In [181]:

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


from sklearn.decomposition import PCA
from scipy.stats import zscore

# Read the dataset

mpg_df = pd.read_csv("d:\gli\dse\data\car-mpg.csv")  # download data from UCI and add
ed column header


# drop the car name column as it is useless for the model
car_name = mpg_df['car_name']
mpg_df = mpg_df.drop('car_name', axis=1)
mpg_df.head()
```

Out[181]:

	mpg	cyl	disp	hp	wt	acc	yr	origin	car_type
0	18.0	8	307.0	130	3504	12.0	70	1	0
1	15.0	8	350.0	165	3693	11.5	70	1	0
2	18.0	8	318.0	150	3436	11.0	70	1	0
3	16.0	8	304.0	150	3433	12.0	70	1	0
4	17.0	8	302.0	140	3449	10.5	70	1	0

In [182]:

```
# horsepower is an object type though it is supposed to be numeric. Check if all the
 rows in this column are digits

temp = pd.DataFrame(mpg_df.hp.str.isdigit())  # if the string is made of digits store
True else False  in the hp column
temp[temp['hp'] == False]   # from temp take only those rows where hp has false

# On inspecting records number 32, 126 etc, we find "?" in the columns. Replace them
 with "nan"
#Replace them with nan and remove the records from the data frame that have "nan"

mpg_df = mpg_df.replace('?', np.nan)
mpg_df = mpg_df.apply(lambda x: x.fillna(x.median()),axis=0)


# converting the hp column from object / string type to float
mpg_df['hp'] = mpg_df['hp'].astype('float64')
```

In [184]:

```
X = mpg_df[mpg_df.columns[1:-1]]   # The last column index is -1 and that is not cons
idered in the range
y = mpg_df["mpg"]
```

In [185]:

```
X = X.drop(['yr' , 'acc' , 'cyl' , 'origin' ] , axis=1)
```

In [186]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_stat
e=1)


print(X_train.shape)
print(X_test.shape)
print(y_test.shape)
```

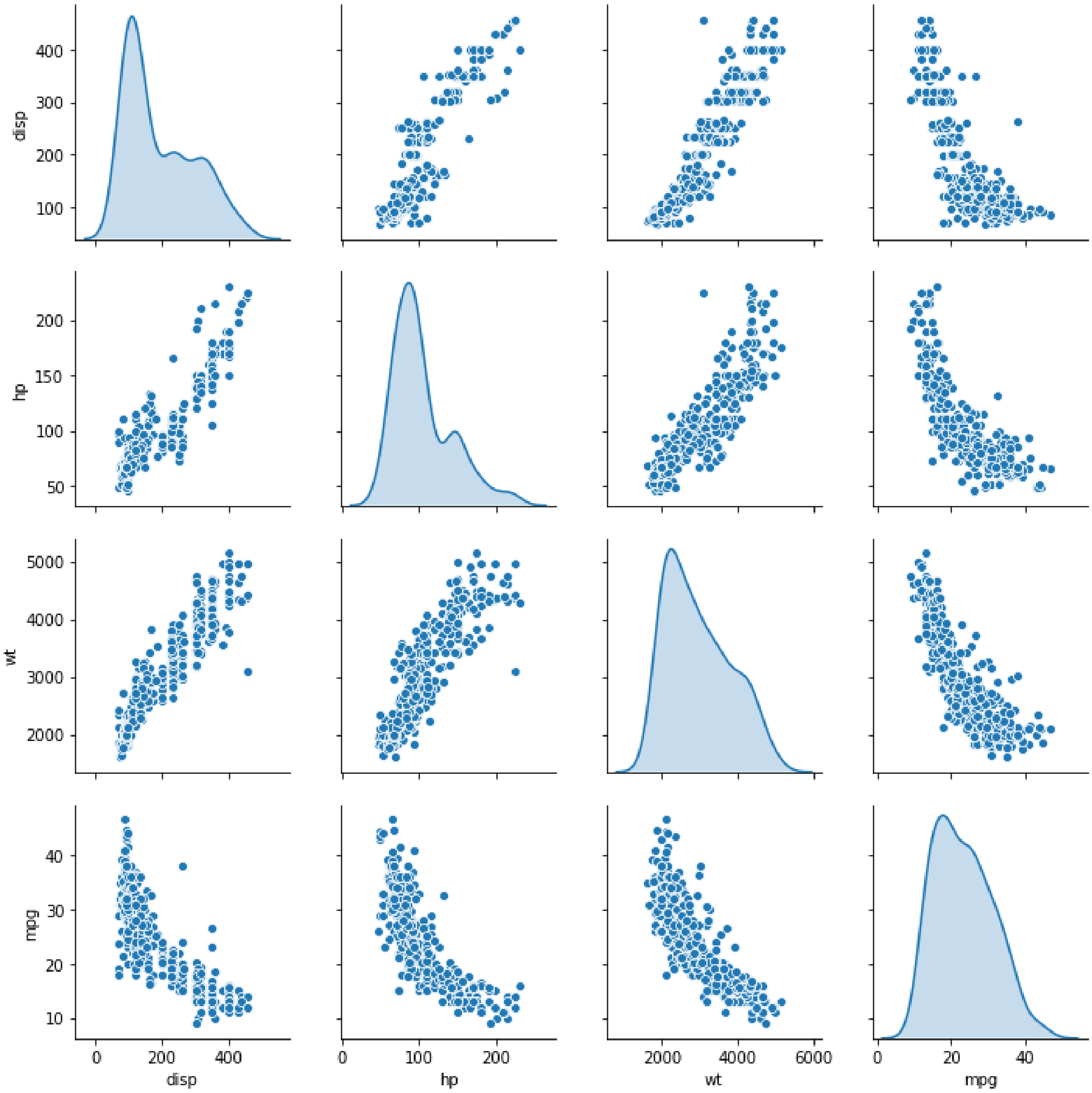
```
(278, 3)
(120, 3)
(120,)
```

In [187]:

```
#Visually inspect the covariance between independent dimensions and between mpg and i
ndependent dimensions

#sns.pairplot(mpg_df, diag_kind='kde')
sns.pairplot(X.join(y), diag_kind='kde')
```

Out[187]:

```
<seaborn.axisgrid.PairGrid at 0x1f32bf13388>
```



PCA starts from here

Step 1 - centre the data in independent variables

In [188]:

```
sc = StandardScaler()
X_train_std =  sc.fit_transform(X_train)

train_cov_matrix = np.cov(X_train_std.T)

print('Covariance Matrix \n%s', train_cov_matrix)  # The quantities in the matrix sho
uld reflect the observations in pairplot
```

```
Covariance Matrix
%s [[1.00361011 0.89032763 0.93969096]
 [0.89032763 1.00361011 0.87229407]
 [0.93969096 0.87229407 1.00361011]]
```

Step2 - Matrix decomposition

In [189]:

```
eigenvalues, eigenvectors = np.linalg.eig(train_cov_matrix)
print('Eigen Vectors \n%s', eigenvectors)
print('\n Eigen Values \n%s', eigenvalues)
```

```
Eigen Vectors
%s [[ 0.58340201  0.75238213  0.30588107]
 [ 0.56889058 -0.10976035 -0.81505593]
 [ 0.5796599  -0.64951813  0.49205752]]

 Eigen Values
%s [2.80545542 0.06250723 0.14286768]
```

Step 3 - Sort the Eigen values in descending order (The order may not be descending by default as in the example above)

In [190]:

```
# Make a set of (eigenvalue, eigenvector) pairs
train_eig_pairs = [(eigenvalues[index], eigenvectors[index, :]) for index in range(le
n(eigenvalues))]

# Sort the (eigenvalue, eigenvector) pairs from highest to lowest with respect to eig
envalue
train_eig_pairs.sort(reverse = True)

train_eig_pairs
```

Out[190]:

```
[(2.8054554166299472, array([0.58340201, 0.75238213, 0.30588107])),
 (0.14286768248591242, array([ 0.5796599 , -0.64951813,  0.49205752])),
 (0.0625072257938851, array([ 0.56889058, -0.10976035, -0.81505593]))]
```

Step 4 - Separate the sorted Eigen values and Eigen vectors for subsequent use in graphing

In [191]:

```
train_eigvalues_sorted = [train_eig_pairs[index][0] for index in range(len(eigenvalue
s))]
train_eigvectors_sorted = [train_eig_pairs[index][1] for index in range(len(eigenvalu
es))]

# Let's confirm our sorting worked, print out eigenvalues
print('Eigenvalues in descending order: \n%s' %train_eigvalues_sorted)
```

```
Eigenvalues in descending order:
[2.8054554166299472, 0.14286768248591242, 0.0625072257938851]
```

Step 5 - convert the Eigen values to %age of total covariance explained and create a cumulative sum

In [192]:

```
tot = sum(eigenvalues) # Sum up all the Eigen values to reflect the total covariance
 captured from original feature space


#%age of total covariance explained = [(i / tot) for i in sorted(train_eigvalues_sort
ed, reverse=True)]
# array of variance explained by each Eigen vector will be generated

var_explained = [(i / tot) for i in train_eigvalues_sorted]




# an array of cumulative covariance captured by the Eigen vectors together

cum_var_exp = np.cumsum(var_explained)
```

In [193]:

```
print(var_explained)
```

```
[0.9317879501276931, 0.04745125665299493, 0.020760793219311974]
```

In [194]:

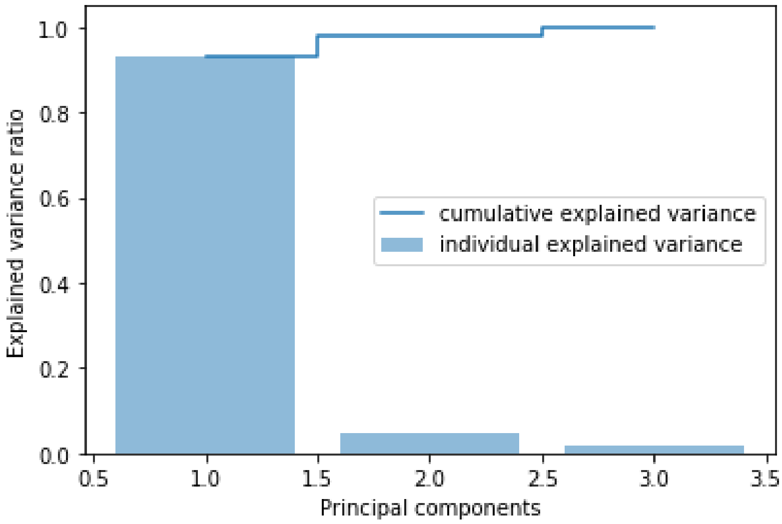
```
print(cum_var_exp)
```

```
[0.93178795 0.97923921 1.        ]
```

Step 6 - Plot the % covariance captured by each Eigen vector and the cumulative covariance

In [195]:

```
plt.bar(range(1,4), var_explained, alpha=0.5, align='center', label='individual expla
ined variance')
plt.step(range(1,4),cum_var_exp, where= 'mid', label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc = 'best')
plt.show()
```



Step 7 - Drop principal components that capture insignificant amount of the covariance compared to others and project the original data into the reduced PC dimension space

In [196]:

```
# P_reduce represents reduced mathematical space....

P_reduce = np.array(train_eigvectors_sorted[0:3])   # In this case using all the thre
e PC dimensions- not reducing  actually

X_std_3D = np.dot(X_train_std,P_reduce.T)   # projecting original data into principal
component dimensions

Proj_train_data_df = pd.DataFrame(X_std_3D)  # converting array to dataframe for pair
plot
```

In [197]:

```
Proj_data_df.head()
```

Out[197]:

	0	1	2
0	-0.839348	0.559449	-0.079805
1	-0.457623	0.845000	-1.197925
2	-0.091813	-0.593066	-0.869853
3	0.774022	-0.404734	1.776637
4	0.436113	0.702259	-2.170559

Step 8 - Build the linear regression model on the training data from PC dimension feature space

In [198]:

```
# Import Linear Regression machine learning library
from sklearn.linear_model import LinearRegression

regression_model = LinearRegression()
regression_model.fit(Proj_train_data_df, y_train)

regression_model.coef_
```

Out[198]:

```
array([-2.88183682, -1.59200383,  3.64614943])
```

In [199]:

```
regression_model.intercept_
```

Out[199]:

```
23.600719424460433
```

In [200]:

```
regression_model.score(Proj_train_data_df, y_train)
```

Out[200]:

```
0.686444212315383
```

Step 9 - Test the model on projected test data i.e. test data mapped to PC dimension feature space

In [201]:

```
X_test_std =  sc.fit_transform(X_test) # Standardize the test data using Zscores -  c
entring the data


X_test_std_3D = np.dot(X_test_std,P_reduce.T)   # projecting original data into princ
ipal component dimensions

Proj_test_data_df = pd.DataFrame(X_test_std_3D)  # converting array to dataframe for
 pairplot
```

In [202]:

```
Proj_test_data_df.shape
```

Out[202]:

```
(120, 3)
```

In [203]:

```
regression_model.score(Proj_test_data_df, y_test)
```

Out[203]:

```
0.7488458012225744
```

Visual Analysis Pitfall

Step 10 - Visually analyise relation between target (mpg) and the principal components.

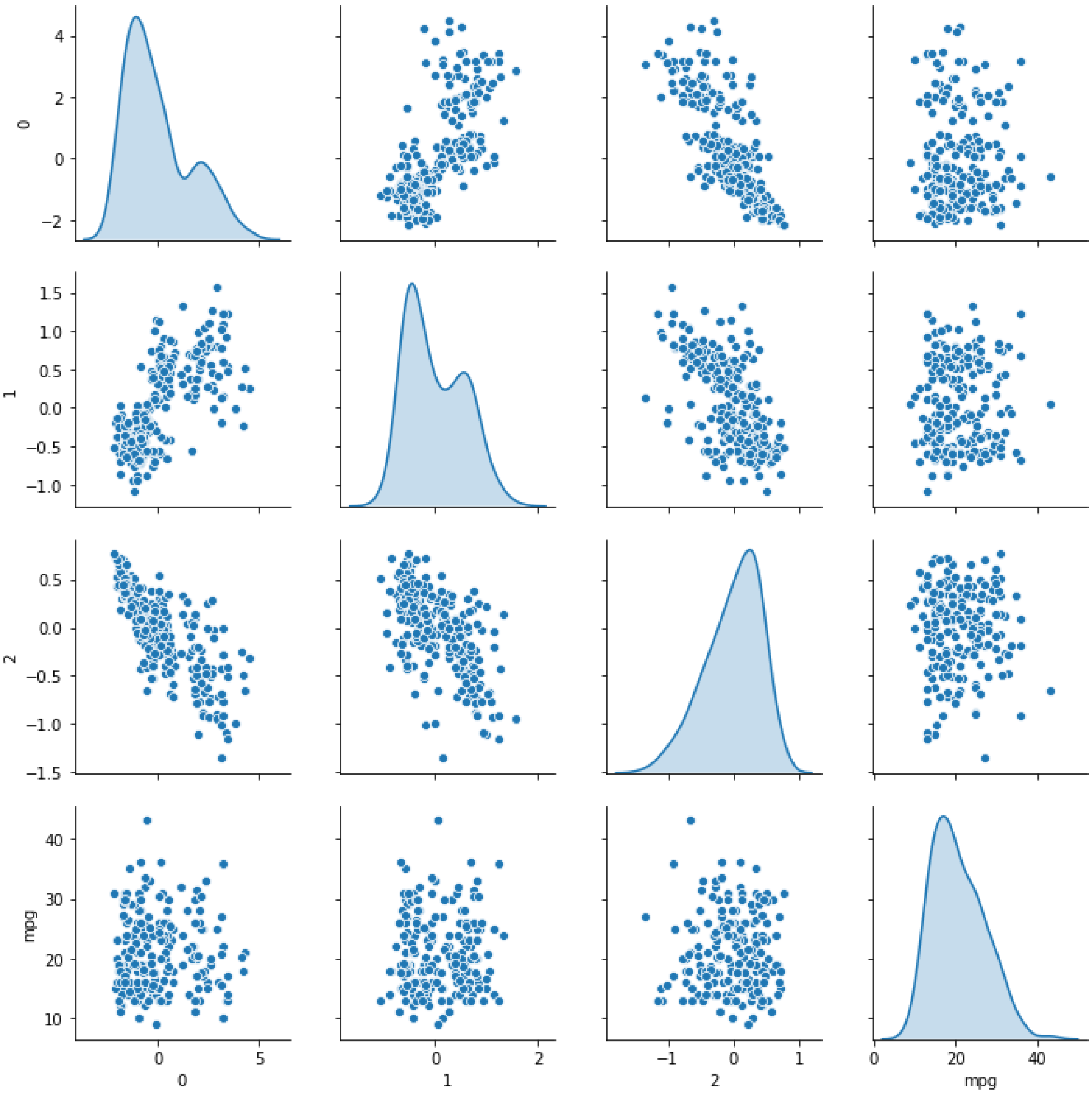
In [204]:

```
#Let us check it visually

Proj_train_data_df = Proj_train_data_df.join(y_train)

sns.pairplot(Proj_train_data_df, diag_kind='kde')
```

```
d:\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:447: Run
timeWarning: invalid value encountered in greater
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two c
olumns.
d:\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:447: Run
timeWarning: invalid value encountered in less
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two c
olumns.
```

Out[204]:

```
<seaborn.axisgrid.PairGrid at 0x1f32c99bb08>
```



Note

1. Strong collinearity between original feature is almost gone as is evidet from the scatter plots. There are some weak collinearity still but that is likely due to the curvilinear distributions in orignal space and could also be noise
2. The scatter plot between "Mpg" and principal components show very poor collinearity!!! This is surprising as the scores in training and testing were 60% and 70%... Not good but not possible with this kind of scatter plots
3. This inconsistency is because of the unwanted changes that happend when we transformed the data using standard scaler
4. When a data is randomly split into training and testing datasets i.e. (X_train, y-train), (X_test, y_test) and the X_train and X_test are transformed using standard scaler or anyother tool, the internal index of the records in both the datasets change. Though the records physically remain the same order and the relation with the y_train and y_test remain intact.
5. When the join is done between the train data and y_train for e.g. Proj_train_data_df and y_train above, the join is done using index leading to incorrect joins! Some cases it will introduce Nan in the y_train as the y_train does not have the index value which Proj_train_data_df has!. This incorrect join leads to incorrect pairplot between the target and independent variables
6. To check this out, list out the records in X_train , Y_train (their indexes will be same). Compare them with X_train_std index. You will see a new set of indexes intialized from 0. (look below)
7. Pairplot selects records by index from X_train, Y_train to do the plot and that is a mistake because the X_train index has changed! It will pick up incorrect combination of X_train_std and y to do the plot! That is what is happening above

In [205]:

```
X_train
```

Out[205]:

	disp	hp	wt
350	105.0	63.0	2215
59	97.0	54.0	2254
120	121.0	112.0	2868
12	400.0	150.0	3761
349	91.0	68.0	1985
...	...	...	...
393	140.0	86.0	2790
255	140.0	88.0	2720
72	304.0	150.0	3892
235	97.0	75.0	2265
37	232.0	100.0	3288

278 rows × 3 columns

In [206]:

```
y_train
```

Out[206]:

```
350    34.7
59     23.0
120    19.0
12     15.0
349    34.1
       ...
393    27.0
255    25.1
72     15.0
235    26.0
37     18.0
Name: mpg, Length: 278, dtype: float64
```

In [207]:

```
Proj_train_data_df
```

Out[207]:

	0	1	2	mpg
0	-1.561221	-0.221144	0.348137	18.0
1	-1.773192	-0.088100	0.292247	15.0
2	-0.251153	-0.597717	-0.336234	18.0
3	2.427868	0.848754	0.245064	16.0
4	-1.624289	-0.521140	0.477503	NaN
...	...	...	...	...
273	-0.691996	-0.085425	-0.079082	23.9
274	-0.677263	-0.160818	-0.017372	NaN
275	1.926872	0.380156	-0.416169	17.0
276	-1.348289	-0.445063	0.220225	21.6
277	0.294583	0.484709	-0.088133	16.2

278 rows × 4 columns

Rectification - To rectify this situation, copy the index of X_train into X_train_std

In [208]:

```
Proj_train_data_df.pop("mpg")  #remove mpg column
Proj_train_data_df.index = X_train.index  # (Restoring the original index into the pr
oject data)
Proj_train_data_df = Proj_train_data_df.join(y_train)  # rejoining after resetting th
e index
```

In [209]:

```
Proj_train_data_df  # observe the index has been restored and there are no visible "N
aN" in the target column
```

Out[209]:

	0	1	2	mpg
350	-1.561221	-0.221144	0.348137	34.7
59	-1.773192	-0.088100	0.292247	23.0
120	-0.251153	-0.597717	-0.336234	19.0
12	2.427868	0.848754	0.245064	15.0
349	-1.624289	-0.521140	0.477503	34.1
...	...	...	...	...
393	-0.691996	-0.085425	-0.079082	27.0
255	-0.677263	-0.160818	-0.017372	25.1
72	1.926872	0.380156	-0.416169	15.0
235	-1.348289	-0.445063	0.220225	26.0
37	0.294583	0.484709	-0.088133	18.0

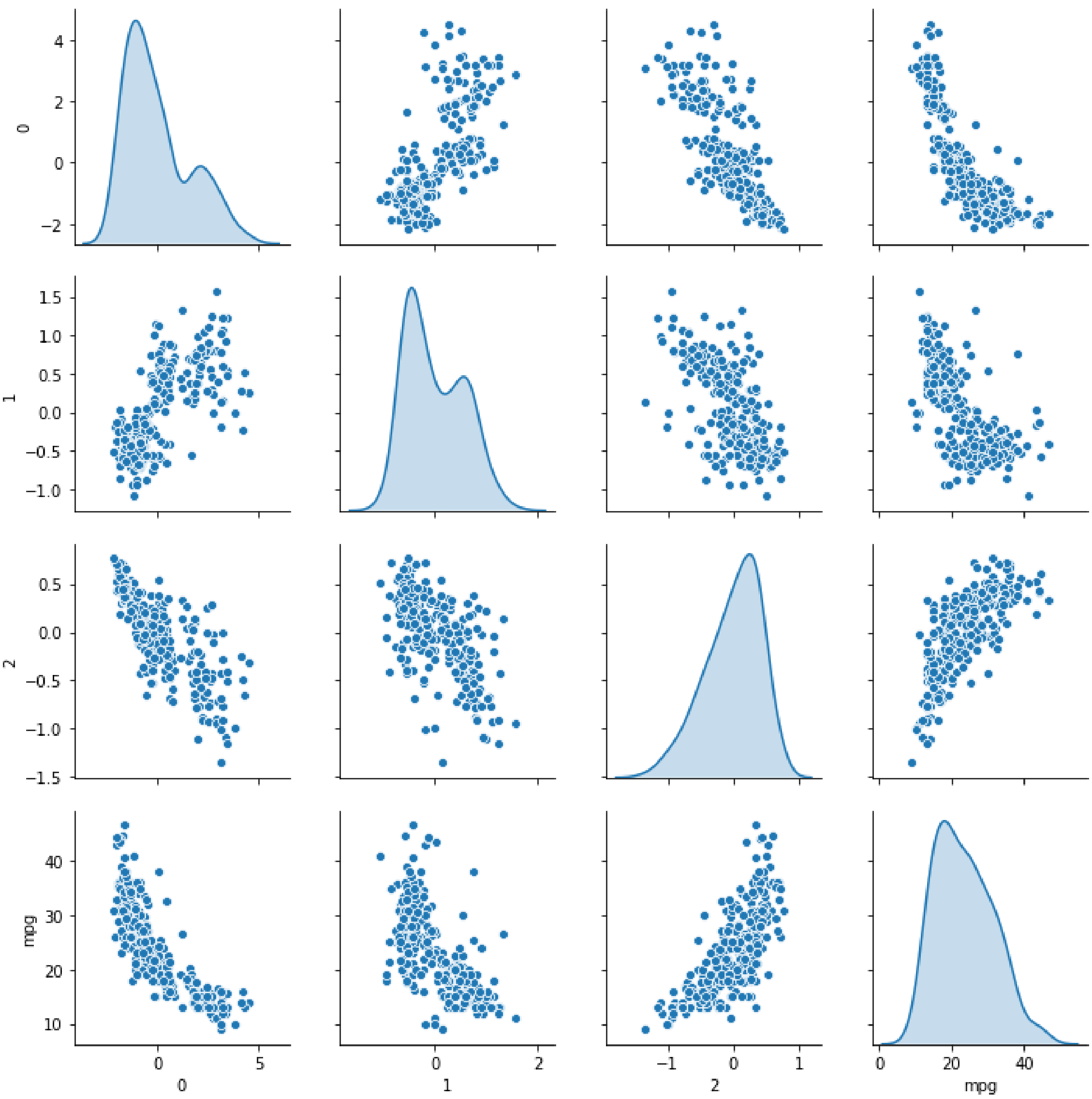
278 rows × 4 columns

In [210]:

```
sns.pairplot(Proj_train_data_df, diag_kind = "kde")
```

Out[210]:

```
<seaborn.axisgrid.PairGrid at 0x1f32e118e48>
```



In [ ]:

```

```