	...	...	
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	N
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	N
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Ye
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	N
214	215	M	62.00	Central	58.00	Others	Science	53.00	Comm&Mgmt	N

```
mean_ds.isnull().sum()
```

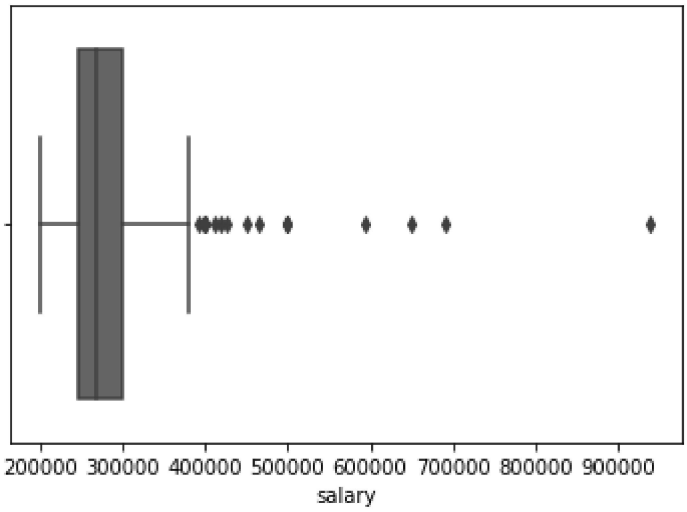
```
sl_no             0
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
hsc_b             0
hsc_s             0
degree_p          0
degree_t          0
workex            0
etest_p           0
```

```
specialisation    0
mba_p             0
status            0
salary            0
dtype: int64
```

Detecting and removing outliers

```
sns.boxplot(mean_ds['salary'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2dd567850>
```

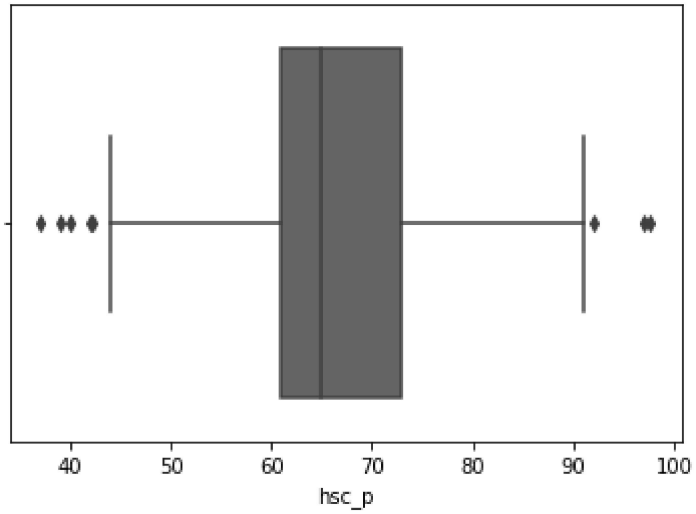



```
print(np.where(mean_ds['salary']>370000))
```

```
(array([  4,  21,  39,  53,  77,  85,  95, 101, 119, 120, 128, 145, 149,
       150, 163, 173, 174, 177, 210]),)
```

```
sns.boxplot(mean_ds['hsc_p'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2dd4400d0>
```



```
print(np.where(mean_ds['hsc_p']>91))
```

```
(array([ 24, 134, 177]),)
```

```
print(np.where(mean_ds['hsc_p']<42.1))
```

```
(array([ 42,  49, 120, 206]),)
```

```
new_ds=mean_ds[(mean_ds['hsc_p']>43) & (mean_ds['hsc_p']<91)]
new_ds
```

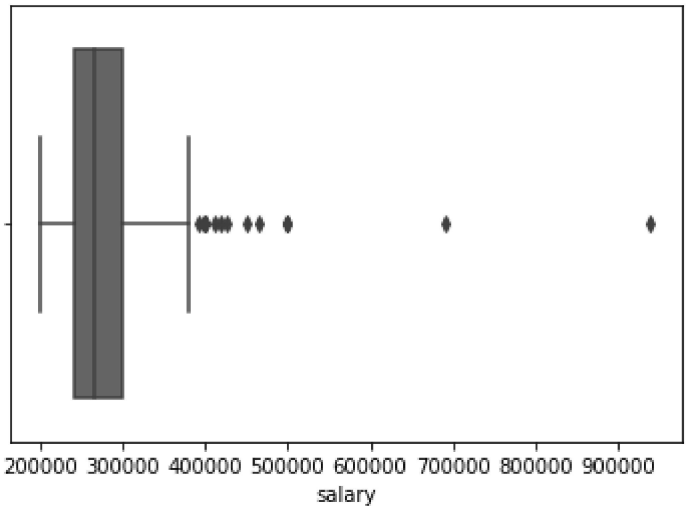
	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	worke
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Ye
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	N
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	N
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	N
5	6	M	55.00	Others	49.80	Others	Science	67.25	Sci&Tech	Ye
...	...	...	...	...	...	...	...	...	...	
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	N
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	N
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Ye
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	N

```
sns.boxplot(new_ds['hsc_p'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2dcfdeaf0>
```

```
sns.boxplot(new_ds['salary'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2dcf32fa0>
```



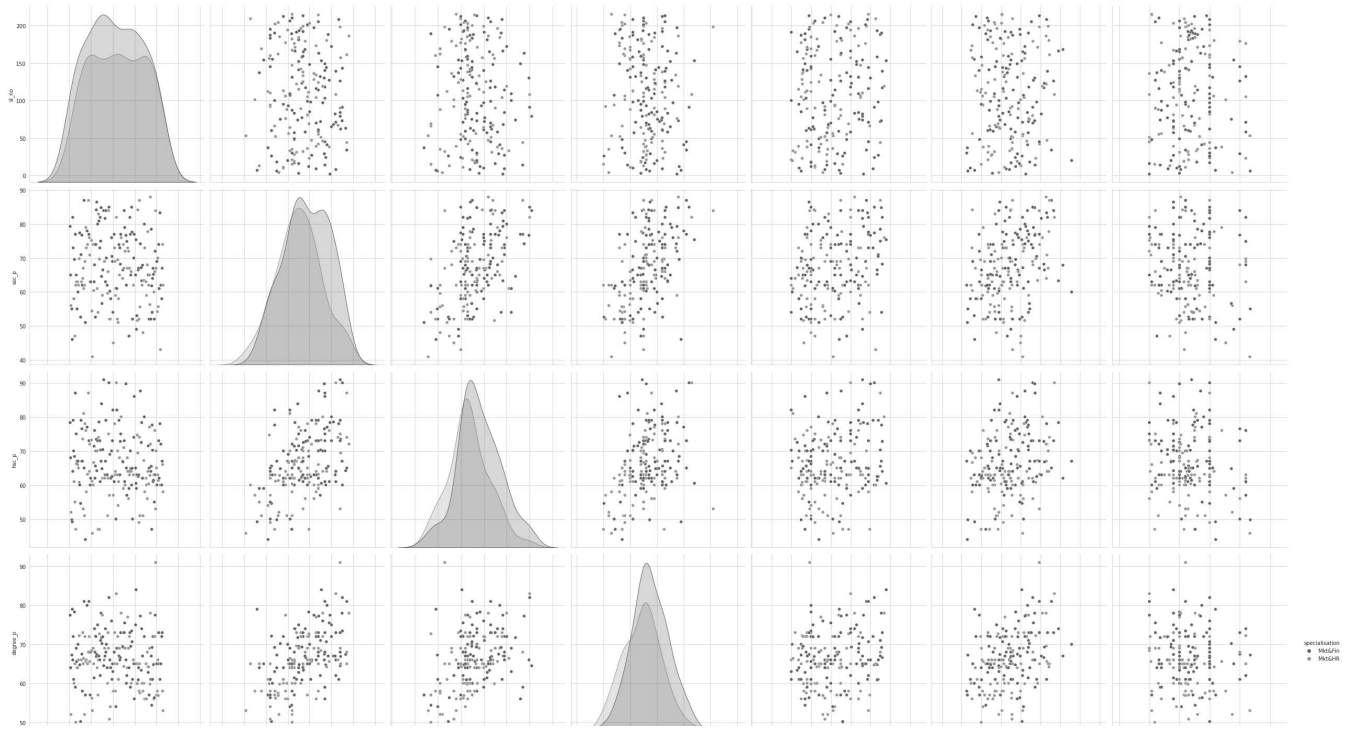

```
wo_outliers_ds=new_ds[(new_ds['salary']<380000)]
wo_outliers_ds
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	worke
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Ye
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	N
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	N
5	6	M	55.00	Others	49.80	Others	Science	67.25	Sci&Tech	Ye
6	7	F	46.00	Others	49.20	Others	Commerce	79.00	Comm&Mgmt	N
...	...	...	...	...	...	...	...	...	...	
209	210	M	62.00	Central	72.00	Central	Commerce	65.00	Comm&Mgmt	N
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	N
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Ye
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	N

```
sns.boxplot(wo_outliers_ds['salary'])
```

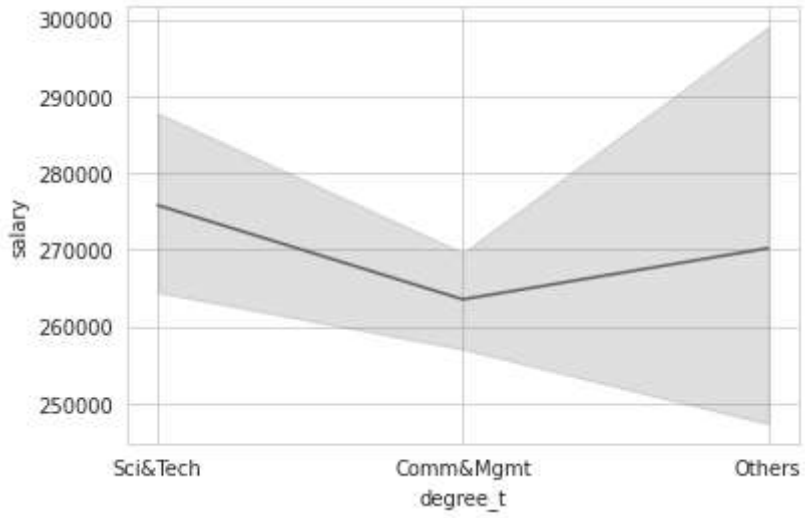
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2dcf136d0>
```

```
plt.close();
sns.set_style("whitegrid");
sns.pairplot(wo_outliers_ds, hue="specialisation", size=5);
plt.show()
```



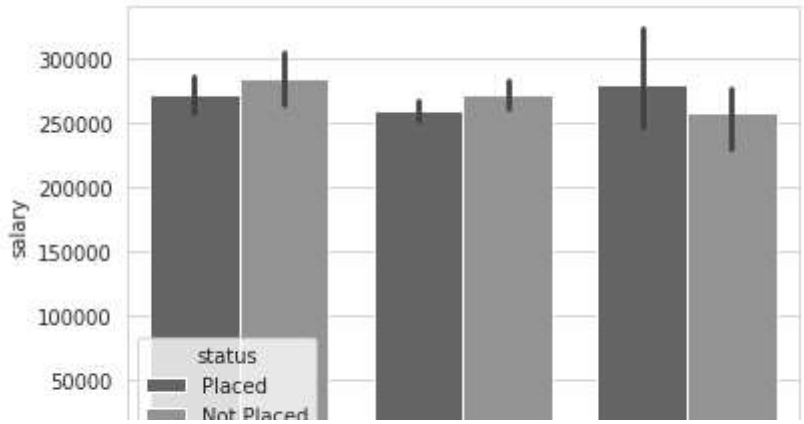
Line plot

```
sns.lineplot(x="degree_t", y="salary", data=wo_outliers_ds)
plt.show()
```



Bar plot

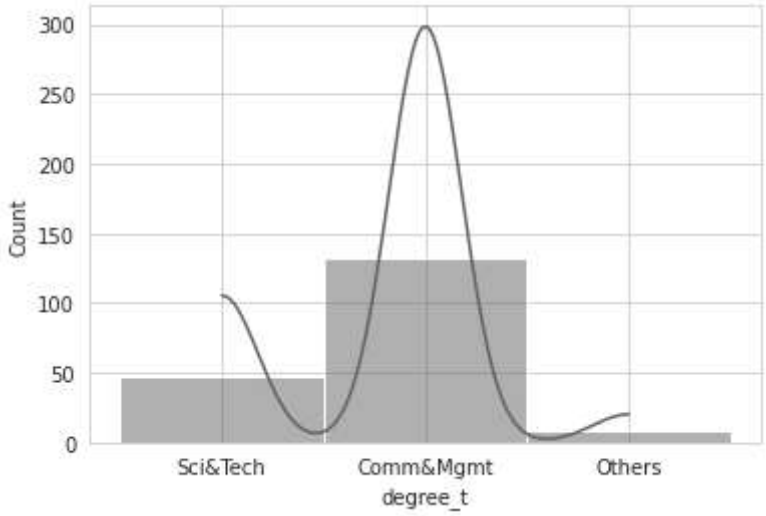
```
sns.barplot(x='degree_t', y='salary',hue='status', data=wo_outliers_ds)
plt.show()
```



Histogram

```
sns.histplot(data=wo_outliers_ds, x="degree_t", kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2d5b8aac0>
```



Density plot

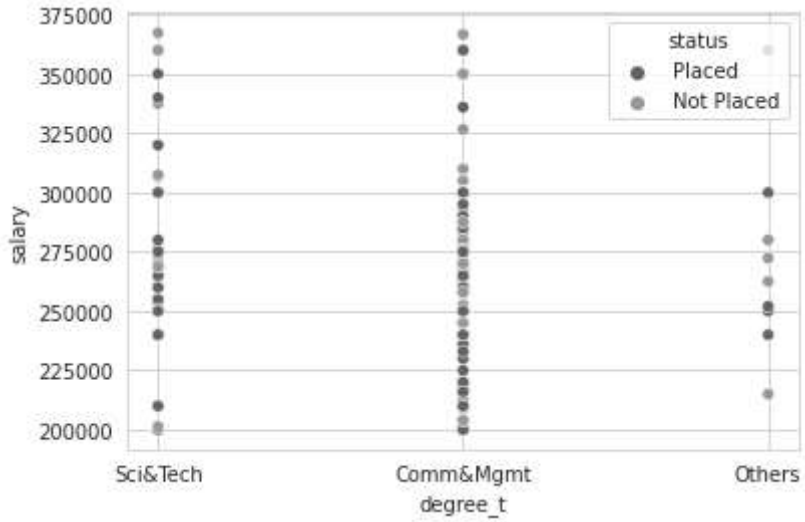
```
sns.kdeplot(wo_outliers_ds['salary'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2d5ae5250>
```

Scatter plot

```
sns.scatterplot(x="degree_t", y="salary", hue="status", data=wo_outliers_ds)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2d5ac4340>
```



Experiment 2

Recognise data Skew-ness,outliers both using statistical function and graphical representation

Skewness

```
wo_outliers_ds.skew()
```

```
sl_no       0.025594
ssc_p      -0.069260
hsc_p       0.256883
degree_p    0.229707
etest_p     0.318875
mba_p       0.277188
salary      0.513317
dtype: float64
```

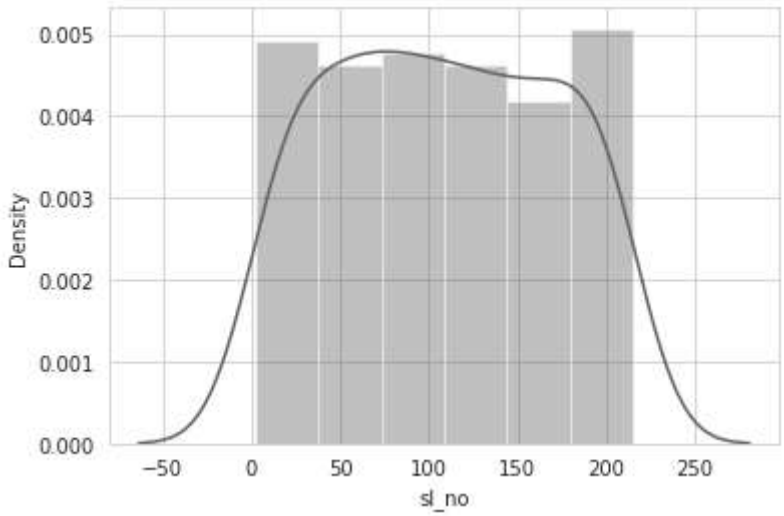
```
ds_num=wo_outliers_ds.select_dtypes(include=np.number)
ds_num
```

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
1	2	79.33	78.33	77.48	86.50	66.28	200000.000000
2	3	65.00	68.00	64.00	75.00	57.80	250000.000000
3	4	56.00	52.00	52.00	66.00	59.43	337500.000000
5	6	55.00	49.80	67.25	55.00	51.58	367333.333333
6	7	46.00	49.20	79.00	74.28	53.29	309666.666667
...	...	...	...	...	...	...	...
209	210	62.00	72.00	65.00	67.00	56.49	216000.000000
211	212	58.00	60.00	72.00	74.00	53.62	275000.000000
212	213	67.00	67.00	73.00	59.00	69.72	295000.000000
213	214	74.00	66.00	58.00	70.00	60.23	204000.000000

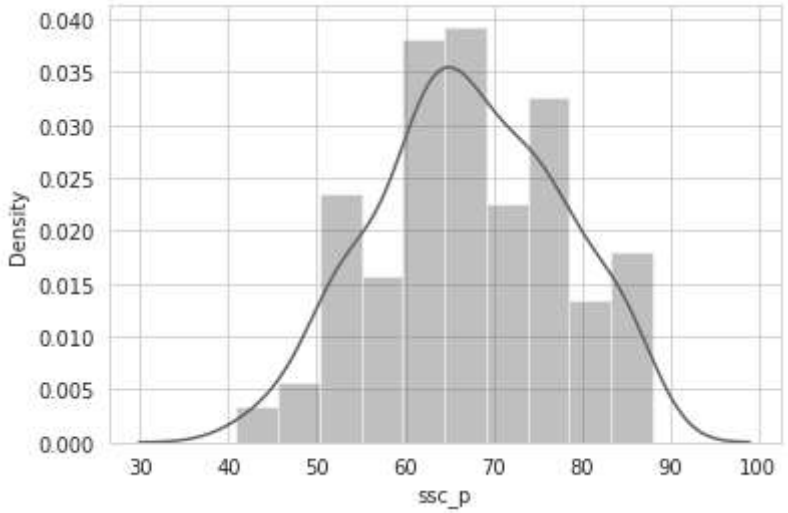
```
for col in ds_num:
  print(col)
  print(skew(ds_num[col]))

  plt.figure()
  sns.distplot(ds_num[col])
  plt.show()
```

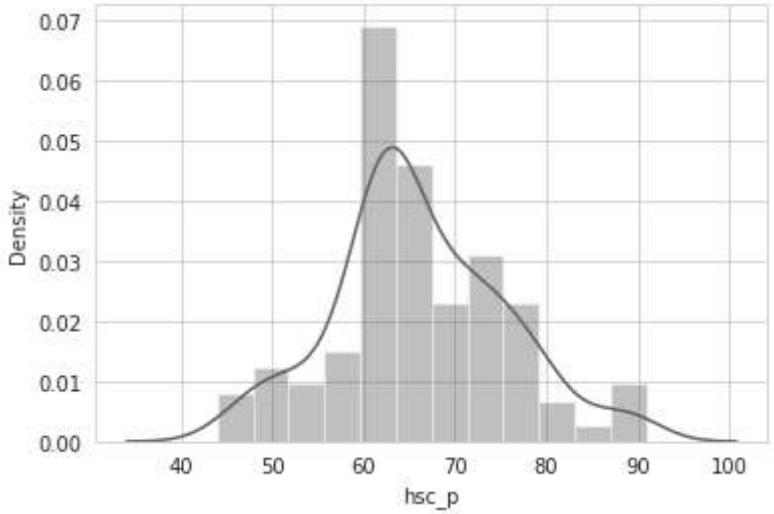
sl_no
0.025390889125873953



ssc_p
-0.06870890555017124



hsc_p
0.25484004999213616



degree_p
0.2278798265738934

Experiment 3

Write a python program to implement Simple Linear Regression to predict if male or female based on Height.

```
df=pd.read_csv('/content/weight-height (1).csv')
df.head()
```

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

```
df.tail()
```

	Gender	Height	Weight
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

```
df.info()
```

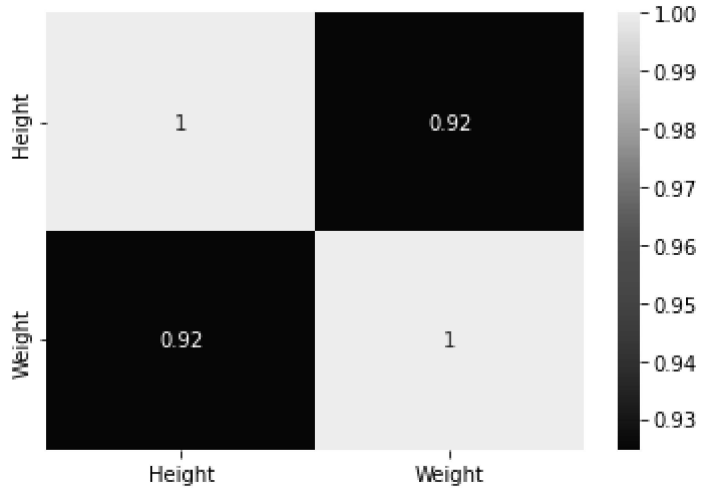
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Gender  10000 non-null  object 
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), object(1)
memory usage: 234.5+ KB
```

```
df.describe()
```

	Height	Weight
count	10000.000000	10000.000000
mean	66.367560	161.440357
std	3.847528	32.108439
min	54.263133	64.700127
25%	63.505620	135.818051
50%	66.318070	161.212928
75%	69.174262	187.169525
max	78.998742	269.989699

```
sns.heatmap(df.corr(),annot=True)
```

```
<Axes: >
```




```
df['Gender'].replace(to_replace={'Male':1,'Female':0})
```

```
0       1
1       1
2       1
3       1
4       1
       ..
9995    0
9996    0
9997    0
9998    0
9999    0
Name: Gender, Length: 10000, dtype: int64
```

```
X=df[['Height','Weight']]
y=df[['Gender']]
```

```
X_train , X_test , y_train ,y_test= train_test_split(X,y, test_size=0.3,random_state=101)
```

```
lr=LogisticRegression()
lr.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
lr.score(X_train,y_train)
```

```
0.9222857142857143
```

```
predict=lr.predict(X_test)
```

```
#accuracy of the model
from sklearn import metrics
print(metrics.accuracy_score(y_test,predict))
print(metrics.confusion_matrix(y_test,predict))
print(metrics.classification_report(y_test,predict))
```

```
0.912
[[1370  126]
 [ 138 1366]]
              precision    recall  f1-score   support

      Female       0.91      0.92      0.91      1496
        Male       0.92      0.91      0.91      1504

    accuracy                           0.91      3000
   macro avg       0.91      0.91      0.91      3000
weighted avg       0.91      0.91      0.91      3000
```

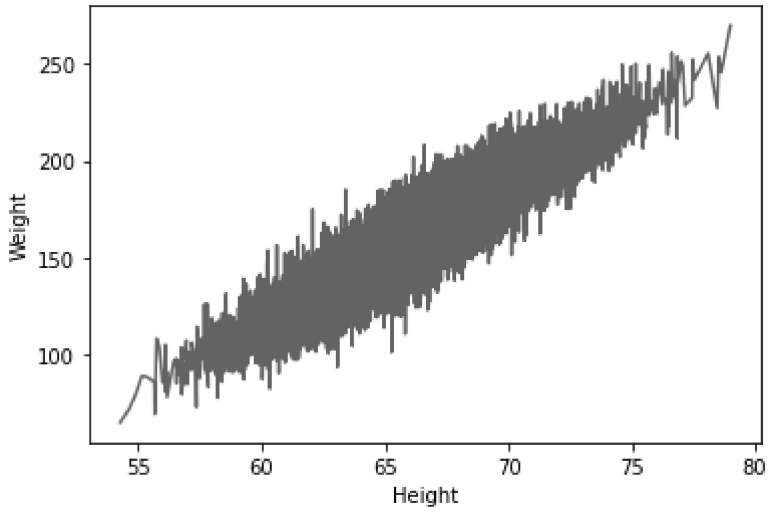
```
result=lr.predict([(65.456,200.5748)])
result
```

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

```
result=lr.predict([(55.456,100.5748)])
result
```

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

```
sns.lineplot(x="Height", y="Weight", data=df)
plt.show()
```



Experiment 4

Implement Various Regression algorithm for House Price Prediction (USA housing Dataset) and compare there accuracy using scikitlearn

- Linear Regression
- Polynomial Regression
- Support Vector machine

Linear regression

```
price_pred=pd.read_csv('USA_Housing.csv')
price_pred.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Ad
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferr 674\nLaurabu 3

```
price_pred.info()
```

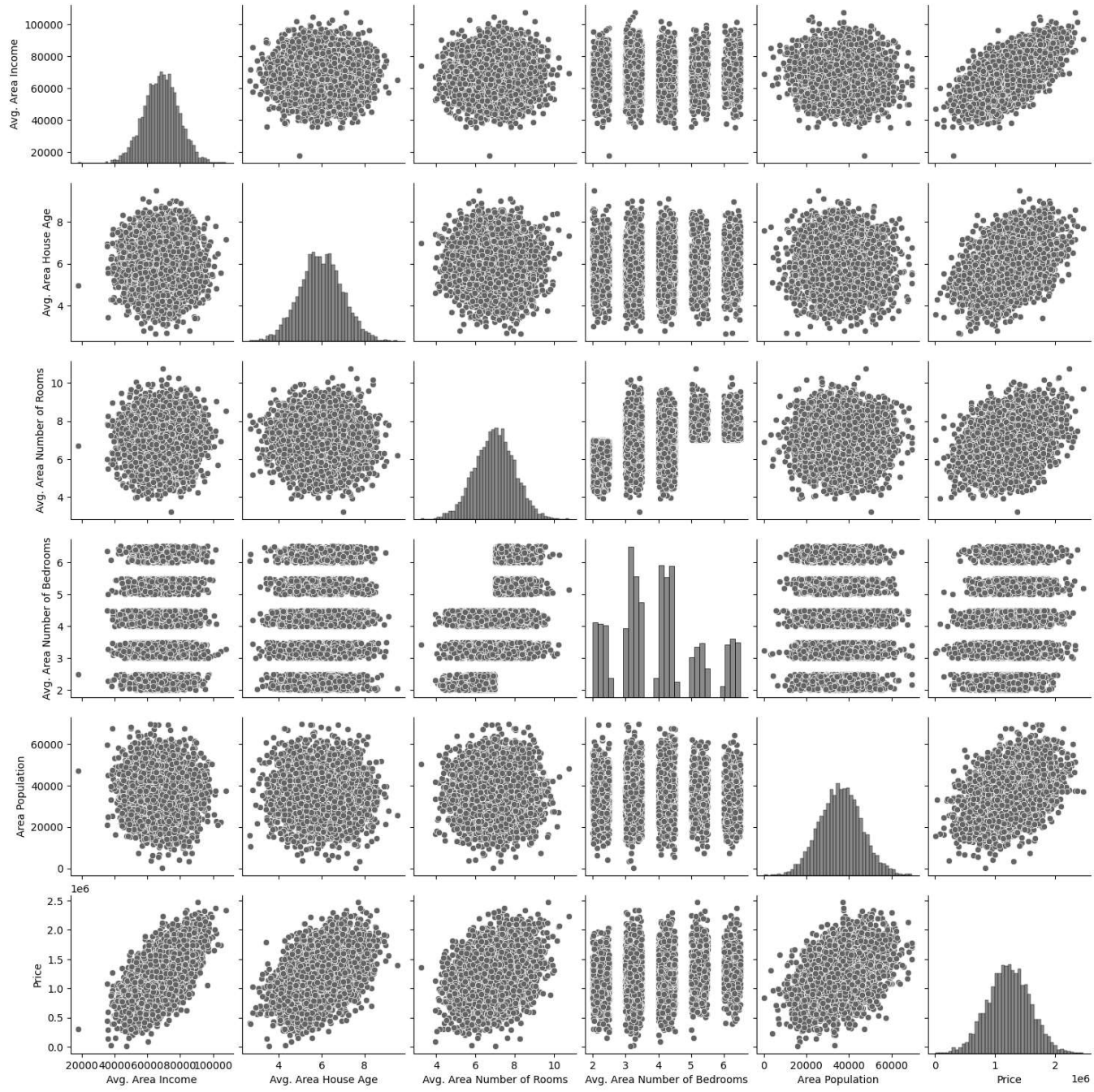
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

```
price_pred.describe(percentiles=[0.1,0.25,0.5,0.75,0.9])
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
10%	55047.633980	4.697755	5.681951	2.310000	23502.845262	7.720318e+05
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
90%	82081.188283	7.243978	8.274222	6.100000	48813.618633	1.684621e+06

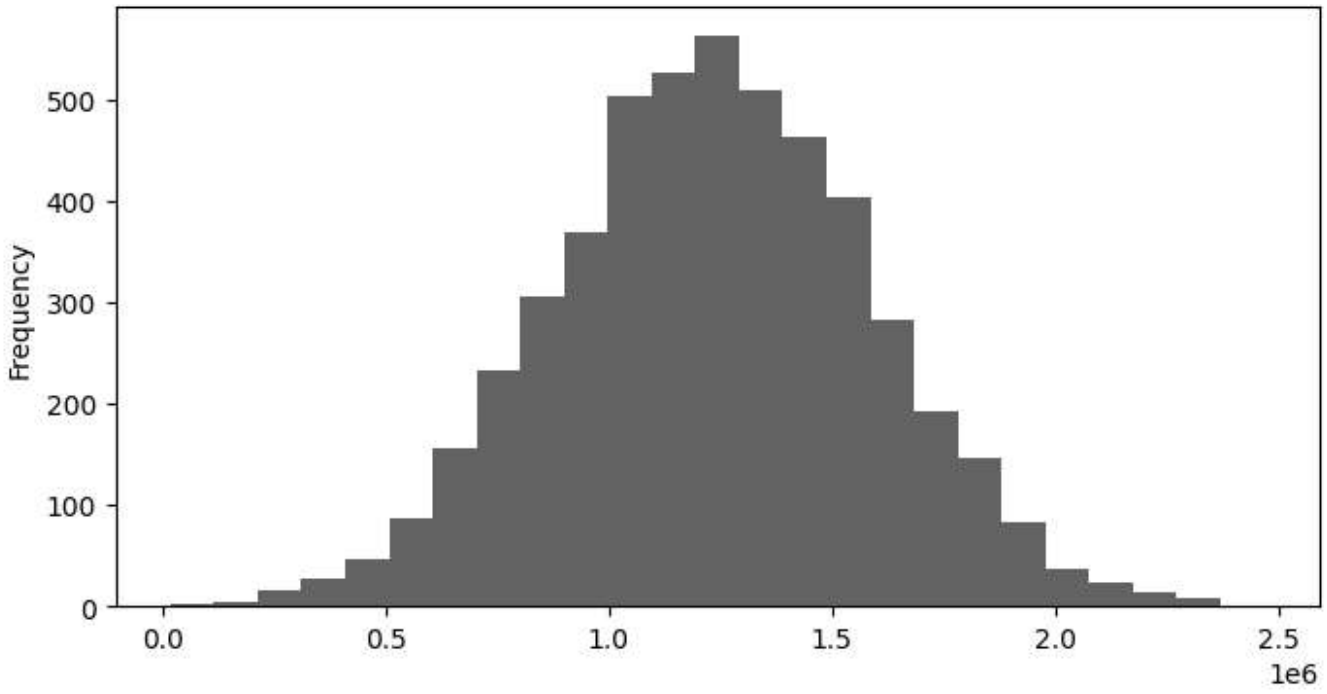
```
sns.pairplot(price_pred)
```

<seaborn.axisgrid.PairGrid at 0x7efe4faea850>



```
price_pred['Price'].plot.hist(bins=25,figsize=(8,4))
```

```
<Axes: ylabel='Frequency'>
```

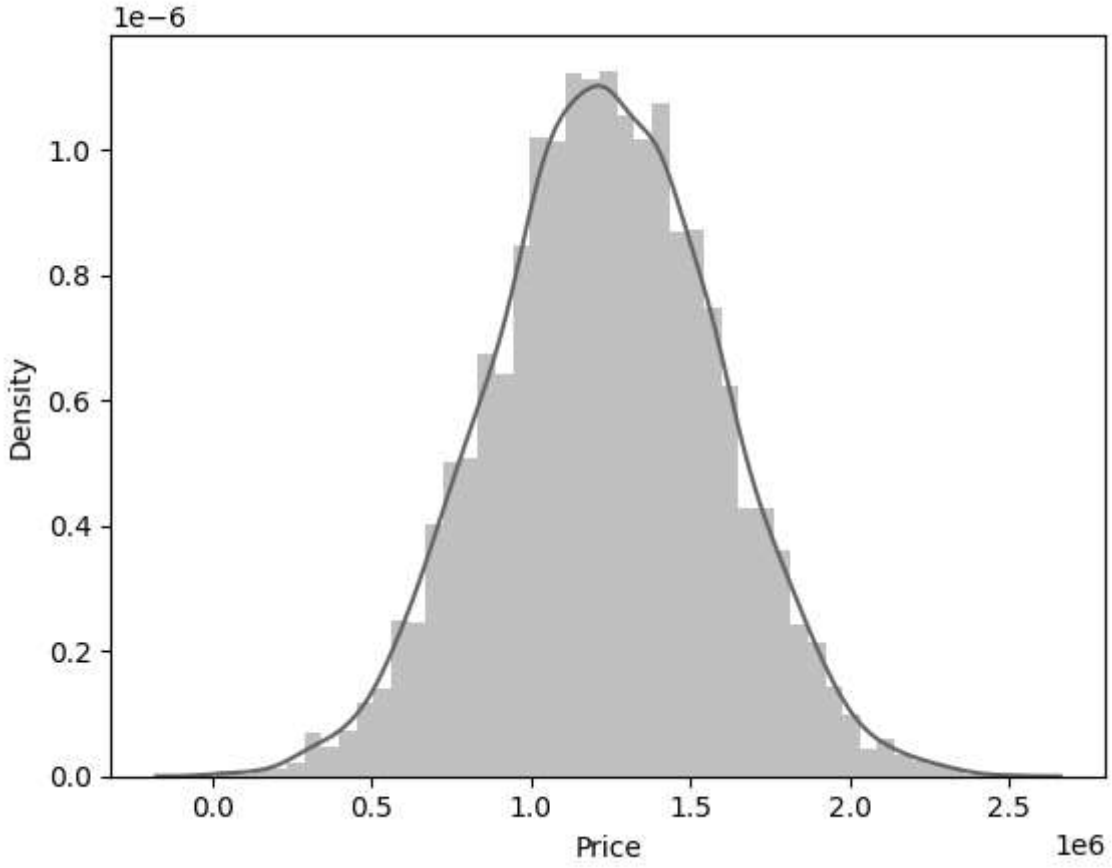



```
price_pred['Price'].plot.density()
```

```
<Axes: ylabel='Density'>
```

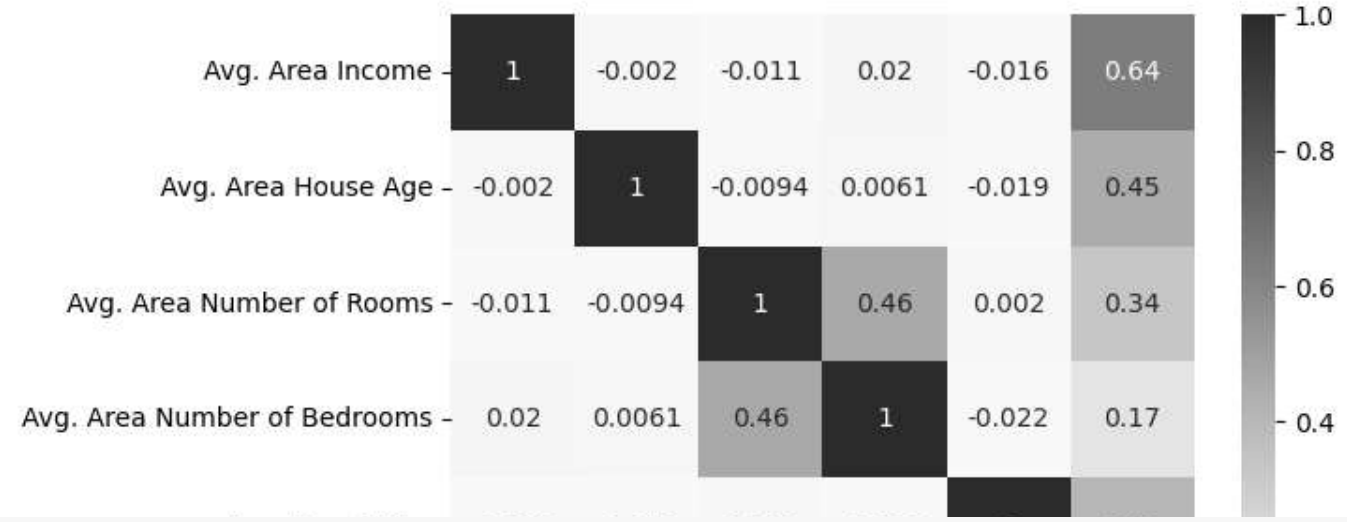
```
sns.distplot(price_pred['Price'])
```

```
<Axes: xlabel='Price', ylabel='Density'>
```




```
sns.heatmap(price_pred.corr(),annot=True,cmap='Greens')
```

```
<Axes: >
```




```
price_pred.columns
```

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

```
X=price_pred[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
```

```
y=price_pred['Price']
y
```

```
0       1.059034e+06
1       1.505891e+06
2       1.058988e+06
3       1.260617e+06
4       6.309435e+05
            ...
4995    1.060194e+06
4996    1.482618e+06
4997    1.030730e+06
4998    1.198657e+06
4999    1.298950e+06
Name: Price, Length: 5000, dtype: float64
```

```
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)
```

```
from sklearn.linear_model import LinearRegression
```

```
model= LinearRegression()
```

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

```
▾ LinearRegression
LinearRegression()
```

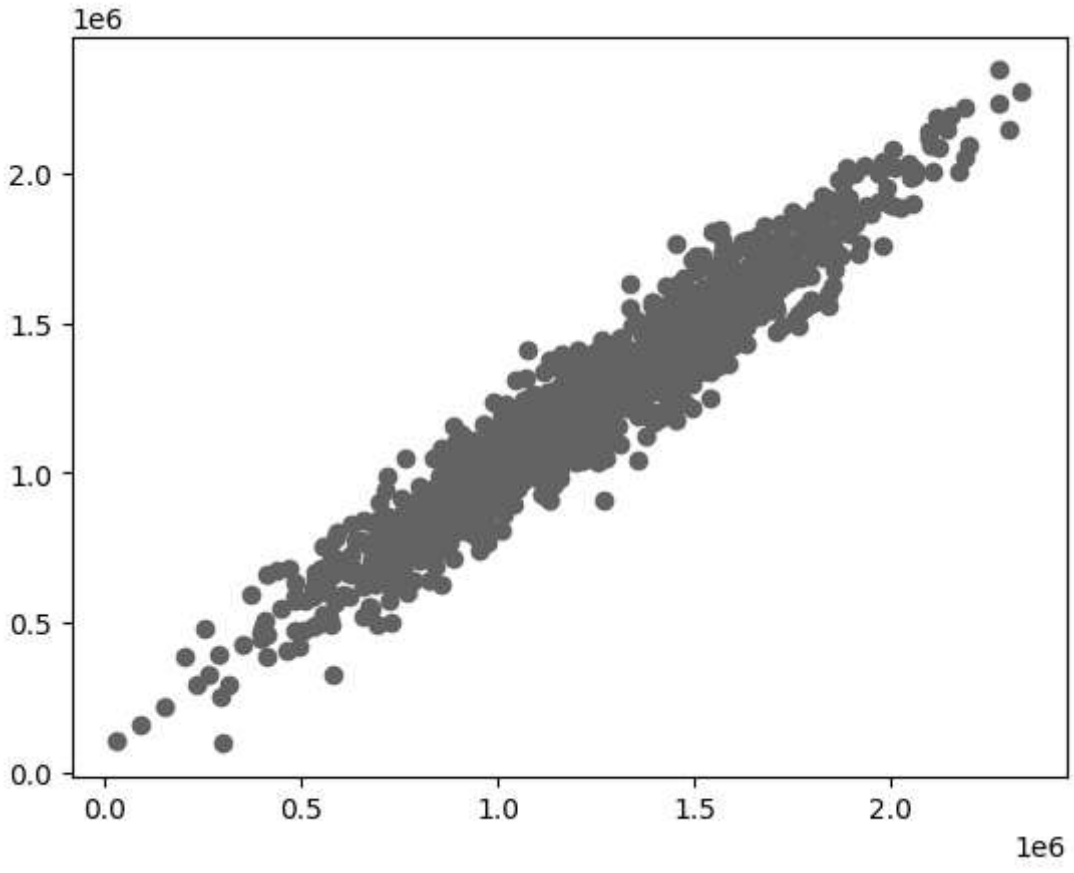
```
pd.DataFrame(model.coef_,X.columns,columns=['coef'])
```

	coef
Avg. Area Income	21.617635
Avg. Area House Age	165221.119872
Avg. Area Number of Rooms	121405.376596
Avg. Area Number of Bedrooms	1318.718783
Area Population	15.225196

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

```
plt.scatter(y_test,predictions)
```

```
<matplotlib.collections.PathCollection at 0x7efe46398be0>
```



```
from sklearn import metrics
from sklearn.metrics import r2_score
```

```
print('MAE:',metrics.mean_absolute_error(y_test,predictions))
print('MSE:',metrics.mean_squared_error(y_test,predictions))
print('RSME:',np.sqrt(metrics.mean_squared_error(y_test,predictions)))
print('R2 Square:',model.score(X_train,y_train))
```

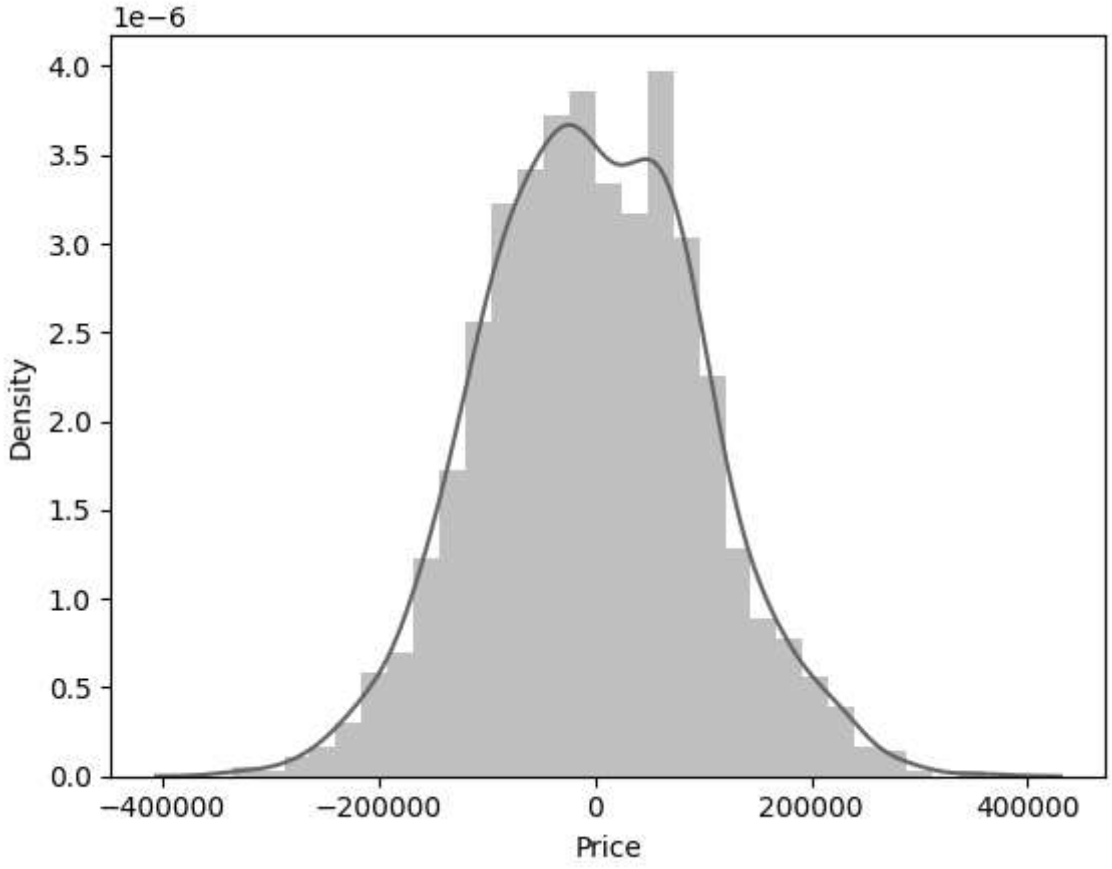
```
MAE: 81257.55795855941
MSE: 10169125565.897606
RSME: 100842.08231635048
R2 Square: 0.9177641115595527
```

```
price_pred['Price'].mean()
```

```
1232072.654142357
```

```
sns.distplot(y_test-predictions)
```

```
<Axes: xlabel='Price', ylabel='Density'>
```



Polynomial Regression

```

```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly_reg = PolynomialFeatures(degree=2)
```

```
X_train_2_d = poly_reg.fit_transform(X_train)
X_test_2_d= poly_reg.transform(X_test)
```

```
lin_reg = LinearRegression()
lin_reg.fit(X_train_2_d,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
test_pred = lin_reg.predict(X_test_2_d)
train_pred = lin_reg.predict(X_train_2_d)
```

```
print('MAE:', metrics.mean_absolute_error(y_test,test_pred))
print('MSE:', metrics.mean_squared_error(y_test,test_pred))
print('RSME:', np.sqrt(metrics.mean_squared_error(y_test,test_pred)))
print('R2 Square:',lin_reg.score(X_train_2_d,y_train))
```

```
MAE: 81711.43034575525
MSE: 10298655026.256853
RSME: 101482.28922455806
R2 Square: 0.9181845240606963
```

Support Vector Machine

```
from sklearn.svm import SVR
```

```
svm_reg = SVR(kernel='rbf', C=1000000,epsilon=0.001)
svm_reg.fit(X_train,y_train)
```

```
▾ SVR
SVR(C=1000000, epsilon=0.001)
```

```
test_pred = svm_reg.predict(X_test)
train_pred = svm_reg.predict(X_train)
```

```
print('MAE:', metrics.mean_absolute_error(y_test,test_pred))
print('MSE:', metrics.mean_squared_error(y_test,test_pred))
```

```
print('RSME:', np.sqrt(metrics.mean_squared_error(y_test,test_pred)))
print('R2 Square:',svm_reg.score(X_train,y_train))
```

```
MAE: 185047.28620378306
MSE: 53792618861.32208
RSME: 231932.3583748548
R2 Square: 0.5890788136150413
```

Experiment 5

Implement Logistic regressor using softmax on iris dataset using sckitlearn

```
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
ds=sns.load_dataset('iris')
```

```
ds.head()
```

	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

```
encoder=LabelEncoder()
ds['species']=encoder.fit_transform(ds['species'])
```

```
ds=ds[['sepal_length','petal_length','species']]
```

```
ds.head()
```

	sepal_length	petal_length	species
0	5.1	1.4	0
1	4.9	1.4	0
2	4.7	1.3	0
3	4.6	1.5	0

```
X=ds.iloc[:,0:2]
y=ds.iloc[:,-1]
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

```
clf=LogisticRegression(multi_class='multinomial')
```

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

```
LogisticRegression
LogisticRegression(multi_class='multinomial')
```

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

```
accuracy_score(y_test,y_pred)
```

```
0.9666666666666667
```

```
pd.DataFrame(confusion_matrix(y_test,y_pred))
```

	0	1	2
0	14	0	0
1	0	7	1
2	0	0	8

Experiment 6

Implement Regularized Regression for house price prediction and evaluate there accuracy using sckitlearn. ● Ridge Regression

● Lasso Regression

```
```

```
!pip install hvplot holoviews
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import hvplot.pandas
%matplotlib inline
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
```

```
USAhousing=pd.read_csv('USA_Housing.csv')
```

```
USAhousing.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Ad
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferr 674\nLaurabu 3

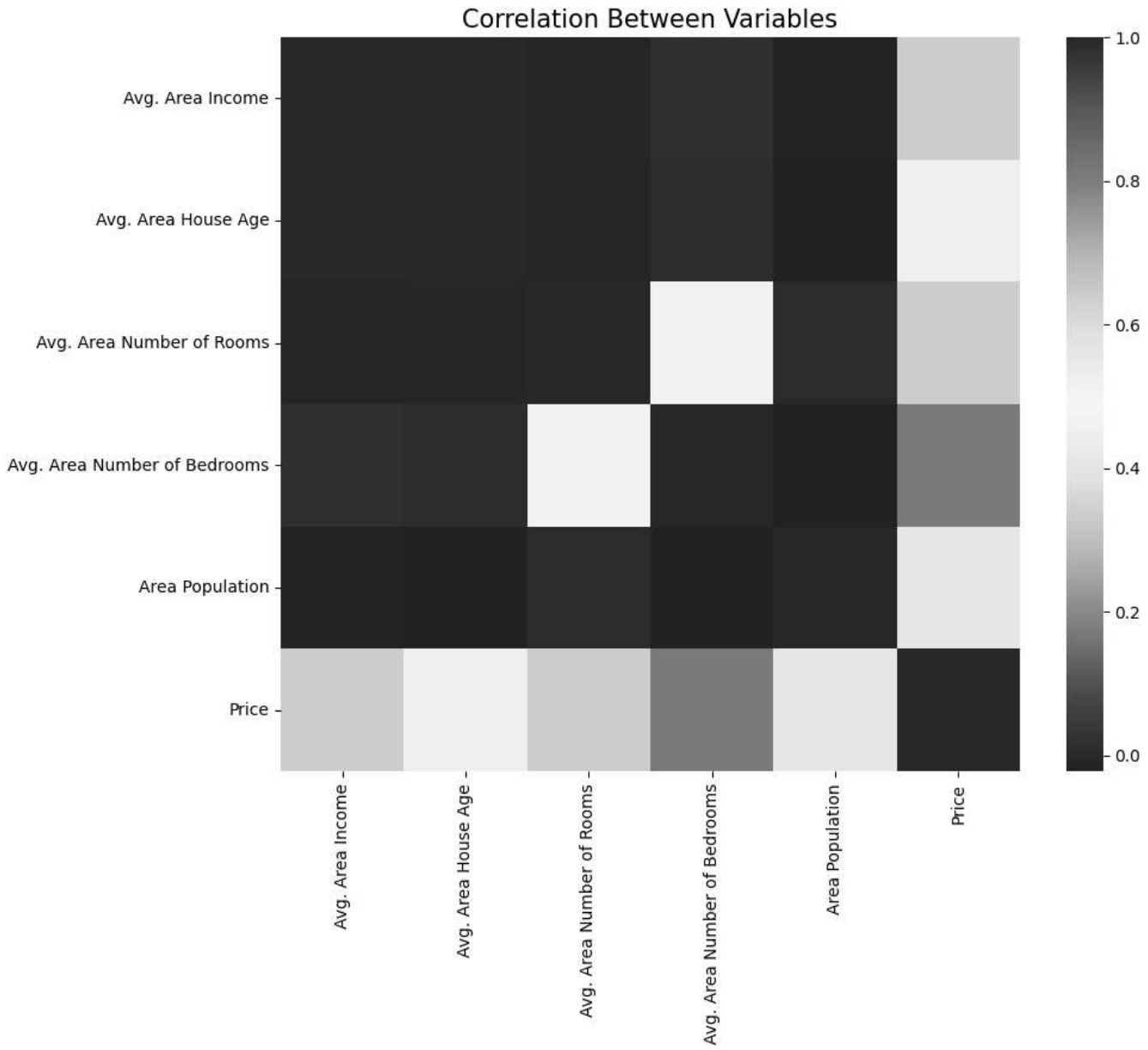
```
USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

```
USAhousing.describe().T
```

	count	mean	std	min	25%	50%
Avg. Area Income	5000.0	6.858311e+04	10657.991214	17796.631190	61480.562388	6.880429e+04
Avg. Area House Age	5000.0	5.977222e+00	0.991456	2.644304	5.322283	5.970429e+00
Avg. Area Number of Rooms	5000.0	6.987792e+00	1.005833	3.236194	6.299250	7.002902e+00

```
plt.figure(figsize=(10,8))
sns.heatmap(USAhousing.corr(),cmap="RdBu")
plt.title("Correlation Between Variables", size=15)
plt.show()
```




```
X=USAhousing[['Avg. Area Income', 'Avg. Area House Age','Avg. Area Number of Rooms', 'Avg. Ar
y=USAhousing['Price']
```

```
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)
```

```
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def cross_val(model):
    pred = cross_val_score(model,X,y,cv=10)
    return pred.mean()

def print_evaluate(true,predicted):
    mae = metrics.mean_absolute_error(true, predicted)
```

```
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_square)
    print('_____________________________________')

def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

```
model=Ridge(alpha=100,solver='cholesky',tol=0.0001,random_state=42)
model.fit(X_train,y_train)
pred=model.predict(X_test)
test_pred=model.predict(X_test)
train_pred=model.predict(X_train)
print('Test set evaluation:/n')
print_evaluate(y_test,test_pred)
print('Train set evaluation:/n')
print_evaluate(y_train,train_pred)
results_df_2=pd.DataFrame(data=[["Ridge Regression",*evaluate(y_test,test_pred),cross_val(Rid
```

```
Test set evaluation:/n
MAE: 81399.92715117308
MSE: 10123755951.873407
RMSE: 100616.8770727526
R2 Square 0.9142129637768774
_____________________________________
Train set evaluation:/n
MAE: 81681.57047883255
MSE: 10322462499.590796
RMSE: 101599.52017401851
R2 Square 0.9190207949590262
_____________________________________
```

```
model = Lasso(alpha=0.1,precompute=True,positive=True,selection='random', random_state=42)
model.fit(X_train,y_train)

test_pred=model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('==================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

```
results_df_2 = pd.DataFrame(data=[["Lasso Regression", *evaluate(y_test, test_pred) , cross_v
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', "Cross Valid
```

```
Test set evaluation:
_____________________________________
MAE: 81135.7008160579
MSE: 10068453919.913456
RMSE: 100341.68585345503
R2 Square 0.9146815840638058
_____________________________________
==================================
Train set evaluation:
_____________________________________
MAE: 81480.62936440232
MSE: 10287043196.82861
RMSE: 101425.06197596632
R2 Square 0.9192986576280261
_____________________________________
```

Experiment 7

Implement Varios Classification algorithm for iris data set and evaluate there performance.

● Navie Bayes Classifier ● Logistic Regression ● Support vector Machine ● Decision tree

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

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

```
iris.head
```

```
<bound method NDFrame.head of      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
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]>
```

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

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

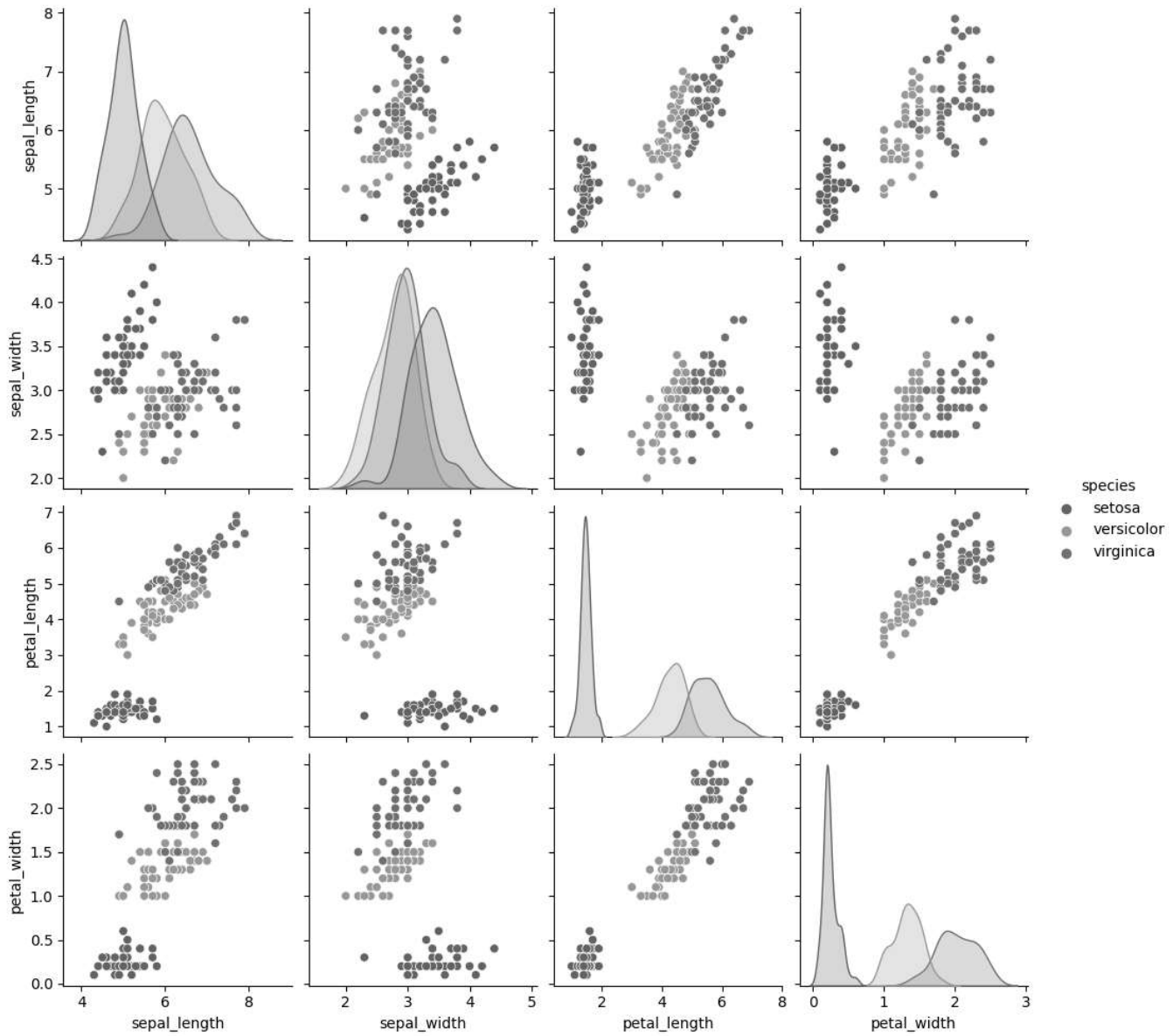
```
iris.describe(include='all')
```

	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	setosa
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.057333	3.758000	1.199333	NaN
std	0.828066	0.435866	1.765298	0.762238	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

```
iris.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
```

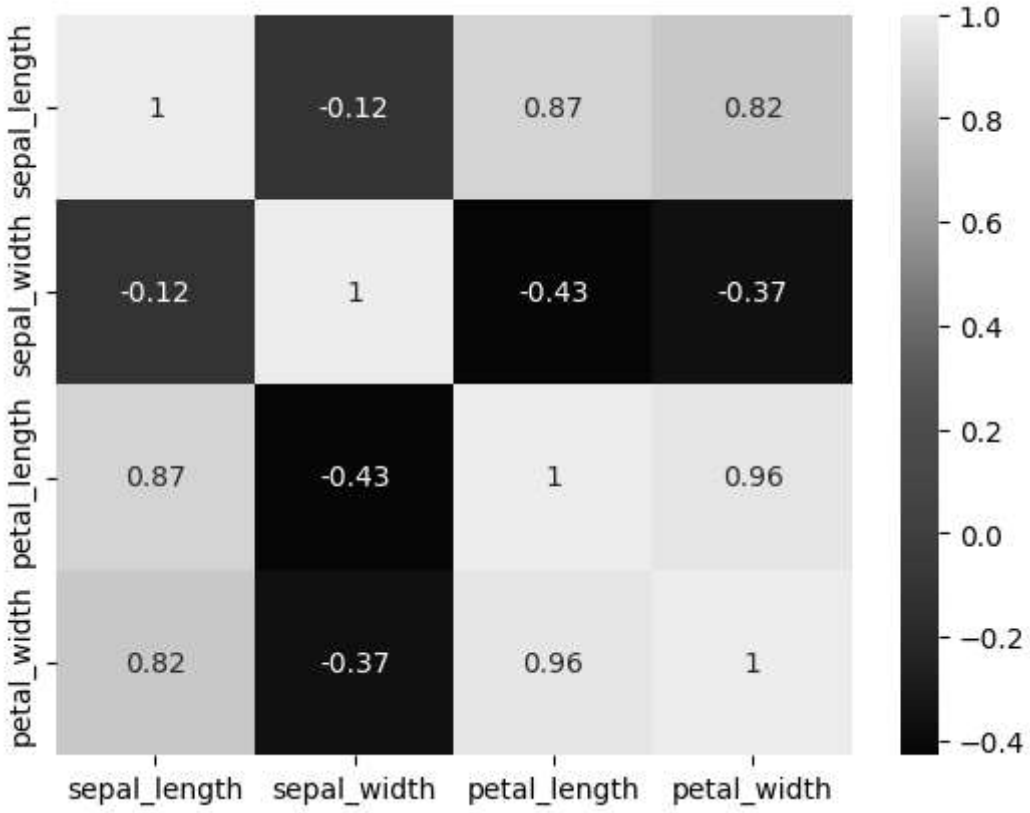
```
sns.pairplot(iris,hue="species")
plt.show()
```


sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica

```
iris.corr()
```

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.117570	0.871754	0.817941
sepal_width	-0.117570	1.000000	-0.428440	-0.366126
petal_length	0.871754	-0.428440	1.000000	0.962865
petal_width	0.817941	-0.366126	0.962865	1.000000

```
sns.heatmap(iris.corr(), annot=True)
plt.show()
```




```
X=iris.iloc[:,0:4].values
y=iris.iloc[:,4].values
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
```

```
from sklearn.metrics import make_scorer,accuracy_score,precision_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.model_selection import KFold,train_test_split,cross_val_score
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC,LinearSVC
from sklearn.naive_bayes import GaussianNB
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

Logistic Regression

```
logreg = LogisticRegression(solver= 'lbfgs',max_iter=400)
logreg.fit(X_train, y_train)
Y_pred = logreg.predict(X_test)
accuracy_lr=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_log = round(logreg.score(X_train, y_train) * 100, 2)


cm = confusion_matrix(y_test, Y_pred,)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for Logistic Regression\n',cm)
print('accuracy_Logistic Regression : %.3f' %accuracy)
print('precision_Logistic Regression : %.3f' %precision)
print('recall_Logistic Regression: %.3f' %recall)
print('f1-score_Logistic Regression : %.3f' %f1)
```

```
Confusion matrix for Logistic Regression
 [[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
accuracy_Logistic Regression : 0.978
precision_Logistic Regression : 0.978
recall_Logistic Regression: 0.978
f1-score_Logistic Regression : 0.978
```

Naive Bayes

```
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
Y_pred = gaussian.predict(X_test)
accuracy_nb=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_gaussian = round(gaussian.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
```

```
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for Naive Bayes\n',cm)
print('accuracy_Naive Bayes: %.3f' %accuracy)
print('precision_Naive Bayes: %.3f' %precision)
print('recall_Naive Bayes: %.3f' %recall)
print('f1-score_Naive Bayes : %.3f' %f1)
```

```
Confusion matrix for Naive Bayes
 [[16  0  0]
 [ 0 18  0]
 [ 0  0 11]]
accuracy_Naive Bayes: 1.000
precision_Naive Bayes: 1.000
recall_Naive Bayes: 1.000
f1-score_Naive Bayes : 1.000
```

SVM

```
linear_svc = LinearSVC(max_iter=4000)
linear_svc.fit(X_train, y_train)
Y_pred = linear_svc.predict(X_test)
accuracy_svc=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_linear_svc = round(linear_svc.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for SVC\n',cm)
print('accuracy_SVC: %.3f' %accuracy)
print('precision_SVC: %.3f' %precision)
print('recall_SVC: %.3f' %recall)
print('f1-score_SVC : %.3f' %f1)
```

```
Confusion matrix for SVC
 [[16  0  0]
 [ 0 15  3]
 [ 0  0 11]]
accuracy_SVC: 0.933
precision_SVC: 0.933
recall_SVC: 0.933
f1-score_SVC : 0.933
```

Decision tree

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
```

```
Y_pred = decision_tree.predict(X_test)
accuracy_dt=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_decision_tree = round(decision_tree.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for DecisionTree\n',cm)
print('accuracy_DecisionTree: %.3f' %accuracy)
print('precision_DecisionTree: %.3f' %precision)
print('recall_DecisionTree: %.3f' %recall)
print('f1-score_DecisionTree : %.3f' %f1)
```

```
Confusion matrix for DecisionTree
 [[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
accuracy_DecisionTree: 0.978
precision_DecisionTree: 0.978
recall_DecisionTree: 0.978
f1-score_DecisionTree : 0.978
```

```
results = pd.DataFrame({
    'Model': ['Logistic Regression',
              'Naive Bayes',
              ' Support Vector Machine',
              'Decision Tree'],
    'Score': [ acc_log,
              acc_gaussian,
              acc_linear_svc,
              acc_decision_tree],
    "Accuracy_score":[accuracy_lr,
                      accuracy_nb,
                      accuracy_svc,
                      accuracy_dt
                     ]})
result_df = results.sort_values(by='Accuracy_score', ascending=False)
result_df = result_df.reset_index(drop=True)
result_df.head(9)
```

	Model	Score	Accuracy_score
0	Naive Bayes	94.29	100.00
1	Logistic Regression	98.10	97.78
2	Decision Tree	100.00	97.78
3	Support Vector Machine	98.10	93.33