K-nearest Neighbours Regression

K-nearest Neighbours Regression non-parametric method It is one of the simplest Machine Learning algorithms based on Supervised Learning technique It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. But,we have knn regressin algorithm

How does Knn Work?

The K-NN working can be explained on the basis of the below algorithm: Step-1: Select the number K of the neighbors Step-2: Calculate the Euclidean distance of K number of neighbors Step-3: Take the K nearest neighbors as per the calculated Euclidean distance. Step-4: Among these k neighbors, count the number of the data points in each category. Step-5: Assign the new data points to that category for which the number of the neighbor is maximum. Step-6: Our model is ready.

Advantages of KNN Algorithm:

It is simple to implement. It is robust to the noisy training data It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

Always needs to determine the value of K which may be complex some time. The computation cost is high because of calculating the distance between the data points for all the training samples.

In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df=pd.read_csv('50_Startups.csv')
df.head()
```

Out[2]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object 
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

In [4]:

```
df.describe()
```

Out[4]:

	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

In [5]:

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

Out[5]:

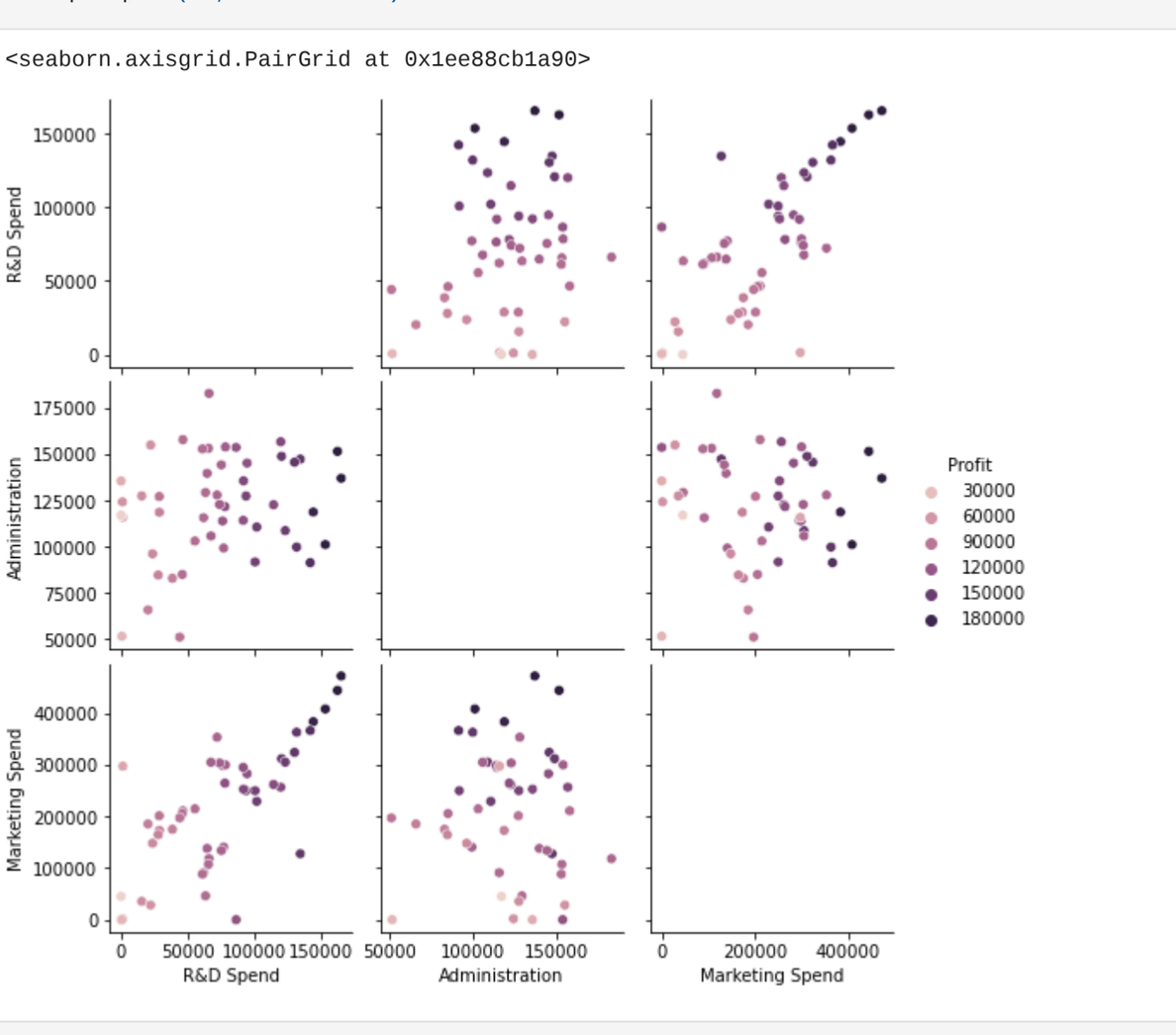
```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

In [6]:

```
sns.pairplot(df,hue='Profit')
```

Out[6]:

```
<seaborn.axisgrid.PairGrid at 0x1ee88cb1a90>
```

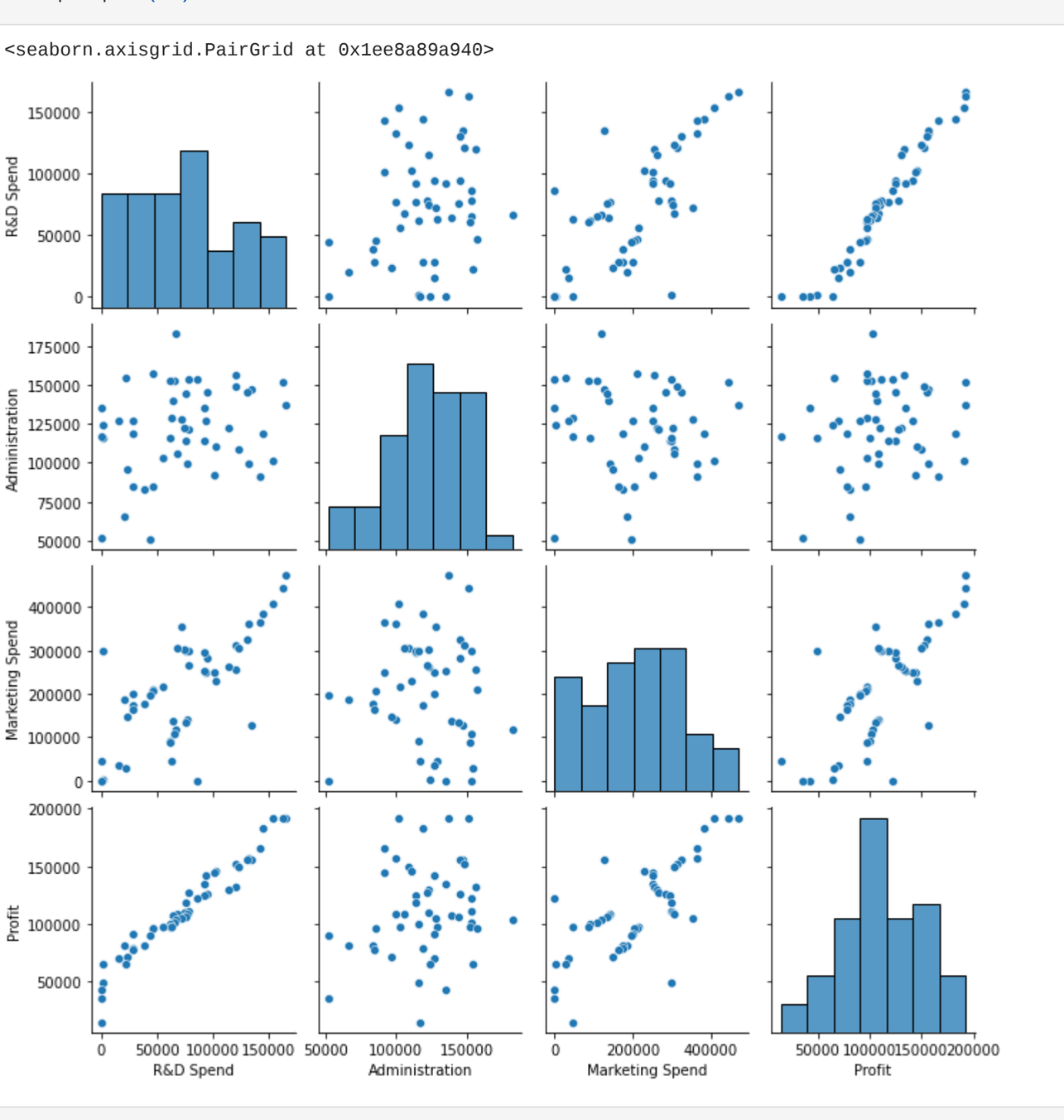


In [7]:

```
sns.pairplot(df)
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x1ee8a89a940>
```



In [8]:

```
df_cat=df.select_dtypes(object)
df_num=df.select_dtypes(float)
```

In [9]:

```
from sklearn.preprocessing import LabelEncoder
```

In [10]:

```
lb=LabelEncoder()
df_cat=pd.DataFrame(lb.fit_transform(df_cat),columns=['State'])
```

In [11]:

```
df_cat
```

Out[11]:

	State
0	2
1	0
2	1
3	2
4	1
5	2
6	0
7	1
8	2
9	0
10	1
11	0
12	1
13	0
14	1
15	2
16	0
17	2
18	1
19	2
20	0
21	2
22	1
23	1
24	2
25	0
26	1
27	2
28	1
29	2
30	1
31	2
32	0
33	1
34	0
35	2
36	1
37	0
38	2
39	0
40	0
41	1
42	0
43	2
44	0
45	2
46	1
47	0
48	2
49	0

In [12]:

```
df=pd.concat([df_cat,df_num],axis=1)
```

In [13]:

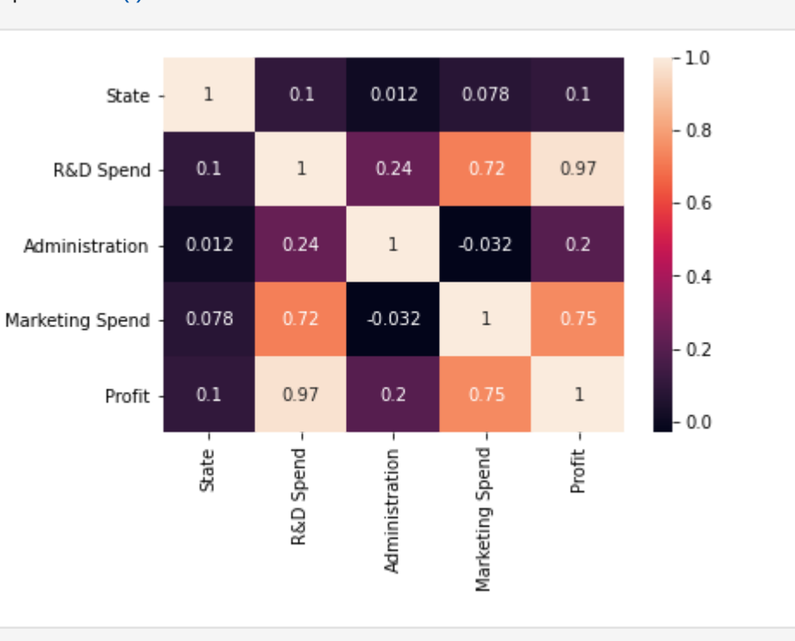
```
df
```

Out[13]:

	State	R&D Spend	Administration	Marketing Spend	Profit
0	2	165349.20	136897.80	471784.10	192261.83
1	0	162597.70	151377.59	443898.53	191792.06
2	1	153441.51	101145.55	407934.54	191050.39
3	2	144372.41	118671.85	383199.62	182901.99
4	1	142107.34	91391.77	366168.42	166187.94
5	2	131876.90	99814.71	362861.36	156991.12
6	0	134615.46	147198.87	127716.82	156122.51
7	1	130298.13	145530.06	323876.68	155752.60
8	2	120542.52	148718.95	311613.29	152211.77
9	0	123334.88	108679.17	304981.62	149759.96
10	1	101913.08	110594.11	229160.95	146121.95
11	0	100671.96	91790.61	249744.55	144259.40
12	1	93863.75	127320.38	249839.44	141585.52
13	0	91992.39	135495.07	252664.93	134307.35
14	1	119943.24	156547.42	256512.92	132602.65
15	2	114523.61	122616.84	261776.23	129917.04
16	0	78013.11	121597.55	264346.06	126992.93
17	2	94657.16	145077.58	282574.31	125370.37
18	1	91749.16	114175.79	294919.57	124266.90
19	2	86419.70	153514.11	0.00	122776.86
20	0	76253.86	113867.30	298664.47	118474.03
21	2	78389.47	153773.43	299737.29	111313.02
22	1	73994.56	122782.75	303319.26	110352.25
23	1	67532.53	105751.03	304768.73	108733.99
24	2	77044.01	99281.34	140574.81	108552.04
25	0	64664.71	139553.16	137962.62	107404.34
26	1	75328.87	144135.98	134050.07	105733.54
27	2	72107.60	127864.55	353183.81	105008.31
28	1	66051.52	182645.56	118148.20	103282.38
29	2	65605.48	153032.06	107138.38	101004.64
30	1	61994.48	115641.28	91131.24	99937.59
31	2	61136.38	152701.92	88218.23	97483.56
32	0	63408.86	129219.61	46085.25	97427.84
33	1	55493.95	103057.49	214634.81	96778.92
34	0	46426.07	157693.92	210797.67	96712.80
35	2	46014.02	85047.44	205517.64	96479.51
36	1	28663.76	127056.21	201126.82	90708.19
37	0	44069.95	51283.14	197029.42	89949.14
38	2	20229.59	65947.93	185265.10	81229.06
39	0	38558.51	82982.09	174999.30	81005.76
40	0	28754.33	118546.05	172795.67	78239.91
41	1	27892.92	84710.77	164470.71	77798.83
42	0	23640.93	96189.63	148001.11	71498.49
43	2	15505.73	127382.30	35534.17	69758.98
44	0	22177.74	154806.14	28334.72	65200.33
45	2	1000.23	124153.04	1903.93	64926.08
46	1	1315.46	115816.21	297114.46	49490.75
47	0	0.00	135426.92	0.00	42559.73
48	2	542.05	51743.15	0.00	35673.41
49	0	0.00	116983.80	45173.06	14681.40

In [14]:

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



In [15]:

```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [16]:

```
from sklearn.model_selection  import  train_test_split as t
xtrain,xtest,ytrain,ytest=t(x,y,random_state=1)
```

In [17]:

```
from sklearn.neighbors import KNeighborsRegressor
```

In [18]:

```
knn=KNeighborsRegressor(n_neighbors=5)
knn.fit(xtrain,ytrain)
ypred=knn.predict(xtest)
```

In [19]:

```
from sklearn.metrics import r2_score  ,mean_squared_error ,mean_absolute_error
```

In [20]:

```
print(f'accuracy   :{r2_score(ytest,ypred)*100}')
print(f'accuracy   :{mean_squared_error(ytest,ypred)}')
print(f'accuracy   :{mean_absolute_error(ytest,ypred)}')
```

```
accuracy   :69.960315030845
accuracy   :536387422.84968525
accuracy   :16358.926615384618
```

As we were getting the accuracy as 71.37852365482746 so we have used for loop for the random state value as i in the loop so we can check on which it is giving the maximum accuracy

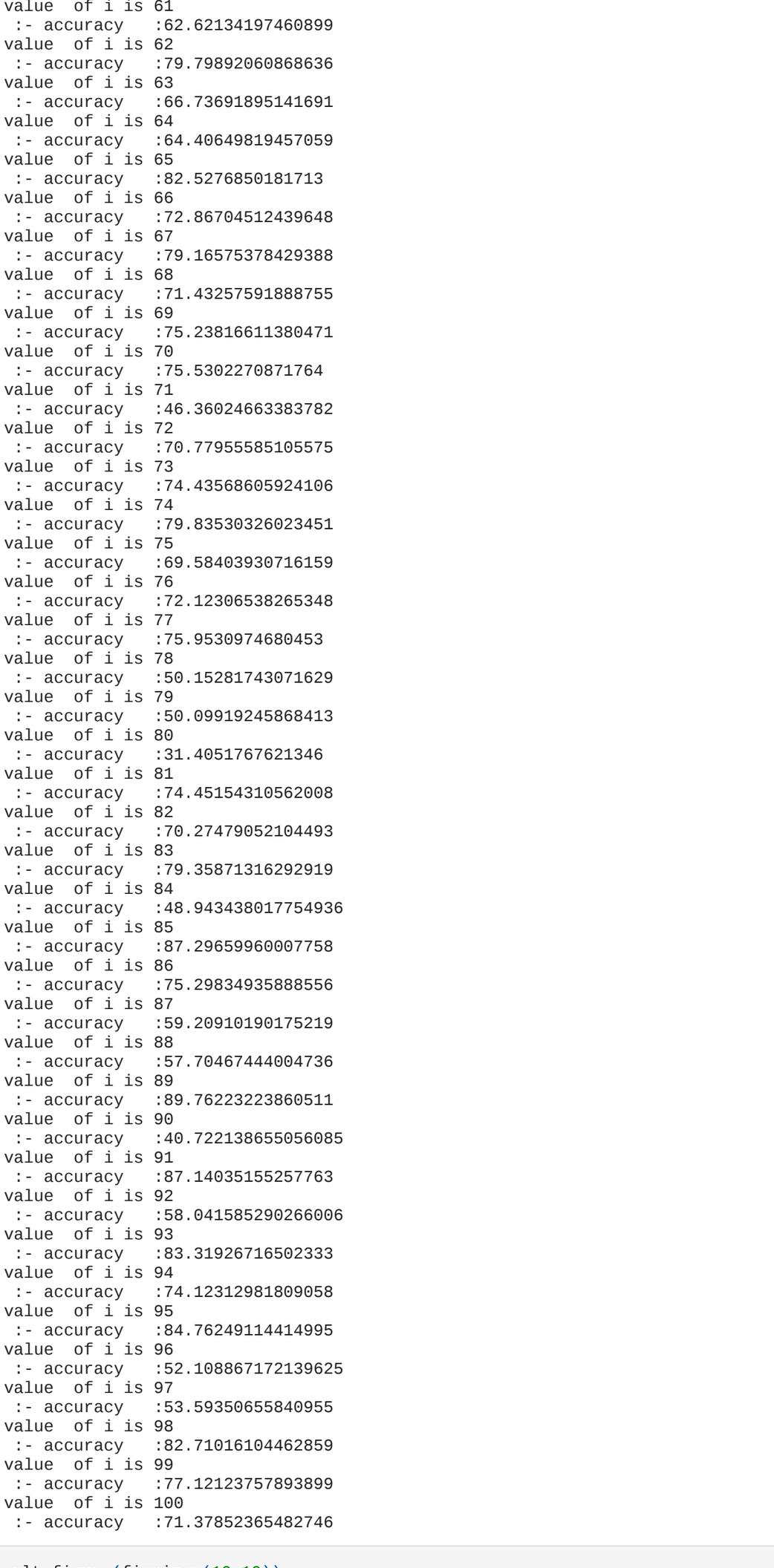
In [21]:

```
ac=[]
for i in range(1,101):
    xtrain,xtest,ytrain,ytest=t(x,y,random_state=i)
    knn=KNeighborsRegressor(n_neighbors=5)
    knn.fit(xtrain,ytrain)
    ypred=knn.predict(xtest)
    print(f'value  of i is {i}  \n :- accuracy   :{r2_score(ytest,ypred)*100}')
    ac.append(r2_score(ytest,ypred)*100)
```

```
value  of i is 1
 :- accuracy   :69.960315030845
value  of i is 2
 :- accuracy   :68.71251026899773
value  of i is 3
 :- accuracy   :70.55337880395052
value  of i is 4
 :- accuracy   :75.78081016144732
value  of i is 5
 :- accuracy   :44.01163969155184
value  of i is 6
 :- accuracy   :60.22549880380505
value  of i is 7
 :- accuracy   :53.04805411200925
value  of i is 8
 :- accuracy   :64.25445062386606
value  of i is 9
 :- accuracy   :92.92501139968044
value  of i is 10
 :- accuracy   :77.69451202168902
value  of i is 11
 :- accuracy   :82.15414711059374
value  of i is 12
 :- accuracy   :40.561995398812
value  of i is 13
 :- accuracy   :50.932173319746674
value  of i is 14
 :- accuracy   :71.85397338024553
value  of i is 15
 :- accuracy   :92.60365955676554
value  of i is 16
 :- accuracy   :65.71527758661988
value  of i is 17
 :- accuracy   :52.9303837605245
value  of i is 18
 :- accuracy   :56.978322964667825
value  of i is 19
 :- accuracy   :79.92257563646497
value  of i is 20
 :- accuracy   :92.272999230936
value  of i is 21
 :- accuracy   :79.86926513700679
value  of i is 22
 :- accuracy   :88.79711661832843
value  of i is 23
 :- accuracy   :82.5575686959371
value  of i is 24
 :- accuracy   :60.12522004489269
value  of i is 25
 :- accuracy   :67.16018372511626
value  of i is 26
 :- accuracy   :63.880397095733485
value  of i is 27
 :- accuracy   :54.793018550907135
value  of i is 28
 :- accuracy   :53.015966150259274
value  of i is 29
 :- accuracy   :84.29071814920019
value  of i is 30
 :- accuracy   :12.90113007502881
value  of i is 31
 :- accuracy   :41.842496084434245
value  of i is 32
 :- accuracy   :78.94951418148146
value  of i is 33
 :- accuracy   :89.78347109114593
value  of i is 34
 :- accuracy   :71.42355384300502
value  of i is 35
 :- accuracy   :86.04640647786287
value  of i is 36
 :- accuracy   :78.54172415414672
value  of i is 37
 :- accuracy   :83.21157866436943
value  of i is 38
 :- accuracy   :71.19736086387812
value  of i is 39
 :- accuracy   :75.72191092953233
value  of i is 40
 :- accuracy   :78.67827714398535
value  of i is 41
 :- accuracy   :74.47952102002608
value  of i is 42
 :- accuracy   :58.69604694082855
value  of i is 43
 :- accuracy   :66.55666782292982
value  of i is 44
 :- accuracy   :71.51748253553345
value  of i is 45
 :- accuracy   :71.17147757178354
value  of i is 46
 :- accuracy   :65.88545535770922
value  of i is 47
 :- accuracy   :57.40388416796809
value  of i is 48
 :- accuracy   :76.58793238656047
value  of i is 49
 :- accuracy   :86.74014737962933
value  of i is 50
 :- accuracy   :67.54687005647257
value  of i is 51
 :- accuracy   :75.74274430056857
value  of i is 52
 :- accuracy   :33.8090260289343
value  of i is 53
 :- accuracy   :47.9388209071123
value  of i is 54
 :- accuracy   :32.79426400452249
value  of i is 55
 :- accuracy   :71.57794663930372
value  of i is 56
 :- accuracy   :83.0505930686644
value  of i is 57
 :- accuracy   :77.60731314310146
value  of i is 58
 :- accuracy   :86.46567314488759
value  of i is 59
 :- accuracy   :87.45645997494864
value  of i is 60
 :- accuracy   :69.96676152062173
value  of i is 61
 :- accuracy   :62.62134197460899
value  of i is 62
 :- accuracy   :79.79892060868636
value  of i is 63
 :- accuracy   :66.73691895141691
value  of i is 64
 :- accuracy   :64.40649819457059
value  of i is 65
 :- accuracy   :82.5276850181713
value  of i is 66
 :- accuracy   :72.86704512439648
value  of i is 67
 :- accuracy   :79.16575378429388
value  of i is 68
 :- accuracy   :71.43257591888755
value  of i is 69
 :- accuracy   :75.23816611380471
value  of i is 70
 :- accuracy   :75.5302270871764
value  of i is 71
 :- accuracy   :46.36024663383782
value  of i is 72
 :- accuracy   :70.77955585105575
value  of i is 73
 :- accuracy   :74.43568605924106
value  of i is 74
 :- accuracy   :79.83530326023451
value  of i is 75
 :- accuracy   :69.58403930716159
value  of i is 76
 :- accuracy   :72.12306538265348
value  of i is 77
 :- accuracy   :75.9530974680453
value  of i is 78
 :- accuracy   :50.15281743071629
value  of i is 79
 :- accuracy   :50.09919245868413
value  of i is 80
 :- accuracy   :31.4051767621346
value  of i is 81
 :- accuracy   :74.45154310562008
value  of i is 82
 :- accuracy   :70.27479052104493
value  of i is 83
 :- accuracy   :79.35871316292919
value  of i is 84
 :- accuracy   :48.943438017754936
value  of i is 85
 :- accuracy   :87.29659960007758
value  of i is 86
 :- accuracy   :75.29834935888556
value  of i is 87
 :- accuracy   :59.20910190175219
value  of i is 88
 :- accuracy   :57.70467444004736
value  of i is 89
 :- accuracy   :89.76223223860511
value  of i is 90
 :- accuracy   :40.722138655056085
value  of i is 91
 :- accuracy   :87.14035155257763
value  of i is 92
 :- accuracy   :58.041585290266006
value  of i is 93
 :- accuracy   :83.31926716502333
value  of i is 94
 :- accuracy   :74.12312981809058
value  of i is 95
 :- accuracy   :84.76249114414995
value  of i is 96
 :- accuracy   :52.108867172139625
value  of i is 97
 :- accuracy   :53.59350655840955
value  of i is 98
 :- accuracy   :82.71016104462859
value  of i is 99
 :- accuracy   :77.12123757893899
value  of i is 100
 :- accuracy   :71.37852365482746
```

In [22]:

```
plt.figure(figsize=(10,10))
plt.plot(range(1,101),ac)
plt.grid()
plt.show()
```



The size of the neighbourhood needs to be set by the analyst or can be chosen using cross-validation (we will see this later) to select the size that minimises the mean-squared error. here we have used for loop and given the range of neighbors so we can which neighbor value is the best

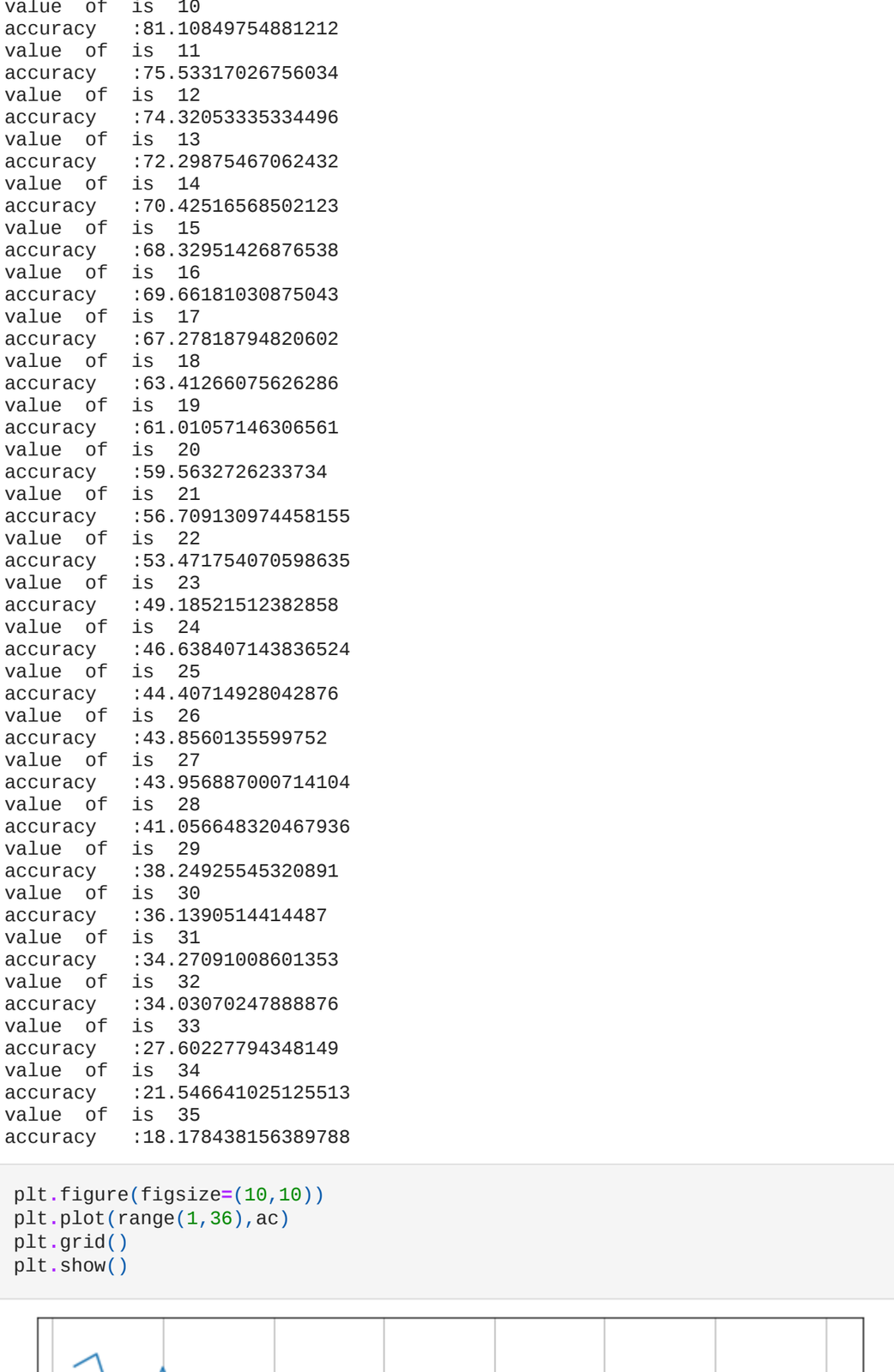
In [23]:

```
ac=[]
for i in range(1,36,1):
    xtrain,xtest,ytrain,ytest=t(x,y,random_state=9)
    knn=KNeighborsRegressor(n_neighbors=i)
    knn.fit(xtrain,ytrain)
    ypred=knn.predict(xtest)
    print(f'value  of  is  {i} \naccuracy   :{r2_score(ytest,ypred)*100}')
    ac.append(r2_score(ytest,ypred)*100)
```

```
value  of  is  1
accuracy   :93.0469092306494
value  of  is  2
accuracy   :94.32285838000702
value  of  is  3
accuracy   :87.12200685370765
value  of  is  4
accuracy   :86.89686202685793
value  of  is  5
accuracy   :92.92501139968044
value  of  is  6
accuracy   :86.71913631825376
value  of  is  7
accuracy   :86.84445625052778
value  of  is  8
accuracy   :85.10638312090006
value  of  is  9
accuracy   :84.09945580487317
value  of  is  10
accuracy   :81.10849754881212
value  of  is  11
accuracy   :75.53317026756034
value  of  is  12
accuracy   :74.32053335334496
value  of  is  13
accuracy   :72.29875467062432
value  of  is  14
accuracy   :70.42516568502123
value  of  is  15
accuracy   :68.32951426876538
value  of  is  16
accuracy   :69.66181030875043
value  of  is  17
accuracy   :67.27818794820602
value  of  is  18
accuracy   :63.41266075626286
value  of  is  19
accuracy   :61.01057146306561
value  of  is  20
accuracy   :59.5632726233734
value  of  is  21
accuracy   :56.709130974458155
value  of  is  22
accuracy   :53.471754070598635
value  of  is  23
accuracy   :49.18521512382858
value  of  is  24
accuracy   :46.638407143836524
value  of  is  25
accuracy   :44.40714928042876
value  of  is  26
accuracy   :43.8560135599752
value  of  is  27
accuracy   :43.956887000714104
value  of  is  28
accuracy   :41.056648320467936
value  of  is  29
accuracy   :38.24925545320891
value  of  is  30
accuracy   :36.1390514414487
value  of  is  31
accuracy   :34.27091008601353
value  of  is  32
accuracy   :34.03070247888876
value  of  is  33
accuracy   :27.60227794348149
value  of  is  34
accuracy   :21.546641025125513
value  of  is  35
accuracy   :18.178438156389788
```

In [24]:

```
plt.figure(figsize=(10,10))
plt.plot(range(1,36),ac)
plt.grid()
plt.show()
```



k =2 is the bestvalue so it can give the highest accuracy on the testing data set

In [25]:

```
xtrain,xtest,ytrain,ytest=t(x,y,random_state=9)
knn=KNeighborsRegressor(n_neighbors=2)
knn.fit(xtrain,ytrain)
ypred=knn.predict(xtest)
print(f'accuracy   :{r2_score(ytest,ypred)*100}')
print(f'accuracy   :{mean_squared_error(ytest,ypred)*100}')
print(f'accuracy   :{mean_absolute_error(ytest,ypred)*100}')
print(f'accuracy   :{np.sqrt(mean_absolute_error(ytest,ypred)*100)}')
```

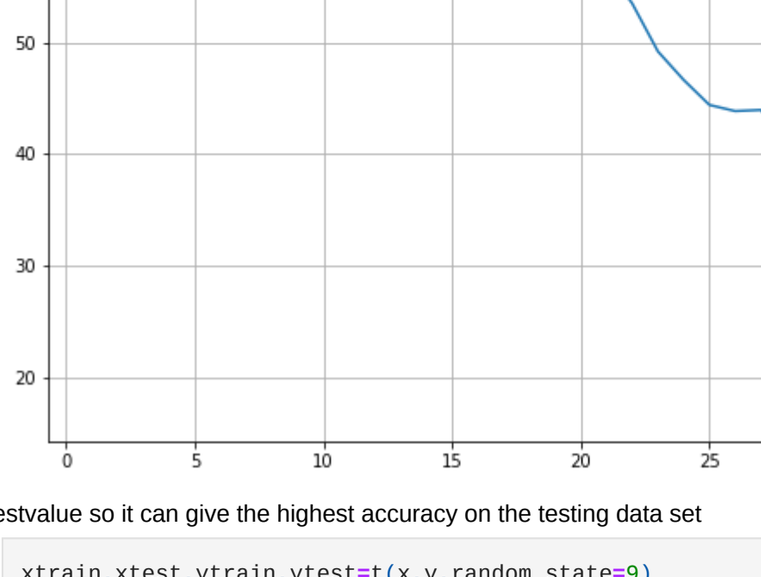
```
accuracy   :94.32285838000702
accuracy   :6620575065.670587
accuracy   :668621.7307692311
accuracy   :817.6929318327456
```

In [26]:

```
plt.scatter(ypred,ytest)
```

Out[26]:

```
<matplotlib.collections.PathCollection at 0x1ee8badfe80>
```



In [27]:

```
knn.predict([[2,1000,20000,1000]])
```

Out[27]:

```
array([25177.405])
```

here a is the value of state b is the value of R&D Spend c is the value of Administration d is the value of Marketing Spend

In [28]:

```
def profit(a,b,c,d):
    return     knn.predict([[a,b,c,d]])
```

In [29]:

```
profit(1,111090,12133,12342134)
```

Out[29]:

```
array([192026.945])
```

In [ ]: