Car accident severity

Capstone Project

“Mohamed Basher”

2. Data

2.1 Data Source

The Original Data Came from:

Data-Collisions.csv

3. Feature

**** Create Data Dictionary ****

1. LOCATION : Description of the general location of the collision
2. SEVERITYCODE : A code that corresponds to the severity of the collision:

   * 3 — fatality

   * 2b —serious injury

   * 2 — injury

   * 1 — prop damage

   * 0 — unknown

3. SEVERITYDESC : A detailed description of the severity of the collision
4. COLLISIONTYPE : Collision type
5. PERSONCOUNT : The total number of people involved in the collision
6. PEDCOUNT : The number of pedestrians involved in the collision. This is entered by the state.
7. PEDCYLCOUNT : The number of bicycles involved in the collision. This is entered by the state.
8. VEHCOUNT : The number of vehicles involved in the collision. This is entered by the state.

9. INJURIES : The number of total injuries in the collision. This is entered by the state.
10. SERIOUSINJURIES : The number of serious injuries in the collision. This is entered by the state.
11. FATALITIES : The number of fatalities in the collision. This is entered by the state.
12. INCDATE : The date of the incident.
13. INCDTTM : The date and time of the incident.
14. JUNCTIONTYPE : Category of junction at which collision took place
15. SDOT_COLCODE : A code given to the collision by SDOT.
16. SDOT_COLDESC : A description of the collision corresponding to the collision code.
17. INATTENTIONIND : Whether or not collision was due to inattention. (Y/N)
18. UNDERINFL : Whether or not a driver involved was under the influence of drugs or alcohol.
19. WEATHER : A description of the weather conditions during the time of the collision.
20. ROADCOND : The condition of the road during the collision.
21. LIGHTCOND : The light conditions during the collision.
22. PEDROWNOTGRNT : Whether or not the pedestrian right of way was not granted. (Y/N)
23. SDOTCOLNUM : A number given to the collision by SDOT.
24. SPEEDING : Whether or not speeding was a factor in the collision. (Y/N)
25. ST_COLCODE : A code provided by the state that describes the collision.
26. ST_COLDESC : A description that corresponds to the state's coding designation.
27. SEGLANEKEY : A key for the lane segment in which the collision occurred.
28. CROSSWALKKEY : A key for the crosswalk at which the collision occurred.
29. HITPARKEDCAR : Whether or not the collision involved hitting a parked car. (Y/N)

4. Methodology

4.1 Data Exploration

We look now for the target values (**SEVERITYCODE**) and do some analysis

```
df.SEVERITYCODE.value_counts()
```

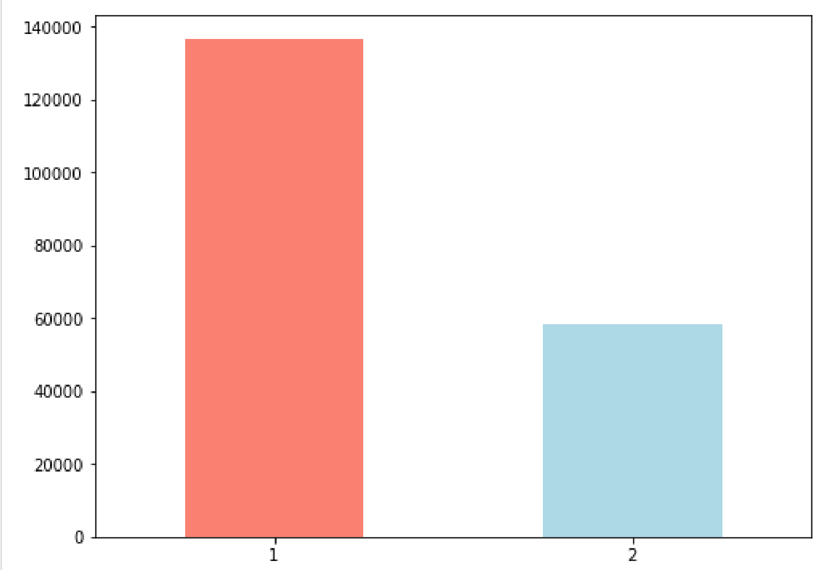
```
1    136485
2     58188
Name: SEVERITYCODE, dtype: int64
```

- We see that most car accident severity :

```
1 – prop damage
2 - injury
```

- Make some visualization

```python
df.SEVERITYCODE.value_counts().plot(kind='bar',figsize=(8,6),color=["salmon", "lightblue"])
plt.xticks(rotation=0);
```

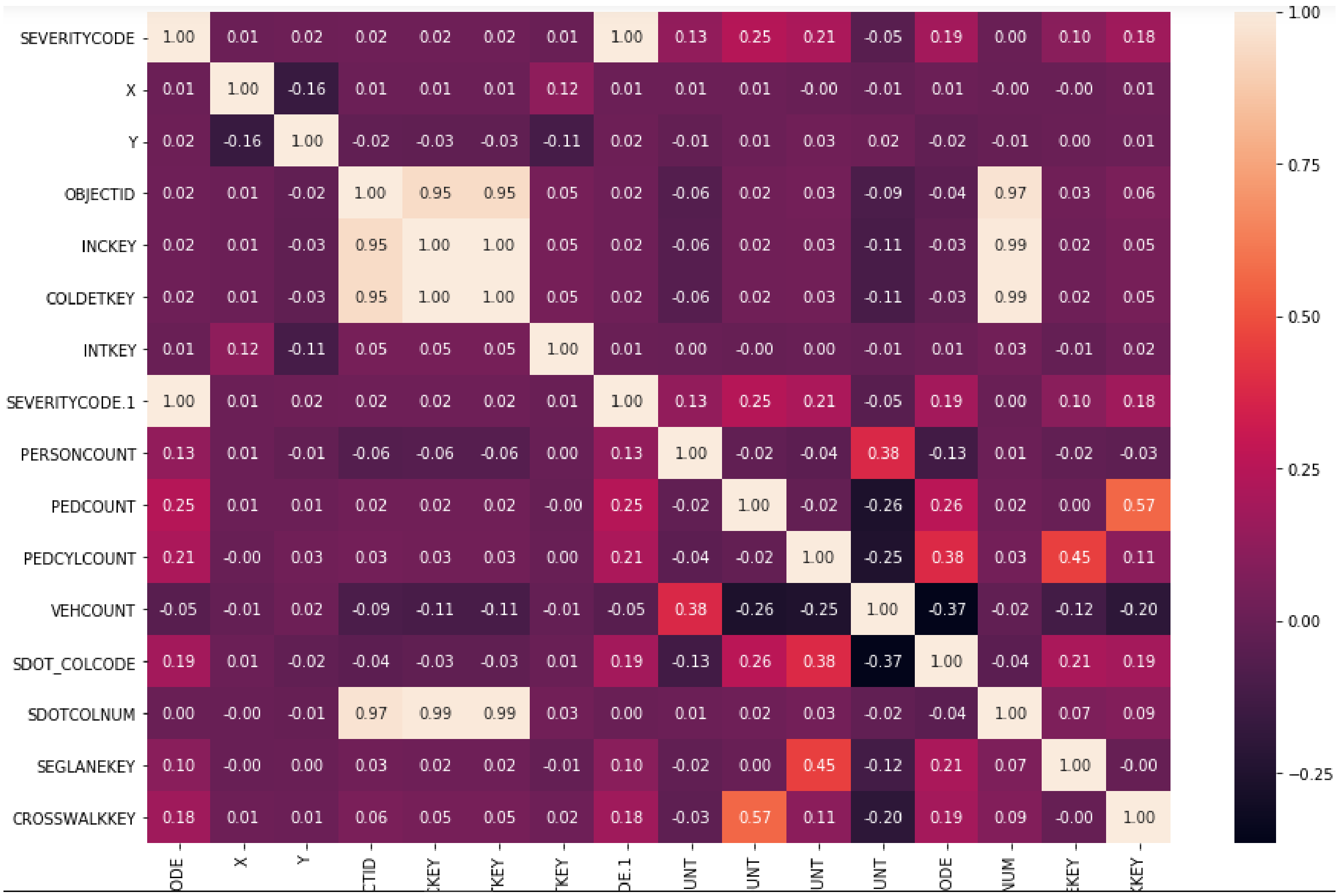


- Here we look to the correlation between Feature

```python
# we now see the corrlation beween Features
data_corr = df.corr()
data_corr
```

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	INTKEY	SEVERITYCODE.1	PERSONCOUNT	PEDCOUNT	PED(
SEVERITYCODE	1.000000	0.010309	0.017737	0.020131	0.022065	0.022079	0.006553	1.000000	0.130949	0.246338	
X	0.010309	1.000000	-0.160262	0.009956	0.010309	0.010300	0.120754	0.010309	0.012887	0.011304	
Y	0.017737	-0.160262	1.000000	-0.023848	-0.027396	-0.027415	-0.114935	0.017737	-0.013850	0.010178	
OBJECTID	0.020131	0.009956	-0.023848	1.000000	0.946383	0.945837	0.046929	0.020131	-0.062333	0.024604	
INCKEY	0.022065	0.010309	-0.027396	0.946383	1.000000	0.999996	0.048524	0.022065	-0.061500	0.024918	
COLDETKEY	0.022079	0.010300	-0.027415	0.945837	0.999996	1.000000	0.048499	0.022079	-0.061403	0.024914	
INTKEY	0.006553	0.120754	-0.114935	0.046929	0.048524	0.048499	1.000000	0.006553	0.001886	-0.004784	
SEVERITYCODE.1	1.000000	0.010309	0.017737	0.020131	0.022065	0.022079	0.006553	1.000000	0.130949	0.246338	
PERSONCOUNT	0.130949	0.012887	-0.013850	-0.062333	-0.061500	-0.061403	0.001886	0.130949	1.000000	-0.023464	
PEDCOUNT	0.246338	0.011304	0.010178	0.024604	0.024918	0.024914	-0.004784	0.246338	-0.023464	1.000000	
PEDCYLCOUNT	0.214218	-0.001752	0.026304	0.034432	0.031342	0.031296	0.000531	0.214218	-0.038809	-0.016920	
VEHCOUNT	-0.054686	-0.012168	0.017058	-0.094280	-0.107528	-0.107598	-0.012929	-0.054686	0.380523	-0.261285	
SDOT_COLCODE	0.188905	0.010904	-0.019694	-0.037094	-0.027617	-0.027461	0.007114	0.188905	-0.128960	0.260393	
SDOTCOLNUM	0.004226	-0.001016	-0.006958	0.969276	0.990571	0.990571	0.032604	0.004226	0.011784	0.021461	
SEGLANEKEY	0.104276	-0.001618	0.004618	0.028076	0.019701	0.019586	-0.010510	0.104276	-0.021383	0.001810	
CROSSWALKKEY	0.175093	0.013586	0.009508	0.056046	0.048179	0.048063	0.018420	0.175093	-0.032258	0.565326	

- Make Correlation more beautiful



- We now will select feature that will help us in Machine learning Model

```
#we will focues in some feature that make result
car_acc = df[['WEATHER','ROADCOND','LIGHTCOND','VEHCOUNT','JUNCTIONTYPE','PERSONCOUNT','SEVERITYCODE']]
```

**Data Dictionary* *

1. WEATHER : A description of the weather conditions during the time of the collision.
2. ROADCOND : The condition of the road during the collision.
3. LIGHTCOND : The light conditions during the collision.
4. VEHCOUNT : The number of vehicles involved in the collision. This is entered by the state.
5. JUNCTIONTYPE : Category of junction at which collision took place
6. PERSONCOUNT : The total number of people involved in thecollision
7. SEVERITYCODE : A code that corresponds to the severity of the collision:

```
* 3 — fatality
* 2b —serious injury
* 2 — injury
* 1 — prop damage
* 0 — unknown
```

- This's our data now

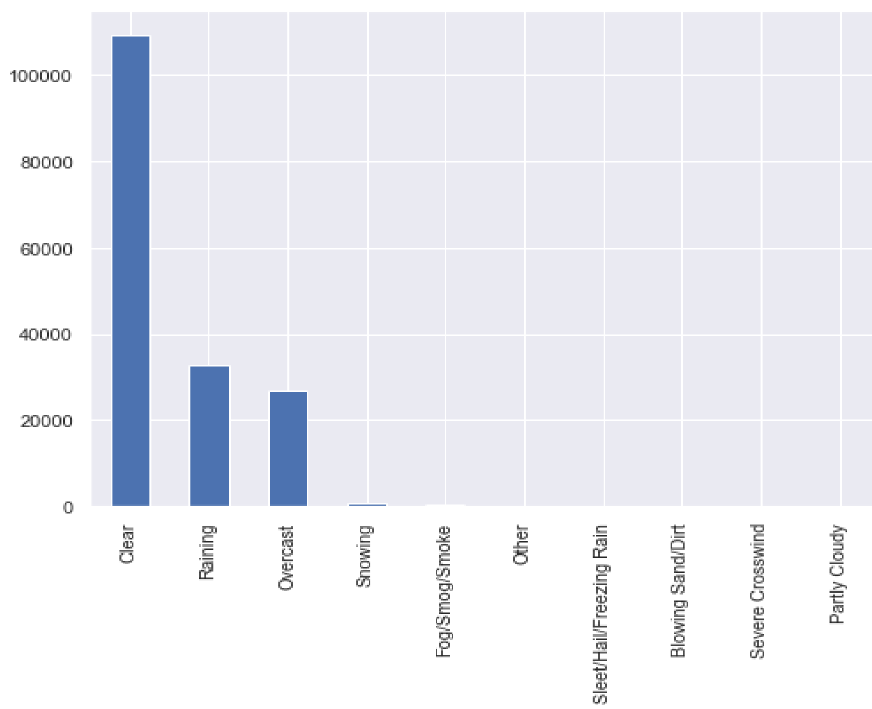
```
1  car_acc
```

	WEATHER	ROADCOND	LIGHTCOND	VEHCOUNT	JUNCTIONTYPE	PERSONCOUNT	SEVERITYCODE
0	Overcast	Wet	Daylight	2	At Intersection (intersection related)	2	2
1	Raining	Wet	Dark - Street Lights On	2	Mid-Block (not related to intersection)	2	1
2	Overcast	Dry	Daylight	3	Mid-Block (not related to intersection)	4	1
3	Clear	Dry	Daylight	3	Mid-Block (not related to intersection)	3	1
4	Raining	Wet	Daylight	2	At Intersection (intersection related)	2	2
...	...	...	...	...	...	...	...
194668	Clear	Dry	Daylight	2	Mid-Block (not related to intersection)	3	2
194669	Raining	Wet	Daylight	2	Mid-Block (not related to intersection)	2	1
194670	Clear	Dry	Daylight	2	At Intersection (intersection related)	3	2
194671	Clear	Dry	Dusk	1	At Intersection (intersection related)	2	2
194672	Clear	Wet	Daylight	2	Mid-Block (not related to intersection)	2	1

194673 rows × 7 columns

- Make Soma Analysis to our data :

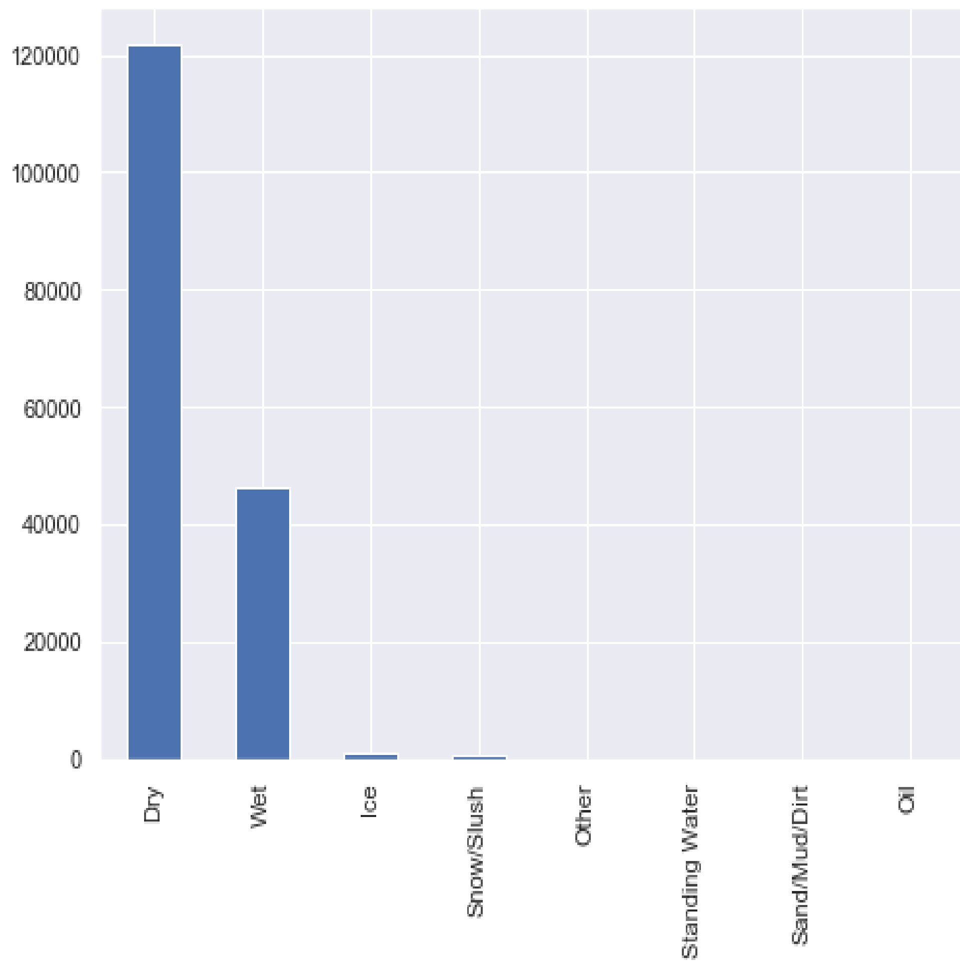
```
1  car_acc['WEATHER'].value_counts().plot(kind='bar',figsize=(8,6));
2
```



Most car accident happen when weather is :

```
- Clear
- Raining
- Overcast
```

```
car_acc['ROADCOND'].value_counts().plot(kind='bar',figsize=(8,6));
```



here The condition of the road during the collision is :

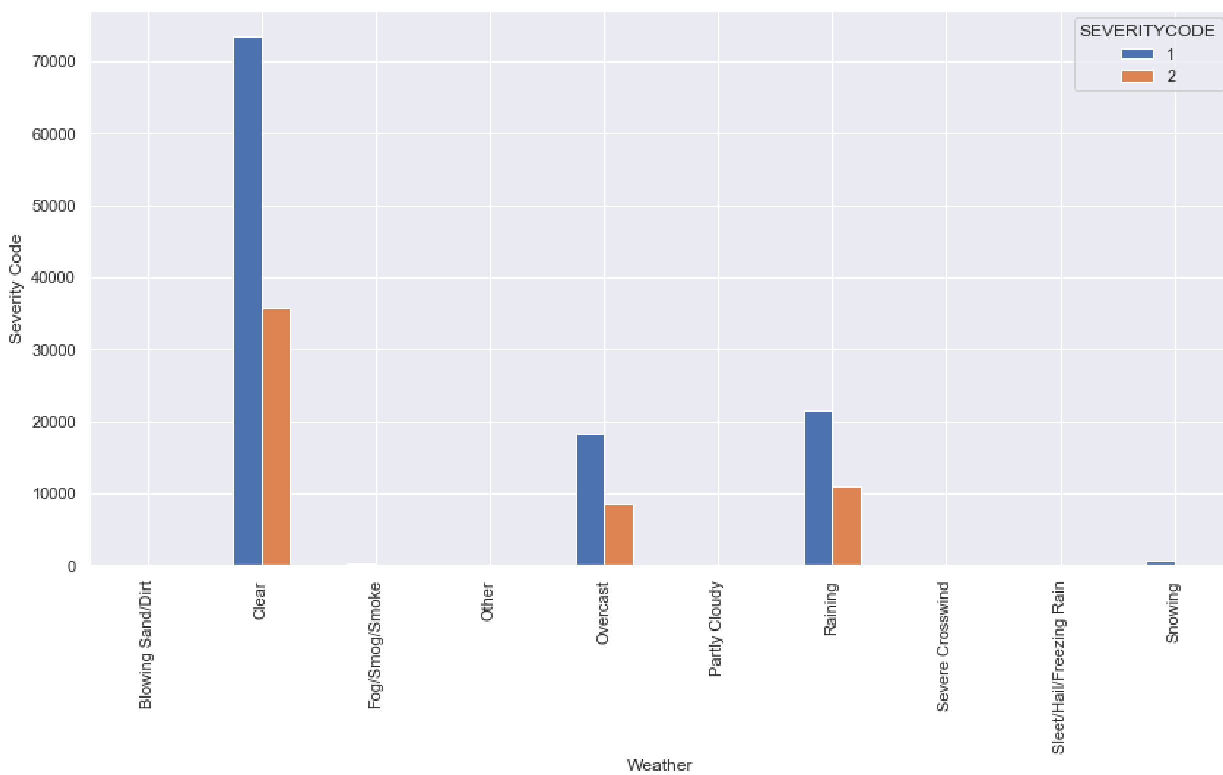
```
- Dry
- Wet
```

Find The Patterns between WEATHER VS SEVERITYCODE

```
pd.crosstab(df.WEATHER,df.SEVERITYCODE)
```

SEVERITYCODE	1	2
WEATHER		
Blowing Sand/Dirt	41	15
Clear	75295	35840
Fog/Smog/Smoke	382	187
Other	716	116
Overcast	18969	8745
Partly Cloudy	2	3
Raining	21969	11176
Severe Crosswind	18	7
Sleet/Hail/Freezing Rain	85	28
Snowing	736	171
Unknown	14275	816

```
pd.crosstab(car_acc.WEATHER,car_acc.SEVERITYCODE).plot(kind='bar',figsize=(14,7));
plt.xlabel('Weather')
plt.ylabel('Severity Code');
```



- Remove Outlier like "Unknown"

```
car_acc.drop(car_acc[car_acc['WEATHER']=='Unknown'].index,inplace=True)
```

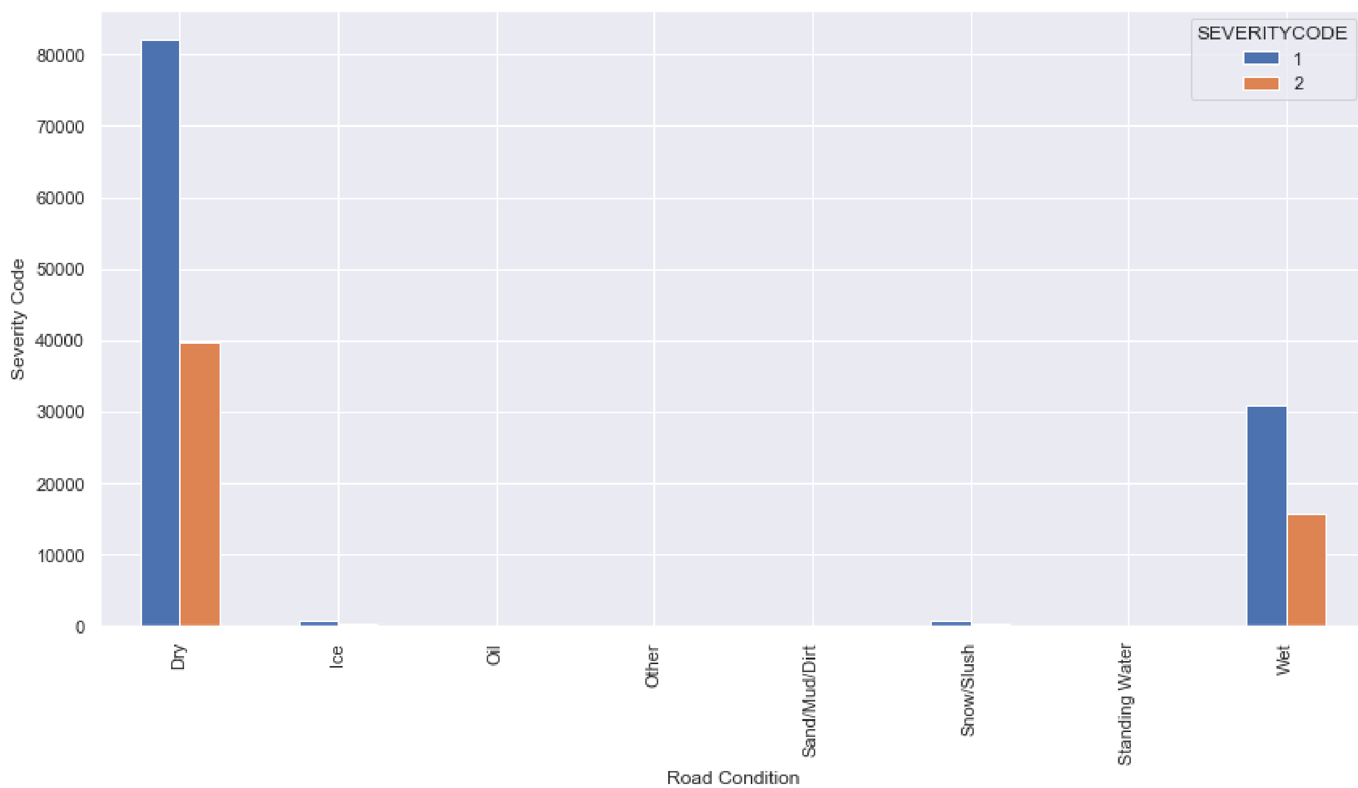
```
C:\Users\Mohamed Basher\Anaconda3\lib\site-packages\pandas\core\frame.py:4102: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

```
pd.crosstab(car_acc.WEATHER,car_acc.SEVERITYCODE).plot(kind='bar',figsize=(14,7));
plt.xlabel('Weather')
plt.ylabel('Severity Code');
```

Find The Patterns between Road Condition VS SEVERITYCODE

```
pd.crosstab(car_acc.ROADCOND,car_acc.SEVERITYCODE).plot(kind='bar',figsize=(14,7));
plt.xlabel('Road Condition')
plt.ylabel('Severity Code');
#we see that most Car accident happen in Road Condition : Dry and Wet
```



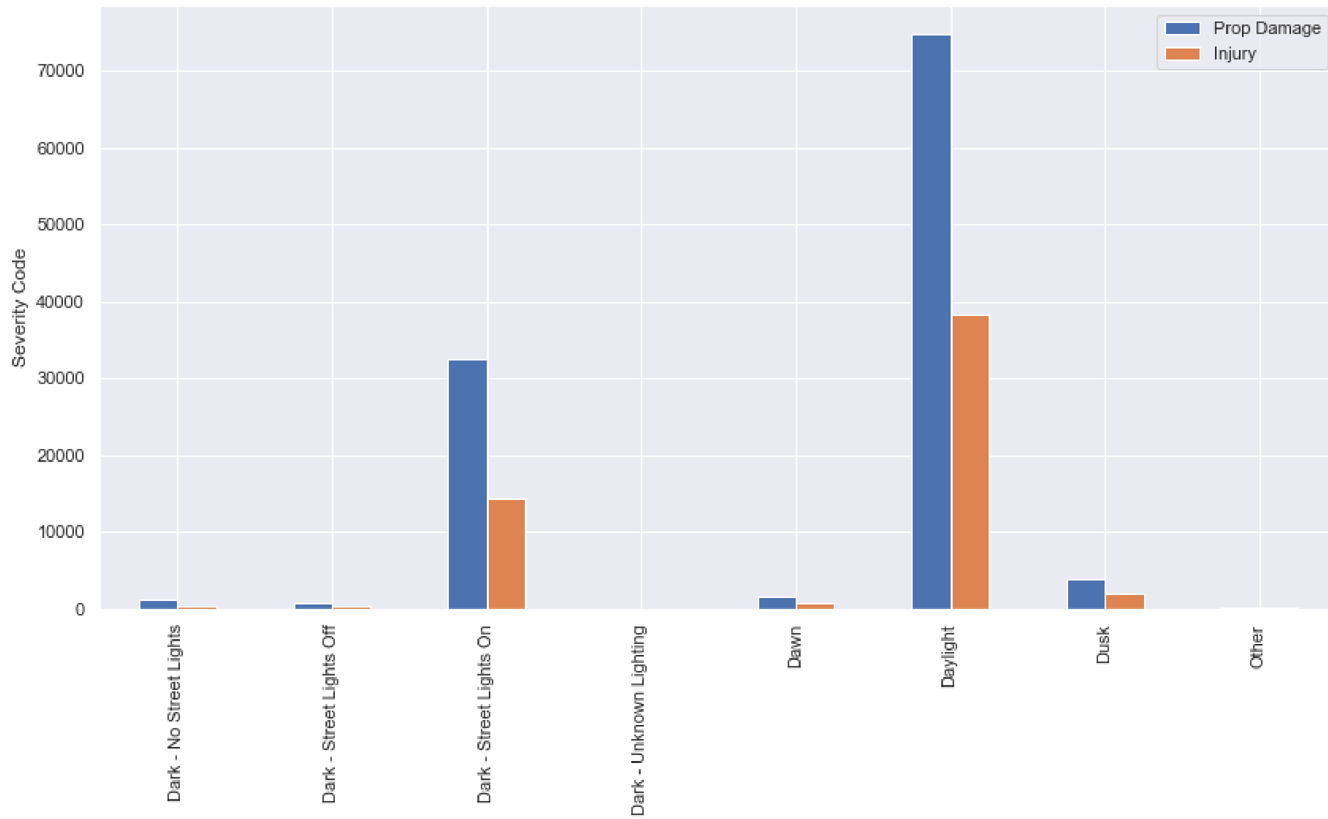

```
#we will drop 'Unknown' Values
car_acc.drop(car_acc[car_acc['ROADCOND']=='Unknown'].index,inplace=True)
```

```
car_acc.ROADCOND.unique()
```

```
array(['Wet', 'Dry', nan, 'Snow/Slush', 'Ice', 'Other', 'Sand/Mud/Dirt',
       'Standing Water', 'Oil'], dtype=object)
```

-Find the Patterns between Light Condition VS SEVERITYCODE

```
pd.crosstab(car_acc.LIGHTCOND,car_acc.SEVERITYCODE).plot(kind='bar',figsize=(14,7));
plt.xlabel('Light Condition ')
plt.ylabel('Severity Code');
plt.legend({'Prop Damage':1,'Injury':2});
```



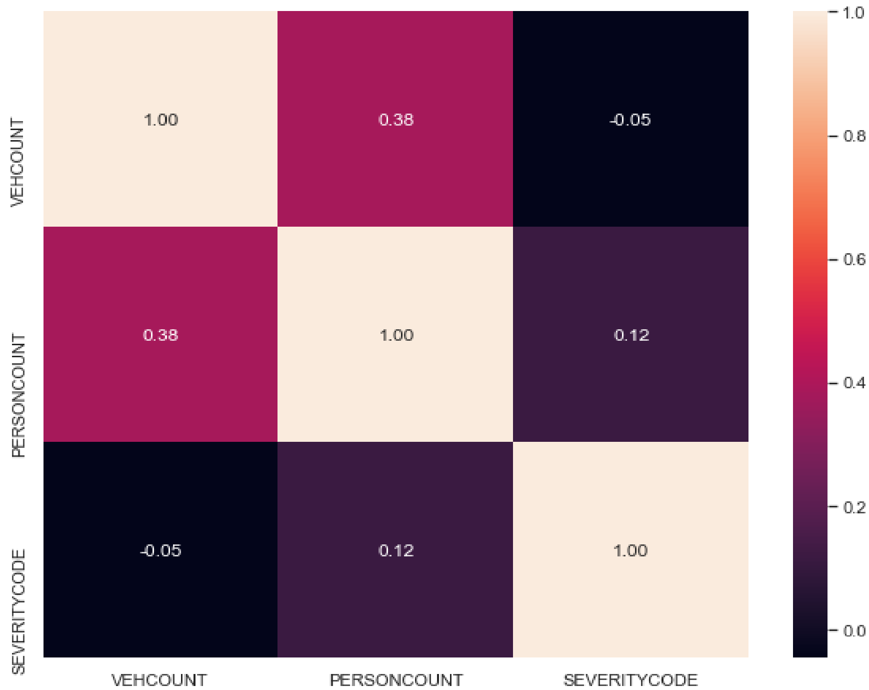

```
#we will drop 'Unknown' Values
car_acc.drop(car_acc[car_acc['LIGHTCOND']=='Unknown'].index,inplace=True)
```

```
car_acc.LIGHTCOND.unique()
```

```
array(['Daylight', 'Dark - Street Lights On', 'Dark - No Street Lights',
       nan, 'Dusk', 'Dawn', 'Dark - Street Lights Off', 'Other',
       'Dark - Unknown Lighting'], dtype=object)
```

Correlation Between Features

```
plt.figure(figsize=(10,8))
ax = sns.heatmap(car_acc.corr(),annot=True,fmt='.2f')
bottom ,top = ax.get_ylim()
ax.set_ylim(bottom + 0.5,top - 0.5);
```



Now we will prepare Data to use in Machine learning Model

- We looking first for Convert Object , String values to Numeric Values
- And Fill Missing Values in our Data

```
data_info = pd.DataFrame(data=car_acc.isna().sum(),columns=['Missing Values'],index=car_acc.columns)
data_info['Data Types'] = car_acc.dtypes
```

```
data_info
```

	Missing Values	Data Types
WEATHER	5067	object
ROADCOND	4995	object
LIGHTCOND	5121	object
VEHCOUNT	0	int64
JUNCTIONTYPE	2676	object
PERSONCOUNT	0	int64
SEVERITYCODE	0	int64

Import library for fill Missing Data and Convert Categorical data to Numerical Values , We will use Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
```

```
#Define Defferent Features and transformer Pipeline

categorical_feature = ['WEATHER','ROADCOND','LIGHTCOND','JUNCTIONTYPE']
categorical_transformer =Pipeline(steps=[
    ('imputer',SimpleImputer(strategy='constant',fill_value='missing')),
    ('onehot',OneHotEncoder(handle_unknown='ignore'))])
```

```
#Split Data

X=car_acc.drop('SEVERITYCODE',axis=1)
y=car_acc['SEVERITYCODE']
```

```
#setup preprocessing Steps (Fill Missing values , then convert to numbers)

preprocessor =ColumnTransformer(transformers=[('cat',categorical_transformer,categorical_feature)])

preprocessor_x =preprocessor.fit_transform(X)
```

```
#bulid Train & Test Data

X_train ,X_test,y_train,y_test = train_test_split(preprocessor_x,y,test_size=0.2,random_state = 42)
```

Try 3 Different ML Models :

1. Logistic Regrassion
2. K-Nearest Neighbours Classifier
3. Random Forest Classifier

- Define Function to Fit and Score all models in one cell:

```
#Create Dictionary containing our Models
models = {'Logistic Regrassion': LogisticRegression(),
          'KNN':KNeighborsClassifier() ,
          'Random Forest ':RandomForestClassifier()}

#Create a Function to fit and score models

def fit_and_score(models ,X_train,X_test,y_train,y_test):

    """
    Fit and Evaluate Models
    Models : a dictionary of Sklearn ML Models
    X_train :Trainning Data (No labels)
    X_test : Testing Data (No labels)
    y_train :Trainnin labels
    y_test :testing Labels
    """

    #set a rondom Seed

    np.random.seed(42)

    #Make a dictionary to keep model score

    models_score ={}

    #Loop Through Models

    for name , model in models.items():

        #fit the model to the data

        model.fit(X_train,y_train)

        #Evaluate the model and append to models_score

        models_score[name]=model.score(X_test,y_test)

    return models_score
```

- it takes times to run

```
#try our Models
models_score = fit_and_score(models ,X_train,X_test,y_train,y_test)
models_score
```

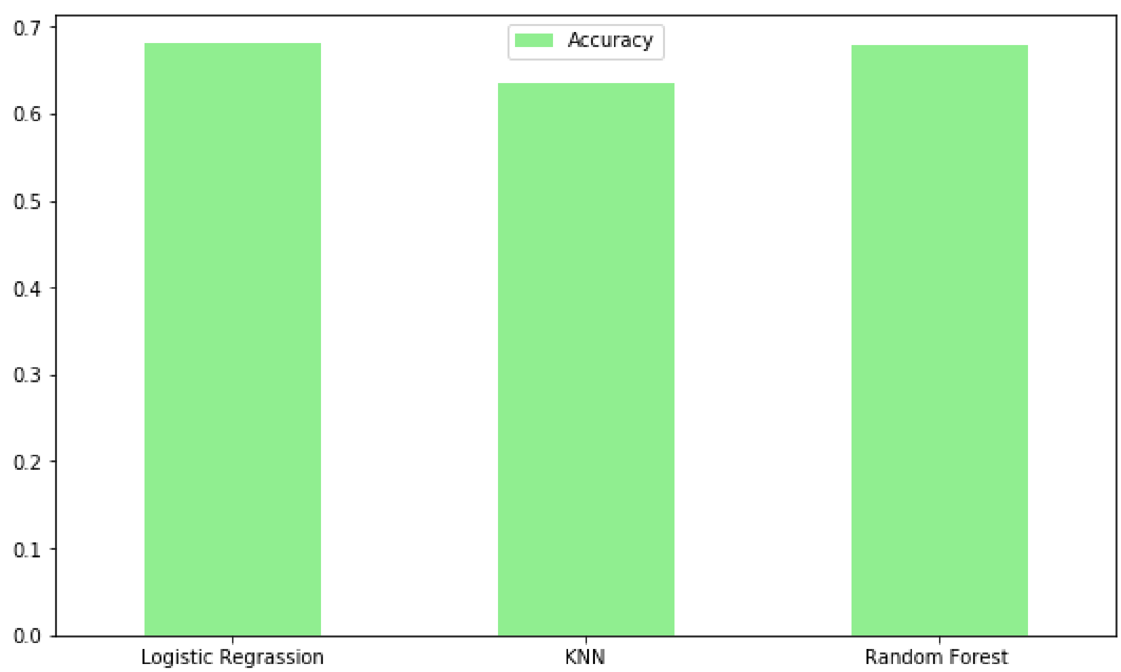
```
C:\Users\Mohamed Basher\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be
changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\Mohamed Basher\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estim
ators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
{'Logistic Regrassion': 0.6803584127435642,
 'KNN': 0.6354999288863604,
 'Random Forest ': 0.6795050490684114}
```

Model Comparison

```
#Create Data Frame
model_comp_df = pd.DataFrame(models_score ,index=['Accuracy'])

#plot our Models
model_comp_df.T.plot.bar(figsize=(10,6),color='lightgreen')
plt.xticks(rotation=0);
```



- We try to improve our model by using different Parameter

Hyperprameter Tuning with GridSearchCV

Using Different parameter in our models to increase model's accuracy

```
from sklearn.model_selection import GridSearchCV
```

```
# We will try to improve Logistic Regrassion

# Different LogisticRegression hyperparameters
log_reg_grid = {"C": np.logspace(-4, 4, 20),
                "solver": ["liblinear"]}
```

- Now let's use GridSearchCV with Logistic Regrassion

```
#Setup hyperprameter in GridSearchCV
gs_log_reg = GridSearchCV(LogisticRegression(),
                          param_grid=log_reg_grid,
                          cv=5,
                          verbose=True)

#Fit hyperprameter using GridSearchCV in LG
gs_log_reg.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

- I face some problem with my PC when I try in Random Forest and KNN, so I will work in Logistic Regression

```
#Best parameter we found
gs_log_reg.best_params_
```

```
{'C': 0.0001, 'solver': 'liblinear'}
```

```
#Score The Model
print(f"Logistic Regression Score : {gs_log_reg.score(X_test,y_test) *100:0.2f}%")
```

```
Logistic Regression Score : 68.04%
```

Evaluating a classification model

- we use :
  - Classification report - classification_report()
  - Precision - precision_score()
  - F1-score - f1_score()

```
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
```

```
y_preds = gs_log_reg.predict(X_test)
```

Classification report

We can make a classification report using classification_report() and passing it the true labels as well as our models predicted labels.

```
print(classification_report(y_test,y_preds))
```

```
              precision    recall  f1-score   support

           1       0.68      1.00      0.81     23921
           2       0.00      0.00      0.00     11234

    accuracy                           0.68     35155
   macro avg       0.34      0.50      0.40     35155
weighted avg       0.46      0.68      0.55     35155
```

Cross validation score

We'll take the best model along with the best hyperparameters and use cross_val_score() along with various scoring parameter values.

cross_val_score() works by taking an estimator (machine learning model) along with data and labels. It then evaluates the machine learning model on the data and labels using cross-validation and a defined scoring parameter.

```
gs_log_reg.best_params_
```

```
{'C': 0.0001, 'solver': 'liblinear'}
```

```
clf =LogisticRegression(C=0.0001,
                        solver='liblinear')
```

```
# Cross-validated accuracy score

cv_acc =cross_val_score(clf,
                        preprocessor_x,
                        y,
                        cv=5,
                        scoring="accuracy")
cv_acc
```

```
array([0.67580714, 0.67580714, 0.67580714, 0.67581714, 0.67581714])
```

```
cv_acc =np.mean(cv_acc)
cv_acc
```

```
0.6758111407684189
```

```
# Cross-validated precision score

cv_pre =np.mean(cross_val_score(clf,
                        preprocessor_x,
                        y,
                        cv=5,
                        scoring="precision"))
cv_pre
```

```
0.6758111407684189
```

```
# Cross-validated F1 score

cv_f1 =np.mean(cross_val_score(clf,
                        preprocessor_x,
                        y,
                        cv=5,
                        scoring="f1"))
cv_f1
```

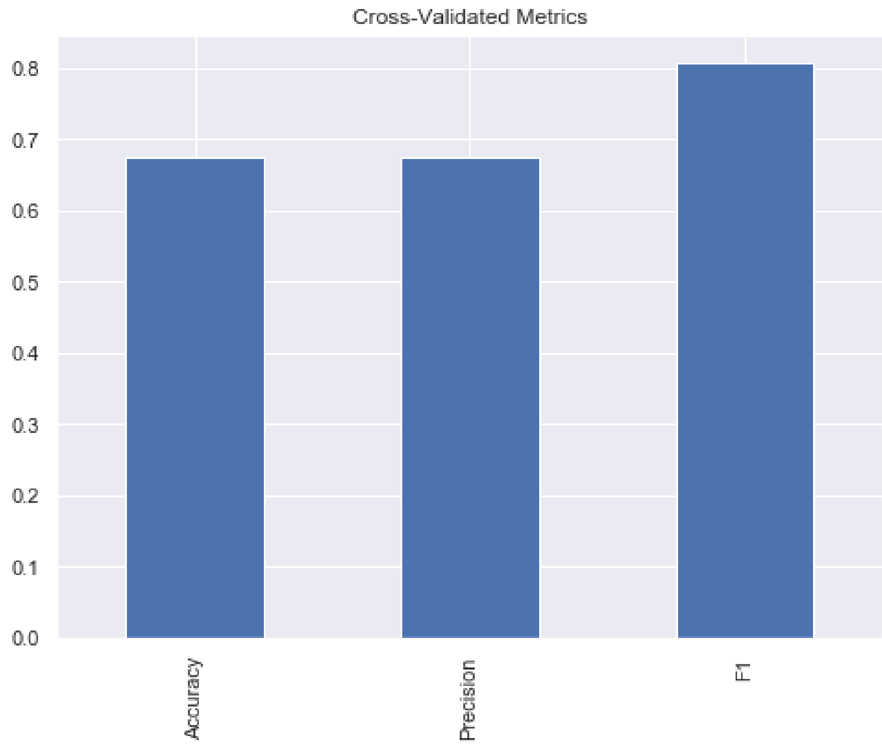
```
0.8065480940173073
```

Plot CV Metrics Score

```
sns.set()

cv_metrics =pd.DataFrame({'Accuracy':cv_acc,'Precision':cv_pre,'F1':cv_f1},index=[0])

cv_metrics.T.plot.bar(figsize=(8,6),title="Cross-Validated Metrics", legend=False);
```



5.Discussion

After the analysis data, we found

1. most car accident severity:
    - prop damage
    - injury
2. Most car accident happens when the weather is:
    - Clear
    - Raining
    - Overcast
3. The condition of the road during the collision is:
    - Dry
    - Wet

6.Conclusion

- We need to add some other models to precisely predict traffic accident severity. We need to improve the accuracy of the model too

- In the future, this model still needs improvement, and we need some additional features to increase prediction accuracy