In [1]:
```
from pydoc import help
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from scipy import stats
from IPython.display import display, HTML
%matplotlib inline
np.set_printoptions(suppress=True)
DISPLAY_MAX_ROWS = 20
pd.set_option('display.max_rows', DISPLAY_MAX_ROWS)
```

In [3]:
```
data = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/w
data.columns = ["V"+str(i) for i in range(1, len(data.columns)+1)]
data.V1 = data.V1.astype(str)
X = data.loc[:, "V2":]
y = data.V1
data
```

Out[3]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows × 14 columns

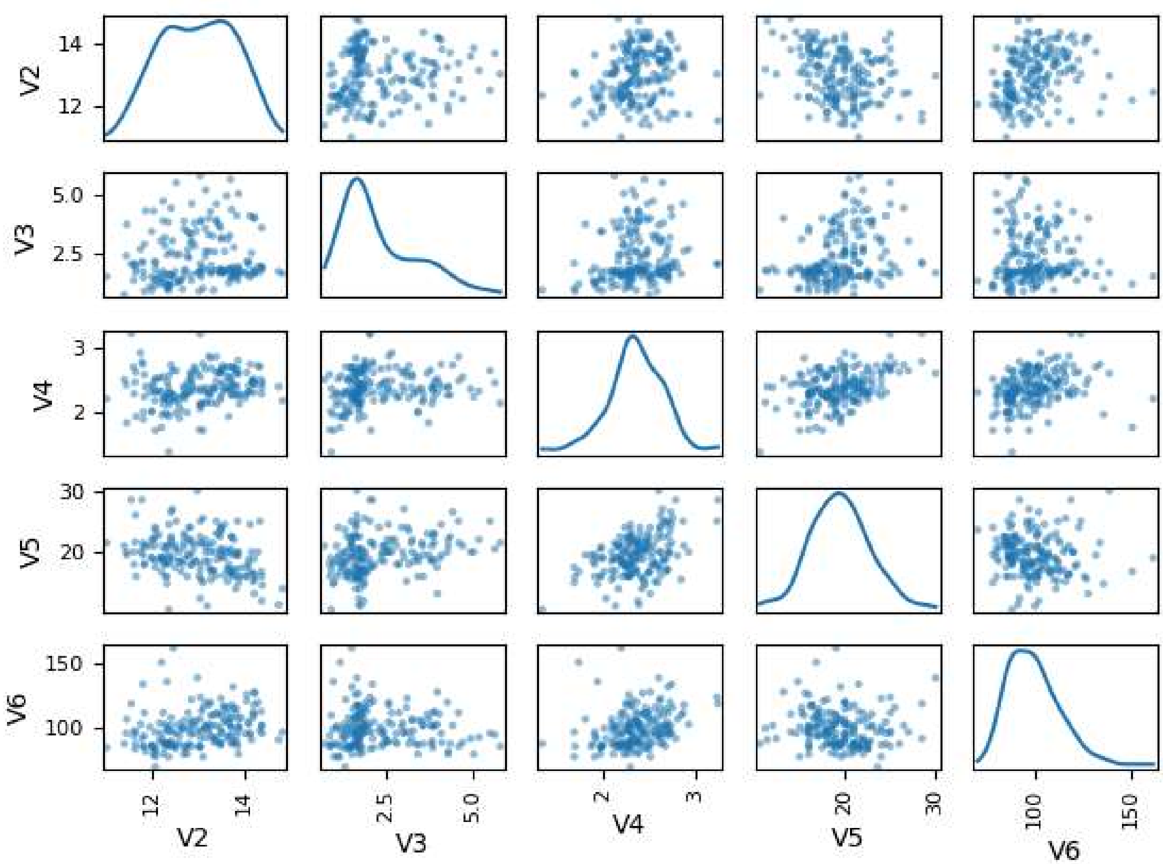
In [4]:
```
data.loc[:, "V2":"V6"]
```

Out[4]:

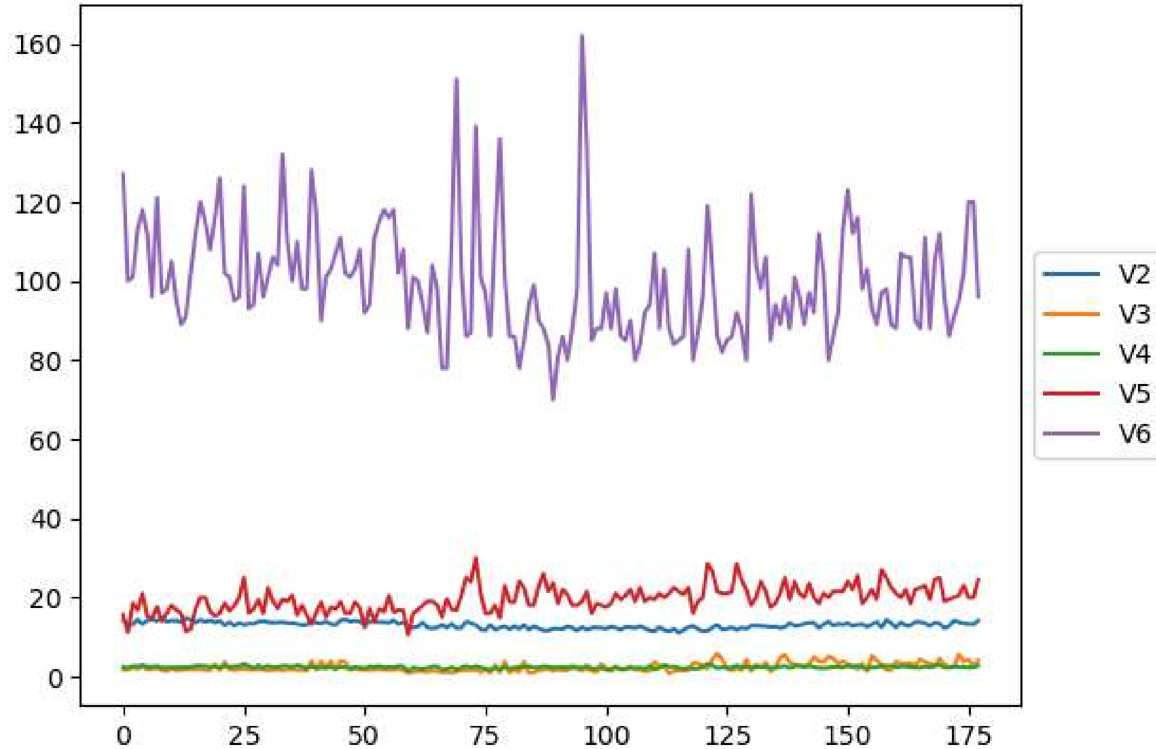
	V2	V3	V4	V5	V6
0	14.23	1.71	2.43	15.6	127
1	13.20	1.78	2.14	11.2	100
2	13.16	2.36	2.67	18.6	101
3	14.37	1.95	2.50	16.8	113
4	13.24	2.59	2.87	21.0	118
...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95
174	13.40	3.91	2.48	23.0	102
175	13.27	4.28	2.26	20.0	120
176	13.17	2.59	2.37	20.0	120
177	14.13	4.10	2.74	24.5	96

178 rows × 5 columns

In [7]:
```
pd.plotting.scatter_matrix(data.loc[:, "V2":"V6"], diagonal="kde")
plt.tight_layout()
plt.show()
```



In [6]:
```
ax = data[["V2","V3","V4","V5","V6"]].plot()
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5));
```



In [8]:
```
X.apply(np.mean)
```

Out[8]:
```
V2       13.000618
V3        2.336348
V4        2.366517
V5       19.494944
V6       99.741573
V7        2.295112
V8        2.029270
V9        0.361854
V10       1.590899
V11       5.058090
V12       0.957449
V13       2.611685
V14     746.893258
dtype: float64
```

In [9]:
```
X.apply(np.std)
```

Out[9]:
```
V2       0.809543
V3       1.114004
V4       0.273572
V5       3.330170
V6      14.242308
V7       0.624091
V8       0.996049
V9       0.124103
V10      0.570749
V11      2.311765
V12      0.227929
V13      0.707993
V14    314.021657
dtype: float64
```

In [10]:
```
class2data = data[y=="2"]
class2data.loc[:, "V2":].apply(np.mean)
class2data.loc[:, "V2":].apply(np.std)
```

Out[10]:
```
V2       0.534162
V3       1.008391
V4       0.313238
V5       3.326097
V6      16.635097
V7       0.541507
V8       0.700713
V9       0.123085
V10      0.597813
V11      0.918393
V12      0.201503
V13      0.493064
V14    156.100173
dtype: float64
```

In [11]:
```
def printMeanAndSdByGroup(variables, groupvariable):
 data_groupby = variables.groupby(groupvariable)
 print("## Means:")
 display(data_groupby.apply(np.mean))
 print("\n## Standard deviations:")
 display(data_groupby.apply(np.std))
 print("\n## Sample sizes:")
 display(pd.DataFrame(data_groupby.apply(len)))
```

In [12]:
```
printMeanAndSdByGroup(X, y)
```

```
## Means:

V1
1    98.081473
2    51.077883
3    60.259487
dtype: float64


## Standard deviations:
```

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	0.458192	0.682689	0.225233	2.524651	10.409595	0.336077	0.394111	0.069453	0.408602
2	0.534162	1.008391	0.313238	3.326097	16.635097	0.541507	0.700713	0.123085	0.597813
3	0.524689	1.076514	0.182756	2.234515	10.776433	0.353233	0.290431	0.122840	0.404555

```
## Sample sizes:
```

V1	0
1	59
2	71
3	48

In [13]:
```
corr = stats.pearsonr(X.V2, X.V3)
print("p-value:\t", corr[1])
print("cor:\t\t", corr[0])
```

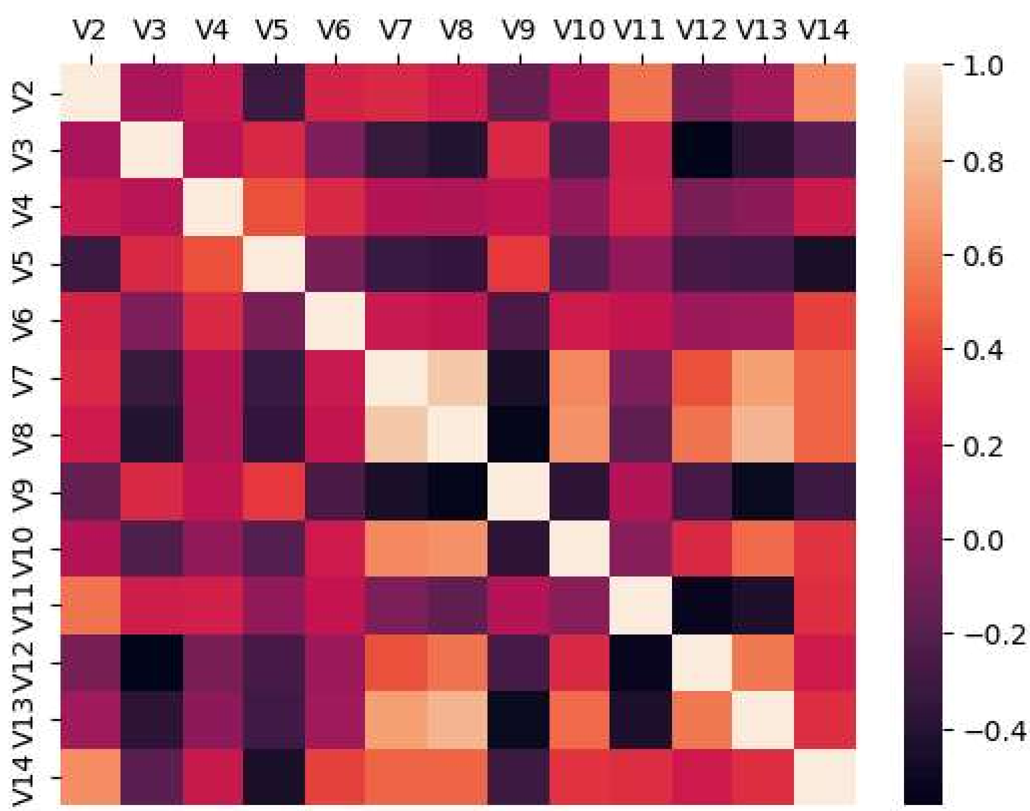
```
p-value:         0.21008198597074274
cor:             0.09439694091041399
```

In [14]:
```
corrmat = X.corr()
corrmat
```

Out[14]:

	V2	V3	V4	V5	V6	V7	V8	V9	
V2	1.000000	0.094397	0.211545	-0.310235	0.270798	0.289101	0.236815	-0.155929	0.136
V3	0.094397	1.000000	0.164045	0.288500	-0.054575	-0.335167	-0.411007	0.292977	-0.220
V4	0.211545	0.164045	1.000000	0.443367	0.286587	0.128980	0.115077	0.186230	0.009
V5	-0.310235	0.288500	0.443367	1.000000	-0.083333	-0.321113	-0.351370	0.361922	-0.197
V6	0.270798	-0.054575	0.286587	-0.083333	1.000000	0.214401	0.195784	-0.256294	0.236
V7	0.289101	-0.335167	0.128980	-0.321113	0.214401	1.000000	0.864564	-0.449935	0.612
V8	0.236815	-0.411007	0.115077	-0.351370	0.195784	0.864564	1.000000	-0.537900	0.652
V9	-0.155929	0.292977	0.186230	0.361922	-0.256294	-0.449935	-0.537900	1.000000	-0.365
V10	0.136698	-0.220746	0.009652	-0.197327	0.236441	0.612413	0.652692	-0.365845	1.000
V11	0.546364	0.248985	0.258887	0.018732	0.199950	-0.055136	-0.172379	0.139057	-0.025
V12	-0.071747	-0.561296	-0.074667	-0.273955	0.055398	0.433681	0.543479	-0.262640	0.295
V13	0.072343	-0.368710	0.003911	-0.276769	0.066004	0.699949	0.787194	-0.503270	0.519
V14	0.643720	-0.192011	0.223626	-0.440597	0.393351	0.498115	0.494193	-0.311385	0.330

In [15]:
```
sns.heatmap(corrmat, vmax=1., square=False).xaxis.tick_top()
```



In [17]:
```
def mosthighlycorrelated(mydataframe, numtoreport):
    # find the correlations
    cormatrix = mydataframe.corr()
    # set the correlations on the diagonal or lower triangle to zero,
    # so they will not be reported as the highest ones:
    cormatrix *= np.tri(*cormatrix.values.shape, k=-1).T
    # find the top n correlations
    cormatrix = cormatrix.stack()
    cormatrix = cormatrix.reindex(cormatrix.abs().sort_values(ascending=False).in
    # assign human-friendly names
    cormatrix.columns = ["FirstVariable", "SecondVariable", "Correlation"]
    return cormatrix.head(numtoreport)
```

In [18]:
```
mosthighlycorrelated(X, 10)
```

Out[18]:

	FirstVariable	SecondVariable	Correlation
0	V7	V8	0.864564
1	V8	V13	0.787194
2	V7	V13	0.699949
3	V8	V10	0.652692
4	V2	V14	0.643720
5	V7	V10	0.612413
6	V12	V13	0.565468
7	V3	V12	-0.561296
8	V2	V11	0.546364
9	V8	V12	0.543479

In [33]:
```
standardisedX = scale(X)
standardisedX = pd.DataFrame(standardisedX, index=X.index, columns=X.columns)
```

In [34]:
```
pca = PCA().fit(standardisedX)
```

In [35]:
```
def pca_summary(pca, standardised_data, out=True):
    names = ["PC"+str(i) for i in range(1, len(pca.explained_variance_ratio_)+1)]
    a = list(np.std(pca.transform(standardised_data), axis=0))
    b = list(pca.explained_variance_ratio_)
    c = [np.sum(pca.explained_variance_ratio_[:i]) for i in range(1, len(pca.expl
    columns = pd.MultiIndex.from_tuples([("sdev", "Standard deviation"), ("varpro
    summary = pd.DataFrame(zip(a, b, c), index=names, columns=columns)
    if out:
        print("Importance of components:")
        display(summary)
    return summary
```

In [36]:
```
summary = pca_summary(pca, standardisedX)
```

```
Importance of components:
```

	sdev	varprop	cumprop
	Standard deviation	Proportion of Variance	Cumulative Proportion
PC1	2.169297	0.361988	0.361988
PC2	1.580182	0.192075	0.554063
PC3	1.202527	0.111236	0.665300
PC4	0.958631	0.070690	0.735990
PC5	0.923704	0.065633	0.801623
PC6	0.801035	0.049358	0.850981
PC7	0.742313	0.042387	0.893368
PC8	0.590337	0.026807	0.920175
PC9	0.537476	0.022222	0.942397
PC10	0.500902	0.019300	0.961697
PC11	0.475172	0.017368	0.979066
PC12	0.410817	0.012982	0.992048
PC13	0.321524	0.007952	1.000000

In [41]:
```
summary.sdev
```

Out[41]:

	Standard deviation
PC1	2.169297
PC2	1.580182
PC3	1.202527
PC4	0.958631
PC5	0.923704
PC6	0.801035
PC7	0.742313
PC8	0.590337
PC9	0.537476
PC10	0.500902
PC11	0.475172
PC12	0.410817
PC13	0.321524

In [38]:
```
lda = LinearDiscriminantAnalysis().fit(X, y)
```

In [39]:
```
def pretty_scalings(lda, X, out=False):
    ret = pd.DataFrame(lda.scalings_, index=X.columns, columns=["LD"+str(i+1)
    for i in range(lda.scalings_.shape[1])])
    if out:
        print("Coefficients of linear discriminants:")
        display(ret)
    return ret
pretty_scalings_ = pretty_scalings(lda, X, out=True)
```

```
Coefficients of linear discriminants:
```

	LD1	LD2
V2	0.403400	0.871793
V3	-0.165255	0.305380
V4	0.369075	2.345850
V5	-0.154798	-0.146381
V6	0.002163	-0.000463
V7	-0.618052	-0.032213
V8	1.661191	-0.491998
V9	1.495818	-1.630954
V10	-0.134093	-0.307088
V11	-0.355056	0.253231
V12	0.818036	-1.515634
V13	1.157559	0.051184
V14	0.002691	0.002853

In [43]:
```
sns.lmplot("LD1", "LD2", lda_values["x"].join(y), hue="V1", fit_reg=False);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[43], line 1
----> 1 sns.lmplot("LD1", "LD2", lda_values["x"].join(y), hue="V1", fit_reg=F
alse)

NameError: name 'lda_values' is not defined
```

In [ ]:
```

```