- **The objective of this case study is to predict the employee salary based on the number of years of experience.**
- **In simple linear regression, we predict the value of one variable Y based on another variable X.**
- **X is called the independent variable and Y is called the dependant variable.**
- **Why simple? Because it examines relationship between two variables only.**
- **Why linear? when the independent variable increases (or decreases), the dependent variable increases (or decreases) in a linear fashion.**

In [1]:

```
# install seaborn library
# !pip install seaborn
# !pip install tensorflow
import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
# read the csv file
salary_df = pd.read_csv('salary.csv')
```

In [3]:

```
salary_df
```

Out[3]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111
15	4.9	67938
16	5.1	66029
17	5.3	83088
18	5.9	81363
19	6.0	93940
20	6.8	91738
21	7.1	98273

	YearsExperience	Salary
22	7.9	101302
23	8.2	113812
24	8.7	109431
25	9.0	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872
30	11.2	127345
31	11.5	126756
32	12.3	128765
33	12.9	135675
34	13.5	139465

In [ ]:

```

```

PERFORM EXPLORATORY DATA ANALYSIS AND VISUALIZATION

In [4]:

```
# check if there are any Null values
sns.heatmap(salary_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[4]:

```
<AxesSubplot:>
```

In [5]:

```
# Check the dataframe info

salary_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35 entries, 0 to 34
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  35 non-null     float64
 1   Salary           35 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 688.0 bytes
```

In [6]:

```python
# Statistical summary of the dataframe

salary_df.describe()
```

Out[6]:

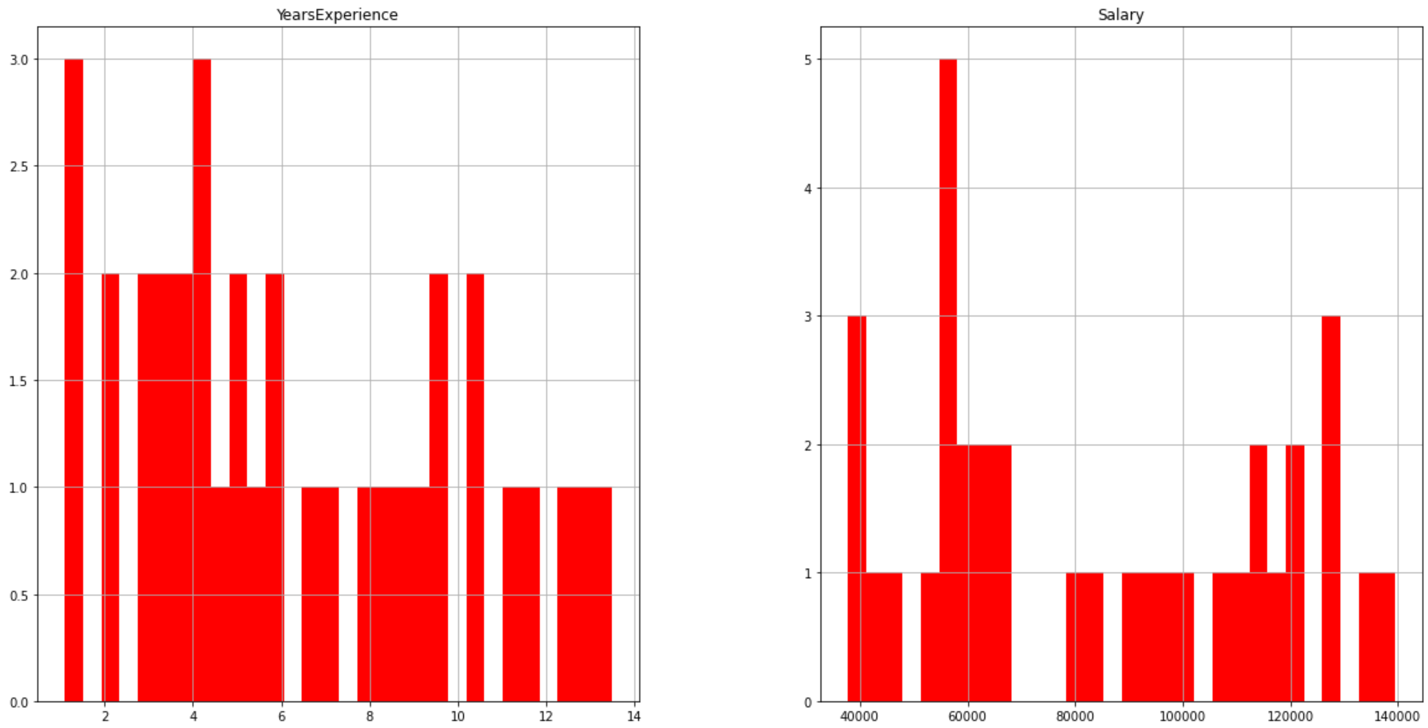
	YearsExperience	Salary
count	35.000000	35.000000
mean	6.308571	83945.600000
std	3.618610	32162.673003
min	1.100000	37731.000000
25%	3.450000	57019.000000
50%	5.300000	81363.000000
75%	9.250000	113223.500000
max	13.500000	139465.000000

In [7]:

```python
salary_df.hist(bins = 30, figsize = (20,10), color = 'r')
```

Out[7]:

```
array([[<AxesSubplot:title={'center':'YearsExperience'}>,
        <AxesSubplot:title={'center':'Salary'}>]], dtype=object)
```



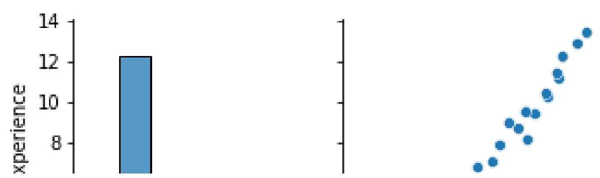
In [8]:

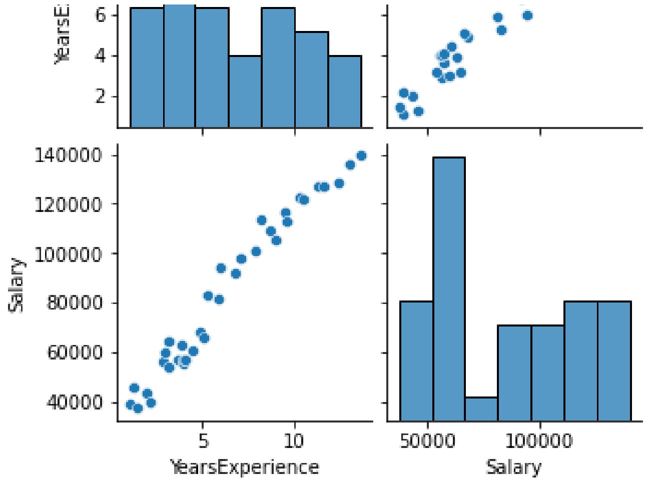
```python
# plot pairplot

sns.pairplot(salary_df)
```

Out[8]:

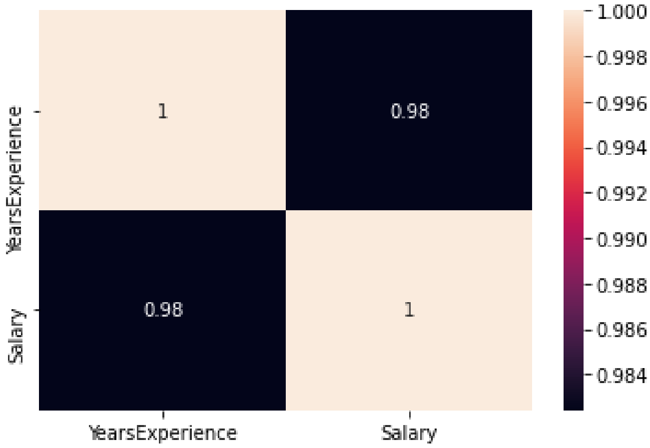
```
<seaborn.axisgrid.PairGrid at 0x7fc6df568be0>
```





In [9]:

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



MINI CHALLENGE

- **Use regplot in Seaborn to obtain a straight line fit between "salary" and "years of experience"**

In [ ]:

```

```

TASK #4: CREATE TRAINING AND TESTING DATASET

In [10]:

```
X = salary_df[['YearsExperience']]
y = salary_df[['Salary']]
```

In [11]:

```
X
```

Out[11]:

	YearsExperience
0	1.1
1	1.3
2	1.5
3	2.0

	YearsExperience
4	2.2
5	2.9
6	3.0
7	3.2
8	3.2
9	3.7
10	3.9
11	4.0
12	4.0
13	4.1
14	4.5
15	4.9
16	5.1
17	5.3
18	5.9
19	6.0
20	6.8
21	7.1
22	7.9
23	8.2
24	8.7
25	9.0
26	9.5
27	9.6
28	10.3
29	10.5
30	11.2
31	11.5
32	12.3
33	12.9
34	13.5

In [12]:

```
y
```

Out[12]:

	Salary
0	39343
1	46205
2	37731
3	43525
4	39891
5	56642
6	60150
7	54445
8	64445

	Salary
9	57189
10	63218
11	55794
12	56957
13	57081
14	61111
15	67938
16	66029
17	83088
18	81363
19	93940
20	91738
21	98273
22	101302
23	113812
24	109431
25	105582
26	116969
27	112635
28	122391
29	121872
30	127345
31	126756
32	128765
33	135675
34	139465

In [13]:

```
X.shape
```

Out[13]:

```
(35, 1)
```

In [14]:

```
y.shape
```

Out[14]:

```
(35, 1)
```

In [15]:

```
X = np.array(X).astype('float32')
y = np.array(y).astype('float32')
```

In [16]:

```
# Only take the numerical variables and scale them
X
```

Out[16]:

```
array([[ 1.1],
       [ 1.3],
```

```
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
       [ 9.6],
       [10.3],
       [10.5],
       [11.2],
       [11.5],
       [12.3],
       [12.9],
       [13.5]], dtype=float32)
```

In [17]:

```
# split the data into test and train sets
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)
```

TRAIN A LINEAR REGRESSION MODEL IN SK-LEARN (NOTE THAT SAGEMAKER BUILT-IN ALGORITHMS ARE NOT USED HERE)

In [18]:

```
# using linear regression model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score

regresssion_model_sklearn = LinearRegression(fit_intercept = True)
regresssion_model_sklearn.fit(X_train, y_train)
```

Out[18]:

```
LinearRegression()
```

In [19]:

```
regresssion_model_sklearn_accuracy = regresssion_model_sklearn.score(X_test, y_test)
regresssion_model_sklearn_accuracy
```

Out[19]:

```
0.9645654317630646
```

In [20]:

```
print('Linear Model Coefficient (m): ', regresssion_model_sklearn.coef_)
```

```
print('Linear Model Coefficient (m): ', regresssion_model_sklearn.coef_)
print('Linear Model Coefficient (b): ', regresssion_model_sklearn.intercept_)
```

```
Linear Model Coefficient (m):  [[8804.039]]
Linear Model Coefficient (b):  [29180.527]
```

EVALUATE TRAINED MODEL PERFORMANCE (NOTE THAT SAGEMAKER BUILT-IN ALGORITHMS ARE NOT USED HERE)

In [21]:

```
y_predict = regresssion_model_sklearn.predict(X_test)
```

In [22]:

```
y_predict
```

Out[22]:

```
array([[105775.67 ],
       [ 91689.2  ],
       [137470.2  ],
       [ 48549.414],
       [ 46788.605],
       [ 65277.086],
       [148035.06 ]], dtype=float32)
```

In [23]:

```
plt.scatter(X_train, y_train, color = 'gray')
plt.plot(X_train, regresssion_model_sklearn.predict(X_train), color = 'red')
plt.ylabel('Salary')
plt.xlabel('Number of Years of Experience')
plt.title('Salary vs. Years of Experience')
```

Out[23]:

```
Text(0.5, 1.0, 'Salary vs. Years of Experience')
```

MINI CHALLENGE

- **Use the trained model, obtain the salary corresponding to eployees who have years of experience = 5**

TRAIN A LINEAR LEARNER MODEL USING SAGEMAKER

In [34]:

```
# Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python
```

```
# Boto3 allows Python developer to write software that makes use of services like Amazon
S3 and Amazon EC2

import sagemaker
import boto3
from sagemaker import Session

# Let's create a Sagemaker session
sagemaker_session = sagemaker.Session()
bucket = Session().default_bucket()
# Let's define the S3 bucket and prefix that we want to use in this session
# bucket = 'sagemaker-practica' # bucket named 'sagemaker-practical' was created beforeha
nd
prefix = 'linear_learner' # prefix is the subfolder within the bucket.

# Let's get the execution role for the notebook instance.
# This is the IAM role that you created when you created your notebook instance. You pass
the role to the training job.
# Note that AWS Identity and Access Management (IAM) role that Amazon SageMaker can assum
e to perform tasks on your behalf (for example, reading training results, called model ar
tifacts, from the S3 bucket and writing training results to Amazon S3).
role = sagemaker.get_execution_role()
print(role)
```

```
arn:aws:iam::542063182511:role/service-role/AmazonSageMaker-ExecutionRole-20191104T033920
```

In [35]:

```
X_train.shape
```

Out[35]:

```
(28, 1)
```

In [36]:

```
y_train = y_train[:,0]
```

In [37]:

```
y_train.shape
```

Out[37]:

```
(28,)
```

In [38]:

```
import io # The io module allows for dealing with various types of I/O (text I/O, binary
I/O and raw I/O).
import numpy as np
import sagemaker.amazon.common as smac # sagemaker common libary

# Code below converts the data in numpy array format to RecordIO format
# This is the format required by Sagemaker Linear Learner

buf = io.BytesIO() # create an in-memory byte array (buf is a buffer I will be writing to
)
smac.write_numpy_to_dense_tensor(buf, X_train, y_train)
buf.seek(0)
# When you write to in-memory byte arrays, it increments 1 every time you write to it
# Let's reset that back to zero
```

Out[38]:

```
0
```

In [39]:

```
import os

# Code to upload RecordIO data to S3
```

```
# Key refers to the name of the file
key = 'linear-train-data'

# The following code uploads the data in record-io format to S3 bucket to be accessed lat
er for training
boto3.resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train', key)).upload_fi
leobj(buf)

# Let's print out the training data location in s3
s3_train_data = 's3://{}/{}/train/{}'.format(bucket, prefix, key)
print('uploaded training data location: {}'.format(s3_train_data))
```

```
uploaded training data location: s3://sagemaker-us-east-2-542063182511/linear_learner/tra
in/linear-train-data
```

In [40]:

```
X_test.shape
```

Out[40]:

```
(7, 1)
```

In [41]:

```
y_test.shape
```

Out[41]:

```
(7, 1)
```

In [42]:

```
# Make sure that the target label is a vector
y_test = y_test[:,0]
```

In [43]:

```
# Code to upload RecordIO data to S3

buf = io.BytesIO() # create an in-memory byte array (buf is a buffer I will be writing to
)
smac.write_numpy_to_dense_tensor(buf, X_test, y_test)
buf.seek(0)
# When you write to in-memory byte arrays, it increments 1 every time you write to it
# Let's reset that back to zero
```

Out[43]:

```
0
```

In [44]:

```
# Key refers to the name of the file
key = 'linear-test-data'

# The following code uploads the data in record-io format to S3 bucket to be accessed lat
er for training
boto3.resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'test', key)).upload_fil
eobj(buf)

# Let's print out the testing data location in s3
s3_test_data = 's3://{}/{}/test/{}'.format(bucket, prefix, key)
print('uploaded training data location: {}'.format(s3_test_data))
```

```
uploaded training data location: s3://sagemaker-us-east-2-542063182511/linear_learner/tes
t/linear-test-data
```

In [45]:

```
# create an output placeholder in S3 bucket to store the linear learner output
```

```
output_location = 's3://{}/{}/output'.format(bucket, prefix)
print('Training artifacts will be uploaded to: {}'.format(output_location))
```

```
Training artifacts will be uploaded to: s3://sagemaker-us-east-2-542063182511/linear_lear
ner/output
```

In [46]:

```
# This code is used to get the training container of sagemaker built-in algorithms
# all we have to do is to specify the name of the algorithm, that we want to use

# Let's obtain a reference to the linearLearner container image
# Note that all regression models are named estimators
# You don't have to specify (hardcode) the region, get_image_uri will get the current reg
ion name using boto3.Session

from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'linear-learner')
```

```
The method get_image_uri has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
Defaulting to the only supported framework/algorithm version: 1. Ignoring framework/algor
ithm version: 1.
```

In [ ]:

```
# We have pass in the container, the type of instance that we would like to use for train
ing
# output path and sagemaker session into the Estimator.
# We can also specify how many instances we would like to use for training
# sagemaker_session = sagemaker.Session()

linear = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count = 1,
                                       train_instance_type = 'ml.c4.xlarge',
                                       output_path = output_location,
                                       sagemaker_session = sagemaker_session)


# We can tune parameters like the number of features that we are passing in, type of pred
ictor like 'regressor' or 'classifier', mini batch size, epochs
# Train 32 different versions of the model and will get the best out of them (built-in pa
rameters optimization!)

linear.set_hyperparameters(feature_dim = 1,
                           predictor_type = 'regressor',
                           mini_batch_size = 5,
                           epochs = 5,
                           num_models = 32,
                           loss = 'absolute_loss')

# Now we are ready to pass in the training data from S3 to train the linear learner model

linear.fit({'train': s3_train_data})

# Let's see the progress using cloudwatch logs
```

DEPLOY AND TEST THE TRAINED LINEAR LEARNER MODEL

In [49]:

```
# Deploying the model to perform inference

linear_regressor = linear.deploy(initial_instance_count = 1,
                                 instance_type = 'ml.m4.xlarge')
```

```
---------------!
```

In [52]:

```python
from sagemaker.predictor import csv_serializer, json_deserializer

# Content type overrides the data that will be passed to the deployed model, since the de
ployed model expects data in text/csv format.

# Serializer accepts a single argument, the input data, and returns a sequence of bytes i
n the specified content type

# Deserializer accepts two arguments, the result data and the response content type, and
return a sequence of bytes in the specified content type.

# Reference: https://sagemaker.readthedocs.io/en/stable/predictors.html

# linear_regressor.content_type = 'text/csv'
linear_regressor.serializer = csv_serializer
linear_regressor.deserializer = json_deserializer
```

In [53]:

```python
# making prediction on the test data

result = linear_regressor.predict(X_test)
```

```
The csv_serializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
The json_deserializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

In [54]:

```python
result # results are in Json format
```

Out[54]:

```
{'predictions': [{'score': 110375.671875},
  {'score': 95310.7265625},
  {'score': 144271.78125},
  {'score': 49174.34375},
  {'score': 47291.2265625},
  {'score': 67063.9609375},
  {'score': 155570.5}]}
```

In [55]:

```python
# Since the result is in json format, we access the scores by iterating through the score
s in the predictions

predictions = np.array([r['score'] for r in result['predictions']])
```

In [56]:

```python
predictions
```

Out[56]:

```
array([110375.671875 ,  95310.7265625, 144271.78125  ,  49174.34375  ,
        47291.2265625,  67063.9609375, 155570.5      ])
```

In [57]:

```python
predictions.shape
```

Out[57]:

```
(7,)
```

In [58]:

```
# VISUALIZE TEST SET RESULTS
plt.scatter(X_test, y_test, color = 'gray')
plt.plot(X_test, predictions, color = 'red')
plt.xlabel('Years of Experience (Testing Dataset)')
plt.ylabel('salary')
plt.title('Salary vs. Years of Experience')
```

Out[58]:

```
Text(0.5, 1.0, 'Salary vs. Years of Experience')
```

In [59]:

```
# Delete the end-point

linear_regressor.delete_endpoint()
```