Name: S. Rayen Subhiksha

MULTICOLLINEARITY

Dataset used: Pre_processed_Placement

1. What is multicollinearity?

   Multicollinearity refers to the situation in which two or more predictor variables in a regression model are highly correlated. This high correlation means that one predictor variable can be linearly predicted from the others with a substantial degree of accuracy.

   Multicollinearity occurs when two or more predictor variables in a regression model are highly correlated. This can lead to:

   - Unstable Estimates:
     Small changes in the data can lead to large changes in the coefficient estimates.
   - Reduced Interpretability:
     It becomes difficult to interpret the effect of each predictor variable due to their intercorrelation.
   - Inflated Variance:
     The variance of the coefficient estimates is increased, which can make the model less reliable.

2. What are the implications in multicollinearity?
   - Redundancy**:**
     It makes the model's estimates of the coefficients unstable and highly sensitive to changes in the model.
   - Interpretation**:**
     It becomes difficult to determine the effect of each predictor on the outcome variable because predictors are not independent of each other.
   - Variance Inflation**:**
     Multicollinearity inflates the variances of the coefficient estimates, making them very large, which can lead to overfitting.

3. What are the ways to control Multicollinearity without removing columns?

- Regularisation
- Principal Component Analysis (PCA)
- Centering and Scaling

4. What is Variance Inflation Factor (VIF)?

Variance Inflation Factor (VIF) is a measure used to detect the presence of multicollinearity in a regression analysis. It quantifies how much the variance of an estimated regression coefficient increases due to multicollinearity.

The VIF of a predictor variable quantifies how much the variance of the estimated regression coefficient for that variable is inflated due to multicollinearity with other predictor variables.

If your VIF values are high (typically VIF > 10 is considered problematic, So use regularisation, centering & scaling, PCA methods to ensuring regression model remains robust and interpretable.

5. What is Regularisation method?

- Ridge Regression**:**
  Adds a penalty proportional to the sum of the squared coefficients to the regression model. This helps to reduce the impact of multicollinearity by shrinking the coefficients of correlated variables.

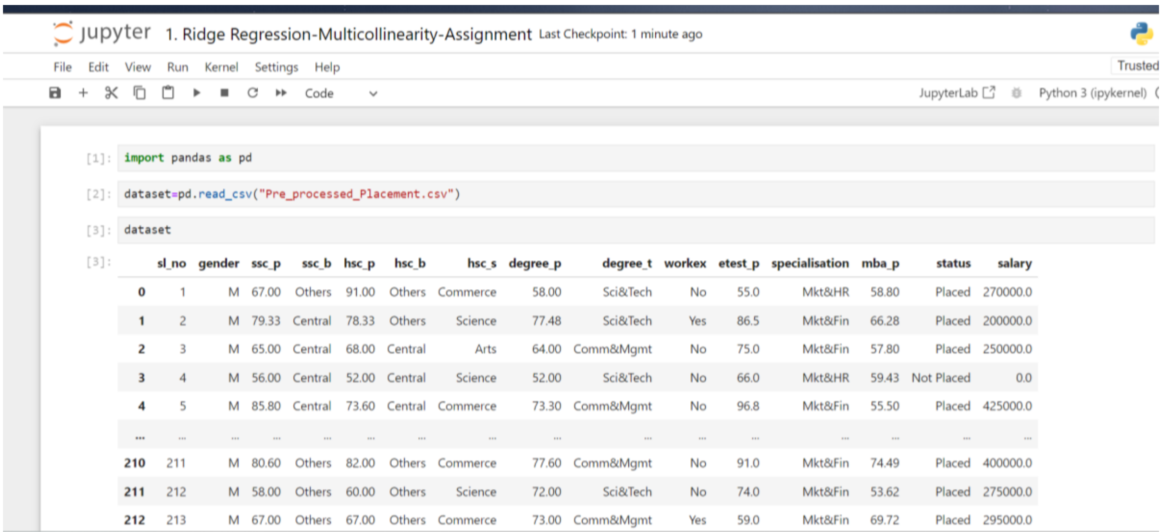
  - How Ridge Regression helps?

  Ridge Regression addresses multicollinearity by adding an L2 penalty term to the regression loss function. The Ridge regression objective function is:
  Objective Function=Sum of Squared Residuals + α ×
  Sum of Squared Co-efficients
  Where:
  Sum of Squared Residuals: Measures how well the model fits the data.
  Sum of Squared Coefficients: The penalty term, which discourages large coefficients.
  α (Alpha): The regularization parameter that controls the strength of the penalty.

- Key Mechanism:
  - Penalization of Coefficients:
    Ridge regression shrinks the coefficients of correlated predictors towards zero, reducing their impact on the model. This helps to mitigate the effects of multicollinearity by making the coefficients less sensitive to changes in the data.
  - Stabilization of Estimates:
    By adding the penalty term, Ridge regression makes the estimates more stable and less sensitive to multicollinearity. This leads to more reliable and robust coefficient estimates.
  - Improved Model Performance:
    The regularization helps to balance the trade-off between fitting the training data and keeping the coefficients small. This can improve the model's performance on unseen data, especially in the presence of multicollinearity.

- Illustration:

  https://github.com/Rayenai/Bivariate-Analysis---Multicollinearity--Assignment

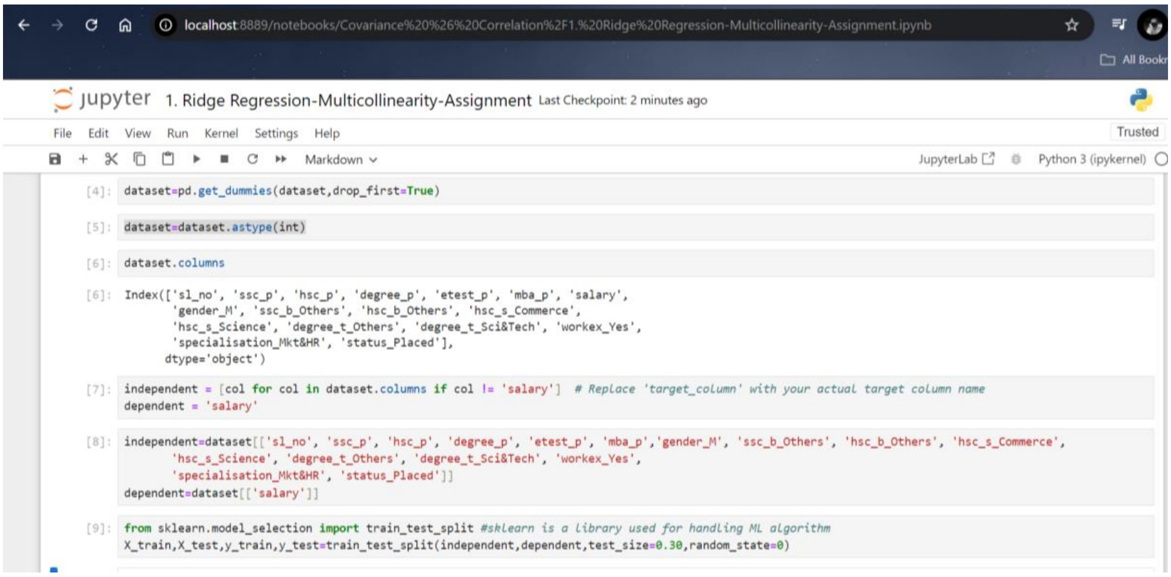

jupyter 1. Ridge Regression-Multicollinearity-Assignment Last Checkpoint: 1 minute ago

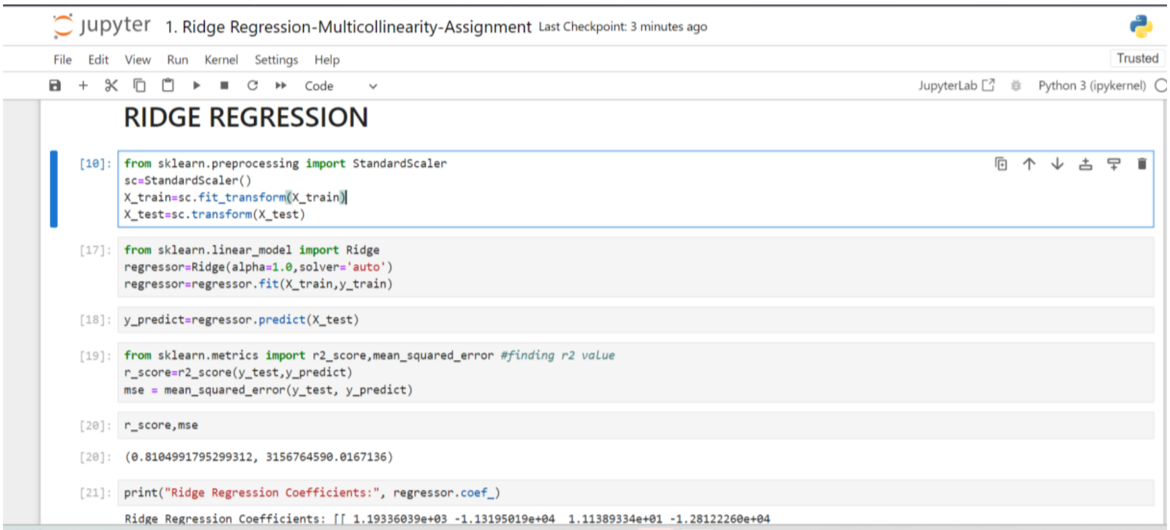
File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```
import pandas as pd
```
[2]:
```
dataset=pd.read_csv("Pre_processed_Placement.csv")
```
[3]:
```
dataset
```
[3]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	0.0
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No	91.0	Mkt&Fin	74.49	Placed	400000.0
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No	74.0	Mkt&Fin	53.62	Placed	275000.0
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes	59.0	Mkt&Fin	69.72	Placed	295000.0





- Summary:
  - Without Ridge Regression:
    Coefficients can be large and unstable, especially if predictors are highly correlated.
  - With Ridge Regression:
    Coefficients are shrunk and more stable, making the model more robust and interpretable.
  - Ridge regression controls multicollinearity by:
  - Penalizing Large Coefficients:
    The L2 penalty discourages large coefficients, helping to stabilize the model.
  - Reducing Variance:
    By shrinking coefficients, it reduces the variance of the coefficient estimates, making them more reliable.

- This approach is particularly useful when dealing with highly correlated predictors, as it helps to produce a more robust and interpretable model.

- Lasso Regression:

Adds a penalty proportional to the sum of the absolute values of the coefficients. This can shrink some coefficients to zero, effectively selecting a subset of features and handling multicollinearity.

Lasso regression (Least Absolute Shrinkage and Selection Operator) is another regularization technique that helps in controlling multicollinearity. Unlike Ridge regression, which uses an L2 penalty, Lasso uses an L1 penalty. This L1 penalty can drive some coefficients to exactly zero, effectively performing feature selection.

- How Lasso Regression Helps with Multicollinearity?

Lasso Regression adds an L1 penalty term to the regression loss function:

Objective Function=Sum of Squared Residuals+α×Sum of Absolute Coefficients

Where:

- Sum of Squared Residuals: Measures how well the model fits the data.
- Sum of Absolute Coefficients: The penalty term, which can set some coefficients to zero.
- α (Alpha): The regularization parameter that controls the strength of the penalty.

- Key Mechanism
  - Feature Selection:

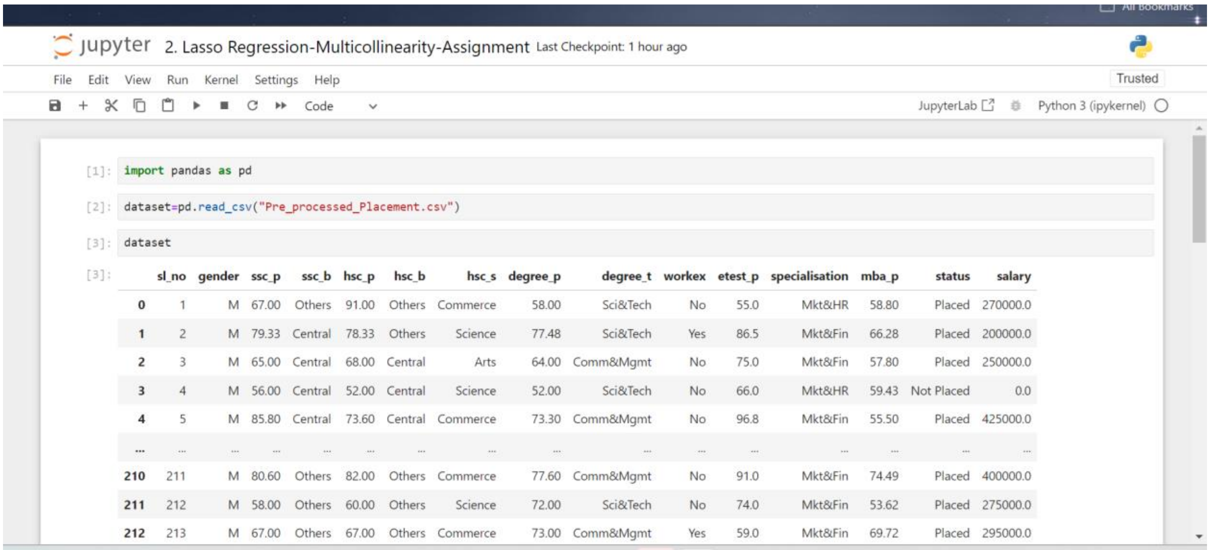
    Lasso regression can set some coefficients to exactly zero, which helps in selecting a subset of features and eliminating irrelevant ones. This is particularly useful in high-dimensional datasets where multicollinearity is a concern.
  - Sparsity:

    By driving some coefficients to zero, Lasso creates a sparse model, which can improve interpretability and reduce the impact of multicollinearity.

- Model Simplification:
  By including fewer features, Lasso can simplify the model, which often improves performance on unseen data and reduces overfitting.

Illustration:

https://github.com/Rayenai/Bivariate-Analysis---Multicollinearity--Assignment


jupyter 2. Lasso Regression-Multicollinearity-Assignment Last Checkpoint: 1 hour ago

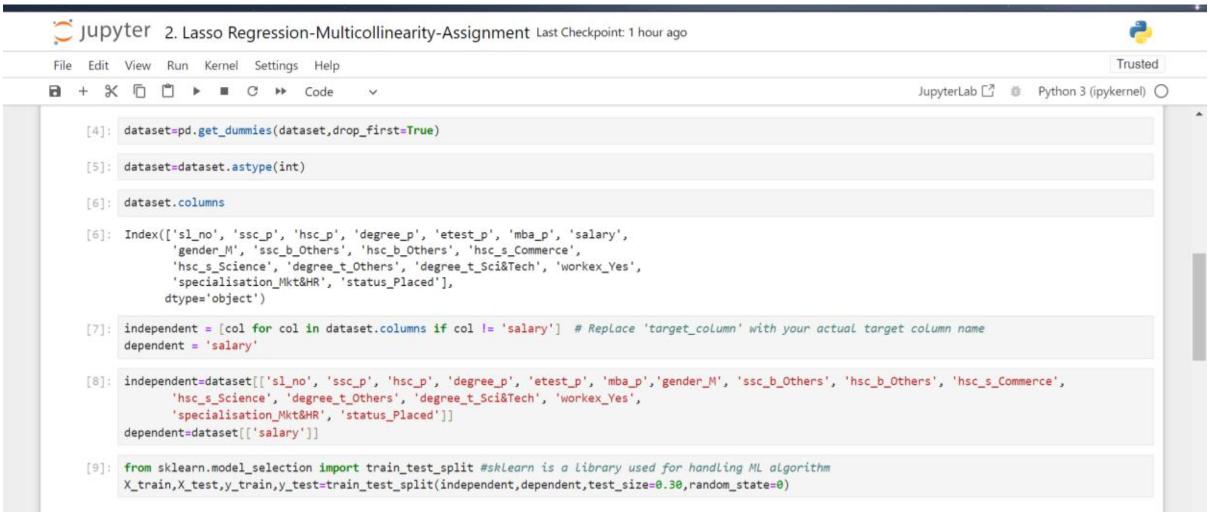
File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
[1]: import pandas as pd
[2]: dataset=pd.read_csv("Pre_processed_Placement.csv")
[3]: dataset
```

[3]:

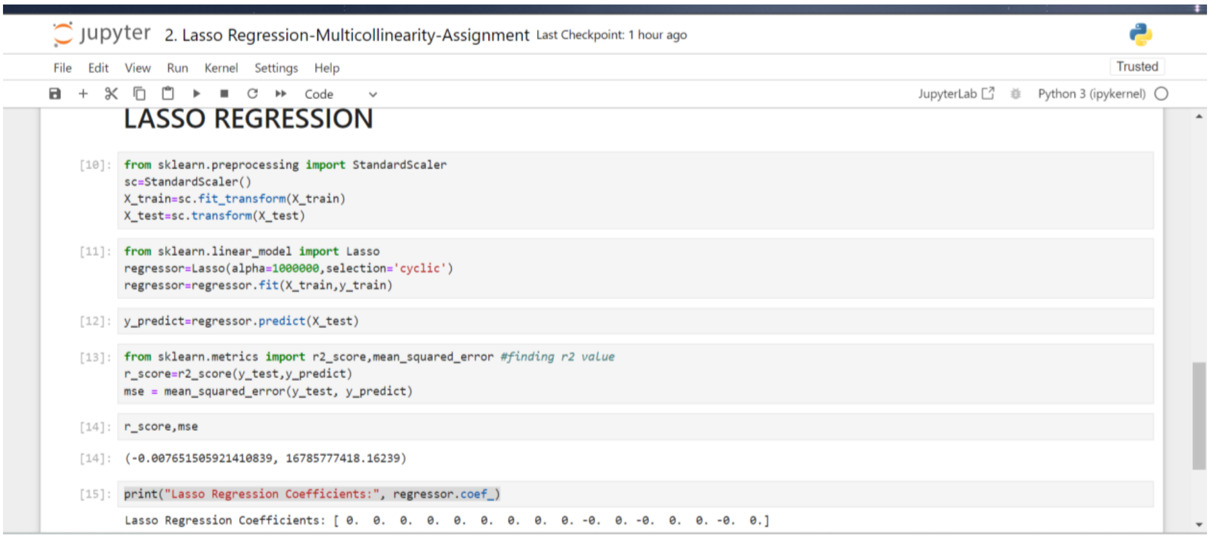
	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	0.0
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No	91.0	Mkt&Fin	74.49	Placed	400000.0
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No	74.0	Mkt&Fin	53.62	Placed	275000.0
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes	59.0	Mkt&Fin	69.72	Placed	295000.0


jupyter 2. Lasso Regression-Multicollinearity-Assignment Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
[4]: dataset=pd.get_dummies(dataset,drop_first=True)
[5]: dataset=dataset.astype(int)
[6]: dataset.columns
[6]: Index(['sl_no', 'ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p', 'salary',
       'gender_M', 'ssc_b_Others', 'hsc_b_Others', 'hsc_s_Commerce',
       'hsc_s_Science', 'degree_t_Others', 'degree_t_Sci&Tech', 'workex_Yes',
       'specialisation_Mkt&HR', 'status_Placed'],
      dtype='object')
[7]: independent = [col for col in dataset.columns if col != 'salary']  # Replace 'target_column' with your actual target column name
     dependent = 'salary'
[8]: independent=dataset[['sl_no', 'ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p','gender_M', 'ssc_b_Others', 'hsc_b_Others', 'hsc_s_Commerce',
       'hsc_s_Science', 'degree_t_Others', 'degree_t_Sci&Tech', 'workex_Yes',
       'specialisation_Mkt&HR', 'status_Placed']]
     dependent=dataset[['salary']]
[9]: from sklearn.model_selection import train_test_split #sklearn is a library used for handling ML algorithm
     X_train,X_test,y_train,y_test=train_test_split(independent,dependent,test_size=0.30,random_state=0)
```



- Summary
  - Lasso Regression is effective in handling multicollinearity by performing feature selection through its L1 penalty. It not only mitigates multicollinearity but also helps in creating a simpler, more interpretable model. The range of coefficients in Lasso regression can vary from very small values close to zero to moderate positive or negative values.
  - Regularization Parameter (α): Controls the strength of the penalty and determines the number of features selected.
  - By applying Lasso regression, we can reduce the impact of multicollinearity and potentially improve our model's performance and interpretability.

6. What is Principal Component Analysis (PCA)?

Principal Component Analysis (PCA) is a technique used to reduce the dimensionality of data while preserving as much variance as possible. It's particularly useful for controlling multicollinearity, which occurs when predictor variables in a regression model are highly correlated.

- How PCA Controls Multicollinearity?
  - Dimensionality Reduction:

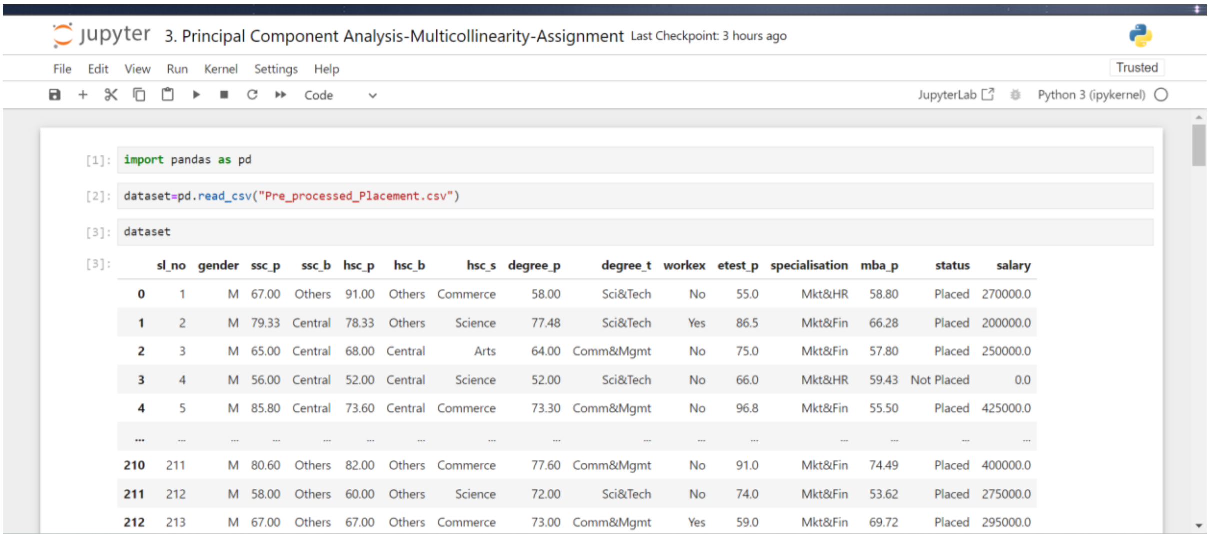
    PCA transforms the original correlated features into a set of linearly uncorrelated components called principal components.

These components are orthogonal to each other, meaning there's no multicollinearity between them.

- Feature Extraction**:**
  The principal components are linear combinations of the original features, and they capture the most variance in the data.
  By using a smaller number of principal components, we can reduce the number of variables in the model, thus addressing multicollinearity.

Illustration:

https://github.com/Rayenai/Bivariate-Analysis---Multicollinearity--Assignment



jupyter 3. Principal Component Analysis-Multicollinearity-Assignment Last Checkpoint: 3 hours ago

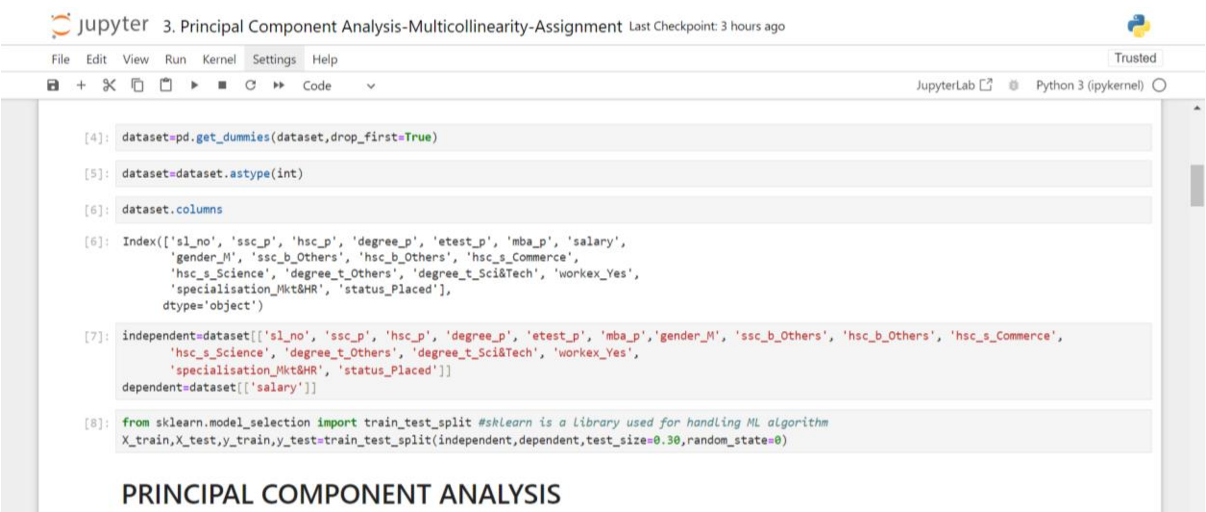
File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
[1]: import pandas as pd
[2]: dataset=pd.read_csv("Pre_processed_Placement.csv")
[3]: dataset
```

[3]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	0.0
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No	91.0	Mkt&Fin	74.49	Placed	400000.0
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No	74.0	Mkt&Fin	53.62	Placed	275000.0
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes	59.0	Mkt&Fin	69.72	Placed	295000.0



jupyter 3. Principal Component Analysis-Multicollinearity-Assignment Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
[4]: dataset=pd.get_dummies(dataset,drop_first=True)
[5]: dataset=dataset.astype(int)
[6]: dataset.columns
[6]: Index(['sl_no', 'ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p', 'salary',
        'gender_M', 'ssc_b_Others', 'hsc_b_Others', 'hsc_s_Commerce',
        'hsc_s_Science', 'degree_t_Others', 'degree_t_Sci&Tech', 'workex_Yes',
        'specialisation_Mkt&HR', 'status_Placed'],
       dtype='object')
[7]: independent=dataset[['sl_no', 'ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p','gender_M', 'ssc_b_Others', 'hsc_b_Others', 'hsc_s_Commerce',
        'hsc_s_Science', 'degree_t_Others', 'degree_t_Sci&Tech', 'workex_Yes',
        'specialisation_Mkt&HR', 'status_Placed']]
     dependent=dataset[['salary']]
[8]: from sklearn.model_selection import train_test_split #sklearn is a library used for handling ML algorithm
     X_train,X_test,y_train,y_test=train_test_split(independent,dependent,test_size=0.30,random_state=0)
```

PRINCIPAL COMPONENT ANALYSIS

All Bookmarks

jupyter 3. Principal Component Analysis-Multicollinearity-Assignment Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[9]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

[10]:
```
from sklearn.decomposition import PCA
pca = PCA()  # Choose number of components based on explained variance
X_pca = pca.fit_transform(X_train)
```

[11]:
```
from sklearn.linear_model import Ridge
regressor=Ridge(alpha=0.25,solver='auto')
regressor=regressor.fit(X_train,y_train)
```

[12]:
```
y_predict=regressor.predict(X_test)
```

[13]:
```
from sklearn.metrics import r2_score,mean_squared_error #finding r2 value
r_score=r2_score(y_test,y_predict)
mse = mean_squared_error(y_test, y_predict)
```

[14]:
```
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r_score}")
```

```
Mean Squared Error: 3232815672.658245
R^2 Score: 0.8059338272056609
```

jupyter 3. Principal Component Analysis-Multicollinearity-Assignment Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[16]:
```
print(pca.explained_variance_ratio_)
```

```
[0.20375502 0.15835064 0.10134497 0.08025629 0.07196321 0.06976795
 0.06122736 0.05408003 0.04625682 0.0371057  0.03065342 0.02887605
 0.02425574 0.01643373 0.01185178 0.00382128]
```

[17]:
```
dataset.isnull().sum()
```

[17]:
```
sl_no                    0
ssc_p                    0
hsc_p                    0
degree_p                 0
etest_p                  0
mba_p                    0
salary                   0
gender_M                 0
ssc_b_Others             0
hsc_b_Others             0
hsc_s_Commerce           0
hsc_s_Science            0
degree_t_Others          0
degree_t_Sci&Tech        0
workex_Yes               0
specialisation_Mkt&HR    0
status_Placed            0
dtype: int64
```

jupyter 3. Principal Component Analysis-Multicollinearity-Assignment Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[18]:
```
dataset.drop("sl_no",inplace=True,axis=1)
```

[19]:
```
dataset
```

[19]:

	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary	gender_M	ssc_b_Others	hsc_b_Others	hsc_s_Commerce	hsc_s_Science	degree_t_Others	degree_t_Sci&Tech	work
0	67	91	58	55	58	270000	1	1	1	1	0	0	1	
1	79	78	77	86	66	200000	1	0	1	0	1	0	1	
2	65	68	64	75	57	250000	1	0	0	0	0	0	0	
3	56	52	52	66	59	0	1	0	0	0	1	0	1	
4	85	73	73	96	55	425000	1	0	0	1	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210	80	82	77	91	74	400000	1	1	1	1	0	0	0	
211	58	60	72	74	53	275000	1	1	1	0	1	0	1	
212	67	67	73	59	69	295000	1	1	1	1	0	0	0	
213	74	66	58	70	60	204000	0	1	1	1	0	0	0	
214	62	58	53	89	60	0	1	0	1	0	1	0	0	

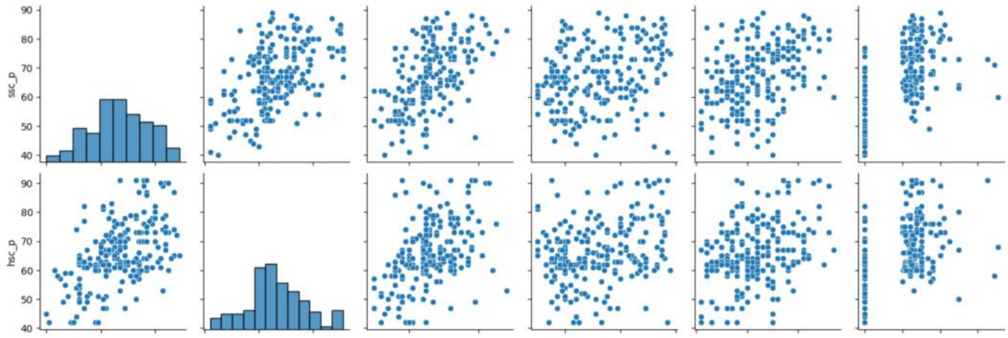
jupyter 3. Principal Component Analysis-Multicollinearity-Assignment Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[20]:
```
import seaborn as sns
sns.pairplot(dataset[['ssc_p','hsc_p','degree_p','etest_p','mba_p','salary']])
```

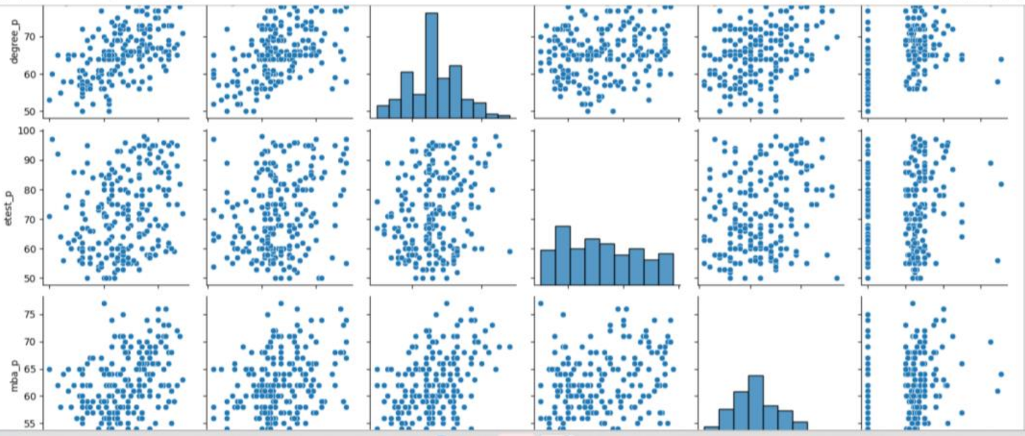
[20]: <seaborn.axisgrid.PairGrid at 0x165b4f67cd0>

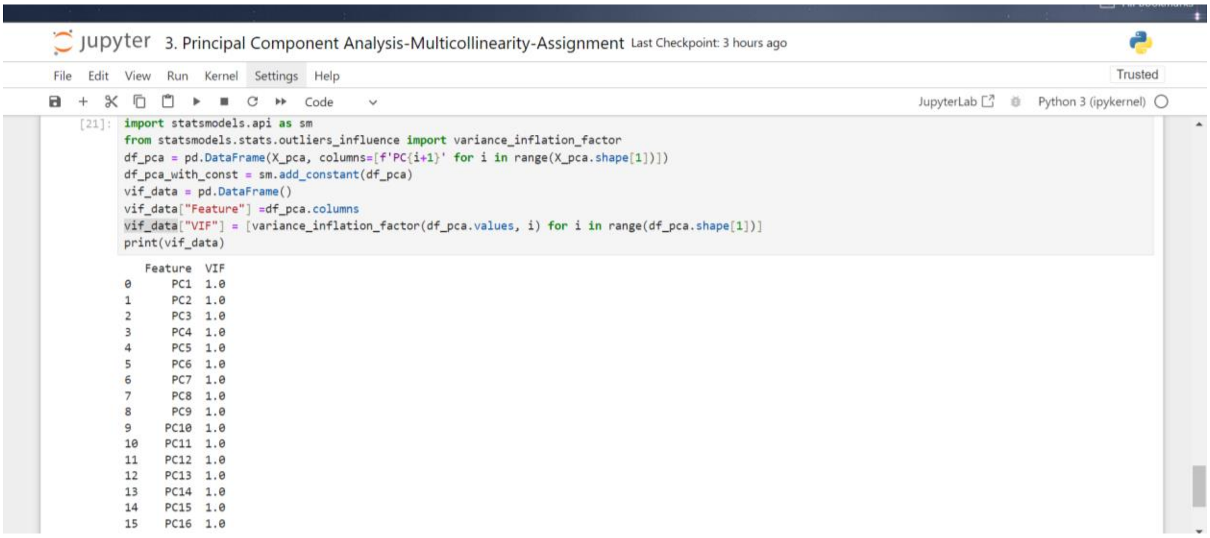


jupyter 3. Principal Component Analysis-Multicollinearity-Assignment Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)






```python
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
df_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(X_pca.shape[1])])
df_pca_with_const = sm.add_constant(df_pca)
vif_data = pd.DataFrame()
vif_data["Feature"] =df_pca.columns
vif_data["VIF"] = [variance_inflation_factor(df_pca.values, i) for i in range(df_pca.shape[1])]
print(vif_data)
```

```
   Feature  VIF
0      PC1  1.0
1      PC2  1.0
2      PC3  1.0
3      PC4  1.0
4      PC5  1.0
5      PC6  1.0
6      PC7  1.0
7      PC8  1.0
8      PC9  1.0
9     PC10  1.0
10    PC11  1.0
11    PC12  1.0
12    PC13  1.0
13    PC14  1.0
14    PC15  1.0
15    PC16  1.0
```

- Summary:

  By using PCA, we address multicollinearity by ensuring that the features used in the regression model are uncorrelated, leading to more stable and interpretable results.

7. What is Centering and Scaling?

Centering and scaling are fundamental preprocessing steps to reduce multicollinearity and improve the performance of regression models. Here's a breakdown of these concepts and how they help, along with code to implement them.

Centering:

This involves subtracting the mean of each feature from the feature values. It repositions the data so that its mean is zero. This is useful for standardizing features and can help in cases where the data features have different means.

Scaling:

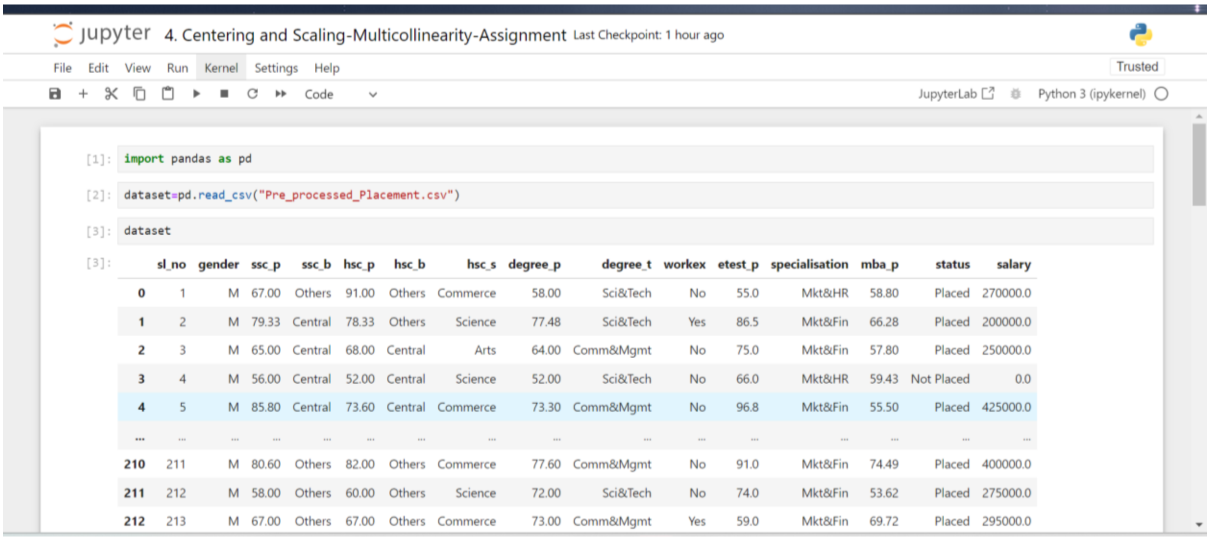
This involves dividing each feature by its standard deviation. Scaling transforms the feature values so that they have unit variance. This ensures that features are on the same scale, which is important for algorithms sensitive to feature scaling, such as Ridge and Lasso regression.

- Why Centering and Scaling help with Multicollinearity?

- **Multicollinearity**:
  When features are highly correlated, it can cause instability in the estimation of regression coefficients. Centering and scaling can mitigate this by ensuring that the features are on a similar scale and have a mean of zero.
- **Regularization**:
  Techniques like Ridge and Lasso regression add a penalty based on the size of coefficients. Centering and scaling make these penalties more effective and consistent across features.

Illustration:

https://github.com/Rayenai/Bivariate-Analysis---Multicollinearity--Assignment



	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	0.0
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No	91.0	Mkt&Fin	74.49	Placed	400000.0
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No	74.0	Mkt&Fin	53.62	Placed	275000.0
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes	59.0	Mkt&Fin	69.72	Placed	295000.0

jupyter 4. Centering and Scaling-Multicollinearity-Assignment Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[4]:
```
dataset=pd.get_dummies(dataset,drop_first=True)
```

[5]:
```
dataset=dataset.astype(int)
```

[6]:
```
independent = [col for col in dataset.columns if col != 'salary']  # Replace 'target_column' with your actual target column name
dependent = 'salary'
```

[7]:
```
independent=dataset[['sl_no', 'ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p','gender_M', 'ssc_b_Others', 'hsc_b_Others', 'hsc_s_Commerce',
       'hsc_s_Science', 'degree_t_Others', 'degree_t_Sci&Tech', 'workex_Yes',
       'specialisation_Mkt&HR', 'status_Placed']]
dependent=dataset[['salary']]
```

[8]:
```
from sklearn.model_selection import train_test_split #sklearn is a library used for handling ML algorithm
X_train,X_test,y_train,y_test=train_test_split(independent,dependent,test_size=0.30,random_state=0)
```

[9]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train_scaled=sc.fit_transform(X_train)
X_test_scaled=sc.transform(X_test)
```

CENTERING AND SCALING

jupyter 4. Centering and Scaling-Multicollinearity-Assignment Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[10]:
```
# Mean should be close to 0 and variance should be close to 1
```

[11]:
```
import numpy as np
#before satndarisation
print("Mean of scaled training features:\n", np.mean(X_train, axis=0))
print("Variance of scaled training features:\n", np.var(X_train, axis=0))
```

```
Mean of scaled training features:
 sl_no                    109.693333
ssc_p                    66.486667
hsc_p                    65.926667
degree_p                 65.646667
etest_p                  72.213333
mba_p                    61.413333
gender_M                  0.680000
ssc_b_Others              0.446667
hsc_b_Others              0.613333
hsc_s_Commerce            0.526667
hsc_s_Science             0.426667
degree_t_Others           0.060000
degree_t_Sci&Tech         0.273333
workex_Yes                0.340000
specialisation_Mkt&HR     0.460000
status_Placed             0.680000
dtype: float64
Variance of scaled training features:
```

jupyter 4. Centering and Scaling-Multicollinearity-Assignment Last Checkpoint: 1 hour ago

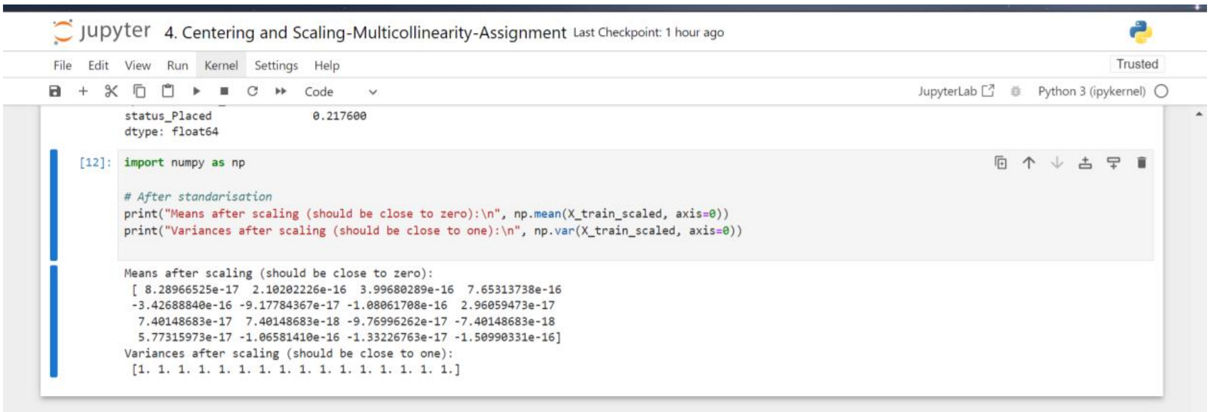
File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
Variance of scaled training features:
 sl_no                    3894.465956
ssc_p                    102.503156
hsc_p                    115.481289
degree_p                  53.961822
etest_p                  178.527822
mba_p                     35.655822
gender_M                   0.217600
ssc_b_Others               0.247156
hsc_b_Others               0.237156
hsc_s_Commerce             0.249289
hsc_s_Science              0.244622
degree_t_Others            0.056400
degree_t_Sci&Tech          0.198622
workex_Yes                 0.224400
specialisation_Mkt&HR      0.248400
status_Placed              0.217600
dtype: float64
```

[12]:
```
import numpy as np

# After standarisation
print("Means after scaling (should be close to zero):\n", np.mean(X_train_scaled, axis=0))
print("Variances after scaling (should be close to one):\n", np.var(X_train_scaled, axis=0))
```



Summary:

- StandardScaler:
  This scales features so that they have zero mean and unit variance. It’s crucial for methods like Ridge and Lasso regression, which are sensitive to the scale of the features.
- Fitting and Transforming:
  We can fit the scaler on the training data and then transform both the training and test data. This ensures that our test data is scaled consistently with our training data.
- Verification:
  After scaling, the mean of the scaled features should be close to zero, and the variance should be close to one.
- Centering and scaling features is a good practice to help manage multicollinearity and ensure that regularization methods like Ridge and Lasso regression work effectively. By standardizing your features, we can ensure that they contribute equally to the model, which helps in obtaining more stable and interpretable regression coefficients.