Multiple linear Regression

Multiple linear Regression means you have more than one feature column in regression problem.

Guideline for Multiple linear Regression:

1- Check for Missing values using

2- Check for Outliers - Only on Feature Columns if performing Inferential Stats else perform for all columns

(Never check outliers for Label Column if performing Supervised Learning)

3- Inferential Stats

- Ensure your data is Complete
- Ensure your data is Strictly Numeric

4- Separate data as features and label and store the same in the form on Numpy Array(Inferential Stats)

5- Split our data as Training Set and Testing Set where Training Set is used for training/convergence purpose(Inferential Stats)

- (whereas Testing set is used for Quality Check Purpose)
- Decide the Split Ratio (65%:35%)(80:20)
- Implement the same using Sci-kit Learn package

6- Build the model:

- Regression Algo ------> Linear Regression
- y = mx + b(Slope Intercept Formula)
- profit = b0(California) + b1(Florida) + b2(New York)+ b3(R&D Spend) + b4(AdminSpend) + b5(MarketingSpend) + intercept
- Goal of this algo is to derive values of b0,b1,b2,b3,b4,b5,intercept based on the historical data !

7- Train the model | Converging Training Set to the Algorithm

8- fit(features,label)

9- Quality Check (Guideline)

1- Ensure your model that is converged is a Generalized Model

- Generalized model is a model that performs well with Known and Unknown data

-Technique: **Accuracy(Test Data) > Accuracy(Train Data)** ------> Model is **Generalized Model**

2- Ensure the Model's accuracy score must be greater than or equal to CL score (*CL = 1 - SL)

- Technique: **Accuracy(Test Data) >= CL**

10- Deployment Test

Use-case: To create a model that can **predict the profit of the company** based on **company's location** and **company's spending pattern**.

```
data = pd.read_csv('50_Startups.csv')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

```
data.State.unique()
```

```
array(['New York', 'California', 'Florida'], dtype=object)
```

Check for Missing values using

```
data.isna().sum()
```

```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

Check for Outliers - Only on Feature Columns if performing Inferential Stats else perform for all columns

Never check outliers for Label Column if performing Supervised Learning

```
data.describe()
```

	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

Inferential Stats
1. Ensure your data is Complete
2. Ensure your data is Strictly Numeric

```
data.head(5)
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
finalData = pd.concat([ pd.get_dummies(data['State']) ,
data.iloc[:,[0,1,2,4]]] , axis = 1)
finalData.head()
```

	California	Florida	New York	R&D Spend	Administration	Marketing Spend	Profit
0	0	0	1	165349.20	136897.80	471784.10	192261.83

		1 1 0 0 162597.70 151377.59 443898.53 191792.06 2 0 1 0 153441.51 101145.55 407934.54 191050.39 3 0 0 1 144372.41 118671.85 383199.62 182901.99 4 0 1 0 142107.34 91391.77 366168.42 166187.94
	Inferential Stats 1.Seperate data as features and label and store the same in the form on Numpy Array 2. Split our data as Training Set and Testing Set where Training Set is used for training/convergence purpose whereas Testing set is used for Quality Check Purpose.	1. Seperate data as features and label and store the same in the form on Numpy Array features = finalData.iloc[:,0:6].values label = finalData.iloc[:,[6]].values
	# 2. Split our data as Training Set and Testing Set where Training Set Step1: Decide the Split Ratio (65%:35%)(80:20) Step2: Implement the same using Sci-kit Learn package X_train --> Training Features y_train --> Training Label X_test --> Testing Features y_test --> Testing Label For training ----------> X_train,y_train For QA ----------> X_test,y_test	from sklearn.model_selection import train_test_split X_train,X_test,y_train,y_test = train_test_split(features, label, test_size=0.2, random_state=10)
	Build the model Regression Algo ------> Linear Regression y = mx + b(Slope Intercept Formula) profit = b0(California) + b1(Florida) + b2(New York)+ b3(R&D Spend) + b4(AdminSpend) + b5(MarketingSpend) + intercept Goal of this algo is to derive values of b0,b1,b2,b3,b4,b5,intercept based on the historical data !	from sklearn.linear_model import LinearRegression model = LinearRegression()
	Train the model | Converging Training Set to the Algorithm # fit(features,label)	model.fit(X_train,y_train)
		model.intercept_ array([50001.73604086])
		model.coef_ array([[8.41023126e+01, 6.95447747e+02, -7.79550060e+02, 8.05859453e-01, -1.79706621e-02, 2.28153524e-02]])
	Profit = 8.41023126e+01(California) + 6.95447747e+02(Florida) - 7.79550060e+02(New York) + 8.05859453e-01 (RDSpend) - 1.79706621e-02(Admin) + 2.28153524e-02(MarkSpend) + 50001.73604086	

	Quality Check (Guideline) 1-Ensure your model that is converged is a Generalized Model **- Generalized model is a model that performs well with Known and Unknown data** -Technique: **Accuracy(Test Data) > Accuracy(Train Data)** ------> Model is **Generalized Model** 2-Ensure the Model's accuracy score must be greater than or equal to CL score (*CL = 1 - SL) -Technique: **Accuracy(Test Data) >= CL**	
	1. Ensure your model that is converged is a Generalized Model #In scikit learn you can get the accuracy score using score function	testAccuracy = model.score(X_test,y_test) trainAccuracy = model.score(X_train,y_train) print("Test Score is {} and train Score is {}".format(testAccuracy,trainAccuracy)) **Test Score** is 0.9901105113397691 and **train Score** is 0.9385918220043519
		As observed above, testScore > trainScore , thus the model is generalized !!!
	SL = 0.05 CL = 0.95 2. Ensure the Model's accuracy score must be greater than or equal to CL score (*CL = 1 - SL) Technique : Accuracy(Test Data) >= CL Since my testScore is greater than CL, model passed the Quality Check !	

	Deployment Test – method one	rdSpend = float(input("Enter R&D Spend: "))
		admSpend = float(input("Enter Administration Spend: "))
		markSpend = float(input("Enter Marketing Spend: "))
		state = input("Enter State: ")
		
		refState = ['California', 'Florida','New York']
		
		if state in refState:
		  if state == "California":
		  stateDummy = np.array([[1,0,0]])
		  elif state == "Florida":
		  stateDummy = np.array([[0,1,0]])
		  else:
		  stateDummy = np.array([[0,0,1]])
		
		  finalFeatures = np.concatenate((stateDummy, np.array([[rdSpend,admSpend,markSpend]])) , axis=1)
		
		  profit = model.predict(finalFeatures)[0][0]
		
		  print("Predicted profit is $ {}".format(profit))
		
		else:
		  print("Model can't predict profit for the given {} state".format(state))
		
		Enter R&D Spend: 234567
		Enter Administration Spend: 45678
		Enter Marketing Spend: 76543
		Enter State: Florida
		Predicted profit is $ 240650.70978691286

	For Deployment Test, method 2 we need to do these before train/test split: -we handle Categorical Data with: Sci-kit Package/ OneHotEncoder. - do FeatureScaling with StandardScaler. 1- Categorical Data Handling Using Sci-kit Package	from sklearn.preprocessing import OneHotEncoder ohe = OneHotEncoder(sparse=False) fState = ohe.fit_transform(np.array(data['State']).reshape(-1,1)) features = np.concatenate([fState , np.array(data.iloc[:,[0,1,2]])] , axis = 1) features
	#FeatureScaling	from sklearn.preprocessing import StandardScaler sc = StandardScaler() features = sc.fit_transform(features)

	Deployment Test / method 2	rdSpend = float(input("Enter R&D Spend: "))
		admSpend = float(input("Enter Administration Spend: "))
		markSpend = float(input("Enter Marketing Spend: "))
		state = input("Enter State: ")
		
		refState = ['California', 'Florida','New York']
		
		if state in refState:
		
		stateDummy = ohe.transform(np.array([[state]]))
		
		finalFeatures = np.concatenate((stateDummy, np.array([[rdSpend,admSpend,markSpend]])) , axis=1)
		
		stdScaleFeatures = sc.transform(finalFeatures)
		
		profit = model.predict(**stdScaleFeatures**)[0][0]
		
		print("Predicted profit is $ {}".format(profit))
		
		else:
		print("Model can't predict profit for the given {} state".format(state))

		data['State'].unique()

		array(['New York', 'California', 'Florida'], dtype=object)
		[0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 1.6534920e+05,1.3689780e+05, 4.7178410e+05] California Florida New York rdSpend admSpend markSpend