Data Preprocessing

In [1]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:

```
# Importing the dataset
df = pd.read_csv('RL_EXAM_3.csv', sep=",")
```

In [3]:

```
df
```

Out[3]:

	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
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	110594.11	229160.95	Florida	146121.95
11	100671.96	91790.61	249744.55	California	144259.40
12	93863.75	127320.38	249839.44	Florida	141585.52
13	91992.39	135495.07	252664.93	California	134307.35
14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37
18	91749.16	114175.79	294919.57	Florida	124266.90
19	86419.70	153514.11	0.00	New York	122776.86
20	76253.86	113867.30	298664.47	California	118474.03
21	78389.47	153773.43	299737.29	New York	111313.02
22	73994.56	122782.75	303319.26	Florida	110352.25
23	67532.53	105751.03	304768.73	Florida	108733.99
24	77044.01	99281.34	140574.81	New York	108552.04
25	64664.71	139553.16	137962.62	California	107404.34
26	75328.87	144135.98	134050.07	Florida	105733.54
27	72107.60	127864.55	353183.81	New York	105008.31
28	66051.52	182645.56	118148.20	Florida	103282.38
29	65605.48	153032.06	107138.38	New York	101004.64
30	61994.48	115641.28	91131.24	Florida	99937.59
31	61136.38	152701.92	88218.23	New York	97483.56
32	63408.86	129219.61	46085.25	California	97427.84
33	55493.95	103057.49	214634.81	Florida	96778.92

	R&D Spend	Administration	Marketing Spend	State	Profit
34	46426.07	157693.92	210797.67	California	96712.80
35	46014.02	85047.44	205517.64	New York	96479.51
36	28663.76	127056.21	201126.82	Florida	90708.19
37	44069.95	51283.14	197029.42	California	89949.14
38	20229.59	65947.93	185265.10	New York	81229.06
39	38558.51	82982.09	174999.30	California	81005.76
40	28754.33	118546.05	172795.67	California	78239.91
41	27892.92	84710.77	164470.71	Florida	77798.83
42	23640.93	96189.63	148001.11	California	71498.49
43	15505.73	127382.30	35534.17	New York	69758.98
44	22177.74	154806.14	28334.72	California	65200.33
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41
49	0.00	116983.80	45173.06	California	14681.40

In [8]:

```
# Function Encoding
def encoding_char(x):
    char_var = list(set(x.columns) - set(x._get_numeric_data().columns))
    for State in char_var:
        f = pd.factorize(x[State])
        x[State] = pd.factorize(x[State])[0]
    return(x)


# Encoding categorical data
df = encoding_char(df)
df
```

Out[8]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	0	192261.83
1	162597.70	151377.59	443898.53	1	191792.06
2	153441.51	101145.55	407934.54	2	191050.39
3	144372.41	118671.85	383199.62	0	182901.99
4	142107.34	91391.77	366168.42	2	166187.94
5	131876.90	99814.71	362861.36	0	156991.12
6	134615.46	147198.87	127716.82	1	156122.51
7	130298.13	145530.06	323876.68	2	155752.60
8	120542.52	148718.95	311613.29	0	152211.77
9	123334.88	108679.17	304981.62	1	149759.96
10	101913.08	110594.11	229160.95	2	146121.95
11	100671.96	91790.61	249744.55	1	144259.40
12	93863.75	127320.38	249839.44	2	141585.52
13	91992.39	135495.07	252664.93	1	134307.35
14	119943.24	156547.42	256512.92	2	132602.65
15	114523.61	122616.84	261776.23	0	129917.04
16	78013.11	121597.55	264346.06	1	126992.93
17	94657.16	145077.58	282574.31	0	125370.37
18	91749.16	114175.79	294919.57	2	124266.90
19	86419.70	153514.11	0.00	0	122776.86
20	76253.86	113867.30	298664.47	1	118474.03
21	78389.47	153773.43	299737.29	0	111313.02
22	73994.56	122782.75	303319.26	2	110352.25
23	67532.53	105751.03	304768.73	2	108733.99
24	77044.01	99281.34	140574.81	0	108552.04
25	64664.71	139553.16	137962.62	1	107404.34
26	75328.87	144135.98	134050.07	2	105733.54
27	72107.60	127864.55	353183.81	0	105008.31
28	66051.52	182645.56	118148.20	2	103282.38
29	65605.48	153032.06	107138.38	0	101004.64
30	61994.48	115641.28	91131.24	2	99937.59
31	61136.38	152701.92	88218.23	0	97483.56
32	63408.86	129219.61	46085.25	1	97427.84
33	55493.95	103057.49	214634.81	2	96778.92

	R&D Spend	Administration	Marketing Spend	State	Profit
34	46426.07	157693.92	210797.67	1	96712.80
35	46014.02	85047.44	205517.64	0	96479.51
36	28663.76	127056.21	201126.82	2	90708.19
37	44069.95	51283.14	197029.42	1	89949.14
38	20229.59	65947.93	185265.10	0	81229.06
39	38558.51	82982.09	174999.30	1	81005.76
40	28754.33	118546.05	172795.67	1	78239.91
41	27892.92	84710.77	164470.71	2	77798.83
42	23640.93	96189.63	148001.11	1	71498.49
43	15505.73	127382.30	35534.17	0	69758.98
44	22177.74	154806.14	28334.72	1	65200.33
45	1000.23	124153.04	1903.93	0	64926.08
46	1315.46	115816.21	297114.46	2	49490.75
47	0.00	135426.92	0.00	1	42559.73
48	542.05	51743.15	0.00	0	35673.41
49	0.00	116983.80	45173.06	1	14681.40

In [21]:

```
df.describe()
# Критических выбросов не наблюдается
```

Out[21]:

	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 [11]:

```
# mean()-3*std
# Let's check how much the data are spread out from the mean.
mean_RD_Spend = np.mean(df['R&D Spend'], axis=0)
sd_RD_Spend = np.std(df['R&D Spend'], axis=0)

mean_Administration = np.mean(df['Administration'], axis=0)
sd_Administration = np.std(df['Administration'], axis=0)

mean_Marketing_Spend = np.mean(df['Marketing Spend'], axis=0)
sd_Marketing_Spend = np.std(df['Marketing Spend'], axis=0)

mean_State = np.mean(df['State'], axis=0)
sd_State = np.std(df['State'], axis=0)

mean_Profit = np.mean(df['Profit'], axis=0)
sd_Profit = np.std(df['Profit'], axis=0)

counter_RD_Spend = 0
counter_Administration = 0
counter_Marketing_Spend = 0
counter_State = 0
counter_Profit = 0

for RD_Spend, Administration, Marketing_Spend, State, Profit in zip(df['R&D Spend'], df['A
dministration'], df['Marketing Spend'], df['State'], df['Profit']):
    if not mean_RD_Spend - 3*sd_RD_Spend <= RD_Spend <= mean_RD_Spend + 3*sd_RD_Spend:
        counter_RD_Spend += 1
    if not mean_Administration - 3*sd_Administration <= Administration <= mean_Administrat
ion + 3*sd_Administration:
        counter_Administration += 1
    if not mean_Marketing_Spend - 3*sd_Marketing_Spend <= counter_Marketing_Spend <= mean_
Marketing_Spend + 3*sd_Marketing_Spend:
        counter_Marketing_Spend += 1
    if not mean_State - 3*sd_State <= counter_State <= mean_State + 3*sd_State:
        counter_State += 1
    if not mean_Profit - 3*sd_Profit <= counter_Profit <= mean_Profit + 3*sd_Profit:
        counter_Profit += 1

counter_dicts = {'counter_RD_Spend': counter_RD_Spend,
                'counter_Administration': counter_Administration,
                'counter_Marketing_Spend': counter_Marketing_Spend,
                'counter_State': counter_State,
                'counter_Profit': counter_Profit}
print(counter_dicts)
```

```
{'counter_actual_power': 0, 'counter_Administration': 0, 'counter_Marketing_S
pend': 0, 'counter_State': 0, 'counter_Profit': 0}
```

In [205]:

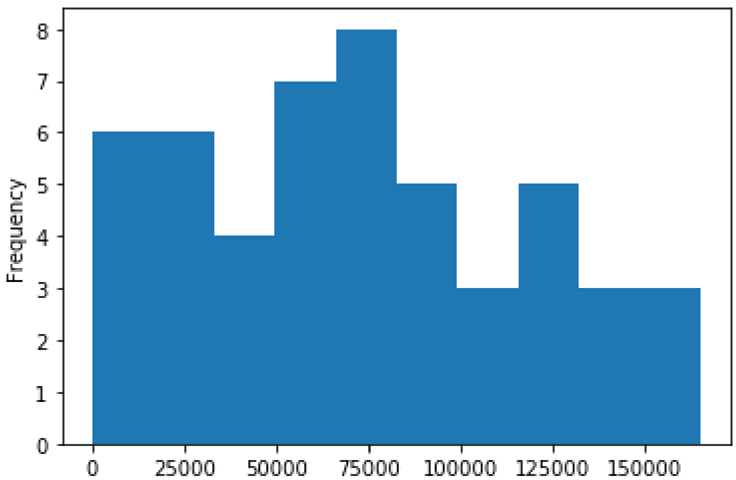
```
# Outliers
RD_Spend = []
for ap in df['R&D Spend']:
    if ap > df['R&D Spend'].mean() + 3 * df['R&D Spend'].std():
        ap = df['R&D Spend'].mean() + 3*df['R&D Spend'].std()
    elif ap < df['R&D Spend'].mean() - 3 * df['R&D Spend'].std():
        ap = df['R&D Spend'].mean() - 3*df['R&D Spend'].std()
    RD_Spend.append(ap)
df['R&D Spend']  = RD_Spend

Administration = []
for m in df['Administration']:
    if m > df['Administration'].mean() + 3 * df['Administration'].std():
        m = df['Administration'].mean() + 3*df['Administration'].std()
    elif m < df['Administration'].mean() - 3 * df['Administration'].std():
        m = df['Administration'].mean() - 3*df['Administration'].std()
    Administration.append(m)
df['Administration']  = Administration

Marketing_Spend = []
for loc in df['Marketing Spend']:
    if loc > df['Marketing Spend'].mean() + 3 * df['Marketing Spend'].std():
        loc = df['Marketing Spend'].mean() + 3*df['Marketing Spend'].std()
    elif loc < df['Marketing Spend'].mean() - 3 * df['Marketing Spend'].std():
        loc = df['Marketing Spend'].mean() - 3*df['Marketing Spend'].std()
    Marketing_Spend.append(loc)
df['Marketing Spend'] = Marketing_Spend

State = []
for loc in df['State']:
    if loc > df['State'].mean() + 3 * df['State'].std():
        loc = df['State'].mean() + 3*df['State'].std()
    elif loc < df['State'].mean() - 3 * df['State'].std():
        loc = df['State'].mean() - 3*df['State'].std()
    State.append(loc)
df['State'] = State

Profit = []
for loc in df['Profit']:
    if loc > df['Profit'].mean() + 3 * df['Profit'].std():
        loc = df['Profit'].mean() + 3*df['Profit'].std()
    elif loc < df['Profit'].mean() - 3 * df['Profit'].std():
        loc = df['Profit'].mean() - 3*df['Profit'].std()
    Profit.append(loc)
df['Profit'] = Profit
```

In [12]:

```
# R&D Spend distribution
df['R&D Spend'].plot(kind = 'hist')
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d145eb4760>
```

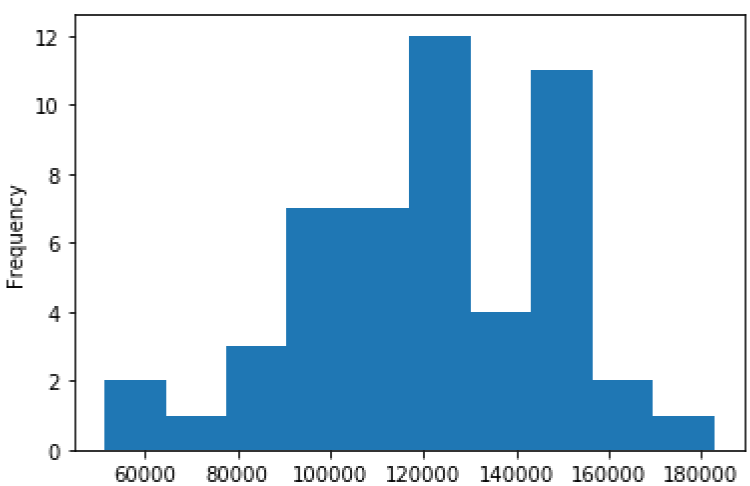


In [15]:

```
# Marketing Spend distribution
df['Administration'].plot(kind = 'hist')
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d14814ce20>
```

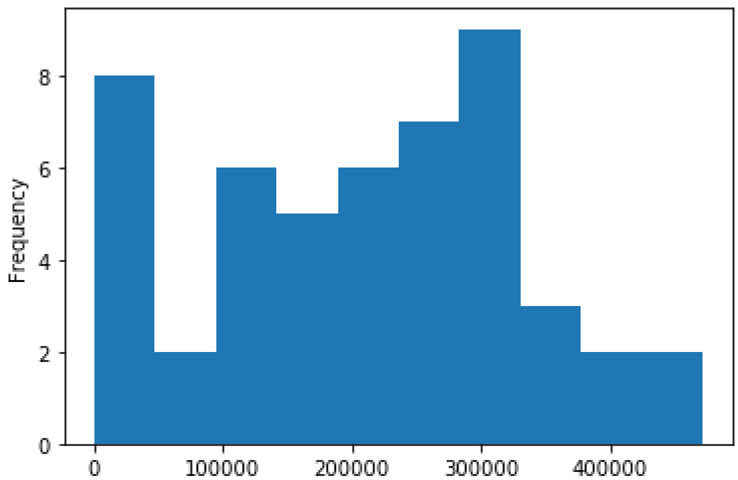


In [17]:

```
# Marketing Spend distribution
df['Marketing Spend'].plot(kind = 'hist')
```

Out[17]:

<matplotlib.axes._subplots.AxesSubplot at 0x2d148232640>

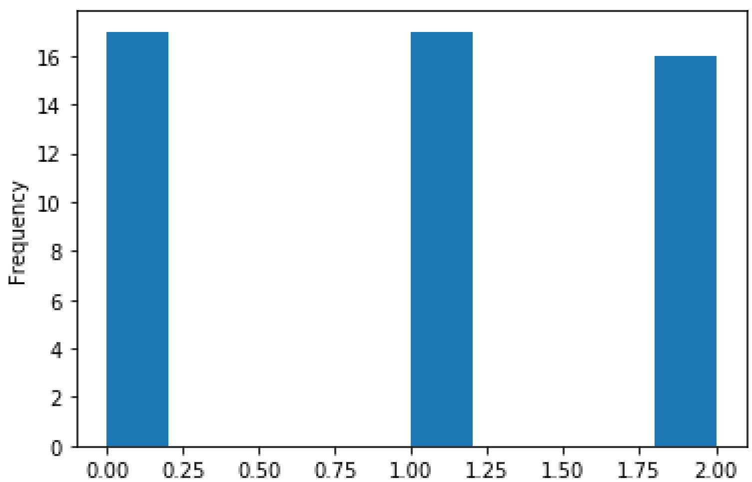


In [18]:

```
# State distribution
df['State'].plot(kind = 'hist')
```

Out[18]:

<matplotlib.axes._subplots.AxesSubplot at 0x2d1482322e0>

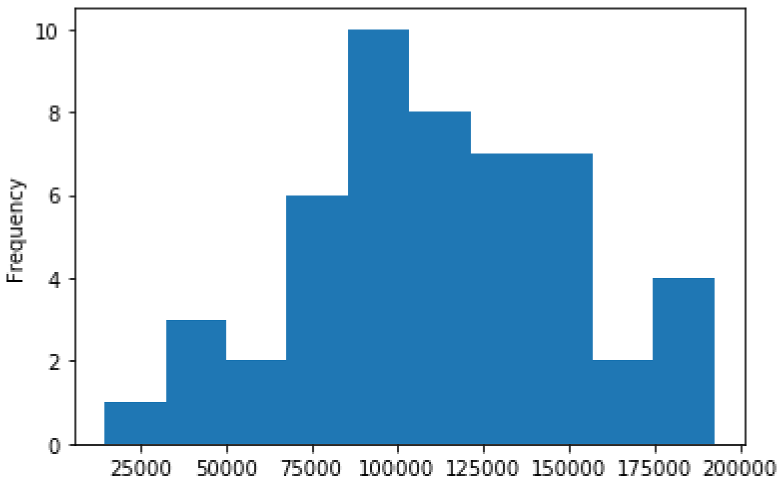


In [20]:

```
# Location 3 distribution
df['Location 3'].plot(kind = 'hist')
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d1482e4f10>
```



In [21]:

```
df.isnull().sum()
# Таким образом мы имеем пропущенные значения в таких колонках:
```

Out[21]:

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

In [22]:

```
# Taking care of missing data
# https://scikit-learn.org/
from sklearn.impute import SimpleImputer
#numeric
df[['R&D Spend']] = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transform(df
[['R&D Spend']]).round()
df[['Administration']] = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transfo
rm(df[['Administration']]).round()
df[['State']] = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transform(df[['S
tate']]).round()
```

In [23]:

```
## Linear Regression
```

In [26]:

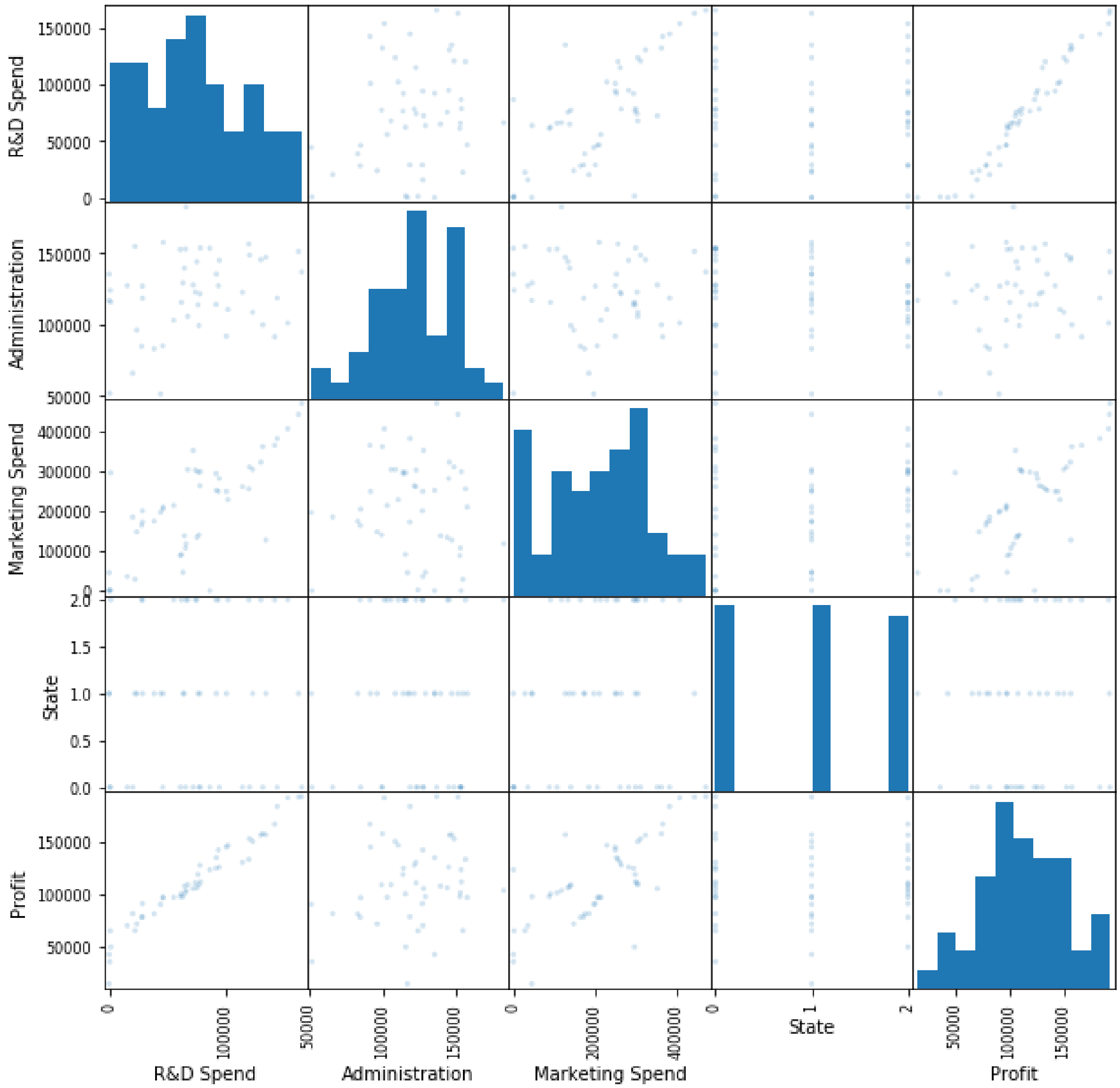
```
# Cheking correlations
correlation = df.corr()
correlation.style.background_gradient(cmap='coolwarm')
```

Out[26]:

	R&D Spend	Administration	Marketing Spend	State	Profit
R&D Spend	1.000000	0.241957	0.724249	0.037930	0.972901
Administration	0.241957	1.000000	-0.032151	0.003026	0.200719
Marketing Spend	0.724249	-0.032151	1.000000	0.137777	0.747766
State	0.037930	0.003026	0.137777	1.000000	0.048471
Profit	0.972901	0.200719	0.747766	0.048471	1.000000

In [27]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, alpha=0.2, figsize=(10, 10))
plt.show()
```


R&D Spend
Administration
Marketing Spend
State
Profit

In [59]:

```
# Splitting the dataset into the Training set and Test set
X = df.iloc[:, 0:4].values
y = df.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Regression Neural Network

In [42]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
dfsc = sc.fit_transform(df)
df['R&D Spend'] = dfsc[:,0]
df['Administration'] = dfsc[:,1]
df['Marketing Spend'] = dfsc[:,2]
df['State'] = dfsc[:,3]
df['Profit'] = dfsc[:,4]
```

In [43]:

```
# Cheking correlations
df.corr()
```

Out[43]:

	R&D Spend	Administration	Marketing Spend	State	Profit
R&D Spend	1.000000	0.241957	0.724249	0.037930	0.972901
Administration	0.241957	1.000000	-0.032151	0.003026	0.200719
Marketing Spend	0.724249	-0.032151	1.000000	0.137777	0.747766
State	0.037930	0.003026	0.137777	1.000000	0.048471
Profit	0.972901	0.200719	0.747766	0.048471	1.000000

In [45]:

```
# Install Tensorflow
# Install Keras
# Importing the Keras libraries and packages
# !pip3 install keras
# !pip install tensorflow
import keras
from keras.models import Sequential
from keras.layers import Dense
```

In [52]:

```
# Initialising the ANN
rnn = Sequential()

# Adding the input layer and the first hidden layer
rnn.add(Dense(units = 4, activation = 'tanh', input_dim = 4))

# Adding the second hidden layer
rnn.add(Dense(units = 3, activation = 'tanh'))

# Adding the output layer
rnn.add(Dense(units = 1, activation = 'linear'))

# Compiling the ANN
rnn.compile(optimizer='adam', loss='mean_squared_error', metrics = ['accuracy'])
```

In [53]:

```
# Fitting the ANN to the Training set
rnn.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

```
Epoch 1/100
4/4 [==============================] - 0s 1ms/step - loss: 0.7222 - accuracy:
0.0000e+00
Epoch 2/100
4/4 [==============================] - 0s 997us/step - loss: 0.7029 - accurac
y: 0.0000e+00
Epoch 3/100
4/4 [==============================] - 0s 1ms/step - loss: 0.6891 - accuracy:
0.0000e+00
Epoch 4/100
4/4 [==============================] - 0s 1ms/step - loss: 0.6764 - accuracy:
0.0000e+00
Epoch 5/100
4/4 [==============================] - 0s 2ms/step - loss: 0.6619 - accuracy:
0.0000e+00
Epoch 6/100
4/4 [==============================] - 0s 1ms/step - loss: 0.6469 - accuracy:
0.0000e+00
Epoch 7/100
4/4 [==============================] - 0s 997us/step - loss: 0.6333 - accurac
y: 0.0000e+00
Epoch 8/100
4/4 [==============================] - 0s 2ms/step - loss: 0.6195 - accuracy:
0.0000e+00
Epoch 9/100
4/4 [==============================] - 0s 2ms/step - loss: 0.6069 - accuracy:
0.0000e+00
Epoch 10/100
4/4 [==============================] - 0s 997us/step - loss: 0.5936 - accurac
y: 0.0000e+00
Epoch 11/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5816 - accuracy:
0.0000e+00
Epoch 12/100
4/4 [==============================] - 0s 1ms/step - loss: 0.5706 - accuracy:
0.0000e+00
Epoch 13/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5557 - accuracy:
0.0000e+00
Epoch 14/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5446 - accuracy:
0.0000e+00
Epoch 15/100
4/4 [==============================] - 0s 1ms/step - loss: 0.5342 - accuracy:
0.0000e+00
Epoch 16/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5222 - accuracy:
0.0000e+00
Epoch 17/100
4/4 [==============================] - 0s 1ms/step - loss: 0.5108 - accuracy:
0.0000e+00
Epoch 18/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5005 - accuracy:
0.0000e+00
Epoch 19/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4910 - accuracy:
0.0000e+00
```

```
Epoch 20/100
4/4 [==============================] - 0s 998us/step - loss: 0.4807 - accurac
y: 0.0000e+00
Epoch 21/100
4/4 [==============================] - 0s 1ms/step - loss: 0.4713 - accuracy:
0.0000e+00
Epoch 22/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4605 - accuracy:
0.0000e+00
Epoch 23/100
4/4 [==============================] - 0s 1ms/step - loss: 0.4509 - accuracy:
0.0000e+00
Epoch 24/100
4/4 [==============================] - 0s 998us/step - loss: 0.4428 - accurac
y: 0.0000e+00
Epoch 25/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4339 - accuracy:
0.0000e+00
Epoch 26/100
4/4 [==============================] - 0s 1ms/step - loss: 0.4258 - accuracy:
0.0000e+00
Epoch 27/100
4/4 [==============================] - 0s 1ms/step - loss: 0.4171 - accuracy:
0.0000e+00
Epoch 28/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4100 - accuracy:
0.0000e+00
Epoch 29/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4018 - accuracy:
0.0000e+00
Epoch 30/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3935 - accuracy:
0.0000e+00
Epoch 31/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3866 - accuracy:
0.0000e+00
Epoch 32/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3802 - accuracy:
0.0000e+00
Epoch 33/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3725 - accuracy:
0.0000e+00
Epoch 34/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3657 - accuracy:
0.0000e+00
Epoch 35/100
4/4 [==============================] - 0s 998us/step - loss: 0.3606 - accurac
y: 0.0000e+00
Epoch 36/100
4/4 [==============================] - 0s 968us/step - loss: 0.3536 - accurac
y: 0.0000e+00
Epoch 37/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3476 - accuracy:
0.0000e+00
Epoch 38/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3416 - accuracy:
0.0000e+00
```

```
Epoch 39/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3355 - accuracy:
0.0000e+00
Epoch 40/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3311 - accuracy:
0.0000e+00
Epoch 41/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3259 - accuracy:
0.0000e+00
Epoch 42/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3207 - accuracy:
0.0000e+00
Epoch 43/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3149 - accuracy:
0.0000e+00
Epoch 44/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3112 - accuracy:
0.0000e+00
Epoch 45/100
4/4 [==============================] - 0s 4ms/step - loss: 0.3075 - accuracy:
0.0000e+00
Epoch 46/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3021 - accuracy:
0.0000e+00
Epoch 47/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2973 - accuracy:
0.0000e+00
Epoch 48/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2941 - accuracy:
0.0000e+00
Epoch 49/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2901 - accuracy:
0.0000e+00
Epoch 50/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2857 - accuracy:
0.0000e+00
Epoch 51/100
4/4 [==============================] - ETA: 0s - loss: 0.3418 - accuracy: 0.0
000e+ - 0s 2ms/step - loss: 0.2825 - accuracy: 0.0000e+00
Epoch 52/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2789 - accuracy:
0.0000e+00
Epoch 53/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2756 - accuracy:
0.0000e+00
Epoch 54/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2722 - accuracy:
0.0000e+00
Epoch 55/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2692 - accuracy:
0.0000e+00
Epoch 56/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2667 - accuracy:
0.0000e+00
Epoch 57/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2629 - accuracy:
0.0000e+00
```

```
Epoch 58/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2601 - accuracy:
0.0000e+00
Epoch 59/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2572 - accuracy:
0.0000e+00
Epoch 60/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2548 - accuracy:
0.0000e+00
Epoch 61/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2522 - accuracy:
0.0000e+00
Epoch 62/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2494 - accuracy:
0.0000e+00
Epoch 63/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2471 - accuracy:
0.0000e+00
Epoch 64/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2446 - accuracy:
0.0000e+00
Epoch 65/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2428 - accuracy:
0.0000e+00
Epoch 66/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2404 - accuracy:
0.0000e+00
Epoch 67/100
4/4 [==============================] - ETA: 0s - loss: 0.2241 - accuracy: 0.0
000e+ - 0s 2ms/step - loss: 0.2377 - accuracy: 0.0000e+00
Epoch 68/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2354 - accuracy:
0.0000e+00
Epoch 69/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2337 - accuracy:
0.0000e+00
Epoch 70/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2312 - accuracy:
0.0000e+00
Epoch 71/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2292 - accuracy:
0.0000e+00
Epoch 72/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2275 - accuracy:
0.0000e+00
Epoch 73/100
4/4 [==============================] - 0s 995us/step - loss: 0.2254 - accurac
y: 0.0000e+00
Epoch 74/100
4/4 [==============================] - ETA: 0s - loss: 0.1099 - accuracy: 0.0
000e+ - 0s 1ms/step - loss: 0.2237 - accuracy: 0.0000e+00
Epoch 75/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2218 - accuracy:
0.0000e+00
Epoch 76/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2199 - accuracy:
0.0000e+00
```

```
Epoch 77/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2182 - accuracy:
0.0000e+00
Epoch 78/100
4/4 [==============================] - 0s 997us/step - loss: 0.2163 - accurac
y: 0.0000e+00
Epoch 79/100
4/4 [==============================] - 0s 997us/step - loss: 0.2150 - accurac
y: 0.0000e+00
Epoch 80/100
4/4 [==============================] - 0s 997us/step - loss: 0.2131 - accurac
y: 0.0000e+00
Epoch 81/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2115 - accuracy:
0.0000e+00
Epoch 82/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2100 - accuracy:
0.0000e+00
Epoch 83/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2084 - accuracy:
0.0000e+00
Epoch 84/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2067 - accuracy:
0.0000e+00
Epoch 85/100
4/4 [==============================] - 0s 3ms/step - loss: 0.2054 - accuracy:
0.0000e+00
Epoch 86/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2037 - accuracy:
0.0000e+00
Epoch 87/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2022 - accuracy:
0.0000e+00
Epoch 88/100
4/4 [==============================] - 0s 997us/step - loss: 0.2008 - accurac
y: 0.0000e+00
Epoch 89/100
4/4 [==============================] - 0s 997us/step - loss: 0.1992 - accurac
y: 0.0000e+00
Epoch 90/100
4/4 [==============================] - 0s 986us/step - loss: 0.1982 - accurac
y: 0.0000e+00
Epoch 91/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1966 - accuracy:
0.0000e+00
Epoch 92/100
4/4 [==============================] - 0s 1ms/step - loss: 0.1952 - accuracy:
0.0000e+00
Epoch 93/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1941 - accuracy:
0.0000e+00
Epoch 94/100
4/4 [==============================] - 0s 997us/step - loss: 0.1923 - accurac
y: 0.0000e+00
Epoch 95/100
4/4 [==============================] - 0s 1ms/step - loss: 0.1914 - accuracy:
0.0000e+00
```

```
Epoch 96/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1900 - accuracy:
0.0000e+00
Epoch 97/100
4/4 [==============================] - 0s 1ms/step - loss: 0.1887 - accuracy:
0.0000e+00
Epoch 98/100
4/4 [==============================] - 0s 1ms/step - loss: 0.1873 - accuracy:
0.0000e+00
Epoch 99/100
4/4 [==============================] - 0s 998us/step - loss: 0.1862 - accurac
y: 0.0000e+00
Epoch 100/100
4/4 [==============================] - 0s 1ms/step - loss: 0.1849 - accuracy:
0.0000e+00
```

Out[53]:

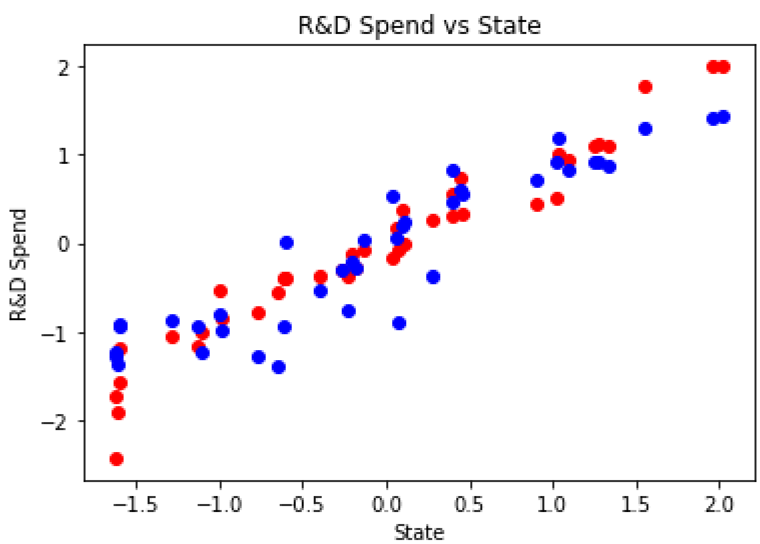
```
<tensorflow.python.keras.callbacks.History at 0x2d1578413a0>
```

In [63]:

```
# Predicting the Test set results
y_pred = rnn.predict(X_test)
```

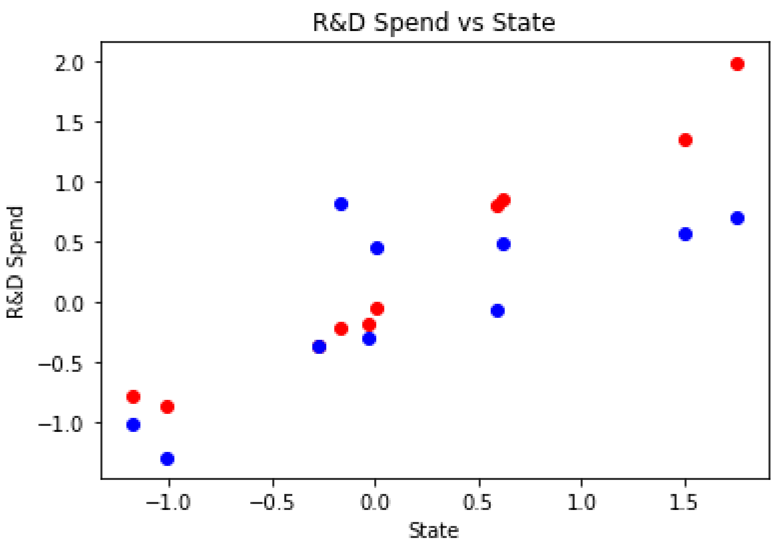
In [65]:

```
# Visualising the Training set results
plt.scatter(X_train[:, 0:1], y_train, color = 'red')
plt.plot(X_train[:, 0:1], rnn.predict(X_train), 'bo')
plt.title('R&D Spend vs State')
plt.xlabel('State')
plt.ylabel('R&D Spend')
plt.show()
```



In [66]:

```
# Visualising the Test set results
plt.scatter(X_test[:, 0:1], y_test, color = 'red')
plt.plot(X_test[:, 0:1], rnn.predict(X_test), 'bo')
plt.title('R&D Spend vs State')
plt.xlabel('State')
plt.ylabel('R&D Spend')
plt.show()
```



In [71]:

```
y_pred_train = rnn.predict(X_train)
y_pred_test = rnn.predict(X_test)
train_mse_nn = sum((y_train - y_pred_train)**2 for y_train, y_pred_train in zip(y_train, y
_pred_train)) / len(y_train)
test_mse_nn = sum((y_test - y_pred_test)**2 for y_test, y_pred_test in zip(y_test, y_pred_
test)) / len(y_test)
print(f"train_mse_nn: {train_mse_nn}, test_mse_nn: {test_mse_nn}")
```

```
train_mse_nn: [0.1840179], test_mse_nn: [0.47062993]
```

Вывод: исходя из величины mse и для тестовой, и для тренировочной выборок, можно утверждать, что модель Regression Neural Network хорошо справилась с задачей прогнозирования.