In [1]:

```python
import pandas as pd

data = pd.read_excel('SoftwareData.xlsx', index_col = 'ProjectID')
```

In [2]:

```python
data.head()
```

Out[2]:

	Subject	Platforms	ManagerID	teamMembers	plannedWork	plannedDuration	com
ProjectID							
1	Supply Chain procurement process. Enable drop ...	SAP	1	12	50	8	
2	Supply Chain procurement process. Build and Sh...	SAP	1	8	40	10	
3	Supply Chain sales processing. Automated stock...	SAP	1	10	30	4	
4	Supply Chain sales processing. Contract based ...	SAP	1	6	45	6	
5	Supply Chain ARIBA integration for buyers	SAP, ARIBA	1, 2	4	30	5	

In [3]:

```python
data["Platforms"] = data["Platforms"].str.lower().str.replace(" ","")
data["ManagerID"] = data["ManagerID"].apply(lambda x: str(x).replace(" ", ""))
platforms = data["Platforms"].str.split(",", expand = True)
managers = data["ManagerID"].str.split(",", expand = True)
```

In [4]:

```python
s = pd.Series()
for column in platforms.columns:
    s = s.append(platforms[column], ignore_index = True)
s.dropna(inplace = True)
platforms = s.unique()
s = pd.Series()
for column in managers.columns:
    s = s.append(managers[column], ignore_index = True)
s.dropna(inplace = True)
managers = s.unique()
```

In [5]:

```
import numpy as np
data_new = data.copy()
for i, v in enumerate(platforms):
    data_new.insert(data_new.shape[1], "Platform_" + v, value = np.zeros(data.shape[0], dtype = np.int8))
for i, v in enumerate(managers):
    data_new.insert(data_new.shape[1], "Manager_" + v, value = np.zeros(data.shape[0], dtype = np.int8))
```

In [6]:

```
data = pd.DataFrame(columns = data_new.columns)
for i, row in data_new.iterrows():
    for v in row["Platforms"].split(","):
        row["Platform_" + v] = 1
    for v in str(row["ManagerID"]).split(","):
        row["Manager_" + v] = 1
    data = data.append(row)
data["plannedSpeed"] = data["plannedWork"] / data["plannedDuration"]
data["remainingSpeed"] = data["remainingWork"] / data["remainingDuration"]
```

In [7]:

```
from sklearn import preprocessing
names = ["teamMembers", "plannedWork", "plannedDuration", "remainingWork", "remainingDuration", "plannedSpeed", "remainingSpeed"]
scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(data[names])
data[names] = scaled_df
data[["percentLevel1", "percentLevel2", "percentLevel3"]] = data[["percentLevel1", "percentLevel2", "percentLevel3"]] / 100
data.head(2)
```

Out[7]:

	Subject	Platforms	ManagerID	teamMembers	plannedWork	plannedDuration	complexity
1	Supply Chain procurement process. Enable drop ...	sap	1	0.650945	-0.423809	0.203553	5
2	Supply Chain procurement process. Build and Sh...	sap	1	-0.390567	-0.834775	0.969869	5

2 rows × 27 columns

In [8]:

```
dummies = pd.get_dummies(data["complexity"], prefix = "complexity")
data = pd.concat([data, dummies], axis = 1)
```

In [9]:

```
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()
def clean(doc):
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    return normalized
doc_complete = data["Subject"]
doc_clean = [clean(doc).split() for doc in doc_complete]
```

In [10]:

```
import gensim
from gensim import corpora

dictionary = corpora.Dictionary(doc_clean)
doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
Lda = gensim.models.ldamodel.LdaModel
ldamodel = Lda(doc_term_matrix, num_topics=5, id2word = dictionary, passes=50)
```

In [11]:

```
print(ldamodel.print_topics(num_topics=5, num_words=3))
```

```
[(0, '0.071*"integration" + 0.070*"supply" + 0.070*"chain"'), (1, '0.084*"man
agement" + 0.084*"software" + 0.058*"using"'), (2, '0.091*"supply" + 0.091*"c
hain" + 0.091*"pricing"'), (3, '0.108*"chain" + 0.108*"supply" + 0.108*"proce
ssing"'), (4, '0.079*"employee" + 0.079*"hr" + 0.079*"u"')]
```

In [11]:

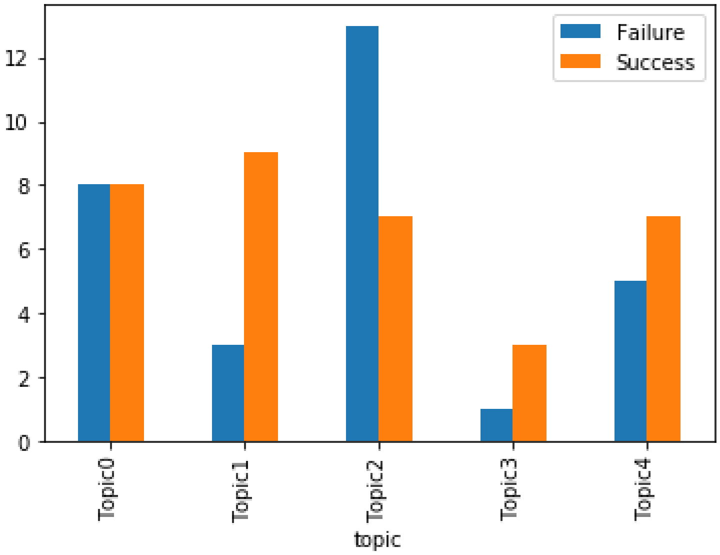
```
topic_distribution = pd.DataFrame(columns = ["Topic0", "Topic1", "Topic2", "To
pic3", "Topic4"])
for text in doc_clean:
    doc_bow = dictionary.doc2bow(text)
    topics = sorted(ldamodel[doc_bow],key=lambda x:x[0],reverse=True)
    row = {}
    for topic in topics:
        row["Topic" + str(topic[0])] = topic[1]
    topic_distribution = topic_distribution.append(row, ignore_index = True)
```

In [12]:

```
topic_distribution.index = data.index
```

In [14]:
```
import matplotlib.pyplot as plt
%matplotlib inline

topics = topic_distribution.idxmax(axis = 1)
df = pd.concat([topics.rename('topic'), data["Success/Failure"]], axis = 1)
df = pd.concat([df, pd.get_dummies(data["Success/Failure"], prefix = 'Outcome'
)], axis = 1)
df = df.drop(["Success/Failure"], axis = 1)
df = pd.DataFrame(df.groupby(['topic'], as_index=False).sum())
df.columns = ['topic', 'Failure', 'Success']
df.plot.bar(x='topic', y=['Failure', 'Success'])
plt.show()
```



In [15]:
```
data = pd.concat([data, topic_distribution], axis = 1)
data_new = data.copy()
target = data_new["Success/Failure"].astype('int32')
data_new = data_new.drop(["Subject", "Platforms","ManagerID", "complexity", "S
uccess/Failure"], axis = 1)
data_new.head(2)
```

Out[15]:

	teamMembers	plannedWork	plannedDuration	remainingWork	remainingDuration	percentLeve
1	0.650945	-0.423809	0.203553	-0.801176	1.00597	0
2	-0.390567	-0.834775	0.969869	-1.18767	-0.232147	0

2 rows × 32 columns

In [18]:

```
from time import time
from sklearn.metrics import f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

def train_classifier(clf, X_train, y_train):
    start = time()
    clf.fit(X_train, y_train)
    end = time()
    print("Trained model in {:.4f} seconds".format(end - start))


def predict_labels(clf, features, target):
    start = time()
    y_pred = clf.predict(features)
    end = time()

    print("Made predictions in {:.4f} seconds.".format(end - start))
    return f1_score(target.values, y_pred)


def train_predict(clf, X_train, y_train, X_test, y_test):
    print("Training a {} using a training set size of {}. . .".format(clf.__class__.__name__, len(X_train)))

    train_classifier(clf, X_train, y_train)
    print("F1 score for training set: {:.4f}.".format(predict_labels(clf, X_train, y_train)))
    print("F1 score for test set: {:.4f}.".format(predict_labels(clf, X_test, y_test)))

clf_A = DecisionTreeClassifier(random_state = 1)
clf_B = GaussianNB()
clf_C = KNeighborsClassifier(n_neighbors=5)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( data_new, target, test_size=0.25, random_state = 1)

for clf in [clf_A, clf_B, clf_C]:
    train_predict(clf, X_train, y_train, X_test, y_test)
```

```
Training a DecisionTreeClassifier using a training set size of 48. . .
Trained model in 0.0080 seconds
Made predictions in 0.0040 seconds.
F1 score for training set: 1.0000.
Made predictions in 0.0040 seconds.
F1 score for test set: 0.7059.
Training a GaussianNB using a training set size of 48. . .
Trained model in 0.0000 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 0.7463.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7826.
Training a KNeighborsClassifier using a training set size of 48. . .
Trained model in 0.1668 seconds
Made predictions in 0.0610 seconds.
F1 score for training set: 0.7143.
Made predictions in 0.0040 seconds.
F1 score for test set: 0.7000.
```

In [19]:

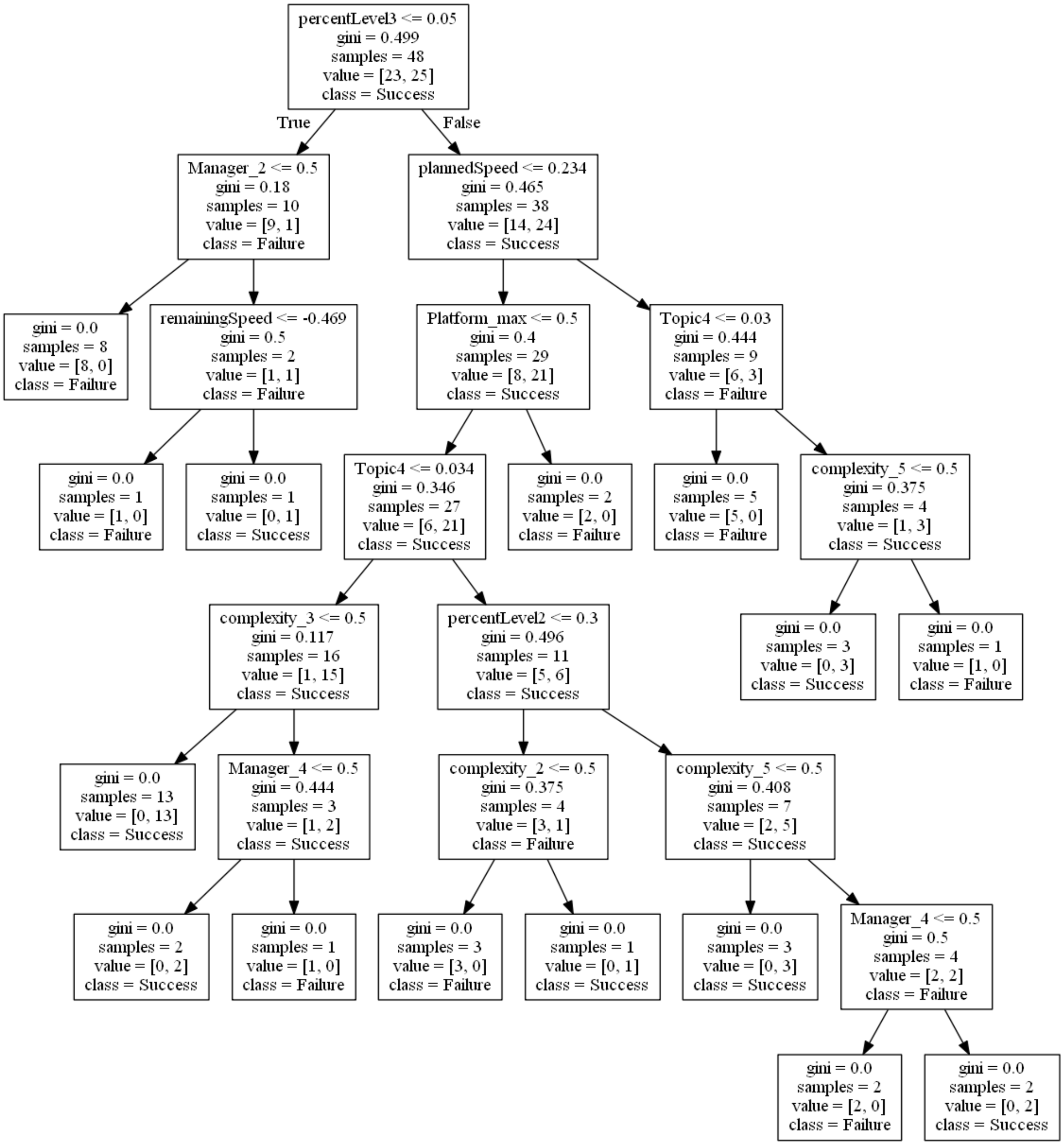
```
import pydotplus

from sklearn import tree
from IPython.display import Image

dot_data = tree.export_graphviz(clf_A, out_file=None,
                                feature_names=data_new.columns,
                                class_names=["Failure", "Success"])


# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Out[19]:



For a project, if failure is predicted using the decision tree classifier then navigating through the tree generated we can observe the features if we improve will reduce the probability of Failure. Thus Decision Tree can help in providing key metrics to improve upon software project failure prediction

In [50]:

```
distances, indices = clf_C.kneighbors(X_train.iloc[8:9])
for d, i in list(zip(*distances, *indices)):
    if data.iloc[i]['Success/Failure'] == 1:
        break
data.iloc[[i,8]]
```

Out[50]:

	Subject	Platforms	ManagerID	teamMembers	plannedWork	plannedDuration	com
13	Software Management automated ticket resolution	jira,azure,docker	3,4	0.911322	1.42554	1.73618	
9	Supply Chain procurement process telemetry to ...	azure,sap	3	-0.390567	-0.0128427	1.73618	

2 rows × 37 columns

By using KNN classifier if we predict a failure then we can look for closest neighbour which has a success and try to improve current project according to the metrics of closest successful project