```
Thanks for viewing this Project.  ||  Pritom Bhowmik   ||
```

Problem Statement

```
Simple Car Purchase Amount Prediction Project for Car Industry

Develop a model to predict the total dollar ammount the customers are willing
to pay to buy a new car.
Given attributes:
    -->  Customer Name
    -->  Customer Email Address
    -->  Country/Region
    -->  Gender
    -->  Age
    -->  Annual Salary
    -->  Credit Card Debt
    -->  Net Worth

The model should predict Car Purchase Amount.
```

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
car_df = pd.read_csv('Car_Purchasing_Data.csv', encoding = 'ISO-8859-1')
```

In [3]:
```
car_df
```

Out[3]:

	Customer Name	Customer e-mail	Country	Gender	Age	
0	Martina Avila	cubilia.Curae.Phasellus@quisaccumsanconvallis.edu	Bulgaria	0	41.851720	62
1	Harlan Barnes	eu.dolor@diam.co.uk	Belize	0	40.870623	66
2	Naomi Rodriquez	vulputate.mauris.sagittis@ametconsectetueradip...	Algeria	1	43.152897	5
3	Jade Cunningham	malesuada@dignissim.com	Cook Islands	1	58.271369	79
4	Cedric Leach	felis.ullamcorper.viverra@egetmollislectus.net	Brazil	1	57.313749	59
...	...	...	...	...	...	
495	Walter	ligula@Cumsociis.ca	Nepal	0	41.462515	7
496	Vanna	Cum.sociis.natoque@Sedmolestie.edu	Zimbabwe	1	37.642000	56
497	Pearl	penatibus.et@massanonante.com	Philippines	1	53.943497	68
498	Nell	Quisque.varius@arcuVivamussit.net	Botswana	1	59.160509	4
499	Marla	Camaron.marla@hotmail.com	marlal	1	46.731152	6

500 rows × 9 columns

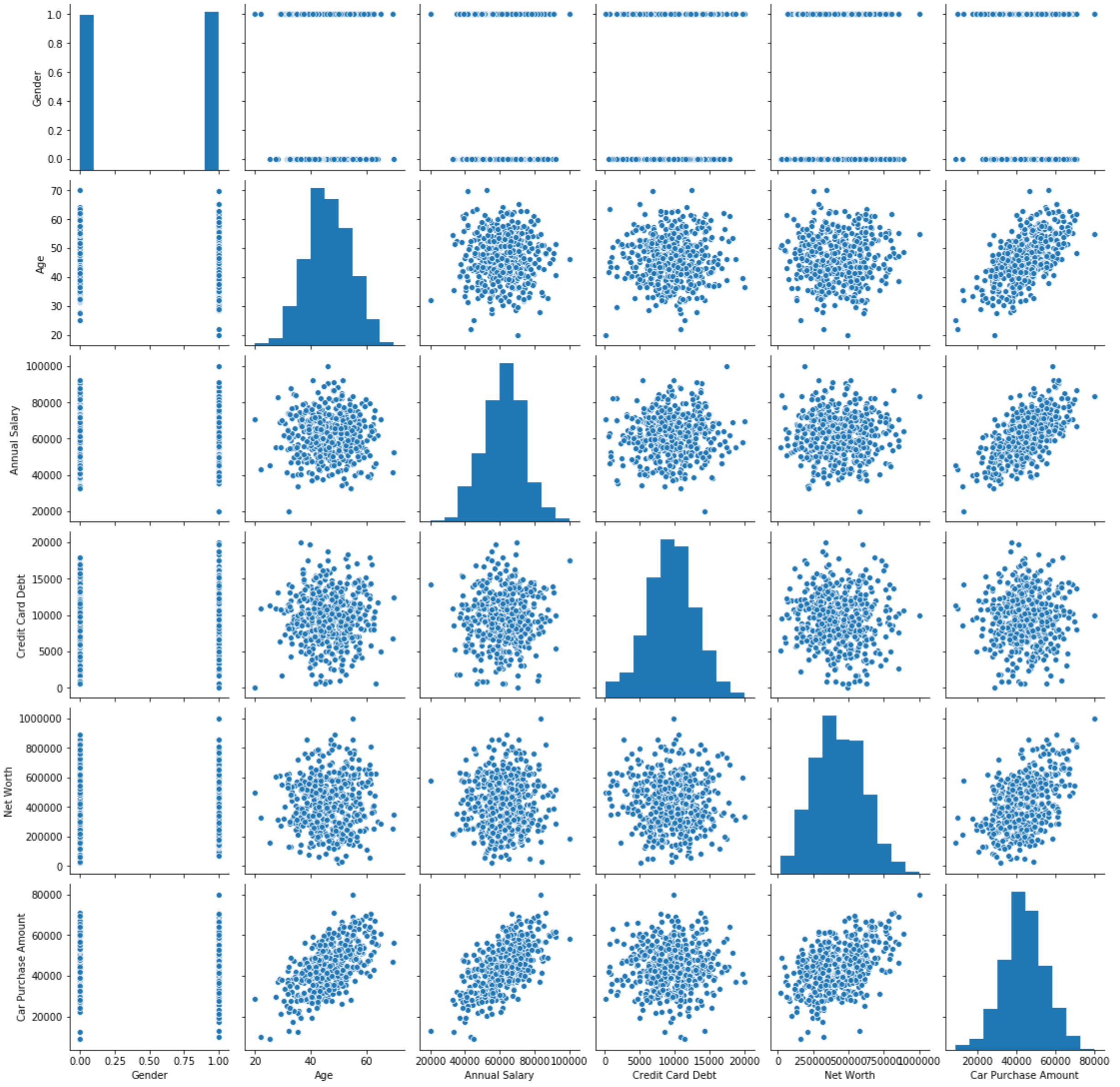
In [4]:
```
car_df.head(5)
```

Out[4]:

	Customer Name	Customer e-mail	Country	Gender	Age	A
0	Martina Avila	cubilia.Curae.Phasellus@quisaccumsanconvallis.edu	Bulgaria	0	41.851720	62812.
1	Harlan Barnes	eu.dolor@diam.co.uk	Belize	0	40.870623	66646.
2	Naomi Rodriquez	vulputate.mauris.sagittis@ametconsectetueradip...	Algeria	1	43.152897	53798
3	Jade Cunningham	malesuada@dignissim.com	Cook Islands	1	58.271369	79370.
4	Cedric Leach	felis.ullamcorper.viverra@egetmollislectus.net	Brazil	1	57.313749	59729.

```
In [5]: sns.pairplot(car_df)
```

```
Out[5]: <seaborn.axisgrid.PairGrid at 0x2766a45f788>
```



Data Cleaning >>> Some features are not needed for the model. Customer name, Email address and Country are droped from the data as we do not need those for the prediction.
Note that we use X as input and y as output in any Machine Learning problem conventionally. So we assign the Car Purchase Amount(Output) to y.

```
In [6]: X = car_df.drop(['Customer Name', 'Customer e-mail', 'Country', 'Car Purchase Amo
```

In [7]:
```
X
```

Out[7]:

	Gender	Age	Annual Salary	Credit Card Debt	Net Worth
0	0	41.851720	62812.09301	11609.380910	238961.2505
1	0	40.870623	66646.89292	9572.957136	530973.9078
2	1	43.152897	53798.55112	11160.355060	638467.1773
3	1	58.271369	79370.03798	14426.164850	548599.0524
4	1	57.313749	59729.15130	5358.712177	560304.0671
...	...	...	...	...	...
495	0	41.462515	71942.40291	6995.902524	541670.1016
496	1	37.642000	56039.49793	12301.456790	360419.0988
497	1	53.943497	68888.77805	10611.606860	764531.3203
498	1	59.160509	49811.99062	14013.034510	337826.6382
499	1	46.731152	61370.67766	9391.341628	462946.4924

500 rows × 5 columns

In [8]:
```
y = car_df['Car Purchase Amount']
```

In [9]:
```
y
```

Out[9]:
```
0      35321.45877
1      45115.52566
2      42925.70921
3      67422.36313
4      55915.46248
          ...
495    48901.44342
496    31491.41457
497    64147.28888
498    45442.15353
499    45107.22566
Name: Car Purchase Amount, Length: 500, dtype: float64
```

In [10]:
```
X.shape
```

Out[10]:
```
(500, 5)
```

In [11]:
```
y.shape
```

Out[11]:
```
(500,)
```

Normalization

In [12]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

In [13]:
```
X_scaled
```

Out[13]:
```
array([[0.        , 0.4370344 , 0.53515116, 0.57836085, 0.22342985],
       [0.        , 0.41741247, 0.58308616, 0.476028  , 0.52140195],
       [1.        , 0.46305795, 0.42248189, 0.55579674, 0.63108896],
       ...,
       [1.        , 0.67886994, 0.61110973, 0.52822145, 0.75972584],
       [1.        , 0.78321017, 0.37264988, 0.69914746, 0.3243129 ],
       [1.        , 0.53462305, 0.51713347, 0.46690159, 0.45198622]])
```

In [14]:
```
scaler.data_max_
```

Out[14]:
```
array([1.e+00, 7.e+01, 1.e+05, 2.e+04, 1.e+06])
```

In [15]:
```
scaler.data_min_
```

Out[15]:
```
array([    0.,    20., 20000.,   100., 20000.])
```

In [16]:
```
y = y.values.reshape(-1,1)
```

In [17]:
```
y_scaled = scaler.fit_transform(y)
```

In [18]:
```
y.shape
```

Out[18]:
```
(500, 1)
```

In [19]:
```
y_scaled
```

Out[19]:
```
array([[0.37072477],
       [0.50866938],
       [0.47782689],
       [0.82285018],
       [0.66078116],
       [0.67059152],
       [0.28064374],
       [0.54133778],
       [0.54948752],
       [0.4111198 ],
       [0.70486638],
       [0.46885649],
       [0.27746526],
       [0.56702642],
       [0.57056385],
       [0.61996151],
       [0.46217916],
       [0.49157341],
       [0.50188722],
       [0.64545808],
```

Training the model

In [20]:
```
X_scaled.shape
```

Out[20]:
```
(500, 5)
```

In [21]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size
```

In [22]:
```
X_train.shape
```

Out[22]:
```
(375, 5)
```

In [23]:
```
X_test.shape
```

Out[23]:
```
(125, 5)
```

In [24]:
```
import tensorflow.keras
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(5, input_dim = 5, activation = 'relu'))
model.add(Dense(5, activation = 'relu'))
model.add(Dense(1, activation = 'linear'))
```

```
Using TensorFlow backend.
```

In [25]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 5)                 30        
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 30        
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 6         
=================================================================
Total params: 66
Trainable params: 66
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [27]:
```
epochs_hist = model.fit(X_train, y_train, epochs = 100, batch_size = 50, verbose
```

```
6/6 [==============================] - 0s 5ms/step - loss: 0.0289 - val_loss:
0.0273
Epoch 29/100
6/6 [==============================] - 0s 6ms/step - loss: 0.0275 - val_loss:
0.0261
Epoch 30/100
6/6 [==============================] - 0s 6ms/step - loss: 0.0263 - val_loss:
0.0251
Epoch 31/100
6/6 [==============================] - 0s 6ms/step - loss: 0.0251 - val_loss:
0.0240
Epoch 32/100
6/6 [==============================] - 0s 6ms/step - loss: 0.0239 - val_loss:
0.0231
Epoch 33/100
6/6 [==============================] - 0s 7ms/step - loss: 0.0229 - val_loss:
0.0223
Epoch 34/100
6/6 [==============================] - 0s 6ms/step - loss: 0.0220 - val_loss:
0.0214
```

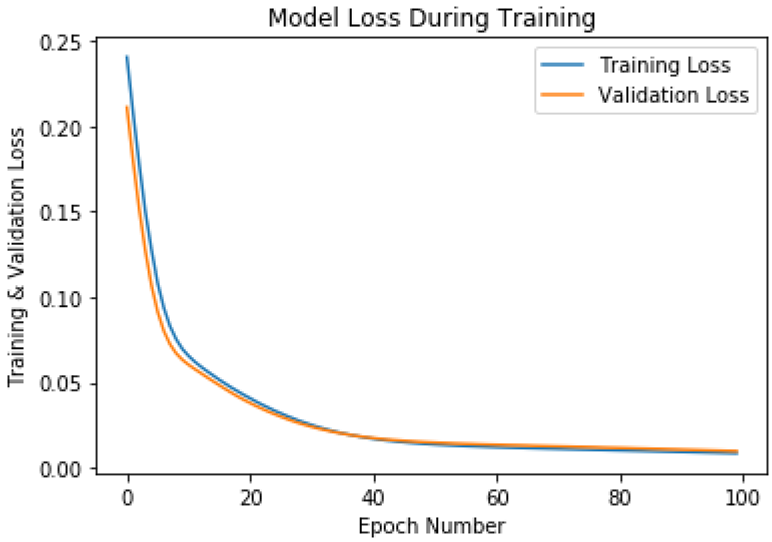
Evaluating The Model

In [28]:
```
epochs_hist.history.keys()
```

Out[28]:
```
dict_keys(['loss', 'val_loss'])
```

In [29]:
```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])
plt.title('Model Loss During Training')
plt.ylabel('Training & Validation Loss')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss' , 'Validation Loss'])
```

Out[29]:
```
<matplotlib.legend.Legend at 0x27679a6d408>
```



In [30]:

```
X_test = np.array([[1, 40, 60000, 20000, 500000]])
y_predict = model.predict(X_test)
```

In [31]:

```
print('Expected Purchase Amount', y_predict)
```

```
Expected Purchase Amount [[97315.18]]
```

In [ ]: