

ARTIFICIAL NEURAL NETWORK - CAR SALES PRICE PREDICTION - SAMADHAN PISAL

Main Context:-

Here we create a model that can estimate the overall amount that consumers would spend given the following characteristics:

- 1) Customer name 2) Customer email 3) Country 4) Gender 5) Age 6) Annual salary 7) Credit card debt 8) net worth

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv("/content/ANN_Car_Sales_Price.csv", encoding='ISO-8859-15')

...

here we have Reads the CSV file named "ANN_Car_Sales_Price.csv" located at the "/content/" path into a pandas DataFrame named 'df'.
The 'encoding' parameter is set to 'ISO-8859-15', indicating the character encoding of the CSV file'''
```

```
df.head()
```

	customer name	customer e-mail	country	gender	age	annual Salary	credit card debt	net worth	car purchase amount
0	Martina Avila	cubilia.Curae.Phasellus@quisaccumsanconvallis.edu	Bulgaria	0	41.851720	62812.09301	11609.380910	238961.2505	35321.45877
1	Harlan Barnes	eu.dolor@diam.co.uk	Belize	0	40.870623	66646.89292	9572.957136	530973.9078	45115.52566
2	Naomi Rodriquez	vulputate.mauris.sagittis@ametconsectetueradip...	Algeria	1	43.152897	53798.55112	11160.355060	638467.1773	42925.70921
3	Jade Cunningham	malesuada@dignissim.com	Cook Islands	1	58.271369	79370.03798	14426.164850	548599.0524	67422.36313
4	Cedric Leach	felis.ullamcorper.viverra@egetmollislectus.net	Brazil	1	57.313749	59729.15130	5358.712177	560304.0671	55915.46248

```
df.shape # Checked size of dataset
```

(500, 9)

```
df.info() # Checked missing Value
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customer name         500 non-null   object
1   customer e-mail       500 non-null   object
2   country               500 non-null   object
3   gender                500 non-null   int64
4   age                   500 non-null   float64
5   annual Salary         500 non-null   float64
6   credit card debt      500 non-null   float64
7   net worth             500 non-null   float64
8   car purchase amount   500 non-null   float64
dtypes: float64(5), int64(1), object(3)
memory usage: 35.3+ KB
```

```
df.duplicated().sum()#checked if any row are duplicate or not
```

0

```
df.describe()
```

	gender	age	annual Salary	credit card debt	net worth	car purchase amount
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000
mean	0.506000	46.241674	62127.239608	9607.645049	431475.713625	44209.799218
std	0.500465	7.978862	11703.378228	3489.187973	173536.756340	10773.178744
min	0.000000	20.000000	20000.000000	100.000000	20000.000000	9000.000000
25%	0.000000	40.949969	54391.977195	7397.515792	299824.195900	37629.896040
50%	1.000000	46.049901	62915.497035	9655.035568	426750.120650	43997.783390
75%	1.000000	51.612263	70117.862005	11798.867487	557324.478725	51254.709517
max	1.000000	70.000000	100000.000000	20000.000000	1000000.000000	80000.000000

```
# Dropping the String columns
df.drop(columns=['customer name', 'customer e-mail', 'country', 'gender'], inplace=True)

...

removing the specified columns ('customer name', 'customer e-mail', 'country', 'gender') from the DataFrame becuase these are not required for predication,
and the changes are applied directly to the original DataFrame due to inplace=True.

...
```

```
df.corr()
```

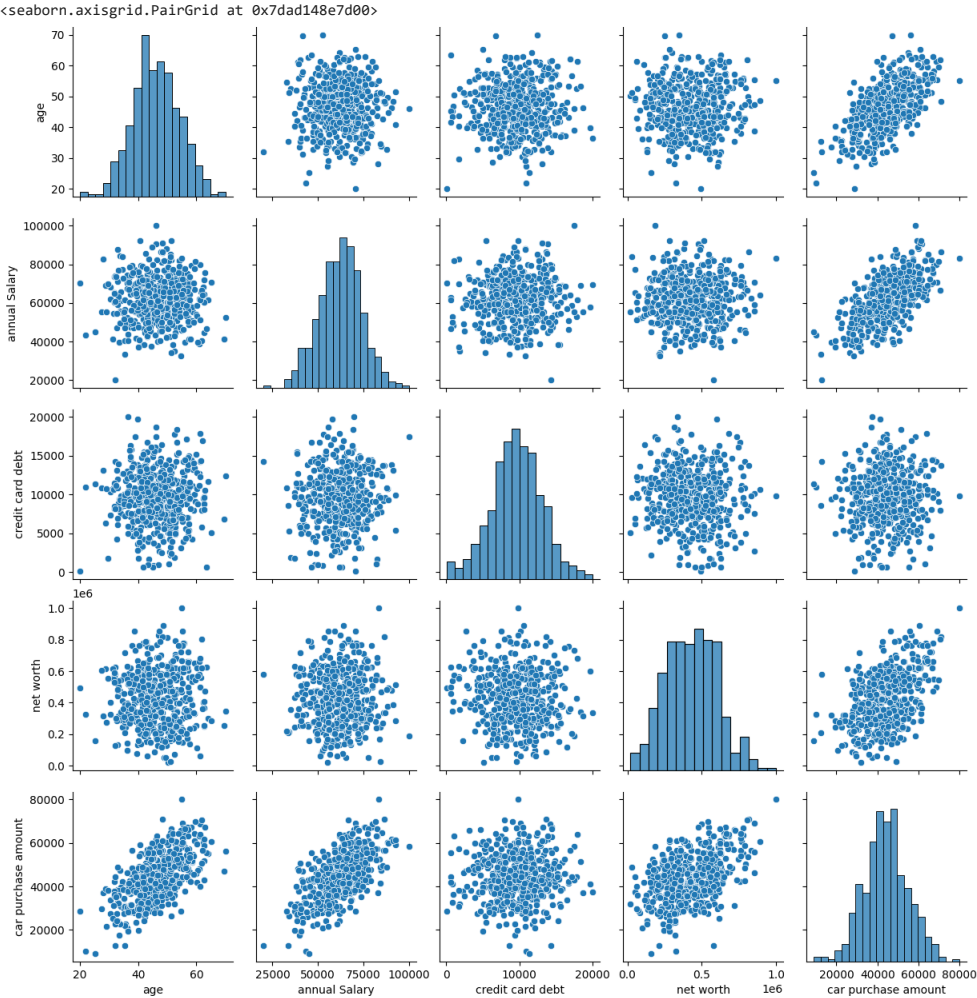
	gender	age	annual Salary	credit card debt	net worth	car purchase amount
gender	1.000000	-0.064481	-0.036499	0.024193	-0.008395	-0.066408
age	-0.064481	1.000000	0.000130	0.034721	0.020356	0.632865
annual Salary	-0.036499	0.000130	1.000000	0.049599	0.014767	0.617862
credit card debt	0.024193	0.034721	0.049599	1.000000	-0.049378	0.028882
net worth	-0.008395	0.020356	0.014767	-0.049378	1.000000	0.488580
car purchase amount	-0.066408	0.632865	0.617862	0.028882	0.488580	1.000000

```
sns.pairplot(df)
```

```
...
```


This function is useful for quickly visualizing the relationships between multiple variables in a dataset. It's especially handy for identifying patterns, trends, or potential correlations.

```
...
```



```
x = df.iloc[:, :-1] # Seperate all features from Datafarama
y = df.iloc[:, -1] # Separate target from Dataframe
```

```
x
```

	age	annual Salary	credit card debt	net worth	
0	41.851720	62812.09301	11609.380910	238961.2505	
1	40.870623	66646.89292	9572.957136	530973.9078	
2	43.152897	53798.55112	11160.355060	638467.1773	
3	58.271369	79370.03798	14426.164850	548599.0524	
4	57.313749	59729.15130	5358.712177	560304.0671	
...	
495	41.462515	71942.40291	6995.902524	541670.1016	
496	37.642000	56039.49793	12301.456790	360419.0988	
497	53.943497	68888.77805	10611.606860	764531.3203	
498	59.160509	49811.99062	14013.034510	337826.6382	
499	46.731152	61370.67766	9391.341628	462946.4924	

500 rows × 4 columns

```
y = y.values.reshape(-1,1)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler= MinMaxScaler()

x=scaler.fit_transform(x)
y=scaler.fit_transform(y)

'''
1) MinMax scaling (or Min-Max normalization) is a data preprocessing technique used in machine learning
and statistics to scale numerical features in a specific range, typically between 0 and 1.
The purpose of MinMax scaling is to ensure that all features contribute equally to the computation,
especially in algorithms that rely on distances between data points, like k-nearest neighbors or support vector machines.
2) Fit transform scaled data to x & y
'''
```

```
y

[[0.4747199 ],
 [0.42114943],
 [0.27669569],
 [0.60775236],
 [0.81194719],
 [0.47759537],
 [0.56687473],
 [0.25779114],
 [0.54746234],
 [0.71808681],
 [0.51086565],
 [0.         ],
 [0.52129727],
 [0.33756622],
 [0.56035434],
 [0.51865585],
 [0.43805795],
 [0.3726359 ],
 [0.2895323 ],
 [0.41187412],
 [0.499023   ],
 [0.59220313],
 [0.61366712],
 [0.73808769],
 [0.27413582],
 [0.26177748],
 [0.54900684],
 [0.26992761],
 [0.85449963],
 [0.55003734],
 [0.39949626],
 [0.50001154],
 [0.36815842],
 [0.6502447 ],
 [0.55469987],
 [0.37779655],
 [0.38757338],
 [0.62128001],
 [0.62041473],
 [0.17564949],
 [0.50726309],
 [0.65320953],
 [0.6691568 ],
 [0.54146119],
 [0.45760058],
 [0.33173992],
 [0.46457217],
 [0.7118085 ],
 [0.4556686 ],
 [0.61669253],
 [0.71996731],
 [0.54592485],
 [0.77729956],
 [0.56199216],
 [0.31678049],
 [0.77672238],
 [0.51326977],
 [0.50855247]])
```

```
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,random_state=0)
```

This line of code is using the `train_test_split` function from the `sklearn.model_selection` module in Python. This function is commonly used in machine learning to split a dataset into two subsets: one for training the model and another for testing the model's performance.

x and y are the input features and corresponding labels respectively of your dataset.

test_size=0.2 specifies that 20% of the data will be used for testing the model, and the remaining 80% will be used for training. This is a common split ratio, but you can adjust it based on your specific use case.

random_state=1 is an optional parameter that sets the random seed for the random number generator. This ensures that every time you run the code, the data is split in the same way. Setting a random seed is important for reproducibility; it allows you to obtain the same train-test split every time you run the code.

After this line of code is executed,data will split into four variables:

x_train: This variable contains the input features for the training set.

x_test: This variable contains the input features for the testing set.

y_train: This variable contains the corresponding labels for the training set.

y_test: This variable contains the corresponding labels for the testing set.

These variables are then typically used to train a machine learning model on the x_train and y_train data and evaluate the model's performance on the x_test and y_test data. This division is essential for assessing how well the trained model generalizes to unseen data.

```
x_train
array([[0.5919433 , 0.28041933, 0.36359421, 0.37352798],
       [0.61344528, 0.64901813, 0.31429008, 0.15276966],
       [0.72431504, 0.46742286, 0.27777908, 0.39982759],
       ...,
       [0.29363923, 0.80213959, 0.63415297, 0.22888834],
       [0.62336278, 0.46847974, 0.10704002, 0.14181603],
       [0.40284696, 0.63935253, 0.39284678, 0.40491578]])
```

```
y_train
[0.65623527],
[0.54372318],
[0.61005611],
[0.44592089],
[0.50211776],
[0.80794216],
[0.39792327],
[0.38927052],
[0.85449963],
[0.25779114],
[0.54380447],
[0.571387 ],
[0.47773971],
[0.56035434],
[0.54839351],
[0.58368485],
[0.27413582],
[0.78232624],
[0.46743215],
[0.42849005],
[0.56687473],
[0.32687852],
[0.43537481],
[0.31967601],
[0.47568675],
[0.61366712],
[0.81860421],
[0.77280198],
[0.41795296],
[0.46343171],
[0.43505067],
[0.49841668],
[0.44900269],
[0.30740999],
[0.65320953],
[0.45217002],
[0.45760058],
[0.64539707],
[0.50997782],
[0.47698891],
[0.79707352],
[0.59067519],
[0.39949626],
[0.30761974],
[0.38056275],
[0.4352791 ],
[0.66900144],
[0.71104101],
[0.54907635],
[0.4111198 ],
[0.4237175 ],
[0.58387492],
[0.2138602 ],
[0.19197549],
[0.40675569],
[0.45706646],
[0.406246 ],
[0.48907732]])
```

```
x_test
```

```
[0.6822764 , 0.77197086, 0.37189841, 0.26345085],
[0.43126874, 0.52116804, 0.53946017, 0.61277794],
[0.4150439 , 0.9058897 , 0.26655265, 0.50583444],
[0.38295346, 0.63079989, 0.5004801 , 0.26408132],
[0.71099221, 0.58474892, 0.71357418, 0.18630267],
[0.44128449, 0.64214907, 0.35149884, 0.30758294],
[0.42709004, 0.73831195, 0.35787272, 0.3529301 ],
[0.46182945, 0.67613141, 0.54159257, 0.20986748],
[0.61922264, 0.68895133, 0.29495812, 0.21694632],
[0.60259845, 0.66685884, 0.41058831, 0.60483532],
[0.33028942, 0.29674973, 0.29312451, 0.50725607],
[0.52612956, 0.41728034, 0.24901687, 0.42703253],
[0.53921561, 0.61446815, 0.7885232 , 0.49692549],
[0.62444944, 0.54837062, 0.64123911, 0.24517255],
[0.46598701, 0.62734379, 0.33877088, 0.65528213],
[0.38240636, 0.2698424 , 0.26618249, 0.30907324],
[0.84141684, 0.54945202, 0.84816721, 0.34774249],
[0.40843091, 0.62564925, 0.23122192, 0.605823 ],
[0.23226977, 0.42993817, 0.53235613, 0.54001268],
[0.50438757, 0.6617479 , 0.38629168, 0.58129557],
[0.6531336 , 0.52322738, 0.25573284, 0.70099699],
[0.6039441 , 0.73147769, 0.5011298 , 0.28010825],
[0.39966992, 0.46819701, 0.60733809, 0.14257946],
[0.61012975, 0.77617634, 0.04852299, 0.56864559],
[0.4525583 , 0.65002569, 0.73415368, 0.56015076],
[0.64579599, 0.5761003 , 0.33513477, 0.54806017],
[0.80346399, 0.41536432, 0.53550798, 0.0940074 ],
[0.56146637, 0.37521727, 0.37258256, 0.14335408],
[0.44530403, 0.45217929, 0.5173152 , 0.3857261 ],
[0.40400854, 0.70108552, 0.46021013, 0.4723128 ],
[0.65033672, 0.42978435, 0.34372143, 0.17466168],
[0.50611883, 0.54451307, 0.18241808, 0.60002003],
[0.64931298, 0.18836774, 0.08803919, 0.34056888],
[0.7879805 , 0.72077639, 0.6705978 , 0.31781681],
[0.45769073, 0.51078722, 0.39867088, 0.33067365],
[0.47152991, 0.34038653, 0.21714339, 0.54898064],
[0.48628725, 0.43647984, 0.44323543, 0.33369167],
[0.27952057, 0.51471359, 0.48930766, 0.32720748],
[0.29456443, 0.66185148, 0.63137289, 0.34740515],
[0.27632596, 0.80584737, 0.38554999, 0.45738652],
[0.39331314, 0.47221444, 0.98456847, 0.59307171],
[0.44116179, 0.3336177 , 0.38842038, 0.60792442],
[0.46080627, 0.28722929, 0.4414126 , 0.48080957]]])
```

y_test

```
[0.30168732],
[0.39864179],
[0.45573492],
[0.50866938],
[0.50109082],
[0.62671101],
[0.41091372],
[0.34705263],
[0.37643596],
[0.45107076],
[0.58012487],
[0.62895125],
[0.43333307],
[0.27746526],
[0.48597146],
[0.68212351],
[0.51422109],
[0.70558126],
[0.4155271 ],
[0.54964825],
[0.47470876],
[0.54534475],
[0.46931782],
[0.57353959],
[0.70486638],
[0.26992761],
[0.43026944],
[0.59095889],
[0.49885366],
[0.61908354],
[0.20447774],
[0.6691568 ],
[0.56409653],
[0.30935321],
[0.63399262],
[0.68227386],
[0.61669253],
[0.27381426],
[0.76406707],
[0.58735466],
[0.6502447 ],
[0.46018938],
[0.31103807],
[0.38794277],
[0.5558489 ],
[0.41088501],
[0.56829414],
[0.3236476 ],
[0.73406295],
[0.41040247],
[0.39802597],
[0.38150608],
[0.3059129 ],
[0.41613807],
[0.54057623],
[0.45278122],
[0.39926954],
[0.33068236]]])
```

```
import tensorflow #Library
from tensorflow import keras #Module
from keras import Sequential #Class
from keras.layers import Dense
#Dense layer is a fully connected layer, meaning that each neuron in the layer is connected to every neuron in the previous layer.
```

```
model=Sequential()

model.add(Dense(4,activation="relu",input_dim=4))
model.add(Dense(4,activation="relu"))
model.add(Dense(1,activation="linear"))
```

- The Sequential() function initializes a linear stack of layers for building the neural network.
- The first Dense layer has 4 units (neurons), uses the ReLU (Rectified Linear Unit) activation function, and specifies an input dimension of 4. This implies that the input data fed into the model is expected to have four features.
- The second Dense layer also has 4 units and uses the ReLU activation function. This layer is connected to the previous layer, and Keras automatically infers the input dimensions from the previous layer.
- The third and final Dense layer has 1 unit and uses the linear activation function. This layer serves as the output layer, and the use of a linear activation function suggests that the network is intended for regression tasks, as linear activation outputs the raw weighted sum of inputs.

```
model.summary()
```

Model: "sequential"		
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 4)	20
dense_1 (Dense)	(None, 4)	20
dense_2 (Dense)	(None, 1)	5

=====
Total params: 45 (180.00 Byte)
Trainable params: 45 (180.00 Byte)
Non-trainable params: 0 (0.00 Byte)

The function model.summary() is typically used in machine learning frameworks like TensorFlow or Keras to display a concise summary of the architecture of a neural network model. This summary provides a quick overview of the layers in the model, along with the number of parameters and the output shapes at each layer.

```
model.compile(loss="mean_squared_error",optimizer="Adam")
```

```
history = model.fit(x_train,y_train,epochs=100,validation_split=0.2)
```

```
Epoch 1/100
10/10 [=====] - 1s 25ms/step - loss: 0.1841 - val_loss: 0.1880
Epoch 2/100
10/10 [=====] - 0s 7ms/step - loss: 0.1440 - val_loss: 0.1492
Epoch 3/100
10/10 [=====] - 0s 7ms/step - loss: 0.1107 - val_loss: 0.1180
Epoch 4/100
10/10 [=====] - 0s 7ms/step - loss: 0.0849 - val_loss: 0.0927
Epoch 5/100
10/10 [=====] - 0s 8ms/step - loss: 0.0639 - val_loss: 0.0729
Epoch 6/100
10/10 [=====] - 0s 5ms/step - loss: 0.0481 - val_loss: 0.0575
Epoch 7/100
10/10 [=====] - 0s 7ms/step - loss: 0.0368 - val_loss: 0.0459
Epoch 8/100
10/10 [=====] - 0s 7ms/step - loss: 0.0286 - val_loss: 0.0378
Epoch 9/100
10/10 [=====] - 0s 6ms/step - loss: 0.0234 - val_loss: 0.0325
Epoch 10/100
10/10 [=====] - 0s 6ms/step - loss: 0.0203 - val_loss: 0.0291
Epoch 11/100
10/10 [=====] - 0s 7ms/step - loss: 0.0187 - val_loss: 0.0270
Epoch 12/100
10/10 [=====] - 0s 5ms/step - loss: 0.0179 - val_loss: 0.0259
Epoch 13/100
10/10 [=====] - 0s 5ms/step - loss: 0.0175 - val_loss: 0.0253
Epoch 14/100
10/10 [=====] - 0s 6ms/step - loss: 0.0173 - val_loss: 0.0248
Epoch 15/100
10/10 [=====] - 0s 7ms/step - loss: 0.0172 - val_loss: 0.0245
Epoch 16/100
10/10 [=====] - 0s 5ms/step - loss: 0.0171 - val_loss: 0.0243
Epoch 17/100
10/10 [=====] - 0s 5ms/step - loss: 0.0170 - val_loss: 0.0242
Epoch 18/100
10/10 [=====] - 0s 7ms/step - loss: 0.0169 - val_loss: 0.0241
Epoch 19/100
10/10 [=====] - 0s 7ms/step - loss: 0.0168 - val_loss: 0.0240
Epoch 20/100
10/10 [=====] - 0s 6ms/step - loss: 0.0167 - val_loss: 0.0239
Epoch 21/100
10/10 [=====] - 0s 7ms/step - loss: 0.0166 - val_loss: 0.0236
Epoch 22/100
10/10 [=====] - 0s 7ms/step - loss: 0.0165 - val_loss: 0.0235
Epoch 23/100
10/10 [=====] - 0s 8ms/step - loss: 0.0164 - val_loss: 0.0233
Epoch 24/100
10/10 [=====] - 0s 7ms/step - loss: 0.0163 - val_loss: 0.0232
Epoch 25/100
10/10 [=====] - 0s 6ms/step - loss: 0.0162 - val_loss: 0.0230
Epoch 26/100
10/10 [=====] - 0s 7ms/step - loss: 0.0161 - val_loss: 0.0229
Epoch 27/100
10/10 [=====] - 0s 7ms/step - loss: 0.0159 - val_loss: 0.0227
Epoch 28/100
10/10 [=====] - 0s 5ms/step - loss: 0.0158 - val_loss: 0.0226
Epoch 29/100
10/10 [=====] - 0s 7ms/step - loss: 0.0157 - val_loss: 0.0223
```

```
y_pred = model.predict(x_test)
```

```
4/4 [=====] - 0s 3ms/step
```

```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

```
0.6736372477519756
```

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
```

```
[<matplotlib.lines.Line2D at 0x7dacbe6aed70>]
```

