# IBM Data Science Capstone:

## Introduction:

I will be analyzing traffic accidents and the various circumstances that contribute to them. I will look at the severity and the conditions that cause them. I will use data analysis and machine learning techniques to try to predict the severity of future accidents.

## Data:

I will be using a data set provided by IBM that includes extensive data on traffic accidents.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df= pd.read_csv('../input/capstone-car-accident-serveity/Data_Collisions.csv')
df.head()
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3063: DtypeWarning: Columns (33) have mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[54]:

| | SEVERITYCODE | X | Y | OBJECTID | INCKEY | COLDETKEY | REPORTNO | STATUS | ADDRTYPE | INTKEY | ... | ROADCOND | LIGHTCOND | PEDROWNOTGRNT | SDC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | -122.323148 | 47.703140 | 1 | 1307 | 1307 | 3502005 | Matched | Intersection | 37475.0 | ... | Wet | Daylight | NaN | |
| 1 | 1 | -122.347294 | 47.647172 | 2 | 52200 | 52200 | 2607959 | Matched | Block | NaN | ... | Wet | Dark - Street Lights On | NaN | |
| 2 | 1 | -122.334540 | 47.607871 | 3 | 26700 | 26700 | 1482393 | Matched | Block | NaN | ... | Dry | Daylight | NaN | |
| 3 | 1 | -122.334803 | 47.604803 | 4 | 1144 | 1144 | 3503937 | Matched | Block | NaN | ... | Dry | Daylight | NaN | |
| 4 | 2 | -122.306426 | 47.545739 | 5 | 17700 | 17700 | 1807429 | Matched | Intersection | 34387.0 | ... | Wet | Daylight | NaN | |

5 rows × 38 columns

```
#Pull in needed columns
df= df[['SEVERITYCODE','ADDRTYPE','COLLISIONTYPE','PERSONCOUNT','PEDCOUNT','VEHCOUNT','INCDATE','JUNCTIONTYPE','WEATHER'
       ,'ROADCOND','LIGHTCOND','SPEEDING','INATTENTIONIND','UNDERINFL']]
```

```
df.select_dtypes(exclude=['int','float']).columns
```

Out[57]:

```
Index(['ADDRTYPE', 'COLLISIONTYPE', 'INCDATE', 'JUNCTIONTYPE', 'WEATHER',
       'ROADCOND', 'LIGHTCOND', 'SPEEDING', 'INATTENTIONIND', 'UNDERINFL'],
      dtype='object')
```

```
#Find null values
df.isnull().sum()
```

Out[61]:

```
SEVERITYCODE          0
ADDRTYPE           1926
COLLISIONTYPE      4904
PERSONCOUNT           0
PEDCOUNT              0
VEHCOUNT              0
INCDATE               0
JUNCTIONTYPE       6329
WEATHER            5081
ROADCOND           5012
LIGHTCOND          5170
SPEEDING         185340
INATTENTIONIND   164868
UNDERINFL          4884
dtype: int64
```

[58]:

```
df.select_dtypes(exclude=['object']).columns
```

Out[58]:

```
Index(['SEVERITYCODE', 'PERSONCOUNT', 'PEDCOUNT', 'VEHCOUNT'], dtype='object')
```

[60]:

```
#View filtered dataset
print('Rows     :',df.shape[0])
print('Columns  :',df.shape[1])
print('\nFeatures :\n     :',df.columns.tolist())
print('\nMissing values    :',df.isnull().values.sum())
print('\nUnique values :  \n',df.nunique())
```

```
Rows     : 194673
Columns  : 14

Features :
     : ['SEVERITYCODE', 'ADDRTYPE', 'COLLISIONTYPE', 'PERSONCOUNT', 'PEDCOUNT', 'VEHCOUNT', 'INCDATE', 'JUNCTIONTYPE', 'WEATHER', 'ROADCOND',
'LIGHTCOND', 'SPEEDING', 'INATTENTIONIND', 'UNDERINFL']

Missing values    : 383514

Unique values :
 SEVERITYCODE         2
ADDRTYPE              3
COLLISIONTYPE        10
PERSONCOUNT          47
PEDCOUNT              7
VEHCOUNT             13
INCDATE            5985
JUNCTIONTYPE          7
WEATHER              11
ROADCOND              9
LIGHTCOND             9
SPEEDING              1
INATTENTIONIND        1
UNDERINFL             4
dtype: int64
```

```
#Analyze meaning of nulls
print('Unique Values of SPEEDING: ',df.SPEEDING.unique(),'\n\n')
print('Unique Values of UNDERINFL: ',df.UNDERINFL.unique(),'\n\n')
print('Unique Values of INATTENTIONIND: ',df.INATTENTIONIND.unique())
```

```
Unique Values of SPEEDING:  [nan 'Y']


Unique Values of UNDERINFL:  ['N' '0' nan '1' 'Y']


Unique Values of INATTENTIONIND:  [nan 'Y']
```

+ Code + Markdown

```
#Change nulls and no to 0 and yes to 1
df.SPEEDING.fillna(value=0,axis=0,inplace=True)
df.SPEEDING.replace(to_replace='Y',value=1,inplace=True)

df.INATTENTIONIND.fillna(value=0,axis=0,inplace=True)
df.INATTENTIONIND.replace(to_replace='Y',value=1,inplace=True)

df.UNDERINFL.replace(to_replace=('Y','N','1','0'),value=(1,0,1,0),inplace=True)

print('SPEEDING unique values: ',df.SPEEDING.unique(),'\n\n')
print('INATTENTIONIND unique values: ',df.INATTENTIONIND.unique(),'\n\n')
print('UNDERINFL unique values:',df.UNDERINFL.unique())
```

```
SPEEDING unique values:  [0 1]


INATTENTIONIND unique values:  [0 1]


UNDERINFL unique values: [ 0. nan  1.]
```

```
#Drop null rows
df.dropna(axis=0,inplace=True)
print('Any null values?','\n', df.isnull().any(),'\n\n')
print('Rows:', df.shape[0])
print('Columns:',df.shape[1])
```

```
Any null values?
 SEVERITYCODE      False
ADDRTYPE           False
COLLISIONTYPE      False
PERSONCOUNT        False
PEDCOUNT           False
VEHCOUNT           False
INCDATE            False
JUNCTIONTYPE       False
WEATHER            False
ROADCOND           False
LIGHTCOND          False
SPEEDING           False
INATTENTIONIND     False
UNDERINFL          False
dtype: bool


Rows: 182895
Columns: 14
```

```
#Format dates
df['INCDATE']=pd.to_datetime(df['INCDATE'],format='%Y-%m-%d %H:%M:%S')
df['YEAR']=df['INCDATE'].dt.year
df['MONTH']=df['INCDATE'].dt.month
df['DAY']=df['INCDATE'].dt.weekday

df.drop(labels='INCDATE',axis=1,inplace=True)
df.drop(labels='JUNCTIONTYPE',axis=1,inplace=True)

df.head()
```

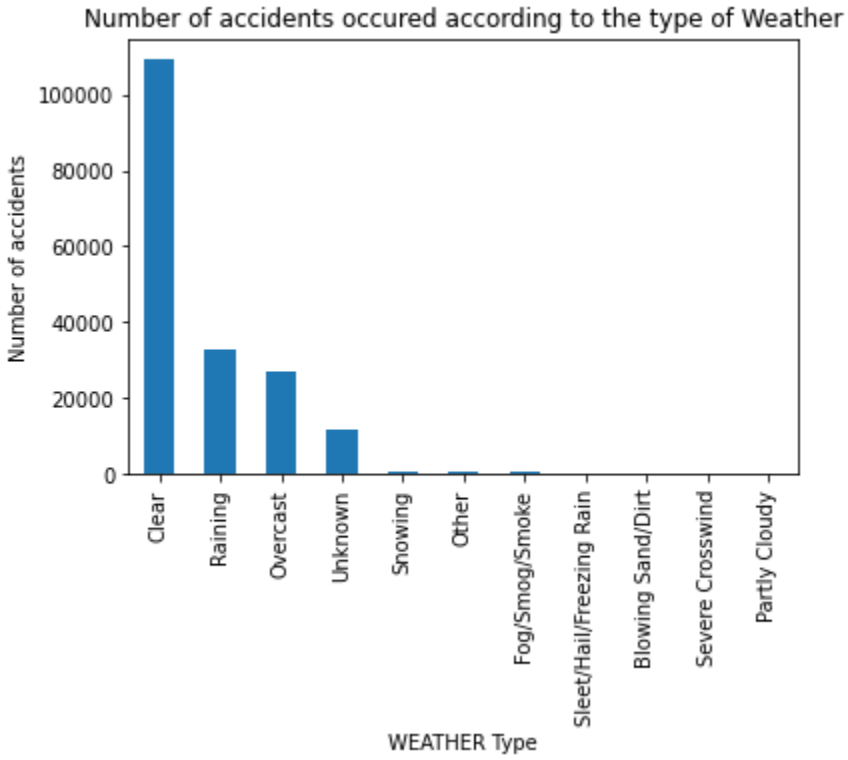
ut[67]:

| | SEVERITYCODE | ADDRTYPE | COLLISIONTYPE | PERSONCOUNT | PEDCOUNT | VEHCOUNT | WEATHER | ROADCOND | LIGHTCOND | SPEEDING | INATTENTIONIND | UNDERINFL | YE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | Intersection | Angles | 2 | 0 | 2 | Overcast | Wet | Daylight | 0 | 0 | 0.0 | 20 |
| 1 | 1 | Block | Sideswipe | 2 | 0 | 2 | Raining | Wet | Dark - Street Lights On | 0 | 0 | 0.0 | 20 |
| 2 | 1 | Block | Parked Car | 4 | 0 | 3 | Overcast | Dry | Daylight | 0 | 0 | 0.0 | 20 |
| 3 | 1 | Block | Other | 3 | 0 | 3 | Clear | Dry | Daylight | 0 | 0 | 0.0 | 20 |
| 4 | 2 | Intersection | Angles | 2 | 0 | 2 | Raining | Wet | Daylight | 0 | 0 | 0.0 | 20 |

## Methodology and Analysis:

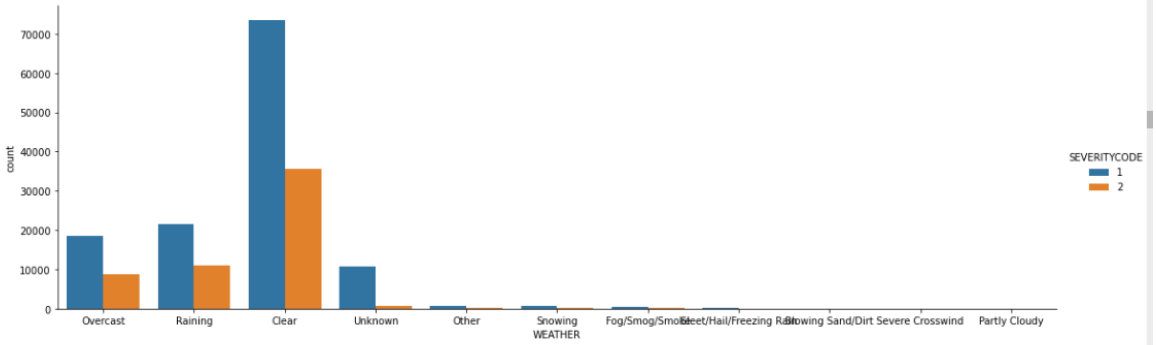
In this section I will look at different factors surrounding traffic accidents and look for patterns.

```
%matplotlib inline

df['WEATHER'].value_counts().plot.bar()

plt.title('Number of accidents occured according to the type of Weather')
plt.xlabel('WEATHER Type')
plt.ylabel('Number of accidents')

plt.show()
```



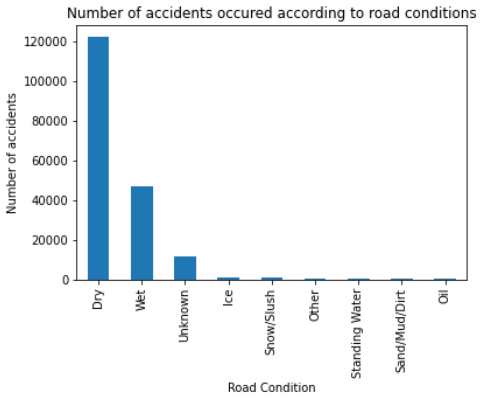

```
[69]: sns.catplot(x="WEATHER", hue="SEVERITYCODE", kind="count", data=df,height=5,aspect=3)
```

```
Out[69]: <seaborn.axisgrid.FacetGrid at 0x7f7c1e0e1150>
```



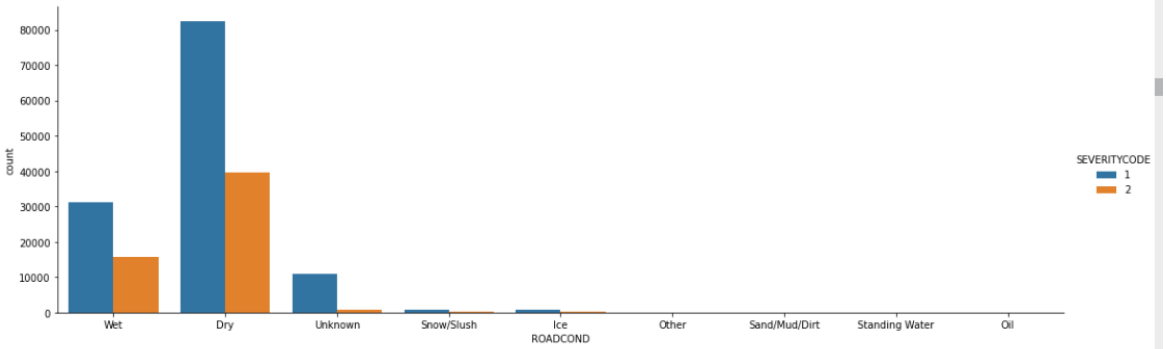

In looking at these two charts, it appears that while it can be treacherous driving in poor weather, it does not seem to cause more severe accidents.

```
%matplotlib inline

df['ROADCOND'].value_counts().plot.bar()

plt.title('Number of accidents occured according to road conditions')
plt.xlabel('Road Condition')
plt.ylabel('Number of accidents')

plt.show()
```



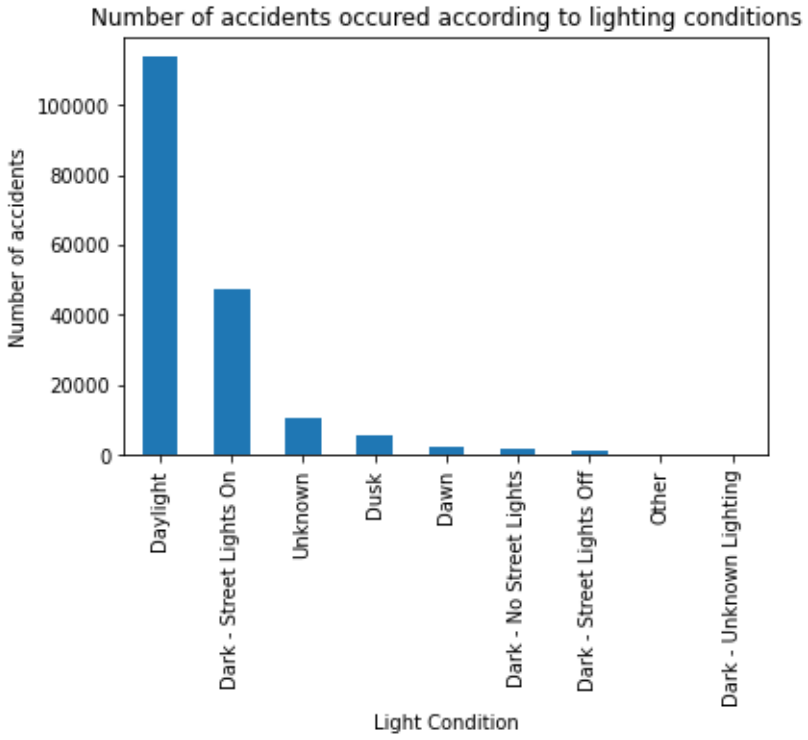

[71]:
```
sns.catplot(x="ROADCOND", hue="SEVERITYCODE", kind="count", data=df,height=5,aspect=3)
```

Out[71]
```
<seaborn.axisgrid.FacetGrid at 0x7f7c1daa6bd0>
```



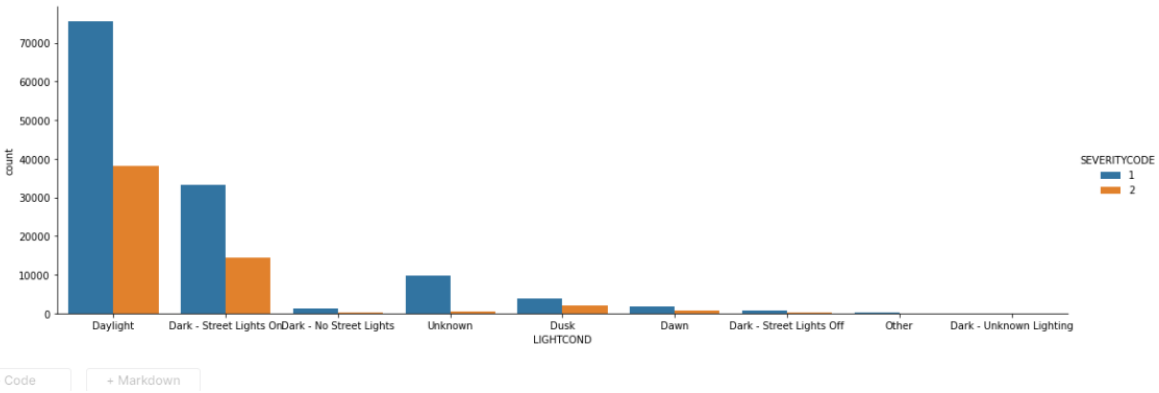

The two charts above show a pattern similar when it comes to road conditions. While it may be more challenging to drive on rougher road conditions, it does not appear to lead to an increase in accident severity.

[72]:
```
%matplotlib inline

df['LIGHTCOND'].value_counts().plot.bar()

plt.title('Number of accidents occured according to lighting conditions')
plt.xlabel('Light Condition')
plt.ylabel('Number of accidents')

plt.show()
```




```
[21]: sns.catplot(x="LIGHTCOND", hue="SEVERITYCODE", kind="count", data=df,height=5,aspect=3)
```

```
Out[21] <seaborn.axisgrid.FacetGrid at 0x7f7c2e545150>
```

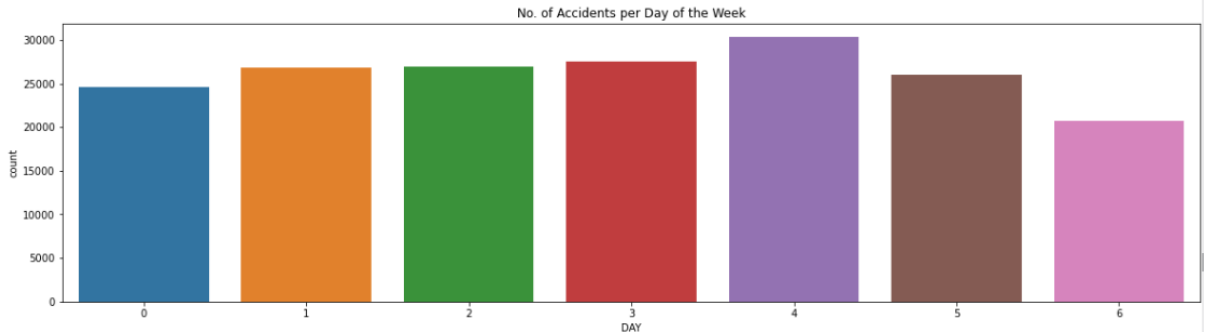



+ Code + Markdown

These two charts show that most severe accidents occur duing daylight hours when it is least challenging to drive.

Taken together, these charts seem to show that most accidents occur when the conditions are seemingly the best for safe driving. It could be theorized that accidents are therefore most likely to occur when the traffic volume is the highest.

```
plt.figure(figsize=(20,5))
sns.countplot(x='DAY',data=df)
plt.title('No. of Accidents per Day of the Week')
plt.show()
```




Looking at this chart, it does appear that the traditional Monday-Friday work week does show a higher volume of crashes.

## Results:

In analyzing the results, it appears that with my initial analysis I cannot conclude that difficult weather, road, or lighting conditions lead to an increase in the number and severity of traffic accidents. It is possible that the increase of accidents during less treacherous driving conditions could be due to the higher volume of cars on the road during this time, but more data would be required to make this conclusion.

## Discussion:

In looking at the data, I employed a number of machine learning techniques to see which would be most helpful in examining the data.

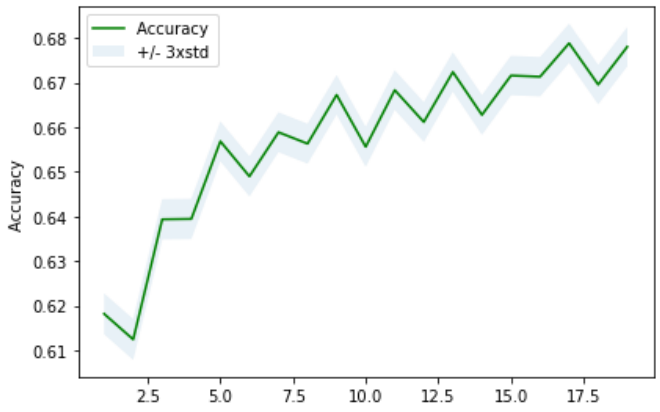
```
#K Nearest Neighbor
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors = 20).fit(X_train,y_train)
KNN_yhat = neigh.predict(X_test)

KNN_acc = round(metrics.accuracy_score(y_test, KNN_yhat),2)
f3= round(f1_score(y_test,KNN_yhat),2)

print('KNN accuracy: ', KNN_acc)
print('KNN f1 score: ',f3)
```

```
KNN accuracy:  0.67
KNN f1 score:  0.66
```

```
]:
from sklearn.neighbors import KNeighborsClassifier
Ks = 20
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test,yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

```
Out[97]
array([0.61818985, 0.61245033, 0.6393819 , 0.6394702 , 0.65686534,
       0.64900662, 0.65889625, 0.65633554, 0.66728477, 0.65562914,
       0.66834437, 0.66119205, 0.67240618, 0.66278146, 0.67161148,
       0.67134658, 0.6788521 , 0.66958057, 0.6780574 ])
```

```
plt.plot(range(1,20),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```




```
#SVM
from sklearn import svm
clf = svm.SVC(kernel='rbf')

clf.fit(X_train, y_train)

SVM_yhat = clf.predict(X_test)

SVM_acc = metrics.accuracy_score(y_test,SVM_yhat)
f4= f1_score(y_test,SVM_yhat)

print('SVM accuracy: ', SVM_acc)
print('SVM f1 score: ',f4)
```

```
SVM accuracy:  0.4879470198675497
SVM f1 score:  0.6558661207050026
```

```
#Jaccard accuracy scores
from sklearn.metrics import jaccard_score
```

[101]:
```
# Logistic Regression
jss1 = round(jaccard_score(y_test, LR_yhat), 2)

# Decision Tree
jss2 = round(jaccard_score(y_test, Tree_yhat), 2)

# KNN
jss3 = round(jaccard_score(y_test, KNN_yhat), 2)

# Support Vector Machine
jss4 = round(jaccard_score(y_test, SVM_yhat), 2)

jss_list = [jss1, jss2, jss3, jss4]
jss_list
```

Out[101]:
```
[0.47, 0.48, 0.5, 0.49]
```

```
f1_list= [f1,f2,f3,round(f4,2)]
acc_list= [LR_acc,Tree_acc,KNN_acc,round(SVM_acc,2)]

columns= ['Logistic Regression','Decision Tree','KNN','SVM']
index= ['Jaccard Score','Model Accuracy','F1 Score']

accuracy_df= pd.DataFrame([jss_list,acc_list,f1_list],index= index,columns=columns)
accuracy_df.head()
```

t[102]:

| | Logistic Regression | Decision Tree | KNN | SVM |
|---|---|---|---|---|
| Jaccard Score | 0.47 | 0.48 | 0.50 | 0.49 |
| Model Accuracy | 0.69 | 0.68 | 0.67 | 0.49 |
| F1 Score | 0.64 | 0.65 | 0.66 | 0.66 |

```
accuracy_df1= accuracy_df.transpose()
accuracy_df1.columns.name= 'Algorithm'
accuracy_df1
```

ut[103]:

| Algorithm | Jaccard Score | Model Accuracy | F1 Score |
|---|---|---|---|
| Logistic Regression | 0.47 | 0.69 | 0.64 |
| Decision Tree | 0.48 | 0.68 | 0.65 |
| KNN | 0.50 | 0.67 | 0.66 |
| SVM | 0.49 | 0.49 | 0.66 |

According to these findings, logistic regression appears to be the best option for further research.