Debashis Saha

DATE: 06 JUNE 2021

Assignment 8 :- Multi Linear Regression

Registration ID : GO_STP_13248

Task - Predicting a Startups Profit/Success Rate using Multiple Linear Regression in Python

Here 50 startups dataset containing 5 columns: “R&D Spend”, “Administration”, “Marketing Spend”, “State”, “Profit”

In this dataset first 3 columns provides you spending on Research , Administration and Marketing respectively. State indicates startup based on that state. Profit indicates how much profits earned by a startup.

Clearly, we can understand that it is a multiple linear regression problem, as the independent variables are more than one. Prepare a prediction model for profit of 50_Startups data in Python

Importing necessary Libraries

In [70]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Reading the CSV file

In [71]:
```
df = pd.read_csv("50_Startups.csv")
df.head(5)
```

Out[71]:

	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

Checking how many null values are there in the datasets

In [72]:
```
df.isna().sum()
```

Out[72]:
```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

Shape of Dataset

In [73]:
```
df.shape
```

Out[73]: (50, 5)

In [74]:
```
df.dtypes
```

Out[74]:
```
R&D Spend          float64
Administration     float64
Marketing Spend    float64
State               object
Profit             float64
dtype: object
```

In [75]:
```
df.size
```

Out[75]: 250

Descriptive Analysis of the dataset using describe( ) function

In [76]:
```
df.describe()
```

Out[76]:

	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

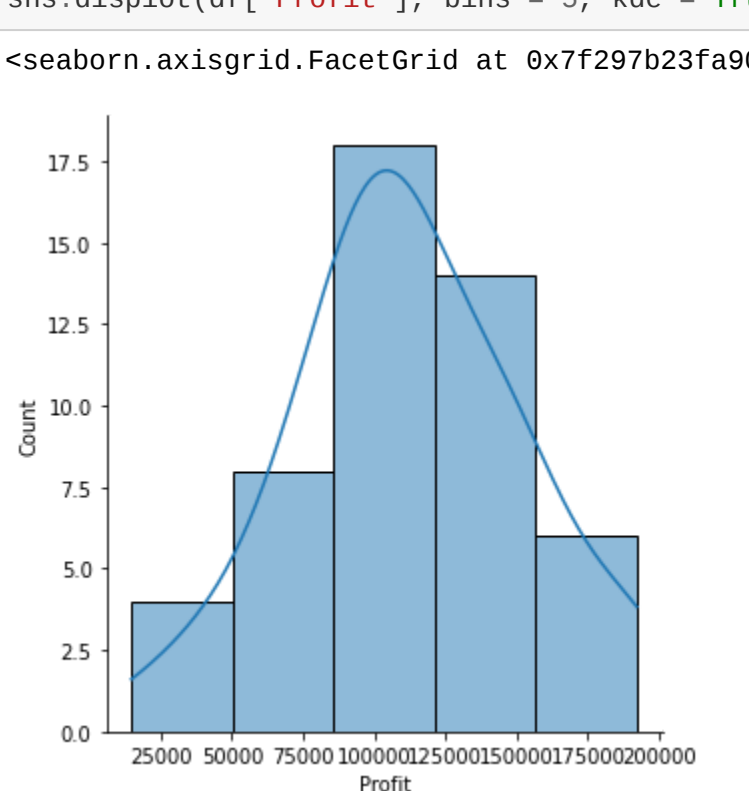
Basic Imformation about the Dataset

In [77]:
```
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 [78]:
```
sns.displot(df["Profit"], bins = 5, kde = True)
```

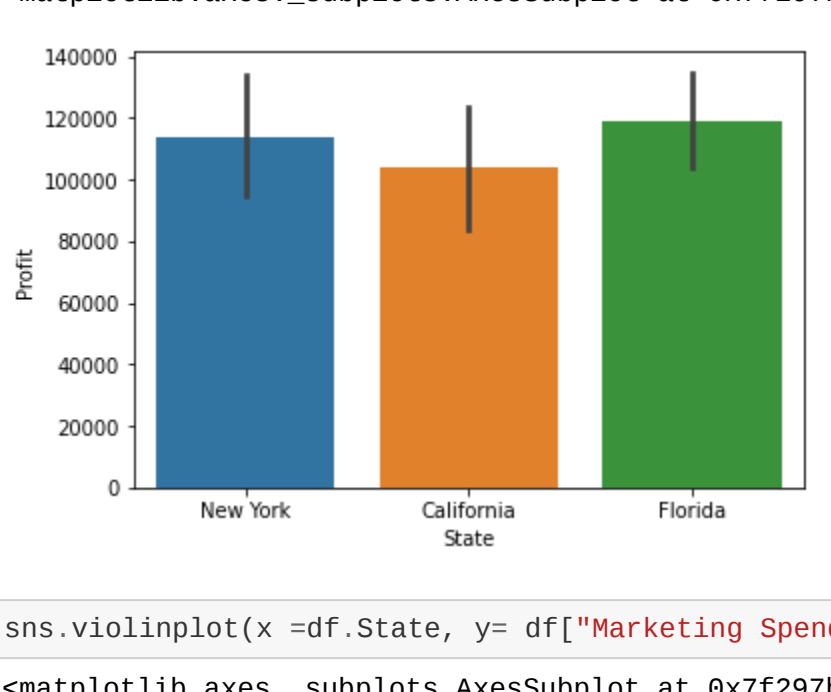
Out[78]: <seaborn.axisgrid.FacetGrid at 0x7f297b23fa90>



Exploratory Data Analysis

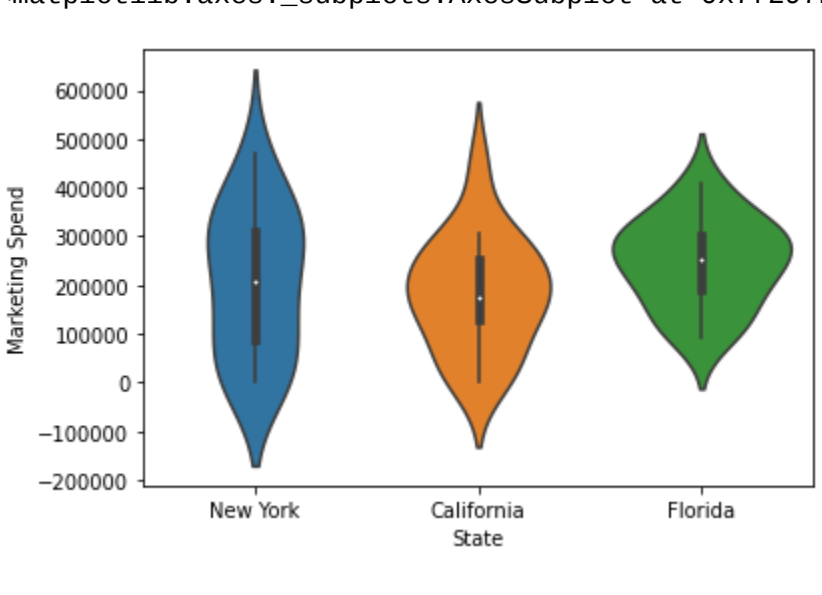
In [79]:
```
sns.barplot(x = "State", y= "Profit", data = df)
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297b2ed990>



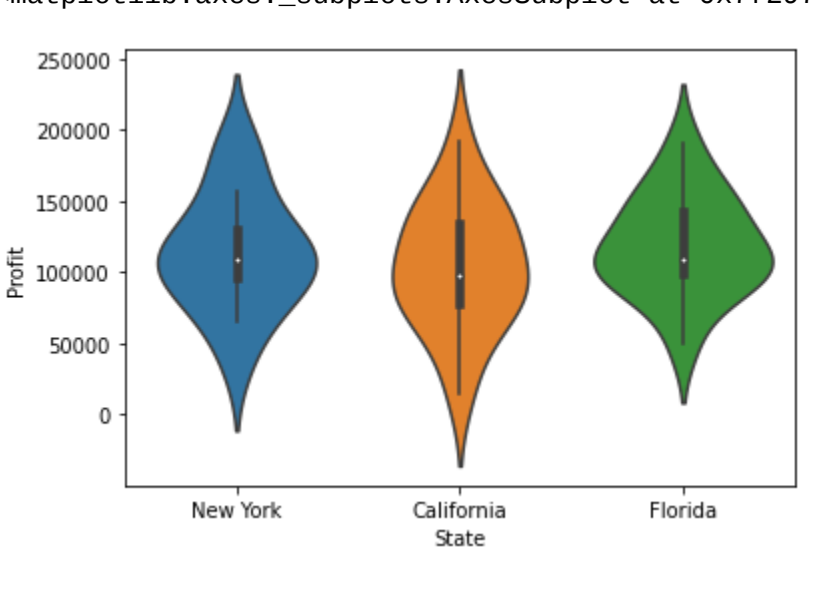
In [80]:
```
sns.violinplot(x =df.State, y= df["Marketing Spend"])
```

Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297b178ed0>



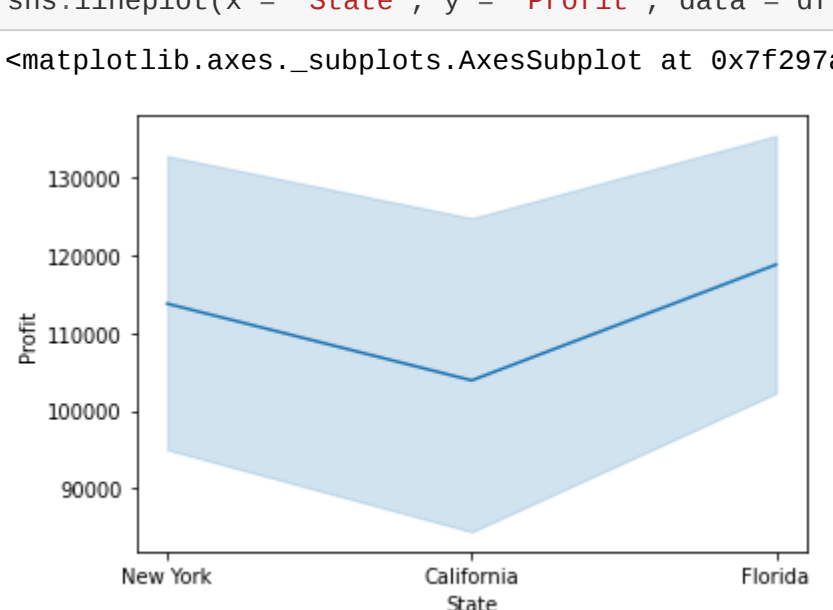
In [81]:
```
sns.violinplot(x = df.State, y = df["Profit"])
```

Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297b059910>



In [82]:
```
sns.lineplot(x = "State", y = "Profit", data = df)
```

Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297affa150>



Conclusion : Correlation and Heatmap Visulaization

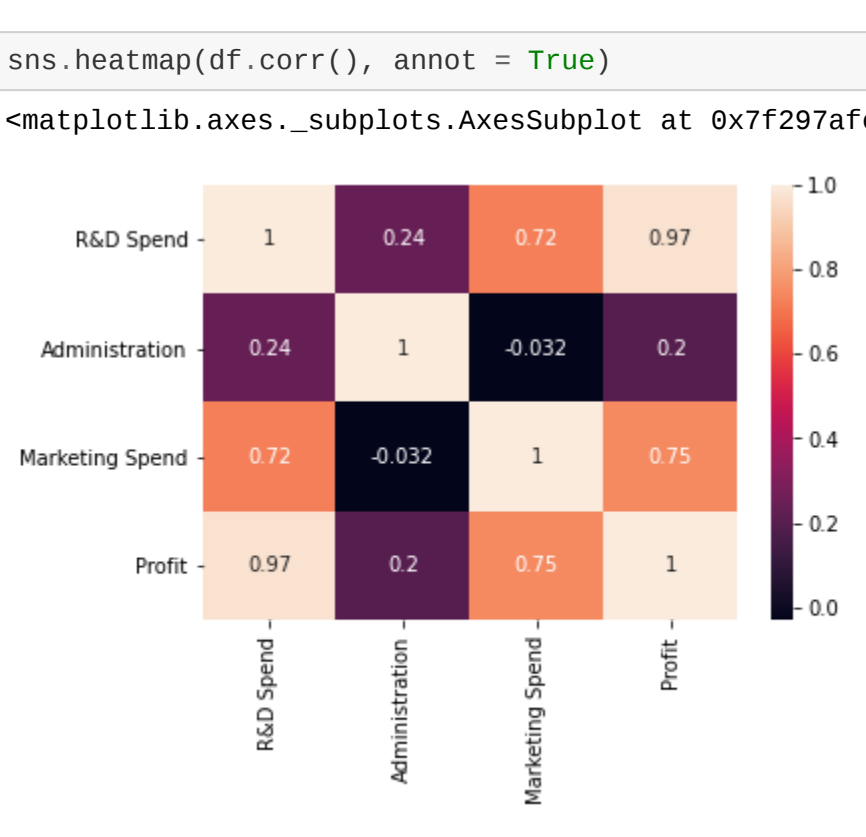
In [83]:
```
df.corr()
```

Out[83]:

	R&D Spend	Administration	Marketing Spend	Profit
R&D Spend	1.000000	0.241955	0.724248	0.972900
Administration	0.241955	1.000000	-0.032154	0.200717
Marketing Spend	0.724248	-0.032154	1.000000	0.747766
Profit	0.972900	0.200717	0.747766	1.000000

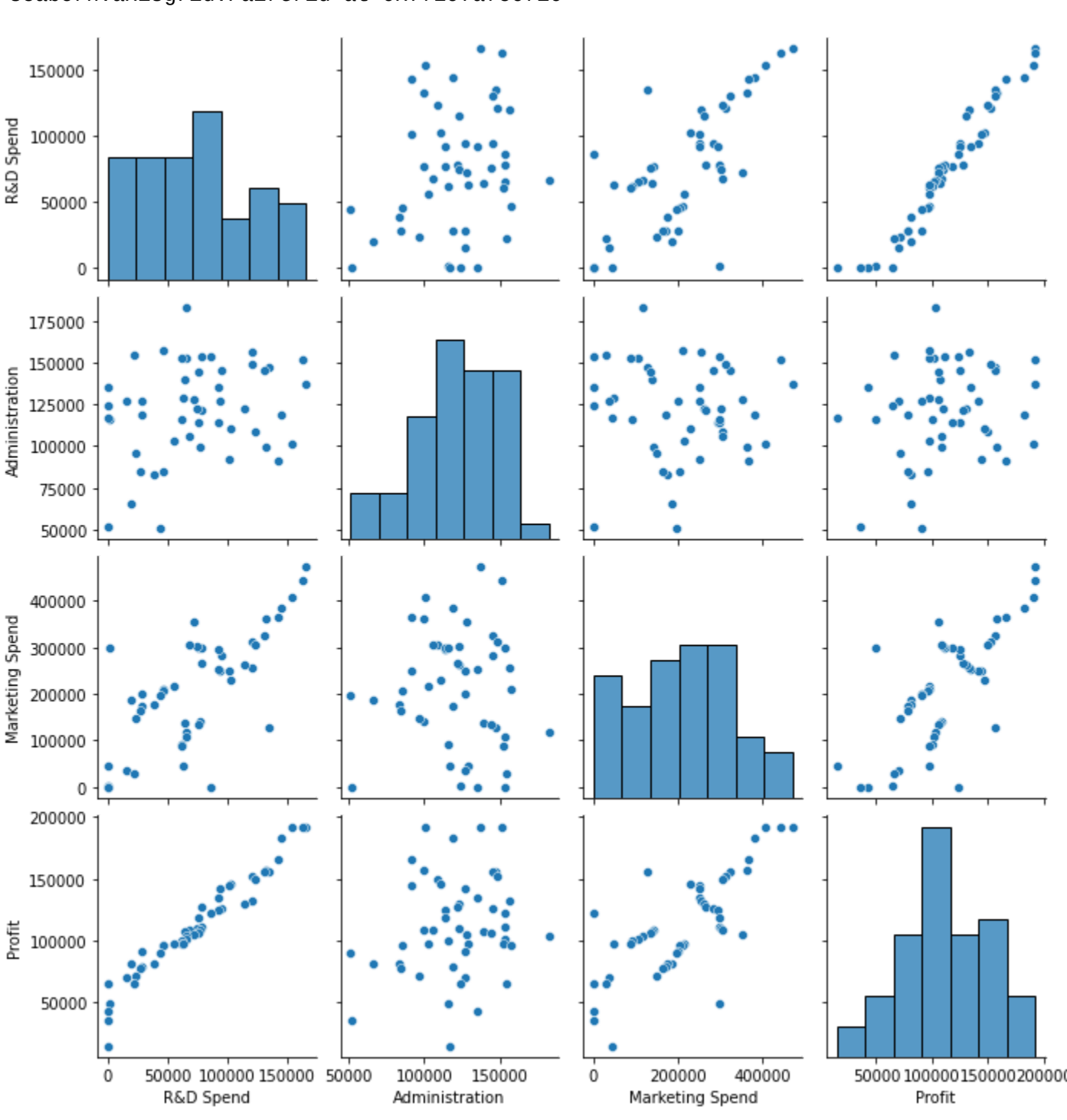
In [84]:
```
sns.heatmap(df.corr(), annot = True)
```

Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297afcd710>



In [85]:
```
sns.pairplot(df)
```

Out[85]: <seaborn.axisgrid.PairGrid at 0x7f297afe6f10>

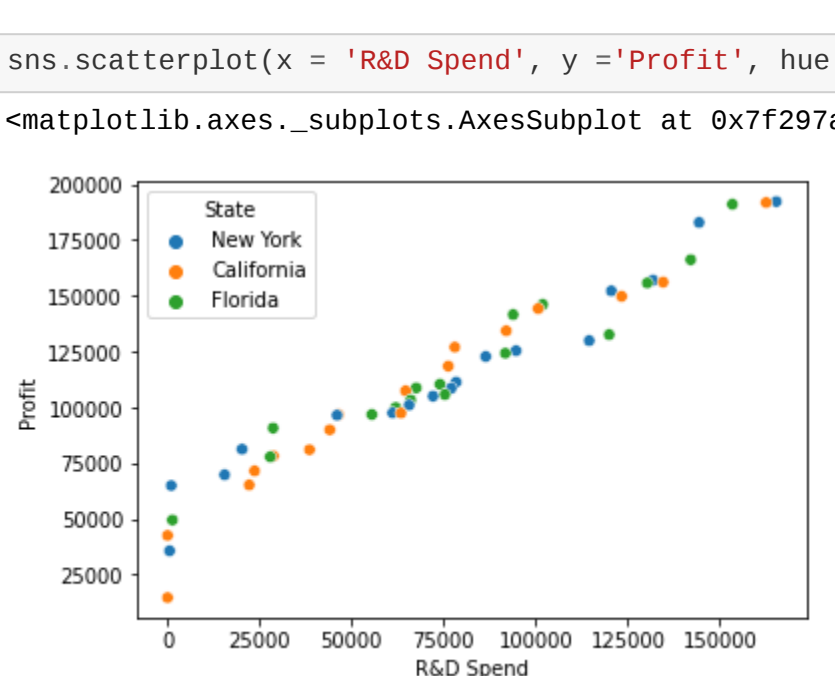


In [86]:
```
df.columns
```

Out[86]: Index(['R&D Spend', 'Administration', 'Marketing Spend', 'State', 'Profit'], dtype='object')

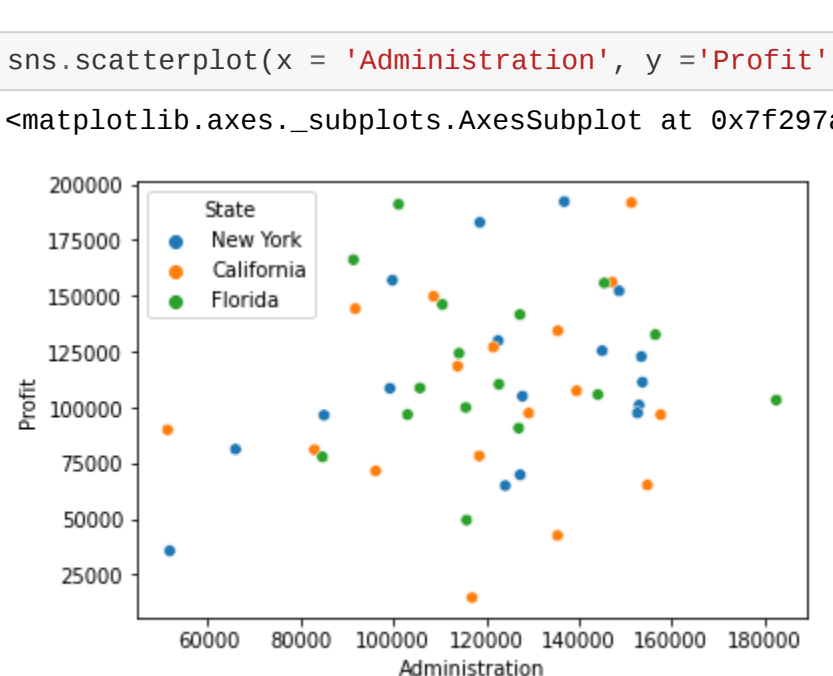
In [87]:
```
sns.scatterplot(x = 'R&D Spend', y ='Profit', hue = "State", data = df)
```

Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297a865b10>



In [88]:
```
sns.scatterplot(x = 'Administration', y ='Profit', hue = "State", data = df)
```

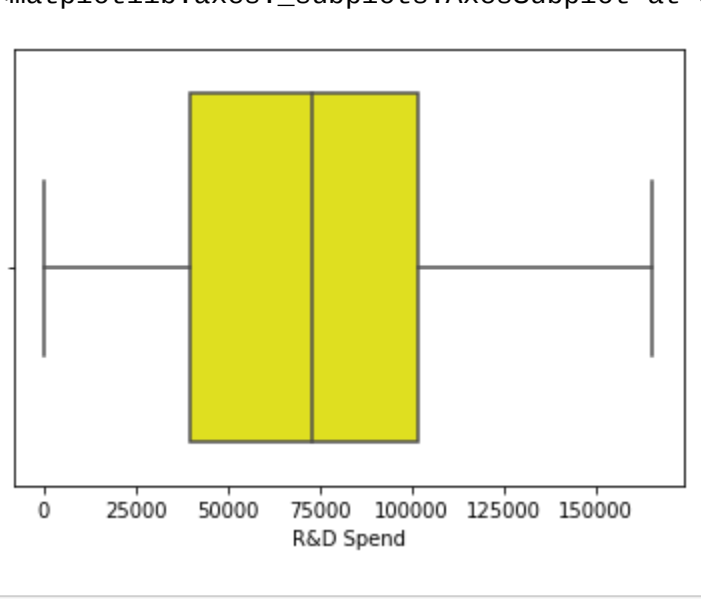
Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297a7f3e90>



In [89]:
```
sns.boxplot(df['R&D Spend'], color = "Yellow")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

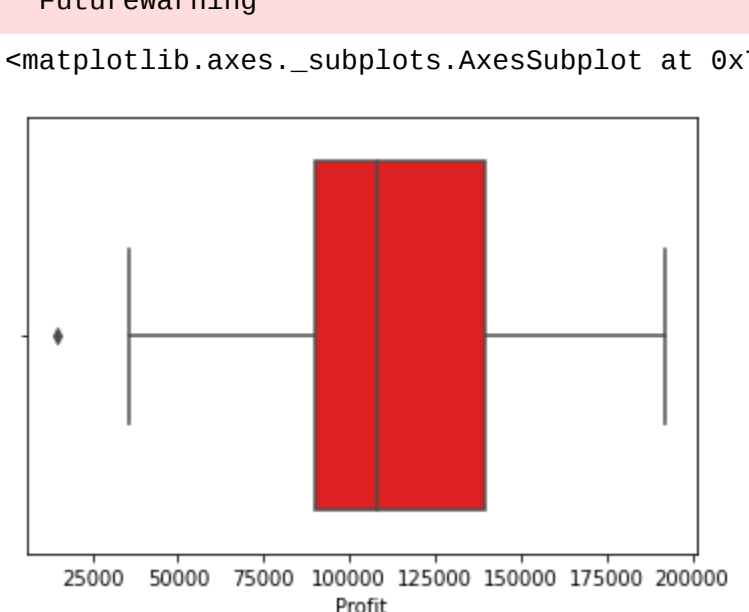
Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297a77c290>



In [90]:
```
sns.boxplot(df["Profit"], color = "Red")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x7f297a6e3fd0>



In [91]:
```
df_new = pd.get_dummies(df, columns = ["State"])
df_new.head()
```

Out[91]:

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

In [92]:
```
df_new.columns
```

Out[92]:
```
Index(['R&D Spend', 'Administration', 'Marketing Spend', 'Profit',
       'State_California', 'State_Florida', 'State_New York'],
      dtype='object')
```

Training a Regressional Model

In [93]:
```
X = df_new[['R&D Spend', 'Administration', 'Marketing Spend',
       'State_California', 'State_Florida', 'State_New York']]
y = df_new[ 'Profit']
```

Train and Test Split

In [94]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, random_state  = 95)
```

Creating and Training the model

In [95]:
```
from sklearn.linear_model import LinearRegression
lin = LinearRegression(normalize = True, n_jobs = -1)
lin.fit(X_train, y_train)
```

Out[95]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=True)

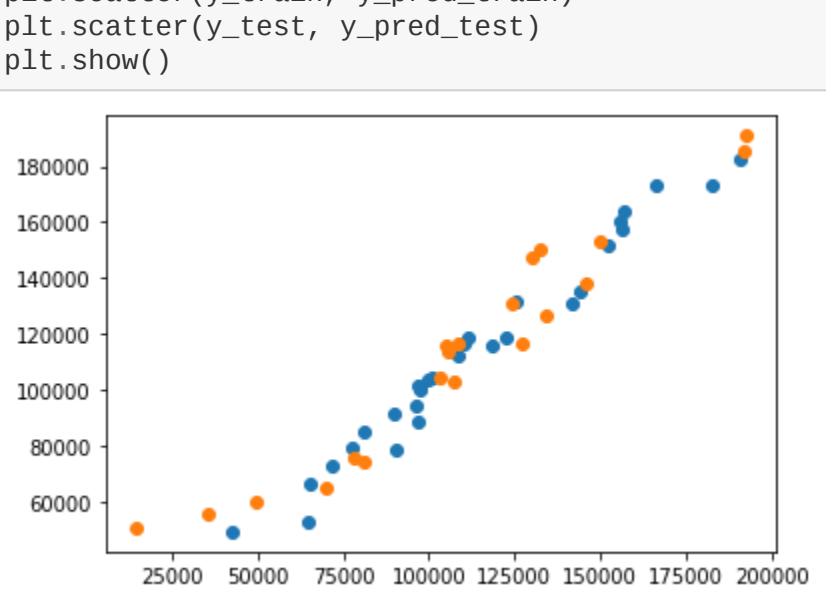
Model Evaluation

In [96]:
```
print(lin.coef_)
print(lin.intercept_)
```

```
[ 7.86174104e-01 -4.50702791e-02  1.96026630e-02 -1.73255032e+03
  9.84138266e+02  7.48412054e+02]
57154.197612032585
```

In [97]:
```
y_pred_train = lin.predict(X_train)
y_pred_test = lin.predict(X_test)
```

In [98]:
```
plt.scatter(y_train, y_pred_train)
plt.scatter(y_test, y_pred_test)
plt.show()
```



Prediction from Model

In [99]:
```
reg = pd.DataFrame(y_pred_test, y_test)
print(reg.head())
```

```
                       0
Profit
103282.38  104150.463404
149759.96  153464.587453
78239.91    76071.908640
107404.34  102674.102603
146121.95  137767.417696
```

Evaluation Metrics

In [100]:
```
from sklearn import metrics
print("MAE :",metrics.mean_absolute_error(y_test, y_pred_test))
print("MSE :", metrics.mean_squared_error(y_test, y_pred_test))
print("RMSE :", np.sqrt(metrics.mean_squared_error(y_test, y_pred_test)))
```

```
MAE : 9661.91629110497
MSE : 157547484.90007424
RMSE : 12551.79209914163
```

In [101]:
```
from sklearn.metrics import r2_score
score = r2_score(y_test, y_pred_test)
print("R2_score is ", score)
```

```
R2_score is  0.9215191980170655
```

In [102]:
```
!pip install nbconvert
```

```
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-packages (5.6.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.7/dist-packages (from nbconvert) (0.3)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-packages (from nbconvert) (0.7.1)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages (from nbconvert) (3.3.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-packages (from nbconvert) (2.6.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.7/dist-packages (from nbconvert) (1.4.3)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-packages (from nbconvert) (4.7.1)
Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.7/dist-packages (from nbconvert) (2.11.3)
Requirement already satisfied: nbformat>=4.4 in /usr/local/lib/python3.7/dist-packages (from nbconvert) (5.1.3)
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-packages (from nbconvert) (0.5.0)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packages (from nbconvert) (5.0.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from bleach->nbconvert) (20.9)
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from bleach->nbconvert) (1.15.0)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2>=2.4->nbconvert) (2.0.1)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.7/dist-packages (from nbformat>=4.4->nbconvert) (0.2.0)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /usr/local/lib/python3.7/dist-packages (from nbformat>=4.4->nbconvert) (2.6.0)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->bleach->nbconvert) (2.4.7)
```

In [104]:
```
!jupyter nbconvert --to html test.ipynb
```

```
[NbConvertApp] WARNING | pattern u'test.ipynb' matched no files
This application is used to convert notebook files (*.ipynb) to various other
formats.

WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
-------

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the cell output (the default behaviour is to abort conversion). This flag is only relevant if '--execute' was specified, too.
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--stdout
    Write notebook output to stdout instead of files.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
-y
    Answer yes to any questions instead of prompting.
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--debug
    set log level to logging.DEBUG (maximize logging output)
--no-prompt
    Exclude input and output prompts from converted document.
--generate-config
    generate default config file
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    Directory to write output(s) to. Defaults to output to the directory of each
    notebook. To recover previous default behaviour (outputting to the current
    working directory) use . as the flag value.
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the  results of the conversion
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL')
    Set the log level by value or name.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
    Default: u''
    The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN,
    but can be any url pointing to a copy  of reveal.js.
    For speaker notes to work, this must be a relative path to a local  copy of
    reveal.js: e.g., "reveal.js".
    If a relative path is given, it must be a subdirectory of the current
    directory (from which the server is run).
    See the usage documentation
    (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
    slideshow) for more details.
--to=<Unicode> (NbConvertApp.export_format)
    Default: 'html'
    The export format to be used, either one of the built-in formats
    ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
    'python', 'rst', 'script', 'slides'] or a dotted object name that represents
    the import path for an `Exporter` class
--template=<Unicode> (TemplateExporter.template_file)
    Default: u''
    Name of the template file to use
--output=<Unicode> (NbConvertApp.output_base)
    Default: ''
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
--post=<DottedOrNone> (NbConvertApp.postprocessor_class)
    Default: u''
    PostProcessor class used to write the results of the conversion
--config=<Unicode> (JupyterApp.config_file)
    Default: u''
    Full path of a config file.

To see all available configurables, use `--help-all`

Examples
--------

    The simplest way to use nbconvert is

    > jupyter nbconvert mynotebook.ipynb

    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX includes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
    can specify the flavor of the format used.

    > jupyter nbconvert --to html --template basic mynotebook.ipynb

    You can also pipe the output to stdout, rather than a file

    > jupyter nbconvert mynotebook.ipynb --stdout

    PDF is generated via latex

    > jupyter nbconvert mynotebook.ipynb --to pdf

    You can get (and serve) a Reveal.js-powered slideshow

    > jupyter nbconvert myslides.ipynb --to slides --post serve

    Multiple notebooks can be given at the command line in a couple of
    different ways:

    > jupyter nbconvert notebook*.ipynb
    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

    or you can specify the notebooks list in a config file, containing::

        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

    > jupyter nbconvert --config mycfg.py
```