## Today topics:

- Polynomial Features
- overfiting,underfitin,bestfit
- use cases of linear regression

## Polynomial Regression

Polynomial Regression is used for apply the non-linear datasets

In [1]:
```
import pandas as pd
import numpy as np
d={"empexp":[1,2,3,4,5,6,7,8,9],"sal":[500,650,400,800,950,1000,1550,1730,2000]}
df=pd.DataFrame(d)
df.head()
```

Out[1]:

| | empexp | sal |
|---|---|---|
| 0 | 1 | 500 |
| 1 | 2 | 650 |
| 2 | 3 | 400 |
| 3 | 4 | 800 |
| 4 | 5 | 950 |

In [2]:
```
X=df[['empexp']]
y=df[['sal']]
```

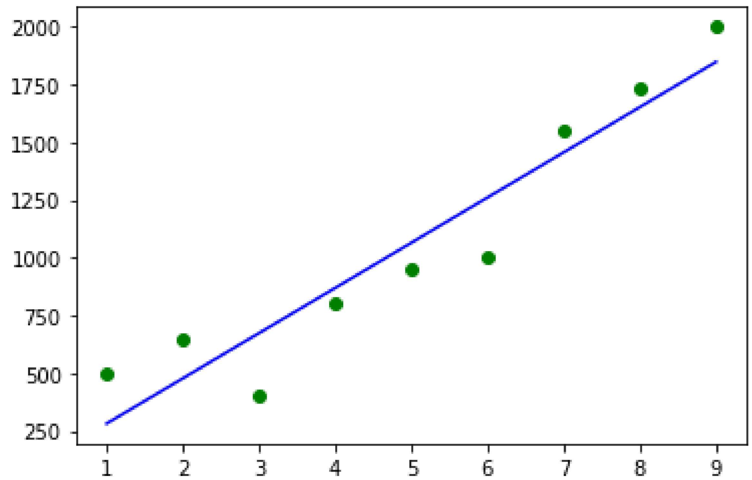
In [3]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X,y)
```

Out[3]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [4]:
```
model.score(X,y)
```

Out[4]:
```
0.8927737377575873
```

In [6]:
```
# Data visulization
import matplotlib.pyplot as plt
plt.scatter(X,y,color='g')
plt.plot(X,model.predict(X),color='b')
plt.show()
```




In [7]:
```
# find the error (distance between actual values and predicted values)

from sklearn.metrics import mean_absolute_error
y_pred=model.predict(X)
mean_absolute_error(y,y_pred)
```

Out[7]: 159.20987654320987

In [8]:
```
model.predict([[4]])
```

Out[8]: array([[868.77777778]])

## Polynomial regression

- y=mx+c
- y=m1x1+m2x2+m3x3+.....+c
- y=m1x1+m1x1^2+m1x1^3+....+mn+x1^n+c

In [32]:
```
from sklearn.preprocessing import PolynomialFeatures
p1=PolynomialFeatures(degree=7)  # y=m1x1+m1x1^2+m1x1^3+c
p_x=p1.fit_transform(X)
p_x
```

Out[32]:
```
array([[1.000000e+00, 1.000000e+00, 1.000000e+00, 1.000000e+00,
        1.000000e+00, 1.000000e+00, 1.000000e+00, 1.000000e+00],
       [1.000000e+00, 2.000000e+00, 4.000000e+00, 8.000000e+00,
        1.600000e+01, 3.200000e+01, 6.400000e+01, 1.280000e+02],
       [1.000000e+00, 3.000000e+00, 9.000000e+00, 2.700000e+01,
        8.100000e+01, 2.430000e+02, 7.290000e+02, 2.187000e+03],
       [1.000000e+00, 4.000000e+00, 1.600000e+01, 6.400000e+01,
        2.560000e+02, 1.024000e+03, 4.096000e+03, 1.638400e+04],
       [1.000000e+00, 5.000000e+00, 2.500000e+01, 1.250000e+02,
        6.250000e+02, 3.125000e+03, 1.562500e+04, 7.812500e+04],
       [1.000000e+00, 6.000000e+00, 3.600000e+01, 2.160000e+02,
        1.296000e+03, 7.776000e+03, 4.665600e+04, 2.799360e+05],
       [1.000000e+00, 7.000000e+00, 4.900000e+01, 3.430000e+02,
        2.401000e+03, 1.680700e+04, 1.176490e+05, 8.235430e+05],
       [1.000000e+00, 8.000000e+00, 6.400000e+01, 5.120000e+02,
        4.096000e+03, 3.276800e+04, 2.621440e+05, 2.097152e+06],
       [1.000000e+00, 9.000000e+00, 8.100000e+01, 7.290000e+02,
        6.561000e+03, 5.904900e+04, 5.314410e+05, 4.782969e+06]])
```

In [33]:
```
from sklearn.linear_model import LinearRegression
pereg=LinearRegression()
pereg.fit(p_x,y)
```

Out[33]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

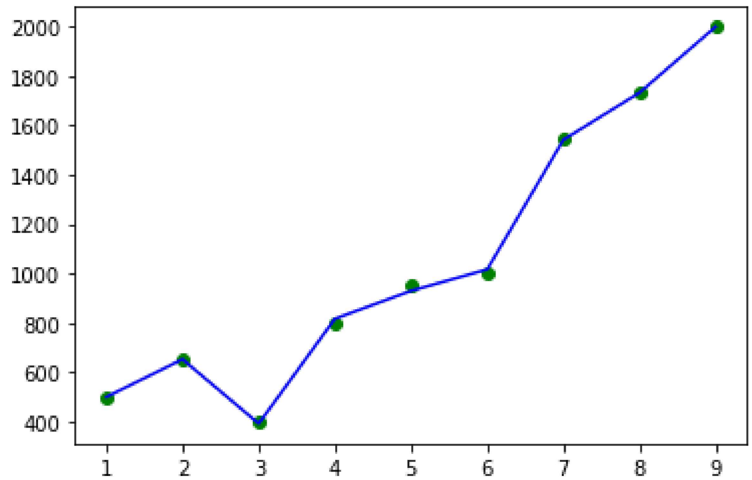
In [34]:
```
pereg.score(p_x,y)
```

Out[34]:
```
0.999573073015455
```

In [35]:
```
from sklearn.metrics import mean_absolute_error
y_pred=pereg.predict(p_x)
mean_absolute_error(y,y_pred)
```

Out[35]:
```
8.310111229092096
```

In [36]:
```
# visulize the polynomial data
import matplotlib.pyplot as plt
plt.scatter(X,y,color='g')
plt.plot(X,pereg.predict(p_x),color='b')
plt.show()
```




In [ ]:
```
```