In [1]:
```
import pandas as pd
import numpy as np
```

In [5]:
```
sales=pd.ExcelFile('ElecmartSales.xlsx')
df=sales.parse('Data')
df.head()
```

Out[5]:

	Date	Day	Time	Region	Card Type	Gender	Buy Category	Items Ordered	Total Cost	High Item	Unnamed: 10	U
0	2016-03-06	Sun	Morning	West	ElecMart	Female	High	4	136.97	79.97	NaN	
1	2016-03-06	Sun	Morning	West	Other	Female	Medium	1	25.55	25.55	NaN	
2	2016-03-06	Sun	Afternoon	West	ElecMart	Female	Medium	5	113.95	90.47	NaN	
3	2016-03-06	Sun	Afternoon	NorthEast	Other	Female	Low	1	6.82	6.82	NaN	
4	2016-03-06	Sun	Afternoon	West	ElecMart	Male	Medium	4	147.32	83.21	NaN	

In [6]:
```
df=df[['Date','Day','Time','Region','Card Type','Gender','Buy Category', 'Items Ordered
df.head()
```

Out[6]:

	Date	Day	Time	Region	Card Type	Gender	Buy Category	Items Ordered	Total Cost	High Item
0	2016-03-06	Sun	Morning	West	ElecMart	Female	High	4	136.97	79.97
1	2016-03-06	Sun	Morning	West	Other	Female	Medium	1	25.55	25.55
2	2016-03-06	Sun	Afternoon	West	ElecMart	Female	Medium	5	113.95	90.47
3	2016-03-06	Sun	Afternoon	NorthEast	Other	Female	Low	1	6.82	6.82
4	2016-03-06	Sun	Afternoon	West	ElecMart	Male	Medium	4	147.32	83.21

In [7]:
```
df.dtypes
```

Out[7]:
```
Date            datetime64[ns]
Day                     object
Time                    object
Region                  object
Card Type               object
```

```
Gender                 object
Buy Category           object
Items Ordered           int64
Total Cost            float64
High Item             float64
dtype: object
```

In [8]:
```
df['Date'].dtypes
```

Out[8]:
```
dtype('<M8[ns]')
```

In [9]:
```
pd.get_dummies(df['Day'], prefix='Day_of_the_week')
```

Out[9]:

	Day_of_the_week_Fri	Day_of_the_week_Mon	Day_of_the_week_Sat	Day_of_the_week_Sun	Day_of_the
0	0	0	0	1	
1	0	0	0	1	
2	0	0	0	1	
3	0	0	0	1	
4	0	0	0	1	
...	...	...	...	...	
395	0	0	1	0	
396	0	0	1	0	
397	0	0	1	0	
398	0	0	1	0	
399	0	0	1	0	

400 rows × 7 columns

In [12]:
```
pd.get_dummies(df['Day'], prefix='Day_of_the_week', drop_first=True)
```

Out[12]:

	Day_of_the_week_Mon	Day_of_the_week_Sat	Day_of_the_week_Sun	Day_of_the_week_Thu	Day_of_th
0	0	0	1	0	
1	0	0	1	0	
2	0	0	1	0	
3	0	0	1	0	
4	0	0	1	0	
...	...	...	...	...	
395	0	1	0	0	
396	0	1	0	0	

	Day_of_the_week_Mon	Day_of_the_week_Sat	Day_of_the_week_Sun	Day_of_the_week_Thu	Day_of_th
397	0	1	0	0	
398	0	1	0	0	
399	0	1	0	0	

400 rows × 6 columns

In [13]:
```
df = pd.concat([df,pd.get_dummies(df['Day'], prefix='Day_of_the_week', drop_first=True)
df.drop(['Day'],axis=1, inplace=True)
df.head()
```

Out[13]:

	Date	Time	Region	Card Type	Gender	Buy Category	Items Ordered	Total Cost	High Item	Day_of_the_week_M
0	2016-03-06	Morning	West	ElecMart	Female	High	4	136.97	79.97	
1	2016-03-06	Morning	West	Other	Female	Medium	1	25.55	25.55	
2	2016-03-06	Afternoon	West	ElecMart	Female	Medium	5	113.95	90.47	
3	2016-03-06	Afternoon	NorthEast	Other	Female	Low	1	6.82	6.82	
4	2016-03-06	Afternoon	West	ElecMart	Male	Medium	4	147.32	83.21	

In [14]:
```
df = pd.concat([df,pd.get_dummies(df['Time'], prefix='Time_of_the_Day', drop_first=True
df.drop(['Time'],axis=1, inplace=True)
df = pd.concat([df,pd.get_dummies(df['Region'], prefix='Region', drop_first=True)],axis
df.drop(['Region'],axis=1, inplace=True)
df = pd.concat([df,pd.get_dummies(df['Card Type'], prefix='Card', drop_first=True)],axi
df.drop(['Card Type'],axis=1, inplace=True)
df = pd.concat([df,pd.get_dummies(df['Gender'], prefix='Gender', drop_first=True)],axis
df.drop(['Gender'],axis=1, inplace=True)
df = pd.concat([df,pd.get_dummies(df['Buy Category'], prefix='Buy_Cat', drop_first=True
df.drop(['Buy Category'],axis=1, inplace=True)
df.head()
```

Out[14]:

	Date	Items Ordered	Total Cost	High Item	Day_of_the_week_Mon	Day_of_the_week_Sat	Day_of_the_week_Sun	Da
0	2016-03-06	4	136.97	79.97	0	0	1	
1	2016-03-06	1	25.55	25.55	0	0	1	

	Date	Items Ordered	Total Cost	High Item	Day_of_the_week_Mon	Day_of_the_week_Sat	Day_of_the_week_Sun	Da
2	2016-03-06	5	113.95	90.47	0	0	1	
3	2016-03-06	1	6.82	6.82	0	0	1	
4	2016-03-06	4	147.32	83.21	0	0	1	

In [15]:
```
from sklearn import preprocessing
```

In [16]:
```
df_date=df['Date'] #min_max scaler does not work with dates.
##I removed it from the dataframe and saved it so I can add it later if I need it
df.drop(['Date'],axis=1, inplace=True)
df1 = df.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
#this scales all the data by column to a value between 0 and 1. you can also scale spec
df_scaled = min_max_scaler.fit_transform(df1)
df = pd.DataFrame(df_scaled) #this process loses the names of the columns
df.head()
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0.3	0.272172	0.195322	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.039169	0.050012	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
2	0.4	0.224032	0.223359	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
3	0.0	0.000000	0.000000	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0
4	0.3	0.293816	0.203973	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0

In [17]:
```
data = pd.read_excel('employee_data.xlsx')
data.head()
# copy the data
data_z_scaled = data.copy()
data_z_scaled.head()
```

Out[17]:

	Employee	Gender	Age	Prior Experience	Beta Experience	Education	Annual Salary
0	1	1	39	5	12	4	57700
1	2	0	44	12	8	6	76400
2	3	0	24	0	2	4	44000
3	4	1	25	2	1	4	41600
4	5	0	56	5	25	8	163900

In [18]:
```
column = 'Age'
data_z_scaled[column] = (data_z_scaled[column] - data_z_scaled[column].mean()) / data_z

# view normalized data
display(data_z_scaled)
```

	Employee	Gender	Age	Prior Experience	Beta Experience	Education	Annual Salary
0	1	1	-0.051942	5	12	4	57700
1	2	0	0.404793	12	8	6	76400
2	3	0	-1.422148	0	2	4	44000
3	4	1	-1.330801	2	1	4	41600
4	5	0	1.500957	5	25	8	163900
...	...	...	...	...	...	...	...
199	200	1	0.404793	10	18	4	100000
200	201	1	0.404793	2	4	4	39300
201	202	1	0.496140	0	7	2	20400
202	203	0	0.313446	0	12	6	74300
203	204	0	-0.600025	11	19	4	114500

204 rows × 7 columns

In [19]:
```
column = 'Annual Salary'
data_z_scaled[column] = (data_z_scaled[column] - data_z_scaled[column].mean()) / data_z

# view normalized data
display(data_z_scaled)
```

	Employee	Gender	Age	Prior Experience	Beta Experience	Education	Annual Salary
0	1	1	-0.051942	5	12	4	-0.448706
1	2	0	0.404793	12	8	6	0.169423
2	3	0	-1.422148	0	2	4	-0.901561
3	4	1	-1.330801	2	1	4	-0.980893
4	5	0	1.500957	5	25	8	3.061743
...	...	...	...	...	...	...	...
199	200	1	0.404793	10	18	4	0.949523
200	201	1	0.404793	2	4	4	-1.056920
201	202	1	0.496140	0	7	2	-1.681661
202	203	0	0.313446	0	12	6	0.100008
203	204	0	-0.600025	11	19	4	1.428822

204 rows × 7 columns

In [20]:
```
#produce a range of -1 to 1
# copy the data
data_max_scaled = data.copy()

# apply normalization techniques on Column 1
column = 'Prior Experience'
data_max_scaled[column] = data_max_scaled[column] /data_max_scaled[column].abs().max()

# view normalized data
display(data_max_scaled)
```

	Employee	Gender	Age	Prior Experience	Beta Experience	Education	Annual Salary
0	1	1	39	0.25	12	4	57700
1	2	0	44	0.60	8	6	76400
2	3	0	24	0.00	2	4	44000
3	4	1	25	0.10	1	4	41600
4	5	0	56	0.25	25	8	163900
...	...	...	...	...	...	...	...
199	200	1	44	0.50	18	4	100000
200	201	1	44	0.10	4	4	39300
201	202	1	45	0.00	7	2	20400
202	203	0	43	0.00	12	6	74300
203	204	0	33	0.55	19	4	114500

204 rows × 7 columns

In [21]:
```
data_copy = data.copy()
train_set = data_copy.sample(frac=0.80, random_state=0) #random state can be omitted, b
#frac sets the split proportion: can typically be anywhere from 0.70 to 0.80
test_set = data_copy.drop(train_set.index)

train_set.head()
```

Out[21]:

	Employee	Gender	Age	Prior Experience	Beta Experience	Education	Annual Salary
18	19	1	34	10	1	4	55800
45	46	0	38	6	6	6	59200
33	34	0	58	9	22	4	133100
37	38	1	35	3	7	4	55400
109	110	1	24	2	7	2	27100

In [22]:
```
test_set.head()
```

Out[22]:

	Employee	Gender	Age	Prior Experience	Beta Experience	Education	Annual Salary
9	10	0	23	0	1	4	39200
21	22	0	63	16	20	4	140400
25	26	1	45	20	2	4	76900
29	30	0	40	4	13	6	82400
31	32	1	27	0	6	0	27000

In [23]:
```
test_set_labels = test_set.pop('Annual Salary')
train_set_labels = train_set.pop('Annual Salary')
```

In [24]:
```
from sklearn.linear_model import LinearRegression
```

In [25]:
```
regressor = LinearRegression()
regressor.fit(train_set, train_set_labels)
```

Out[25]:
```
LinearRegression()
```

In [26]:
```
y_pred = regressor.predict(train_set)
y_pred
```

Out[26]:
```
array([ 59171.86991142,  80985.47449101, 114795.26730015,  52967.81217694,
        34724.28542735,  82034.04749566,  78992.76035731,  95938.70107472,
        88477.18057981,  66358.7750899 ,  65468.27126024,  58023.01232711,
        76336.1653849 , 103986.19397876,  50696.50087044,  81668.08447073,
        94892.06555803,  68229.9147447 ,  42719.35368705,  52592.1773555 ,
        30699.77894005,  64342.83883272, 105177.81775959, 119954.92227798,
       103792.89081894,  71025.54721823,  64972.25438134,  54201.69501453,
       103316.98153275,  25269.74809327, 113122.576394  ,  36111.88947002,
        99245.24600083,  78359.88366363,  60701.72659979,  46028.55509936,
        97030.70676875,  85981.36274507, 141334.90225394,   2969.18722052,
       127676.86601599,  47137.87932272,  72800.63953396,  54490.28672464,
        75827.81353608,  73040.02248563,  69180.04689183, 118003.82057891,
        64325.73686179,  55611.62317711,  62740.29186043, 111871.26269418,
        69206.81831493,  33007.9568358 , 107616.04895507,  39135.2615163 ,
       120962.23467701,  81812.51407649,  73271.65754597,  76774.38755355,
        52550.18997296,  88932.2347036 ,  91724.43441846,  99433.3053407 ,
        79648.44106856,  76053.10484176,  84190.06684996,  61829.9551451 ,
        35627.07890473,  -1033.97914769,  41899.87341045,  30103.75384495,
        13829.96706244, 110677.27011948,  54585.09667604,  32739.06450258,
       100312.62617601,  78484.68864555,  21726.67267108,  56816.21173773,
       101133.49936153,  36820.73986589,  73936.30132867,  77240.2107008 ,
        50673.55418897,  99934.53932025,  79854.99096045,  89782.86198628,
        38877.58036233,  50206.21530333,  63632.32514439,  60203.45744341,
        49197.42401458,  73839.60937288,  68646.10503324, 145456.10629329,
        84275.59459009,  62118.27271253,  17939.30373542,  46145.90282413,
        35706.51788699,  55717.42357599,  65437.06481686,  54802.38940176,
       101524.50187997,  99159.67925793,  72187.29263307,  52676.05840811,
       129339.45564818,  44401.60730873,  56675.82522266,  69667.33406357,
       114974.13781671,  72960.36892212, 119148.74922645,  62753.00504404,
        66016.93331062,  25476.09605023,  69437.367157  ,  60459.83995084,
```

```
 40926.21405913,  77378.36790491,  77003.39248991, 124345.68329248,
 97883.29192748,  74805.35594812,  67936.18304202,  58633.13468902,
 90374.2869994 ,  57983.52031327,  75634.90158158,  64991.73650757,
113787.53210018, 103803.80399208,  78454.67300354,  77769.48598571,
 90445.40412689,  72410.5576184 ,  96118.48064189,  52791.04349963,
 17021.16386481, 121691.93789093,  34550.14687401,  14304.9871751 ,
 62430.01340246, 125973.49369926, 104577.52831603,  22812.53092456,
111790.69125001,  82404.53078673,  96890.16328707,  18256.96846473,
 56155.64413092,  57687.71912213, 127541.20603415,  93111.6306044 ,
 93593.21213228,  85829.32507673,  74744.13693771,  53499.07514719,
 29433.23644365,  92715.64453032,  56750.07132743])
```

In [27]:
```
from sklearn import metrics
```

In [28]:
```
coeff_df = pd.DataFrame(regressor.coef_, train_set.columns, columns=['Coefficient'])
coeff_df
```

Out[28]:

	Coefficient
Employee	-13.368459
Gender	-6593.849027
Age	-90.400920
Prior Experience	3059.935899
Beta Experience	2593.315868
Education	7607.735946

In [29]:
```
df_pred = pd.DataFrame({'Actual':train_set_labels, 'Predicted': y_pred})
df_pred.head()
```

Out[29]:

	Actual	Predicted
18	55800	59171.869911
45	59200	80985.474491
33	133100	114795.267300
37	55400	52967.812177
109	27100	34724.285427

In [30]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(train_set_labels, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(train_set_labels, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(train_set_labels,
```

```
Mean Absolute Error: 5447.957734818786
Mean Squared Error: 71033531.0260981
Root Mean Squared Error: 8428.13923865156
```

In [31]:
```
y_pred_test = regressor.predict(test_set)
```

In [32]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(test_set_labels, y_pred_test)
print('Mean Squared Error:', metrics.mean_squared_error(test_set_labels, y_pred_test))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(test_set_labels, y
```

```
Mean Absolute Error: 4184.218969403912
Mean Squared Error: 43237689.943195514
Root Mean Squared Error: 6575.537236089194
```

In [ ]:
```
```