Wine Quality Dataset

Notebook that will used unsupervised learning techniques to orgaize data into clusters based on the features of the wine

PCA will be performed on the dataset to observe the effect of dimensionality reduction, before fitting clustering models to the data

Load Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Load Data & Inspect

In [3]:
```
df = pd.read_csv('Wine_Quality_Data.csv')
```

In [5]:
```
df.head()
```

Out[5]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphat
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  color                 6497 non-null   object 
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

In [7]:
```
df.describe()
```

Out[7]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000
mean	7.215307	0.339666	0.318633	5.443235	0.056034	30.525319	115.744574	0.994697
std	1.296434	0.164636	0.145318	4.757804	0.035034	17.749400	56.521855	0.002999
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980

In [8]:
```
df.isna().sum()
```

Out[8]:
```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
color                   0
dtype: int64
```

No null values

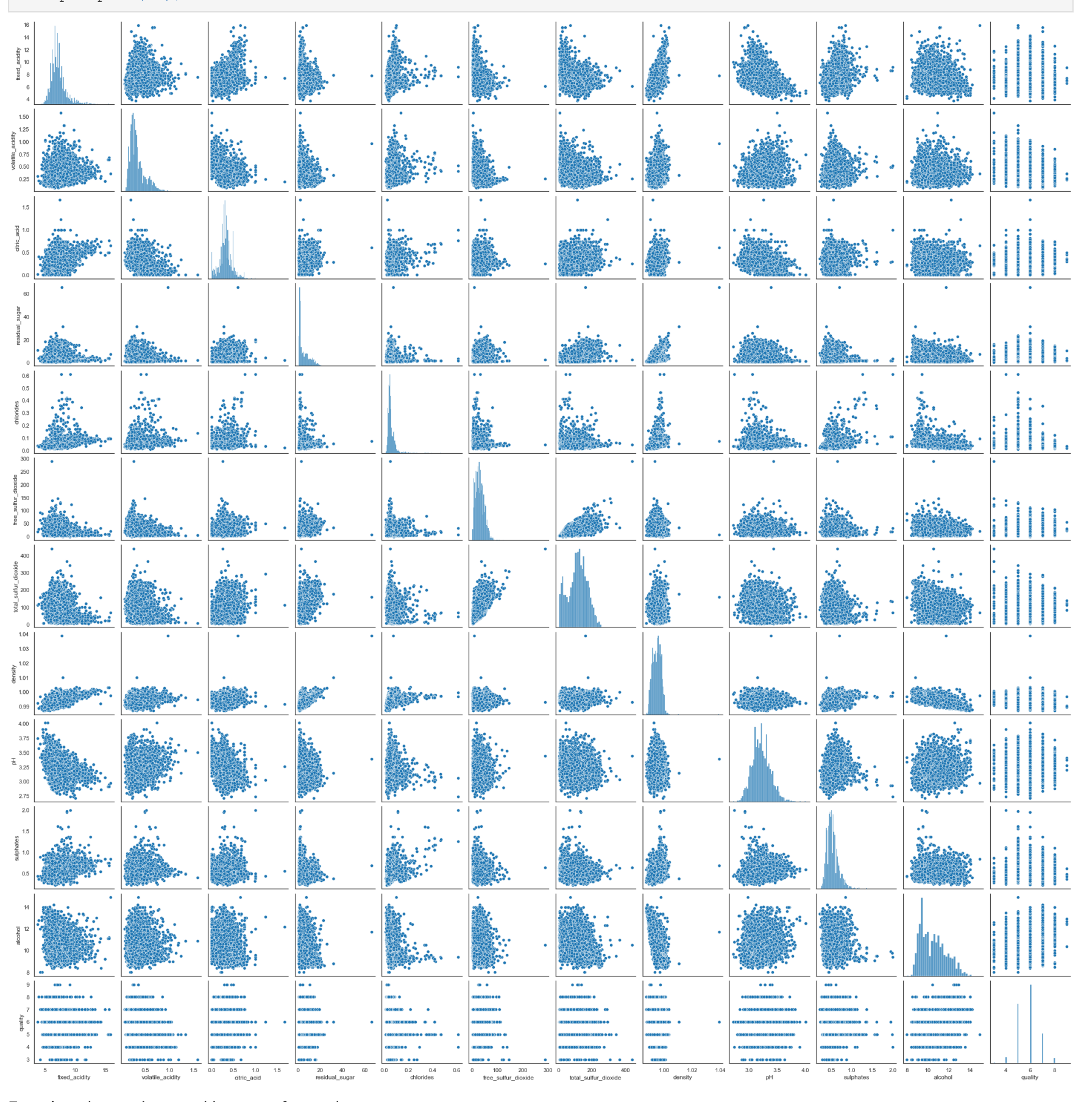
Check for Correlations

In [9]:
```
corr_mat = df.corr()

# Strip the diagonal for future examination
for x in range(corr_mat.shape[0]):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

Out[9]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	d
fixed_acidity	0.000000	0.219008	0.324436	-0.111981	0.298195	-0.282735	-0.329054	0.4
volatile_acidity	0.219008	0.000000	-0.377981	-0.196011	0.377124	-0.352557	-0.414476	0.
citric_acid	0.324436	-0.377981	0.000000	0.142451	0.038998	0.133126	0.195242	0.0
residual_sugar	-0.111981	-0.196011	0.142451	0.000000	-0.128940	0.402871	0.495482	0.
chlorides	0.298195	0.377124	0.038998	-0.128940	0.000000	-0.195045	-0.279630	0.
free_sulfur_dioxide	-0.282735	-0.352557	0.133126	0.402871	-0.195045	0.000000	0.720934	0.
total_sulfur_dioxide	-0.329054	-0.414476	0.195242	0.495482	-0.279630	0.720934	0.000000	0.0
density	0.458910	0.271296	0.096154	0.552517	0.362615	0.025717	0.032395	0.0
pH	-0.252700	0.261454	-0.329808	-0.267320	0.044708	-0.145854	-0.238413	0.
sulphates	0.299568	0.225984	0.056197	-0.185927	0.395593	-0.188457	-0.275727	0.2
alcohol	-0.095452	-0.037640	-0.010493	-0.359415	-0.256916	-0.179838	-0.265740	-0.6
quality	-0.076743	-0.265699	0.085532	-0.036980	-0.200666	0.055463	-0.041385	-0.3

In [10]:
```
sns.set_context('notebook')
sns.set_style('white')
sns.pairplot(df);
```



Examine skew values and log transform where necesary

In [11]:
```
log_columns = df.skew().sort_values(ascending=False)
log_columns = log_columns.loc[log_columns > 0.75]

log_columns
```

Out[11]:
```
chlorides              5.399828
sulphates              1.797270
fixed_acidity          1.723290
volatile_acidity       1.495097
residual_sugar         1.435404
free_sulfur_dioxide    1.220066
dtype: float64
```

In [12]:
```
# The log transformations
for col in log_columns.index:
    df[col] = np.log1p(df[col])
```

In [14]:
```
df.head()
```

Out[14]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphat
0	2.128232	0.530628	0.00	1.064711	0.073250	2.484907	34.0	0.9978	3.51	0.4446
1	2.174752	0.631272	0.00	1.280934	0.093490	3.258097	67.0	0.9968	3.20	0.5187
2	2.174752	0.565314	0.04	1.193922	0.088011	2.772589	54.0	0.9970	3.26	0.5007
3	2.501436	0.246860	0.56	1.064711	0.072321	2.890372	60.0	0.9980	3.16	0.4574
4	2.128232	0.530628	0.00	1.064711	0.073250	2.484907	34.0	0.9978	3.51	0.4446

Scale Data

In [17]:
```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()

for col in df.columns[:-1]:
    df[col] = mms.fit_transform(df[[col]]).squeeze()
```

Pipeline if needed

In [ ]:
```
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline

# The custom NumPy log transformer
log_transformer = FunctionTransformer(np.log1p)

# The pipeline
estimators = [('log1p', log_transformer), ('minmaxscale', MinMaxScaler())]
pipeline = Pipeline(estimators)

# Convert the original data
data_pipe = pipeline.fit_transform(data_orig)
```

PCA

In this section, we will:

- Perform PCA with `n_components` ranging from 1 to 5.
- Store the amount of explained variance for each number of dimensions.
- Also store the feature importance for each number of dimensions. *Hint:* PCA doesn't explicitly provide this after a model is fit, but the `components_` properties can be used to determine something that approximates importance. How you decided to do so is entirely up to you.
- Plot the explained variance and feature importances.

In [21]:
```
data = df.iloc[:, :-1]
```

In [22]:
```
from sklearn.decomposition import PCA

pca_list = list()
feature_weight_list = list()

# Fit a range of PCA models

for n in range(1, 6):

    # Create and fit the model
    PCAmod = PCA(n_components=n)
    PCAmod.fit(data)

    # Store the model and variance
    pca_list.append(pd.Series({'n':n, 'model':PCAmod,
                               'var': PCAmod.explained_variance_ratio_.sum()}))

    # Calculate and store feature importances
    abs_feature_values = np.abs(PCAmod.components_).sum(axis=0)
    feature_weight_list.append(pd.DataFrame({'n':n,
                                             'features': data.columns,
                                             'values':abs_feature_values/abs_feature_values.sum()}))

pca_df = pd.concat(pca_list, axis=1).T.set_index('n')
pca_df
```

Out[22]:

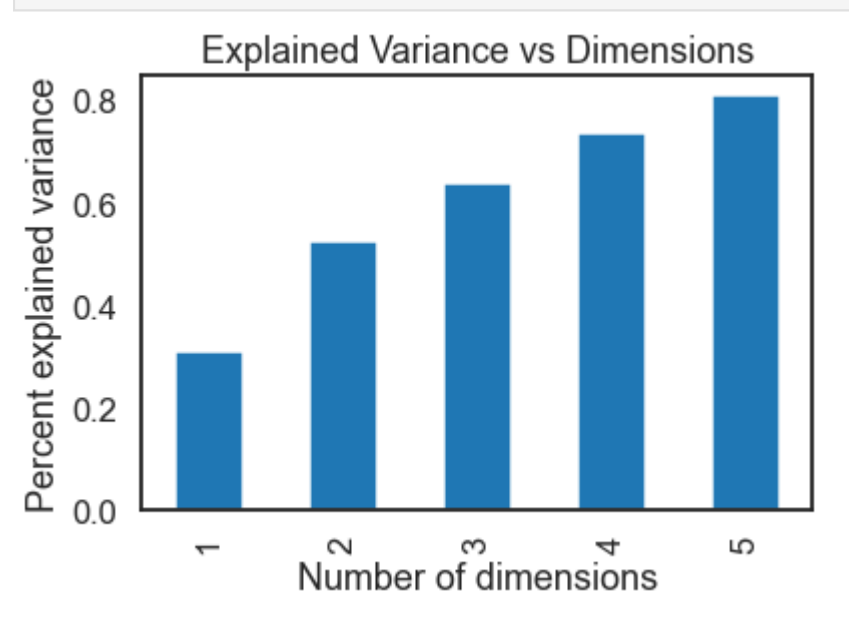
n	model	var
1	PCA(n_components=1)	0.311284
2	PCA(n_components=2)	0.525984
3	PCA(n_components=3)	0.638541
4	PCA(n_components=4)	0.737239
5	PCA(n_components=5)	0.811488

In [23]:
```
features_df = (pd.concat(feature_weight_list)
               .pivot(index='n', columns='features', values='values'))

features_df
```

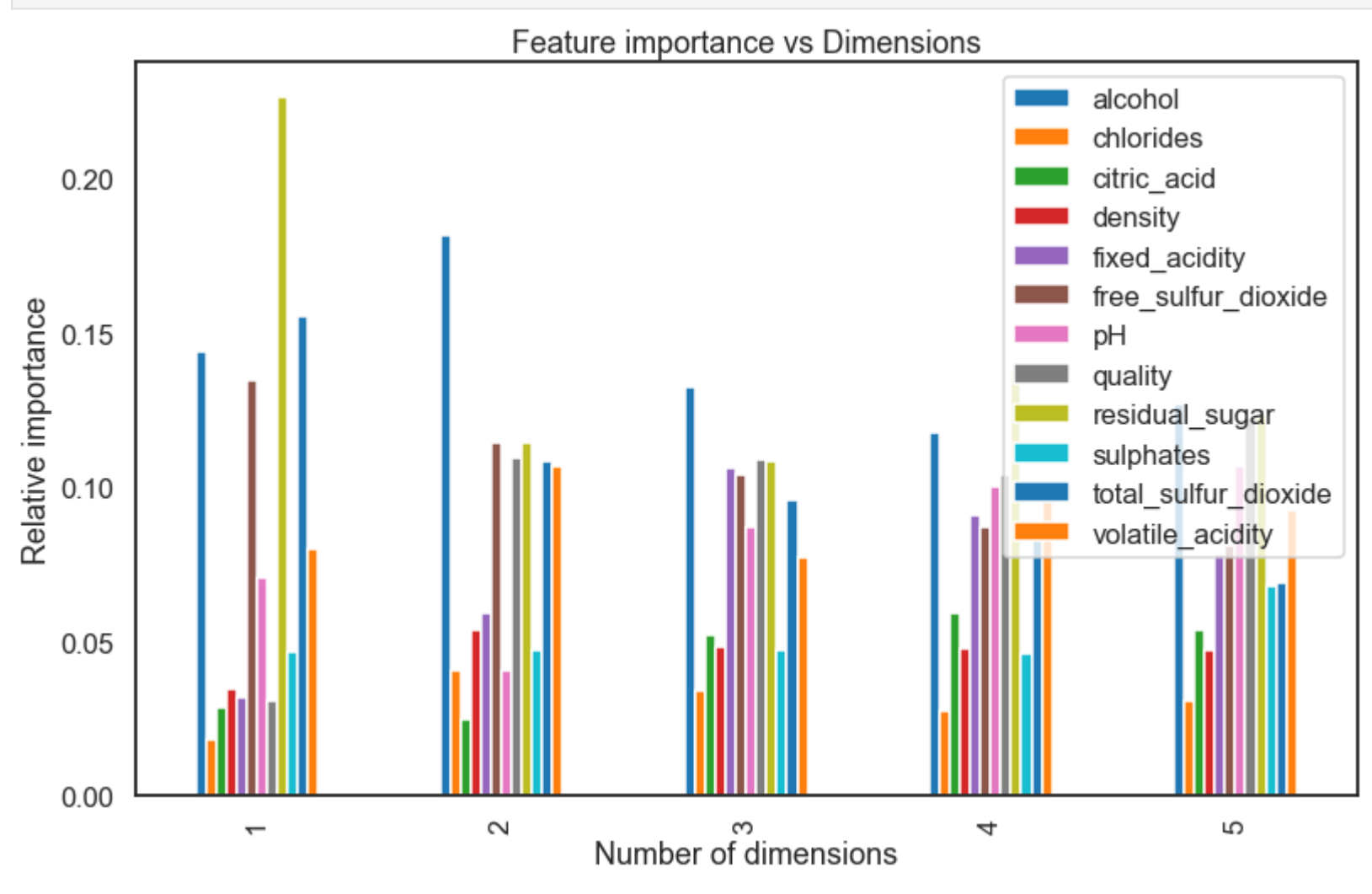
Out[23]:

features n	alcohol	chlorides	citric_acid	density	fixed_acidity	free_sulfur_dioxide	pH	quality	residual_sugar	sulphates
1	0.143785	0.018045	0.028676	0.034374	0.031602	0.134565	0.070385	0.030701	0.226326	0.046723
2	0.181576	0.040521	0.024795	0.053388	0.059247	0.114185	0.040582	0.109303	0.114359	0.047103
3	0.132363	0.034222	0.051910	0.048139	0.106143	0.103952	0.086728	0.108606	0.108319	0.047024
4	0.117626	0.027649	0.059095	0.047603	0.091051	0.086824	0.100261	0.104000	0.137302	0.045829
5	0.126735	0.030821	0.053471	0.047088	0.078425	0.080936	0.106521	0.122545	0.123838	0.068052

In [24]:
```
sns.set_context('talk')
ax = pca_df['var'].plot(kind='bar')

ax.set(xlabel='Number of dimensions',
       ylabel='Percent explained variance',
       title='Explained Variance vs Dimensions');
```



In [25]:
```
ax = features_df.plot(kind='bar', figsize=(13,8))
ax.legend(loc='upper right')
ax.set(xlabel='Number of dimensions',
       ylabel='Relative importance',
       title='Feature importance vs Dimensions');
```



Using Pipeline

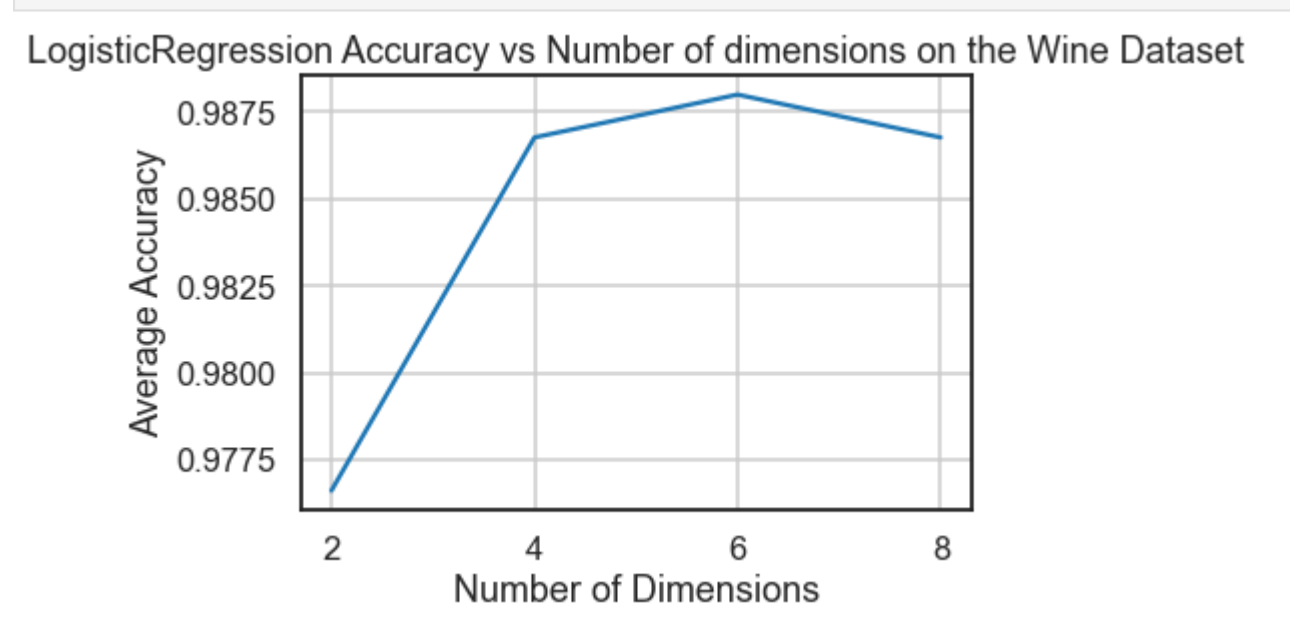
Let's explore how our model accuracy may change if we include a `PCA` in our model building pipeline. Let's plan to use sklearn's `Pipeline` class and create a pipeline that has the following steps:

1. A scaler
2. $PCA(n_compo \neq nts = n)$
3. Log$isticRegression$

- Load the Human Activity data from the datasets.
- Write a function that takes in a value of `n` and makes the above pipeline, then predicts the "Activity" column over a 5-fold StratifiedShuffleSplit, and returns the average test accuracy
- For various values of n, call the above function and store the average accuracies.
- Plot the average accuracy by number of dimensions.

In [28]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X = data
y = df.color
sss = StratifiedShuffleSplit(n_splits=5, random_state=42)

def get_avg_score(n):
    pipe = [
        ('scaler', StandardScaler()),
        ('pca', PCA(n_components=n)),
        ('estimator', LogisticRegression(solver='liblinear'))
    ]
    pipe = Pipeline(pipe)
    scores = []
    for train_index, test_index in sss.split(X, y):
        X_train, X_test = X.loc[train_index], X.loc[test_index]
        y_train, y_test = y.loc[train_index], y.loc[test_index]
        pipe.fit(X_train, y_train)
        scores.append(accuracy_score(y_test, pipe.predict(X_test)))
    return np.mean(scores)


ns = [2, 4, 6, 8]
score_list = [get_avg_score(n) for n in ns]
```

In [30]:
```
sns.set_context('talk')

ax = plt.axes()
ax.plot(ns, score_list)
ax.set(xlabel='Number of Dimensions',
       ylabel='Average Accuracy',
       title='LogisticRegression Accuracy vs Number of dimensions on the Wine Dataset')
ax.grid(True)
```



K Means Clustering

In [32]:
```
from sklearn.cluster import KMeans
```

In [38]:
```
data.values
```

Out[38]:
```
array([[0.44459904, 0.52096054, 0.        , ..., 0.27322238, 0.20289855,
        0.33333333],
       [0.48155788, 0.63653271, 0.        , ..., 0.35558641, 0.26086957,
        0.33333333],
       [0.48155788, 0.56079107, 0.02409639, ..., 0.33556054, 0.26086957,
        0.33333333],
       ...,
       [0.35456258, 0.15864244, 0.11445783, ..., 0.19959244, 0.20289855,
        0.5       ],
       [0.24087297, 0.20403697, 0.18072289, ..., 0.13696146, 0.69565217,
        0.66666667],
       [0.29974968, 0.13051862, 0.22891566, ..., 0.08755752, 0.55072464,
        0.5       ]])
```

In [41]:
```
inertias = []
for i in range(1,10):
    model = KMeans(n_clusters = i, init='random')
    model.fit(data.values)
    inertia = model.inertia_
    inertias.append(inertia)
```

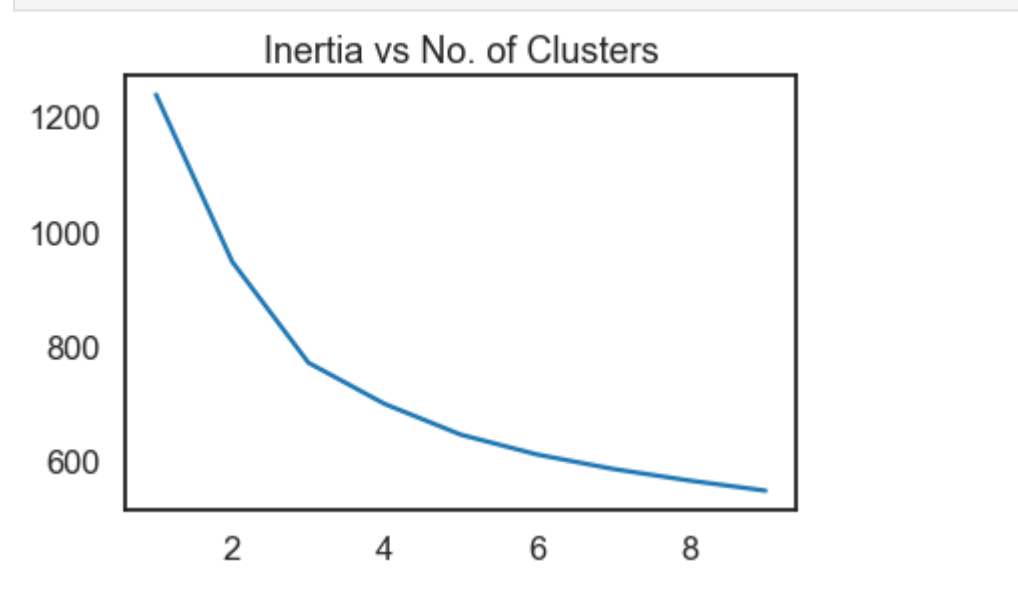
In [43]:
```
inertias
```

Out[43]:
```
[1238.6133387004722,
 947.8241390672694,
 772.0587185966506,
 700.3068406151962,
 646.7645461866077,
 612.2339888444768,
 587.0412769979399,
 566.8767197217977,
 549.2670168815022]
```

In [47]:
```
list(range(1,10))
```

Out[47]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [48]:
```
plt.plot(list(range(1,10)), inertias)
plt.title('Inertia vs No. of Clusters')
plt.show()
```



DSBCAN

In [49]:
```
from sklearn.cluster import DBSCAN
```

In [50]:
```
db = DBSCAN(eps = 3, min_samples = 3)
```

In [51]:
```
db.fit(data.values)
```

Out[51]:
```
DBSCAN(eps=3, min_samples=3)
```

In [53]:
```
db.labels_
```

Out[53]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [ ]: