step 0 importing the required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

step 1 importing the dataset

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	Ag
0	Martina Avila	cubilia.Curae.Phasellus@quisaccumsanconvallis.edu	Bulgaria	0	41.85172
1	Harlan Barnes	eu.dolor@diam.co.uk	Belize	0	40.87062
2	Naomi Rodriquez	vulputate.mauris.sagittis@ametconsectetueradip...	Algeria	1	43.15289
3	Jade Cunningham	malesuada@dignissim.com	Cook Islands	1	58.27136
4	Cedric Leach	felis.ullamcorper.viverra@egetmollislectus.net	Brazil	1	57.31374
5	Carla Hester	mi@Aliquamerat.edu	Liberia	1	56.82489
6	Griffin Rivera	vehicula@at.co.uk	Syria	1	46.60731
7	Orli Casey	nunc.est.mollis@Suspendissetristiqueneque.co.uk	Czech Republic	1	50.19301
8	Marny Obrien	Phasellus@sedsemegestas.org	Armenia	0	46.58474
9	Rhonda Chavez	nec@nuncest.com	Somalia	1	43.32378
10	Jerome Rowe	ipsum.cursus@dui.org	Sint Maarten	1	50.12992
11	Akeem Gibson	turpis.egestas.Fusce@purus.edu	Greenland	1	53.18015
12	Quin Smith	nulla@ipsum.edu	Nicaragua	0	44.39649
13	Tatum Moon	Cras.sed.leo@Seddiamlorem.ca	Palestine, State of	0	48.49651
14	Sharon Sharpe	eget.metus@aaliquetvel.co.uk	United Arab Emirates	0	55.24486
15	Thomas Williams	aliquet.molestie@ut.org	Gabon	1	53.28976
16	Blaine Bender	ultrices.posuere.cubilia@pedenonummyut.net	Tokelau	0	44.74220
17	Stephen Lindsey	erat.eget.ipsum@tinciduntpede.org	Portugal	1	48.12708
18	Sloane Mann	at.augue@augue.net	Chad	1	51.85347
19	Athena Wolf	volutpat.Nulla.facilisis@primis.ca	Iraq	0	58.74184
20	Blythe Romero	Sed.eu@risusNuncac.co.uk	Sudan	1	51.90047

step 2 visualising the data

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

Out[4]:

	Customer Name	Customer e-mail	Country	Gender	Age	
0	Martina Avila	cubilia.Curae.Phasellus@quisaccumsanconvallis.edu	Bulgaria	0	41.851720	6281
1	Harlan Barnes	eu.dolor@diam.co.uk	Belize	0	40.870623	6664
2	Naomi Rodriquez	vulputate.mauris.sagittis@ametconsectetueradip...	Algeria	1	43.152897	5379
3	Jade Cunningham	malesuada@dignissim.com	Cook Islands	1	58.271369	7937
4	Cedric Leach	felis.ullamcorper.viverra@egetmollislectus.net	Brazil	1	57.313749	5972

In [5]:
```
#to visualise the last rows
car_df.tail(10
           )
```

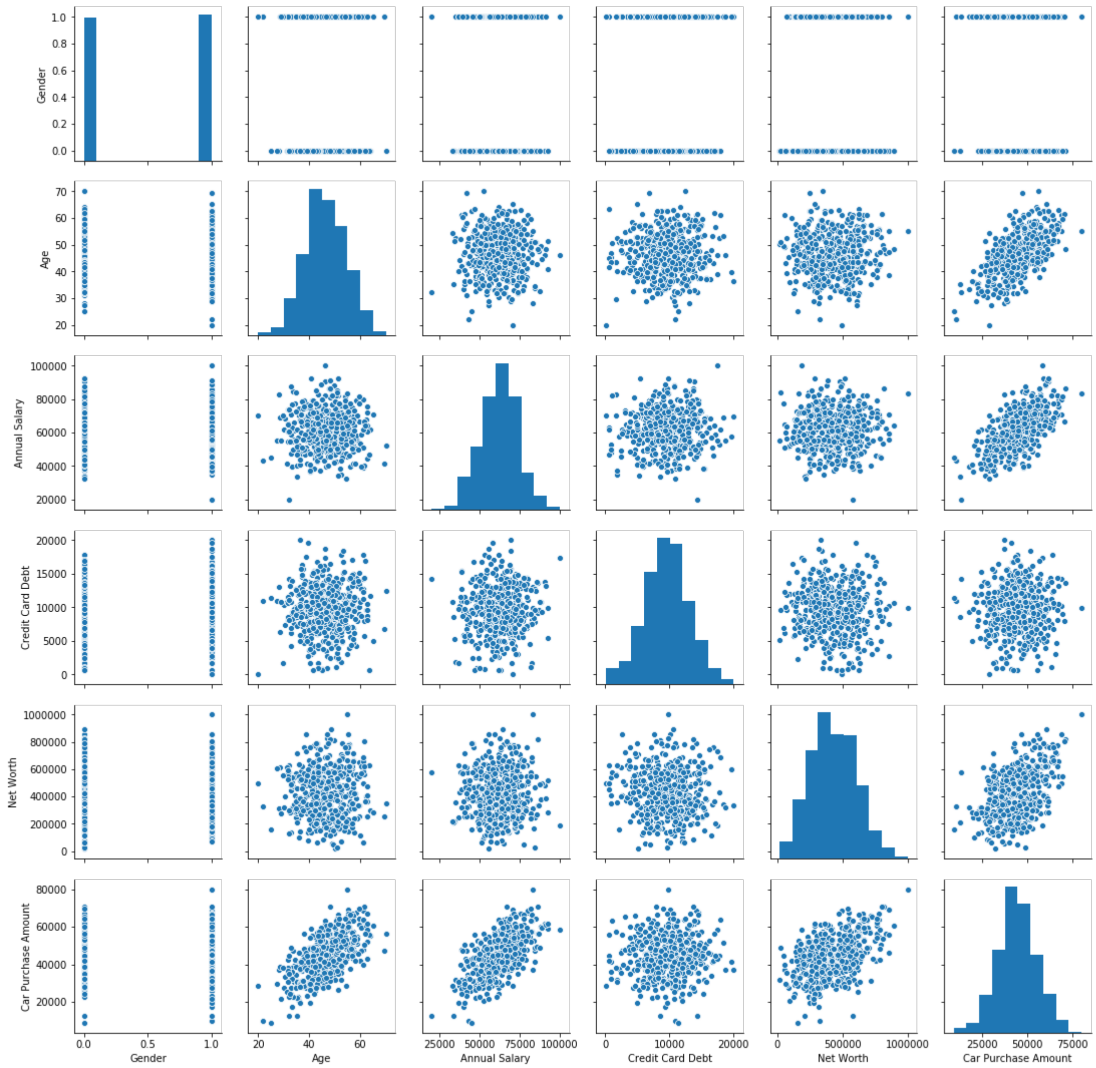
Out[5]:

	Customer Name	Customer e-mail	Country	Gender	Age	Annu Sala
490	Jonah	augue@risusNuncac.co.uk	Myanmar	1	45.752698	63722.001
491	Merrill	dolor.sit@turpisIn.com	Egypt	1	50.197205	78518.215
492	Nolan	Donec.at@neccursus.co.uk	Latvia	0	55.087720	72424.801
493	Winter	egestas.urna.justo@maurissagittis.edu	Wallis and Futuna	0	42.900187	77665.171
494	Rigel	egestas.blandit.Nam@semvitaealiquam.com	Sao Tome and Principe	0	51.767418	77345.616
495	Walter	ligula@Cumsociis.ca	Nepal	0	41.462515	71942.402
496	Vanna	Cum.sociis.natoque@Sedmolestie.edu	Zimbabwe	1	37.642000	56039.497
497	Pearl	penatibus.et@massanonante.com	Philippines	1	53.943497	68888.778
498	Nell	Quisque.varius@arcuVivamussit.net	Botswana	1	59.160509	49811.990
499	Marla	Camaron.marla@hotmail.com	marlal	1	46.731152	61370.677

step 2 visualising the data

In [6]:
```
sns.pairplot(car_df)
```

Out[6]: <seaborn.axisgrid.PairGrid at 0x7f62239323c8>



step 3 creating the testing and training dataset and doing the data cleaning of the data

In [7]:
```
X = car_df.drop(['Customer Name', 'Customer e-mail', 'Country'], axis = 1)
```

In [8]:
```
X
```

Out[8]:

	Gender	Age	Annual Salary	Credit Card Debt	Net Worth	Car Purchase Amount
0	0	41.851720	62812.09301	11609.380910	238961.2505	35321.45877
1	0	40.870623	66646.89292	9572.957136	530973.9078	45115.52566
2	1	43.152897	53798.55112	11160.355060	638467.1773	42925.70921
3	1	58.271369	79370.03798	14426.164850	548599.0524	67422.36313
4	1	57.313749	59729.15130	5358.712177	560304.0671	55915.46248
5	1	56.824893	68499.85162	14179.472440	428485.3604	56611.99784
6	1	46.607315	39814.52200	5958.460188	326373.1812	28925.70549
7	1	50.193016	51752.23445	10985.696560	629312.4041	47434.98265
8	0	46.584745	58139.25910	3440.823799	630059.0274	48013.61410
9	1	43.323782	53457.10132	12884.078680	476643.3544	38189.50601
10	1	50.129923	73348.70745	8270.707359	612738.6171	59045.51309
11	1	53.180158	55421.65733	10014.969290	293862.5123	42288.81046
12	0	44.396494	37336.33830	10218.320920	430907.1673	28700.03340
13	0	48.496515	68304.47298	9466.995128	420322.0702	49258.87571
14	0	55.244866	72776.00382	10597.638140	146344.8965	49510.03356
15	1	53.289768	64662.30061	11326.034340	481433.4324	53017.26723
16	0	44.742200	63259.87837	11495.549990	370356.2223	41814.72067
17	1	48.127085	52682.06401	12514.520290	549443.5886	43901.71244
18	1	51.853474	54503.14423	7377.820914	431098.9998	44633.99241
19	0	58.741842	55368.23716	13272.946470	566022.1306	54827.52403
20	1	51.900471	63435.86304	11878.037790	480588.2345	51130.95379
21	0	48.081120	64347.34531	10905.366280	307226.0977	43402.31525
22	1	45.531842	65176.69055	7698.552234	497526.4566	47240.86004
23	1	47.022284	52027.63837	11960.853770	688466.0503	46635.49432
24	0	39.942995	69612.01230	8125.598993	499086.3442	45078.40193
25	0	52.577441	53065.57175	17805.576070	429440.3297	44387.58412
26	0	28.009676	82842.53385	13102.158050	315775.3207	37161.55393
27	0	55.630317	61388.62709	14270.007310	341691.9337	49091.97185
28	1	46.124036	100000.00000	17452.921790	188032.0778	58350.31809
29	1	40.245327	62891.86556	12522.940520	583230.9760	43994.35972
...	...	...	...	...	...	...
470	0	59.619615	81565.95967	9072.063059	544291.9504	69669.47402
471	0	43.542528	65364.06334	7839.414396	579640.7982	48052.65091
472	1	39.281245	65019.15701	4931.560160	341330.7344	37364.23474
473	1	41.679623	58243.17992	15149.034260	649323.7878	44500.81936

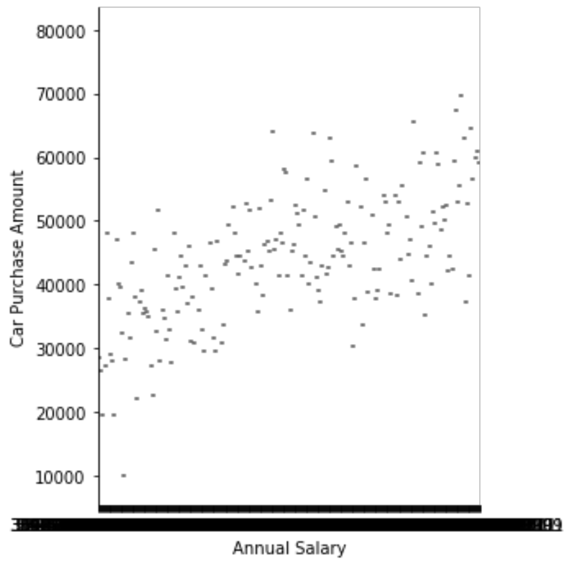
In [9]:
```
#X is for inputs and y is for outputs
X = car_df.drop(['Customer Name', 'Customer e-mail', 'Country', 'Car Purcha
se Amount'], axis = 1)
```

In [10]:
```
X
```

Out[10]:

	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
5	1	56.824893	68499.85162	14179.472440	428485.3604
6	1	46.607315	39814.52200	5958.460188	326373.1812
7	1	50.193016	51752.23445	10985.696560	629312.4041
8	0	46.584745	58139.25910	3440.823799	630059.0274
9	1	43.323782	53457.10132	12884.078680	476643.3544
10	1	50.129923	73348.70745	8270.707359	612738.6171
11	1	53.180158	55421.65733	10014.969290	293862.5123
12	0	44.396494	37336.33830	10218.320920	430907.1673
13	0	48.496515	68304.47298	9466.995128	420322.0702
14	0	55.244866	72776.00382	10597.638140	146344.8965
15	1	53.289768	64662.30061	11326.034340	481433.4324
16	0	44.742200	63259.87837	11495.549990	370356.2223
17	1	48.127085	52682.06401	12514.520290	549443.5886
18	1	51.853474	54503.14423	7377.820914	431098.9998
19	0	58.741842	55368.23716	13272.946470	566022.1306
20	1	51.900471	63435.86304	11878.037790	480588.2345
21	0	48.081120	64347.34531	10905.366280	307226.0977
22	1	45.531842	65176.69055	7698.552234	497526.4566
23	1	47.022284	52027.63837	11960.853770	688466.0503
24	0	39.942995	69612.01230	8125.598993	499086.3442
25	0	52.577441	53065.57175	17805.576070	429440.3297
26	0	28.009676	82842.53385	13102.158050	315775.3207
27	0	55.630317	61388.62709	14270.007310	341691.9337
28	1	46.124036	100000.00000	17452.921790	188032.0778
29	1	40.245327	62891.86556	12522.940520	583230.9760
...	...	...	...	...	...
470	0	59.619615	81565.95967	9072.063059	544291.9504
471	0	43.542528	65364.06334	7839.414396	579640.7982
472	1	39.281245	65019.15701	4931.560160	341330.7344
473	1	41.679623	58243.17992	15149.034260	649323.7878

In [11]:
```
sns.catplot(x = 'Annual Salary', y = 'Car Purchase Amount', kind = 'violin'
, data = car_df);
```



In [12]:
```
y = car_df['Car Purchase Amount']
```

In [13]:
```
y
```

```
Out[13]: 0      35321.45877
         1      45115.52566
         2      42925.70921
         3      67422.36313
         4      55915.46248
         5      56611.99784
         6      28925.70549
         7      47434.98265
         8      48013.61410
         9      38189.50601
         10     59045.51309
         11     42288.81046
         12     28700.03340
         13     49258.87571
         14     49510.03356
         15     53017.26723
         16     41814.72067
         17     43901.71244
         18     44633.99241
         19     54827.52403
         20     51130.95379
         21     43402.31525
         22     47240.86004
         23     46635.49432
         24     45078.40193
         25     44387.58412
         26     37161.55393
         27     49091.97185
         28     58350.31809
         29     43994.35972
                   ...
         470    69669.47402
         471    48052.65091
         472    37364.23474
         473    44500.81936
         474    35139.24793
         475    55167.37361
         476    48383.69071
         477    35823.55471
         478    36517.70996
         479    53110.88052
         480    53049.44567
         481    21471.11367
         482    45015.67953
         483    55377.87697
         484    56510.13294
         485    47443.74443
         486    41489.64123
         487    32553.53423
         488    41984.62412
         489    59538.40327
         490    41352.47071
         491    52785.16947
         492    60117.67886
         493    47760.66427
         494    64188.26862
         495    48901.44342
         496    31491.41457
         497    64147.28888
         498    45442.15353
         499    45107.22566
         Name: Car Purchase Amount, Length: 500, dtype: float64
```

In [14]:
```
X.shape
```
Out[14]:
```
(500, 5)
```

In [15]:
```
y.shape
```
Out[15]:
```
(500,)
```

In [16]:
```
#the dataset is not normalised and it is needed to normalise the dataset th
at can be done by min max scaling
```

In [17]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

In [18]:
```
X_scaled
```
Out[18]:
```
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 [19]:
```
scaler.data_max_
```
Out[19]:
```
array([1.e+00, 7.e+01, 1.e+05, 2.e+04, 1.e+06])
```

In [20]:
```
scaler.data_min_
```
Out[20]:
```
array([    0.,    20., 20000.,   100., 20000.])
```

In [21]:
```
y = y.values.reshape(-1,1)
y_scaled = scaler.fit_transform(y)
y_scaled
```

```
Out[21]: 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],
               [0.59339372],
               [0.48453965],
               [0.53860366],
               [0.53007738],
               [0.50814651],
               [0.49841668],
               [0.3966416 ],
               [0.56467566],
               [0.6950749 ],
               [0.49287831],
               [0.12090943],
               [0.50211776],
               [0.80794216],
               [0.62661214],
               [0.43394857],
               [0.60017103],
               [0.42223485],
               [0.01538345],
               [0.37927499],
               [0.64539707],
               [0.51838974],
               [0.45869677],
               [0.26804521],
               [0.2650104 ],
               [0.84054134],
               [0.84401542],
               [0.35515157],
               [0.406246  ],
               [0.40680623],
               [0.55963883],
               [0.2561583 ],
               [0.77096325],
               [0.55305289],
               [0.5264948 ],
               [0.3236476 ],
               [0.55070832],
               [0.54057623],
               [0.45669016],
               [0.41053254],
               [0.33433524],
               [0.39926954],
               [0.5420261 ],
               [0.57366948],
```

In [22]:
```
y.shape
```
Out[22]: (500, 1)

In [23]:
```
#training the model that we have created so far
```

In [24]:
```
X_scaled.shape
```
Out[24]: (500, 5)

In [25]:
```
y_scaled.shape
```
Out[25]: (500, 1)

In [29]:
```
X_train.shape
```
Out[29]: (375, 5)

In [28]:
```
X_test.shape
```
Out[28]: (125, 5)

In [27]:
```
#splitting the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, tes
t_size = 0.25)
```

In [30]:
```
import tensorflow.keras
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(45, input_dim = 5, activation = 'relu'))
model.add(Dense(45, activation = 'relu'))
model.add(Dense(1, activation = 'linear'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/py
thon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.
framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

Using TensorFlow backend.
```

In [31]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 45)                270
_________________________________________________________________
dense_2 (Dense)              (None, 45)                2070
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 46
=================================================================
Total params: 2,386
Trainable params: 2,386
Non-trainable params: 0
_________________________________________________________________
```

In [32]:
```
#training the built model
```

In [33]:
```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [34]:
```
epochs_hist = model.fit(X_train, y_train, epochs = 100, batch_size = 25, ve
rbose = 1, validation_split = 0.2)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/py
thon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 300 samples, validate on 75 samples
Epoch 1/100
300/300 [==============================] - 0s 886us/step - loss: 0.6459 - va
l_loss: 0.4127
Epoch 2/100
300/300 [==============================] - 0s 61us/step - loss: 0.2477 - val
_loss: 0.1320
Epoch 3/100
300/300 [==============================] - 0s 76us/step - loss: 0.0606 - val
_loss: 0.0220
Epoch 4/100
300/300 [==============================] - 0s 74us/step - loss: 0.0177 - val
_loss: 0.0183
Epoch 5/100
300/300 [==============================] - 0s 67us/step - loss: 0.0163 - val
_loss: 0.0130
Epoch 6/100
300/300 [==============================] - 0s 89us/step - loss: 0.0112 - val
_loss: 0.0110
Epoch 7/100
300/300 [==============================] - 0s 93us/step - loss: 0.0095 - val
_loss: 0.0100
Epoch 8/100
300/300 [==============================] - 0s 103us/step - loss: 0.0081 - va
l_loss: 0.0082
Epoch 9/100
300/300 [==============================] - 0s 67us/step - loss: 0.0069 - val
_loss: 0.0073
Epoch 10/100
300/300 [==============================] - 0s 91us/step - loss: 0.0061 - val
_loss: 0.0066
Epoch 11/100
300/300 [==============================] - 0s 72us/step - loss: 0.0054 - val
_loss: 0.0060
Epoch 12/100
300/300 [==============================] - 0s 67us/step - loss: 0.0049 - val
_loss: 0.0056
Epoch 13/100
300/300 [==============================] - 0s 87us/step - loss: 0.0043 - val
_loss: 0.0048
Epoch 14/100
300/300 [==============================] - 0s 75us/step - loss: 0.0037 - val
_loss: 0.0043
Epoch 15/100
300/300 [==============================] - 0s 64us/step - loss: 0.0032 - val
_loss: 0.0037
Epoch 16/100
300/300 [==============================] - 0s 71us/step - loss: 0.0028 - val
_loss: 0.0037
Epoch 17/100
300/300 [==============================] - 0s 64us/step - loss: 0.0025 - val
_loss: 0.0028
Epoch 18/100
300/300 [==============================] - 0s 81us/step - loss: 0.0020 - val
_loss: 0.0023
Epoch 19/100
300/300 [==============================] - 0s 80us/step - loss: 0.0016 - val
_loss: 0.0025
```

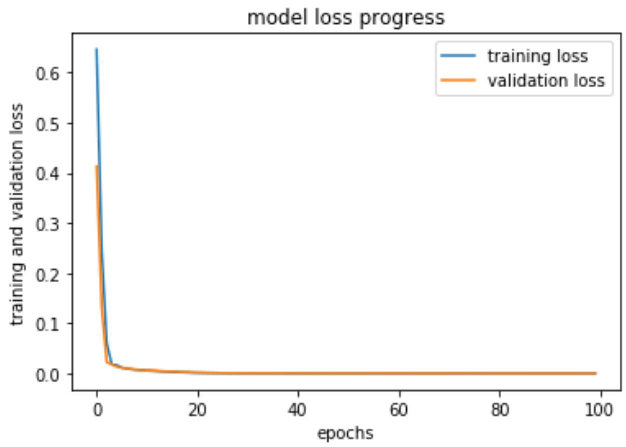
In [35]:
```
#evaluating the results of the model
```

In [36]:
```
epochs_hist.history
```

```
Out[36]: {'loss': [0.6459317033489546,
          0.24766341100136438,
          0.06058388374124964,
          0.017661547210688393,
          0.016292542956459027,
          0.01119504946594437,
          0.009511676849797368,
          0.008114963808717826,
          0.006925621846069892,
          0.0060653333785012364,
          0.0054111888554568095,
          0.004894864939463635,
          0.0043205666782644885,
          0.003677442902699113,
          0.0032214938546530902,
          0.0028249362755256393,
          0.0024506561361098043,
          0.0020383397156062224,
          0.0016281316347885877,
          0.0013434372570676107,
          0.001147749256536675,
          0.000940261884049202,
          0.0007790919529118886,
          0.0006737828989571426,
          0.000562202685008136,
          0.00046925858744846966,
          0.00039858208765508607,
          0.0003452666278462857,
          0.00029962079255104374,
          0.00025311218390318874,
          0.00022438493942900095,
          0.00019162436789580775,
          0.00017384103375661653,
          0.00015194467535669295,
          0.00013613120790978428,
          0.0001230565903824754,
          0.00011013563228819596,
          0.00010294923868059414,
          9.4166321408314e-05,
          8.823376689785316e-05,
          8.878462843616337e-05,
          7.970347148026728e-05,
          7.088848724379204e-05,
          6.686863374246362e-05,
          6.251601098483661e-05,
          5.9319700085325167e-05,
          5.601866420571847e-05,
          5.377084016799927e-05,
          5.1247867456064945e-05,
          4.94882072719823e-05,
          4.843080675224579e-05,
          4.418857800677264e-05,
          4.4163288900260035e-05,
          4.1834216669182446e-05,
          4.040611687135728e-05,
          3.9356037783970045e-05,
          4.029415003969916e-05,
          3.7366848876748314e-05,
          3.906434767486644e-05,
          3.553574712592914e-05,
          3.324686561730535e-05,
          3.285654050462957e-05,
          3.2145976698908875e-05,
```

In [37]:
```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])
plt.title('model loss progress')
plt.ylabel('training and validation loss')
plt.xlabel('epochs')
plt.legend(['training loss', 'validation loss'])
```

Out[37]:
```
<matplotlib.legend.Legend at 0x7f6200604b38>
```



In [38]:
```
#predicting for evaluation
```

In [41]:
```
X_testing = np.array([[1, 50, 50000, 10000, 600000]])
y_predict = model.predict(X_testing)
print('expected purchase amount', y_predict)
y_predict.shape
```

```
expected purchase amount [[211196.47]]
```

Out[41]:
```
(1, 1)
```

In [45]:
```
X_testing_final = np.array([[1, 50, 50000, 10985, 629312]])
y_predict_final = model.predict(X_testing_final)
y_predict_final.shape
print('Expected Purchase Amount=', y_predict_final[:,0])
```

```
Expected Purchase Amount= [220559.56]
```

In [ ]: