CENTENNIAL COLLEGE

Toronto Police Service
PUBLIC SAFETY DATA PORTAL

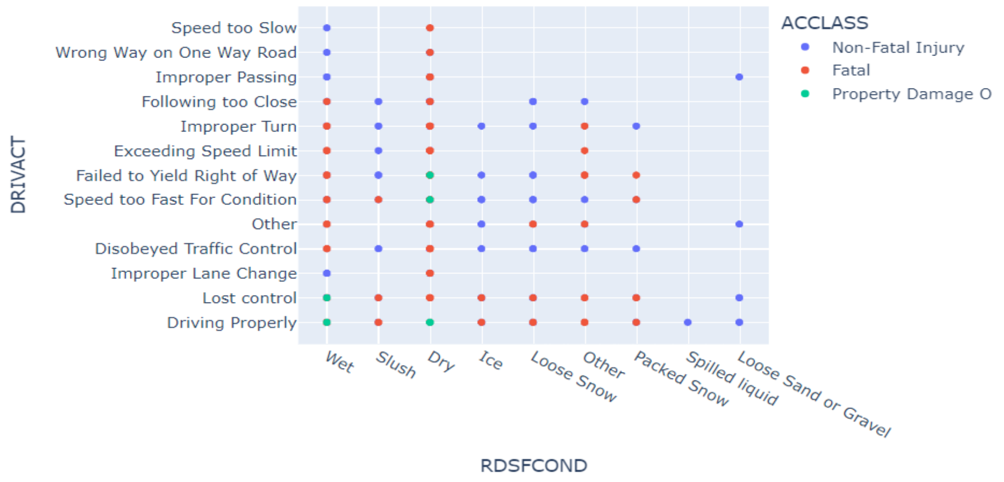
KSI-Related Collisions in Toronto: A predictive model with an app

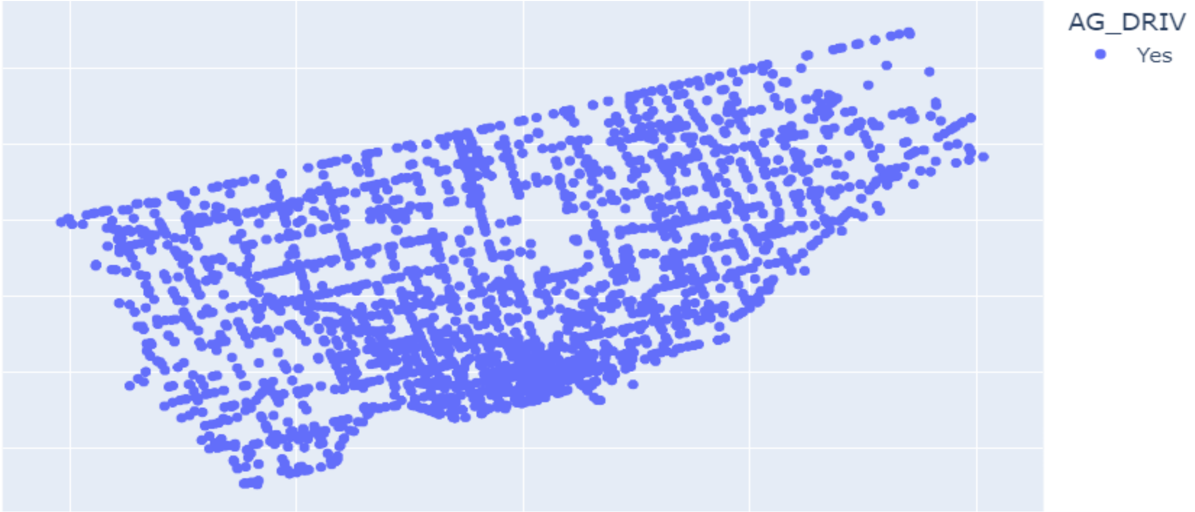
Huaye Zhan[1], Helia Mozaffari[1], Mattia Carganico[1], Jaekyeong Jang[1], Hsin-Tung Chen[1]

[1] Artificial Intelligence - Software engineer technology, Centennial College

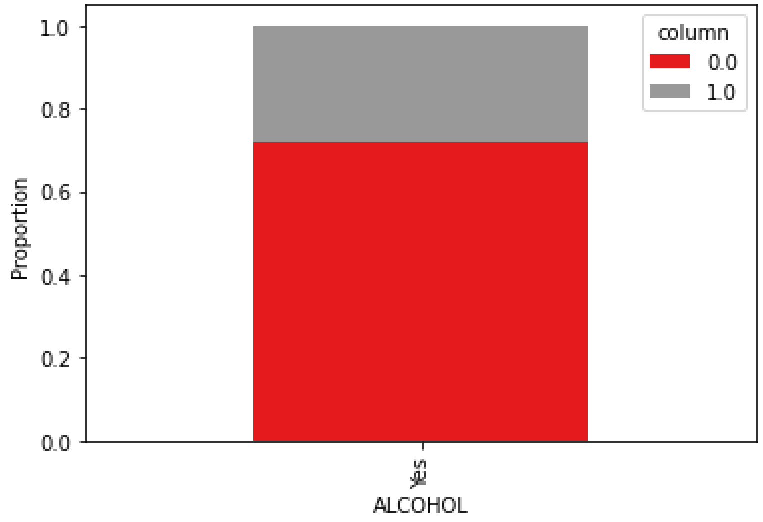
content

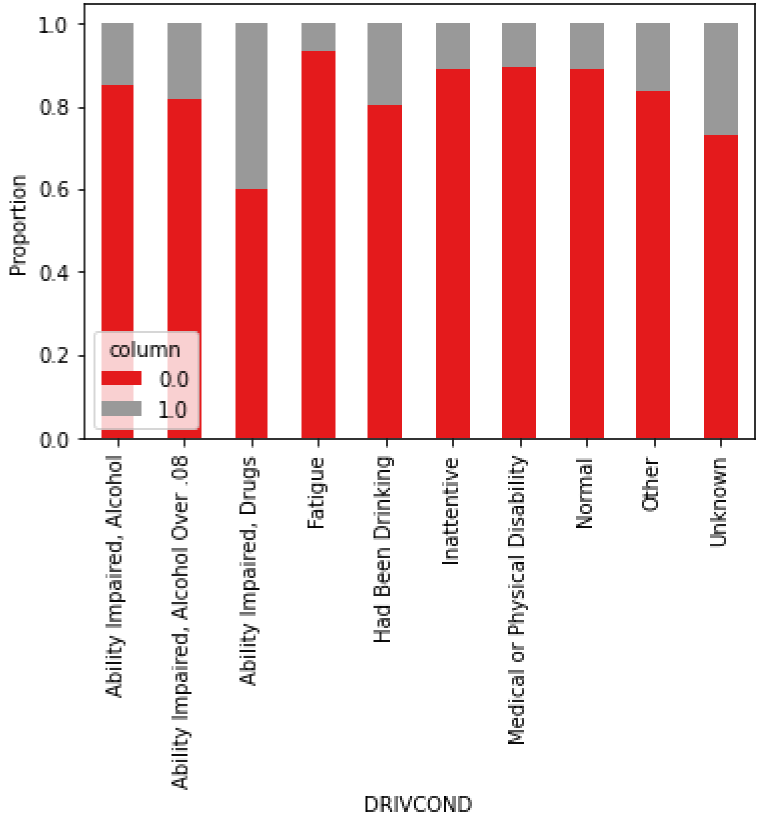
REVISITING VISUALIZATION





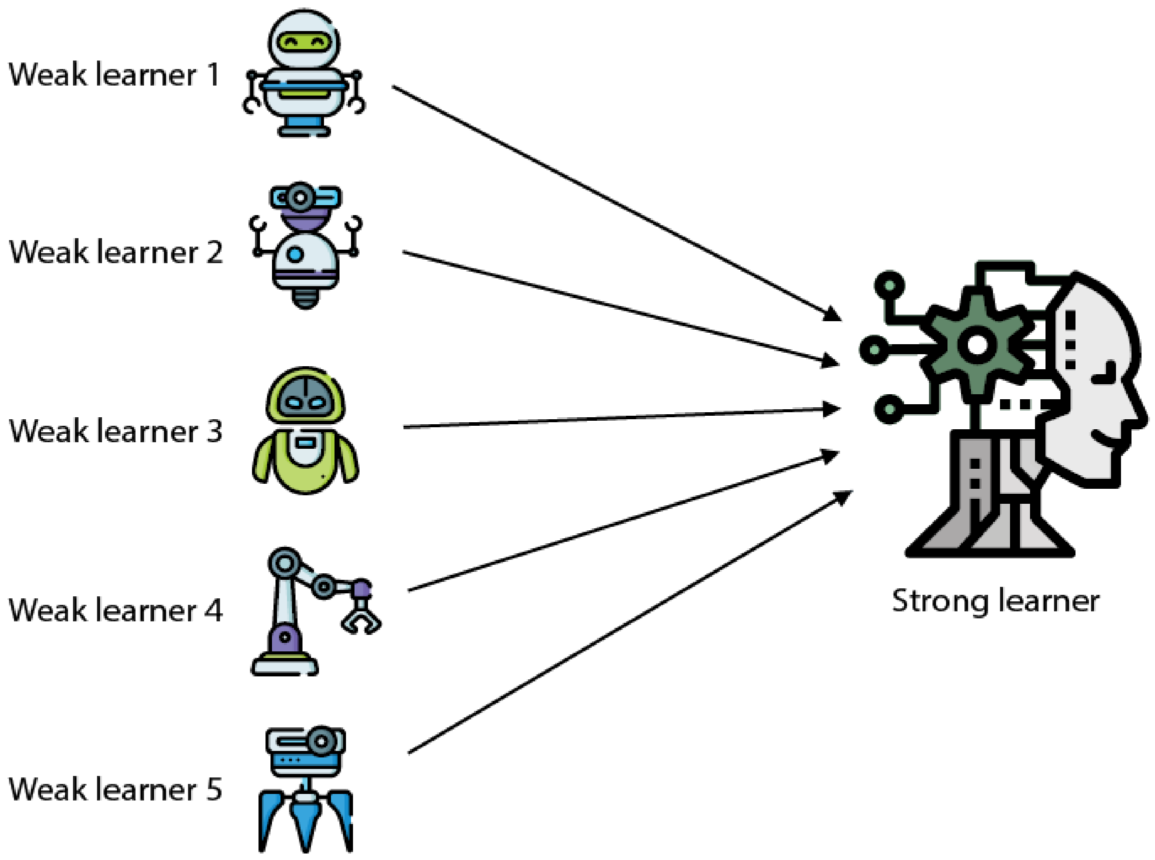
Multicollinearity and nulls





CYCLISTYF	CYCACT	CYCCOND	PEDESTRIAN	CYCLIST	AUTOMOBILE	MOTORCYCLE	TRUCK	TRSN_CITY_VEH	EMERG_VEH	PASSENGER	SPEEDING	AG_DRIV	REDLIGHT	ALCOHOL	DISABILITY
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes	Yes	Yes		Yes	
					Yes					Yes					
					Yes					Yes					
					Yes					Yes					
					Yes					Yes					
					Yes					Yes					

Choosing a model



https://livebook.manning.com/book/grokking-machine-learning/chapter-12/15

ETL: Extract, transform, load

Snow	Dark, artifi	Slush	Non-Fatal	SMV Othe	Driver	20 to 24	None
Snow	Dark, artifi	Slush	Non-Fatal	SMV Othe	Other Prop	unknown	
Other	Dark, artifi	Wet	Non-Fatal	Pedestrian	Driver	30 to 34	None
Other	Dark, artifi	Wet	Non-Fatal	Pedestrian	Pedestrian	45 to 49	Major
Rain	Dark	Wet	Non-Fatal	Pedestrian	Driver	25 to 29	None
Rain	Dark	Wet	Non-Fatal	Pedestrian	Pedestrian	75 to 79	Major
Clear	Dark, artifi	Dry	Non-Fatal	Pedestrian	Driver	50 to 54	None
Clear	Dark, artifi	Dry	Non-Fatal	Pedestrian	Pedestrian	25 to 29	Major
Clear	Dark	Wet	Fatal	Approachi	Driver	50 to 54	Fatal
Clear	Dark	Wet	Fatal	Approachi	Vehicle Ov	unknown	
Clear	Dark	Wet	Fatal	Approachi	Driver	35 to 39	Major
Clear	Daylight	Dry	Non-Fatal	Angle	Driver	40 to 44	Minimal
Clear	Daylight	Dry	Non-Fatal	Angle	Driver	45 to 49	Major
Clear	Daylight	Dry	Non-Fatal	Angle	Other Prop	unknown	
Clear	Daylight	Dry	Non-Fatal	Angle	Other Prop	unknown	
Clear	Dark	Dry	Fatal	Pedestrian	Passenger	20 to 24	None
Clear	Dark	Dry	Fatal	Pedestrian	Passenger	20 to 24	None
Clear	Dark	Dry	Fatal	Pedestrian	Passenger	10 to 14	None
Clear	Dark	Dry	Fatal	Pedestrian	Vehicle Ov	unknown	
Clear	Dark	Dry	Fatal	Pedestrian	Driver	20 to 24	None
Clear	Dark	Dry	Fatal	Pedestrian	Pedestrian	10 to 14	Fatal
Clear	Daylight	Dry	Fatal	Pedestrian	Vehicle Ov	unknown	
Clear	Daylight	Dry	Fatal	Pedestrian	Driver	50 to 54	None
Clear	Daylight	Dry	Fatal	Pedestrian	Pedestrian	75 to 79	Fatal
Clear	Dark, artifi	Dry	Non-Fatal	Pedestrian	Vehicle Ov	unknown	
Clear	Dark, artifi	Dry	Non-Fatal	Pedestrian	Driver	55 to 59	None
Clear	Dark, artifi	Dry	Non-Fatal	Pedestrian	Pedestrian	50 to 54	Major
Clear	Daylight	Dry	Non-Fatal	Approachi	Driver	80 to 84	Minor
Clear	Daylight	Dry	Non-Fatal	Approachi	Driver	55 to 59	Major
Clear	Daylight	Wet	Fatal	SMV Othe	Passenger	15 to 19	Minimal
Clear	Daylight	Wet	Fatal	SMV Othe	Passenger	15 to 19	Minimal
Clear	Daylight	Wet	Fatal	SMV Othe	Passenger	15 to 19	Fatal
Clear	Daylight	Wet	Fatal	SMV Othe	Vehicle Ov	unknown	
Clear	Daylight	Wet	Fatal	SMV Othe	Driver	15 to 19	Major
Clear	Daylight	Wet	Fatal	SMV Othe	Other Prop	unknown	

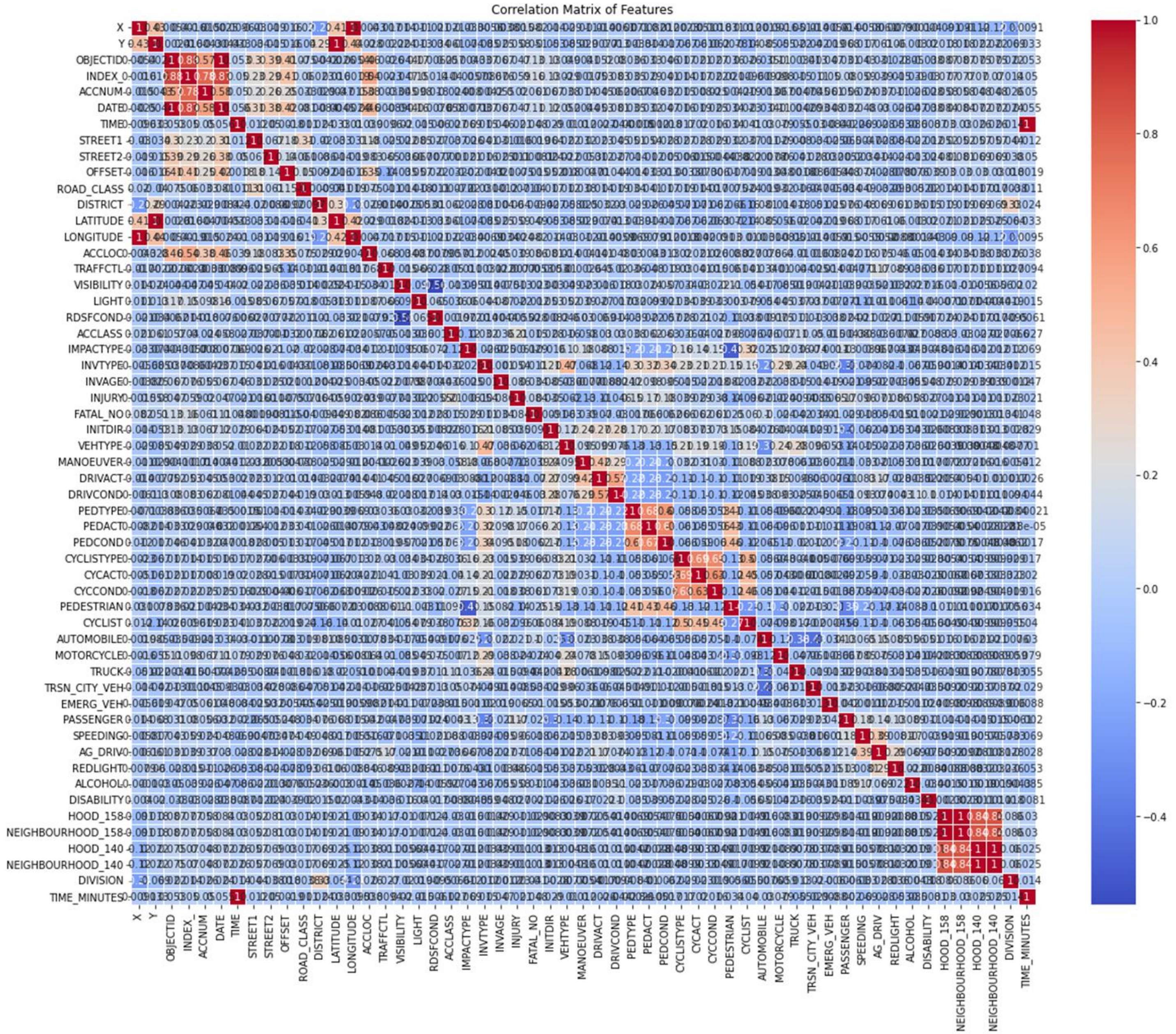
```python
fatal_rows = (load_df['ACCLASS'] == 'Fatal') & (load_df['INJURY'] == 'Fatal')
df_fatal = load_df.loc[fatal_rows]

no_fatal_row = (load_df['ACCLASS'] == 'Non-Fatal Injury')
df_non_fatal = load_df.loc[no_fatal_row]
df_non_fatal = df_non_fatal.drop_duplicates(subset=['ACCNUM'])

df_final = pd.concat([df_fatal, df_non_fatal], ignore_index=True)
df_final.to_csv('allfilter_injury_data2.csv', index=False)
```

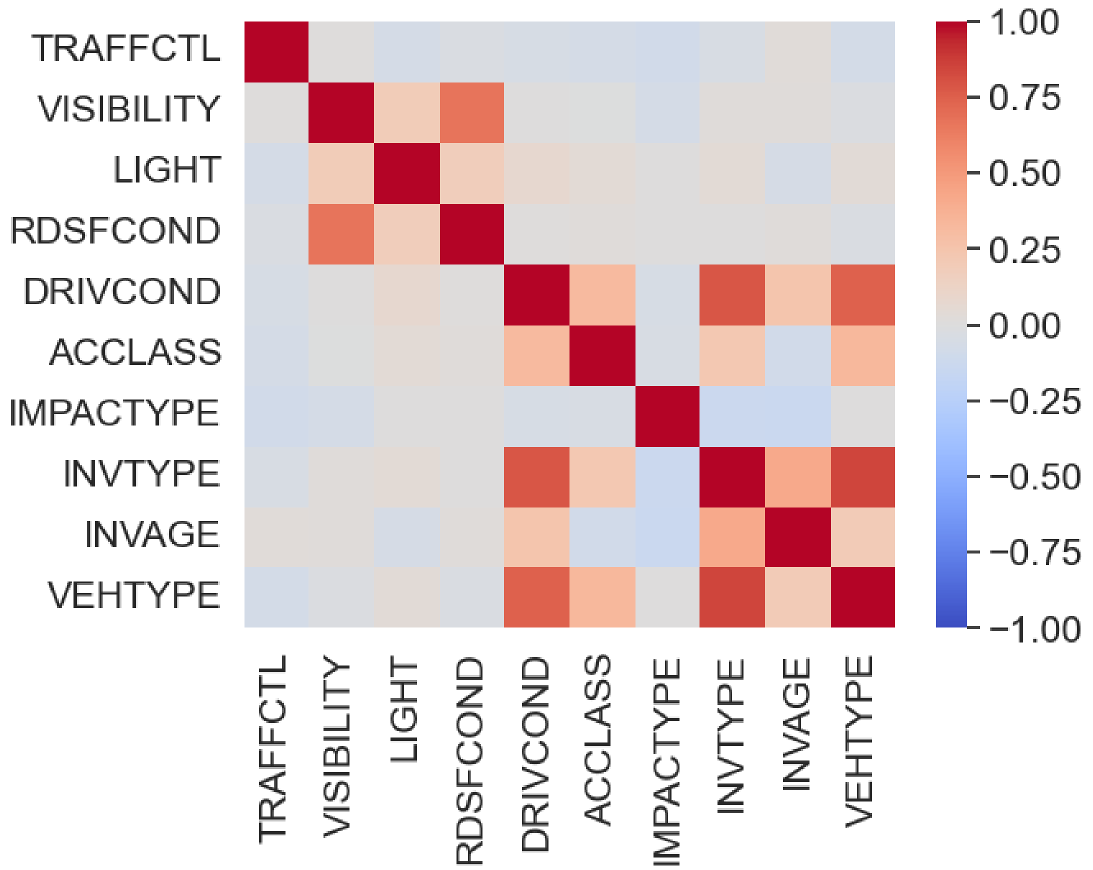
Handcrafted Ordinal Encoder

```python
'''
    1: 'Small Vehicles',
    2: 'Trucks and Vans',
    3: 'Public Transit',
    4: 'Emergency and Unknown',
    5: 'Special Equipment',
    6: 'Off-Road',
    7: 'Bicycles and Mopeds',
    8: 'Motorcycles',
    9: 'Rickshaws',
    10: 'Others'
'''
load_df["VEHTYPE"] = load_df["VEHTYPE"].fillna('Other')

classification = {
    'Automobile, Station Wagon': 1,
    'Bicycle': 7,
    'Motorcycle': 8,
    'Pick Up Truck': 1,
    'Passenger Van': 1,
    'Taxi': 1,
    'Moped': 7,
    'Delivery Van': 2,
    'Truck - Open': 2,
    'Truck - Closed (Blazer, etc)': 2,
    'Truck - Dump': 2,
    'Truck-Tractor': 2,
    'Truck (other)': 2,
    'Truck - Tank': 2,
    'Tow Truck': 2,
    'Truck - Car Carrier': 2,
    'Municipal Transit Bus (TTC)': 3,
    'Street Car': 3,
    'Bus (Other) (Go Bus, Gray Coa': 3,
    'Intercity Bus': 3,
    'School Bus': 3,
    'Other': 10,
    'Unknown': 4,
    'Police Vehicle': 4,
    'Fire Vehicle': 4,
    'Other Emergency Vehicle': 4,
    'Construction Equipment': 5,
    'Rickshaw': 9,
    'Ambulance': 4,
    'Off Road - 2 Wheels': 6,
    'Off Road - 4 Wheels': 6,
    'Off Road - Other': 6
}


load_df['VEHTYPE'] = load_df['VEHTYPE'].map(classification)
```

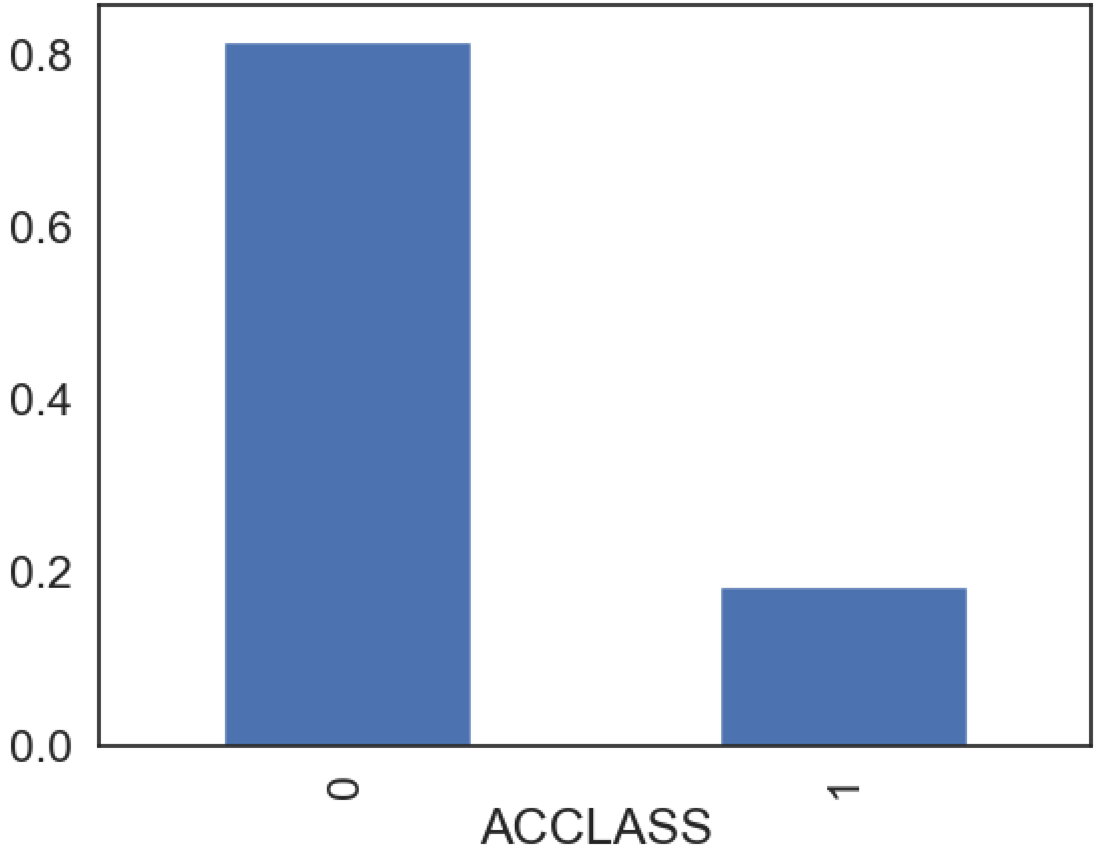
```python
'''
Dry (1)
Wet (2): Includes Wet and Spilled Liquid conditions.
Slushy/Other (3): Includes Slush and any other unspecified conditions.
Loose Surface (4): Includes Loose Snow, Packed Snow, and Loose Sand/Gravel.
Ice (5): Purely icy conditions.
'''
load_df["RDSFCOND"] = load_df["RDSFCOND"].fillna('Other')
load_df['RDSFCOND'].value_counts()
road_condition_classification = {
    'Dry': 1,                       # Category 1: Dry
    'Wet': 2,                       # Category 2: Wet
    'Slush': 3,                     # Category 3: Slushy
    'Loose Snow': 4,                # Category 4: Loose Snow
    'Packed Snow': 4,               # Category 4: Packed Snow
    'Ice': 5,                       # Category 5: Ice
    'Loose Sand or Gravel': 4,      # Category 4: Loose Sand/Gravel
    'Spilled liquid': 2,            # Category 2: Wet (Spilled Liquid)
    'Other': 3                      # Category 3: Slushy/Other
}

load_df['RDSFCOND'] = load_df['RDSFCOND'].map(road_condition_classification)
```

Correlation Analysis of Final Model Features

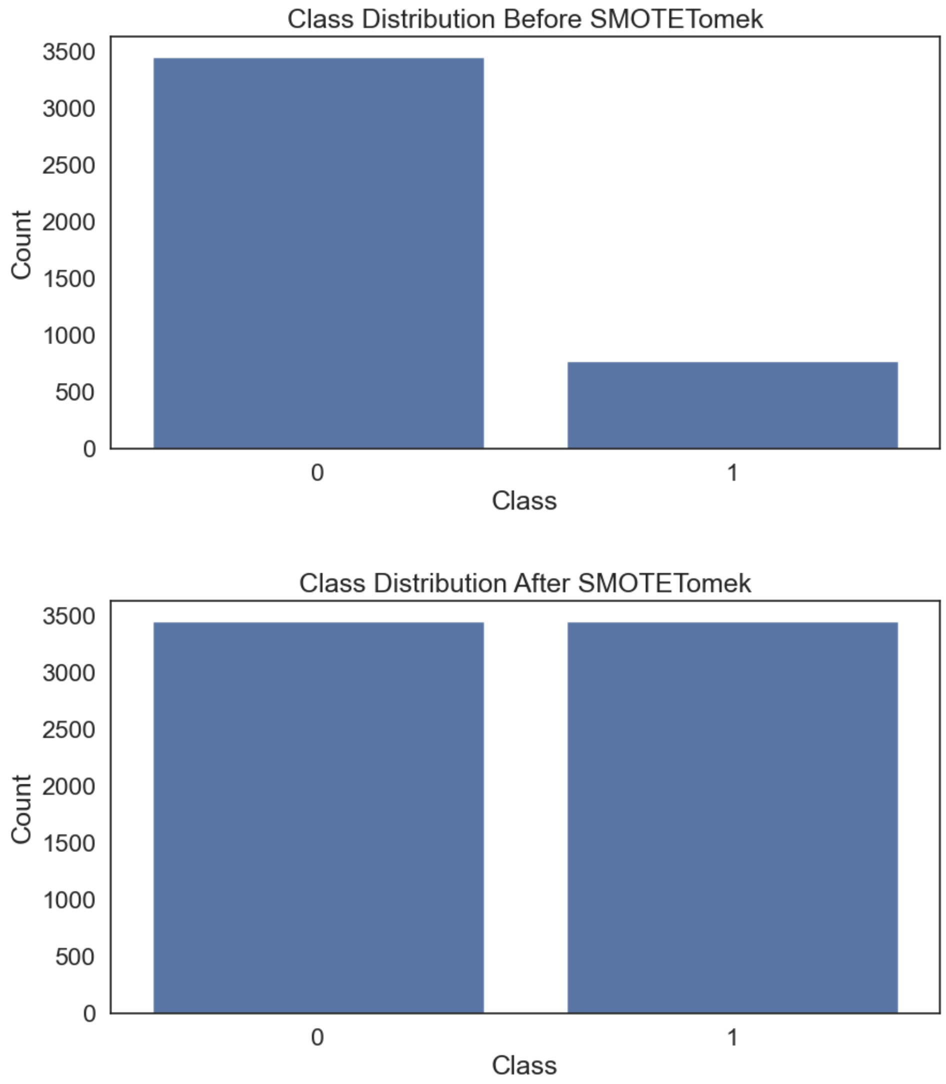




3 Imbalanced Data and Method



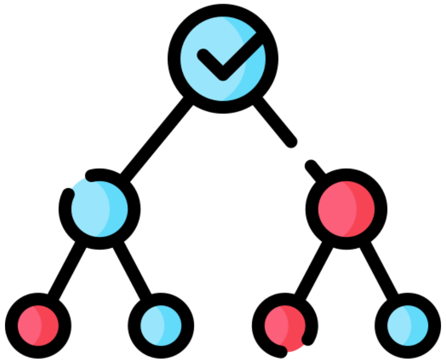
Imbalanced Target Variable



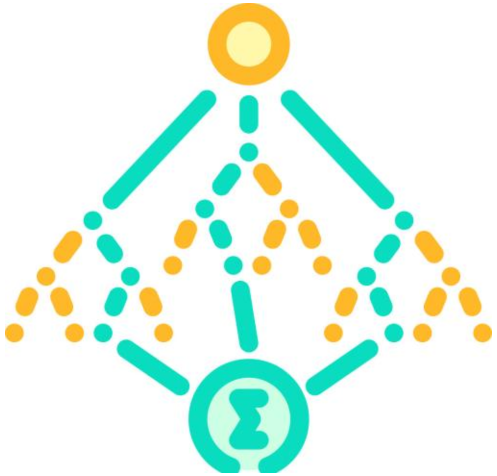
Oversamping: SMOTETomek

3 Model Comparison

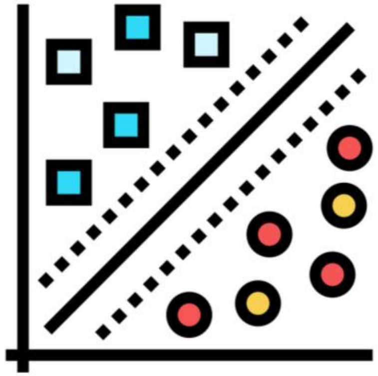
Logistic Regression



Decision Tree



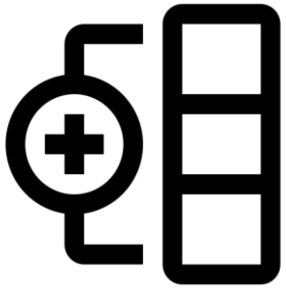
Random Forest



SVM

XGBoost

3 Data Binning VS Dummy Variables



```
new_df = load_df[['TRAFFCTL', 'VISIBILITY', 'LIGHT', 'RDSFCOND','DRIVCOND', 'ACCLASS', 'IMPACTYPE', 'INVTYPE', 'INVAGE', 'VEHTYPE']]
print(new_df)
```

```
      TRAFFCTL  VISIBILITY  LIGHT  RDSFCOND  DRIVCOND  ACCLASS  IMPACTYPE  \
0            1           1      3         2         2        1          2
1            2           1      3         1         3        1          1
2            2           1      1         1         3        1          1
3            1           1      1         2         3        1          2
4            1           1      3         1         3        1          1
...        ...         ...    ...       ...       ...      ...        ...
5294         1           2      2         2         1        0          2
5295         2           1      1         1         1        0          1
5296         2           1      2         1         2        0          2
5297         2           2      2         2         1        0          1
5298         1           2      3         2         1        0          1

      INVTYPE  INVAGE  VEHTYPE
0           1       5        1
1           4       2       10
2           4       5       10
3           3       2       10
4           4       5       10
...       ...     ...      ...
5294        1       4        1
5295        1       5        1
5296        1       5        1
5297        1       5        1
5298        1       5        1
```

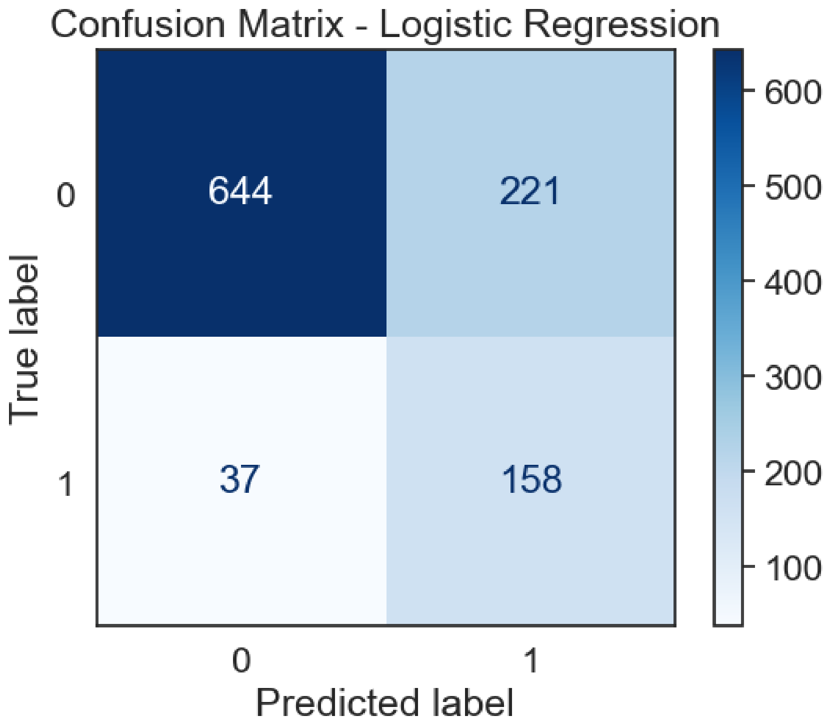
- For example, bin the 8 clarity values into just 3 distinct buckets

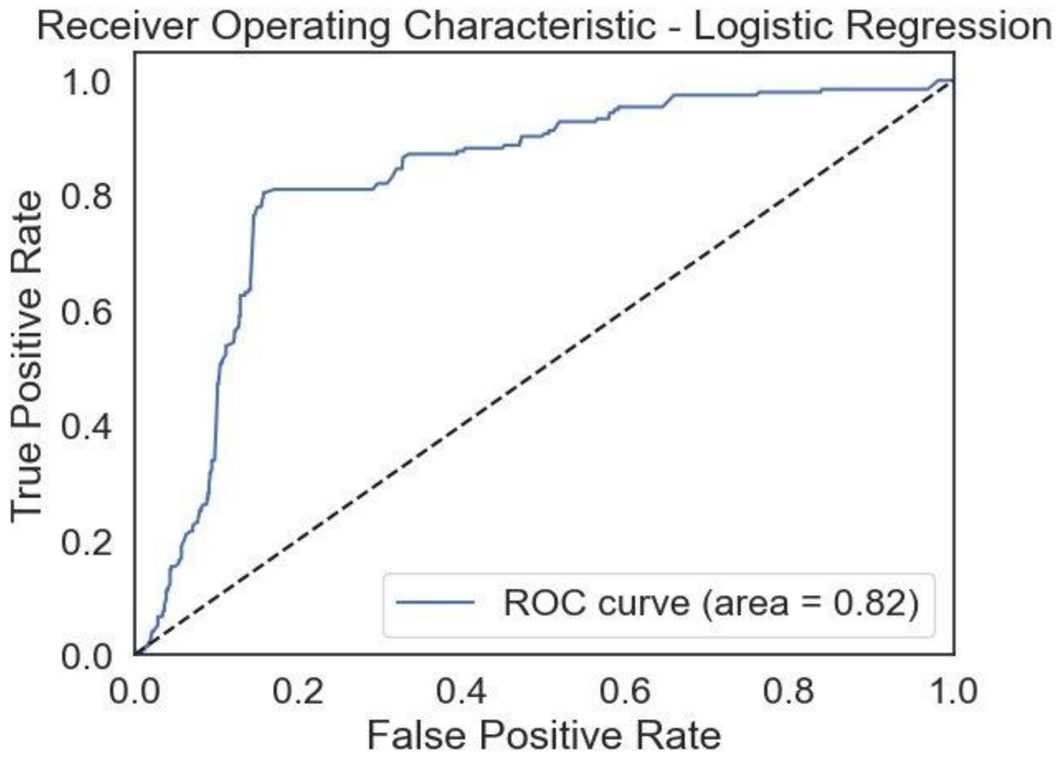
- Adding **dummy variables** for each categorical column can **lead to wide data sets and increase model variance**

- **Data binning** can solve this problem

- In general, we want data to be long rather than wide (**many rows, few columns**)

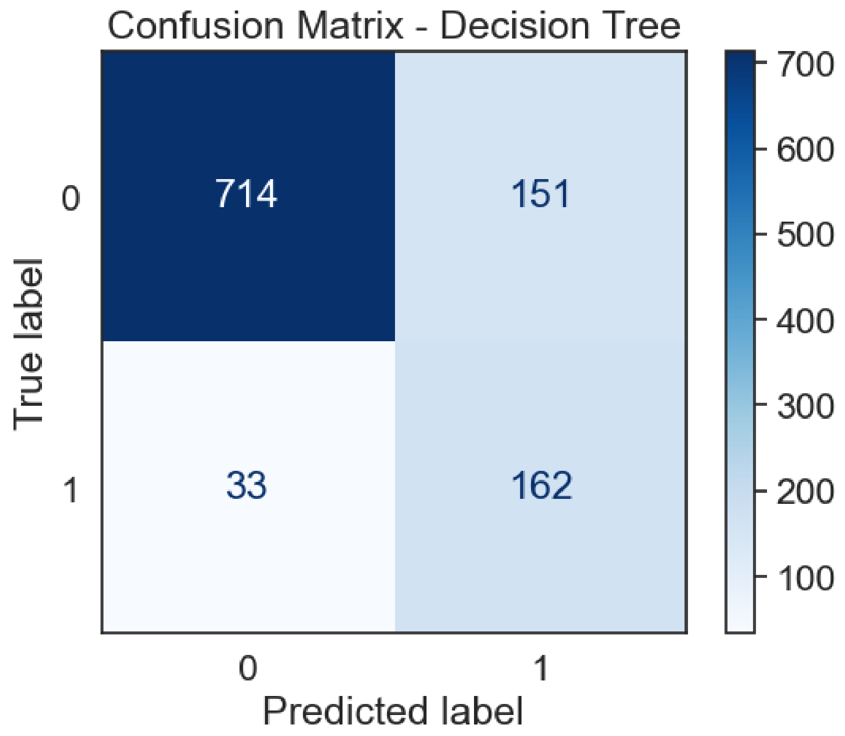
3 Logistic Regression: after Randomized Search

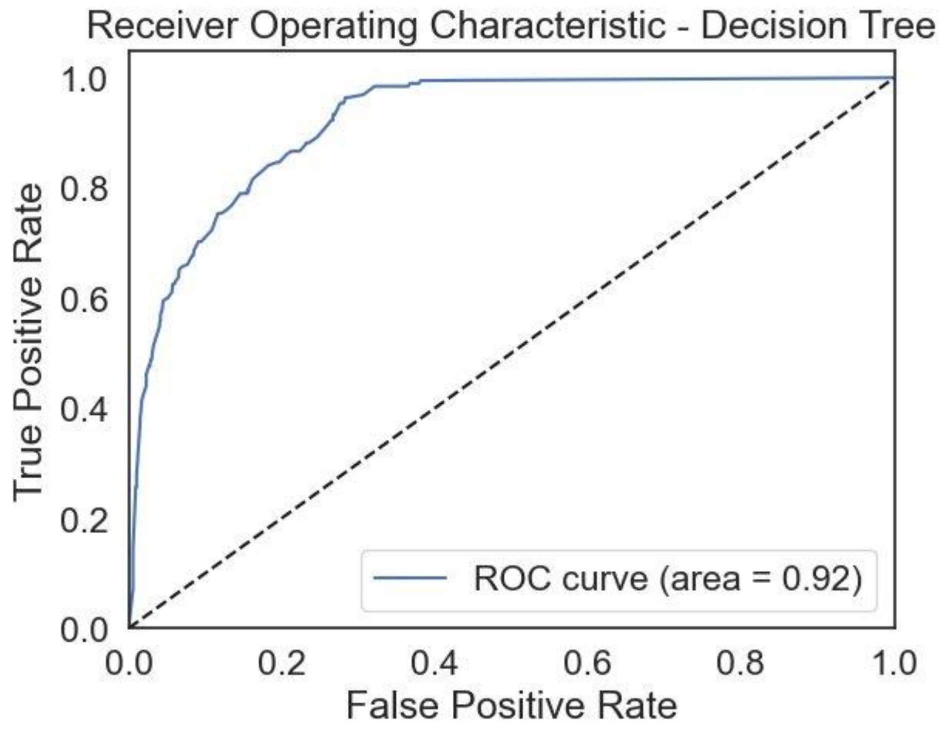




Model	Accuracy	Precision	Recall	F1-Score	AUC Score	Cross-validation Score	Best Parameters
Logistic Regression	0.7566	0.42	0.81	0.55	0.8246	0.795 (+/- 0.012)	{'C': 0.10778765841014329, 'penalty': 'l2'}
Decision Tree	0.8264	0.52	0.83	0.64	0.9203	0.869 (+/- 0.012)	{'max_depth': 17, 'min_samples_leaf': 7, 'min_samples_split': 8}
Random Forest	0.8255	0.52	0.85	0.64	0.9266	0.874 (+/- 0.012)	{'max_depth': 13, 'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 63}
Support Vector Machine	0.8142	0.5	0.88	0.63	0.8975	0.879 (+/- 0.015)	{'C': 3.845401188473625, 'gamma': 0.09607143064099162}
XGBoost	0.8274	0.52	0.83	0.64	0.9236	0.875 (+/- 0.012)	{'learning_rate': 0.06396921323890797, 'max_depth': 9, 'n_estimators': 173}

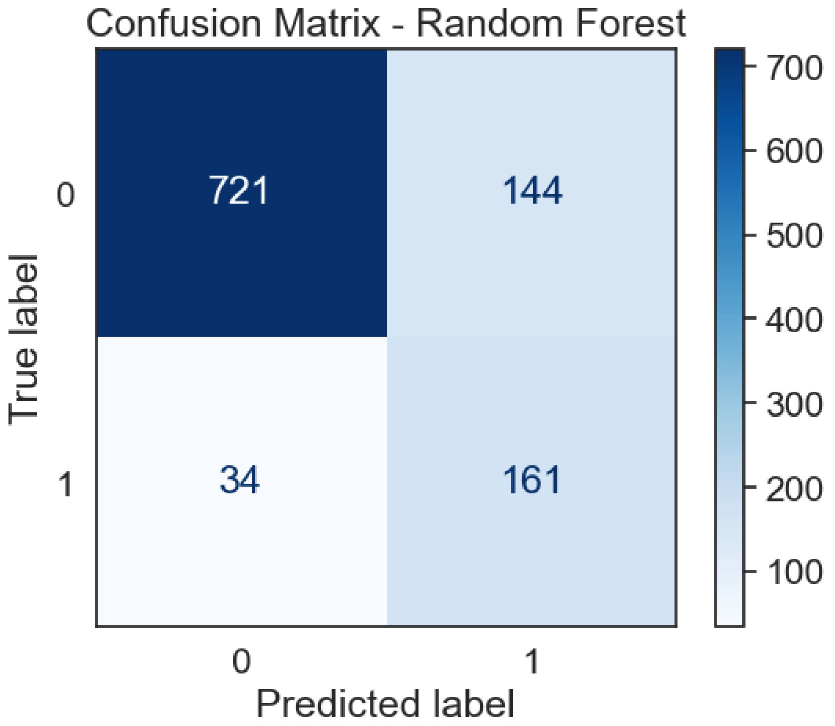
3 Decision Tree: after Randomized Search

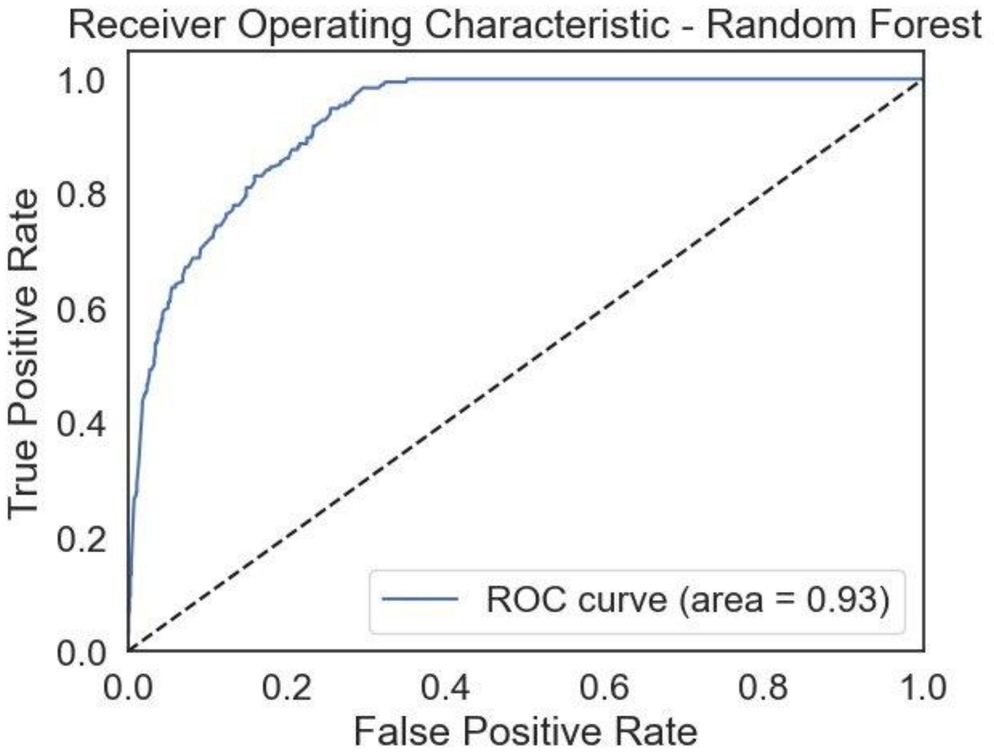




Model	Accuracy	Precision	Recall	F1-Score	AUC Score	Cross-validation Score	Best Parameters
Logistic Regression	0.7566	0.42	0.81	0.55	0.8246	0.795 (+/- 0.012)	{'C': 0.10778765841014329, 'penalty': 'l2'}
Decision Tree	0.8264	0.52	0.83	0.64	0.9203	0.869 (+/- 0.012)	{'max_depth': 17, 'min_samples_leaf': 7, 'min_samples_split': 8}
Random Forest	0.8255	0.52	0.85	0.64	0.9266	0.874 (+/- 0.012)	{'max_depth': 13, 'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 63}
Support Vector Machine	0.8142	0.5	0.88	0.63	0.8975	0.879 (+/- 0.015)	{'C': 3.845401188473625, 'gamma': 0.09607143064099162}
XGBoost	0.8274	0.52	0.83	0.64	0.9236	0.875 (+/- 0.012)	{'learning_rate': 0.06396921323890797, 'max_depth': 9, 'n_estimators': 173}

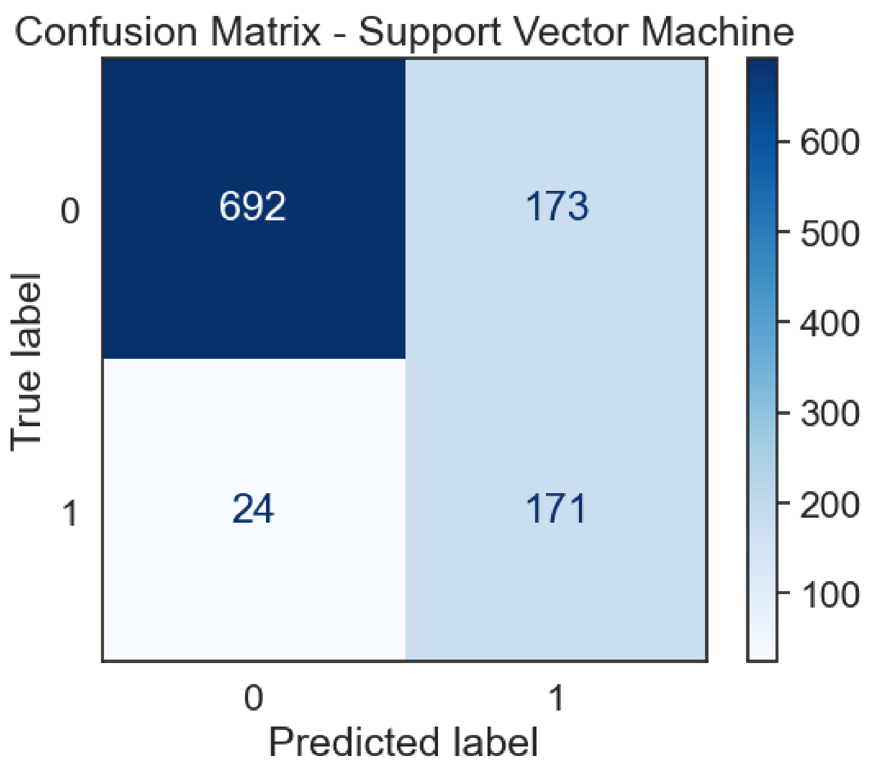
3 Random Forest: after Randomized Search

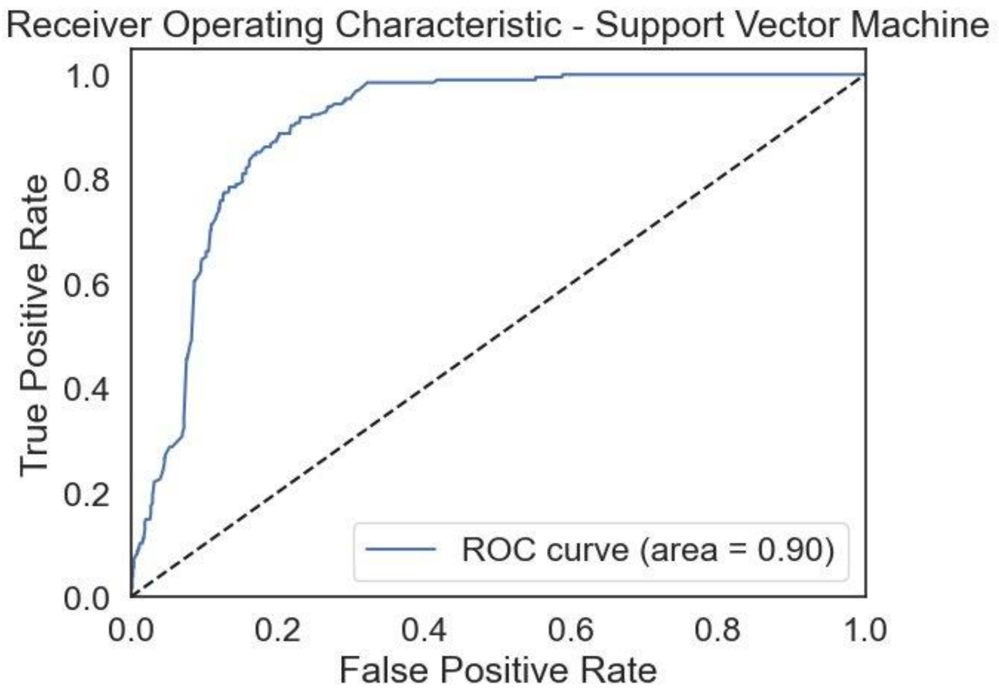




Model	Accuracy	Precision	Recall	F1-Score	AUC Score	Cross-validation Score	Best Parameters
Logistic Regression	0.7566	0.42	0.81	0.55	0.8246	0.795 (+/- 0.012)	{'C': 0.10778765841014329, 'penalty': 'l2'}
Decision Tree	0.8264	0.52	0.83	0.64	0.9203	0.869 (+/- 0.012)	{'max_depth': 17, 'min_samples_leaf': 7, 'min_samples_split': 8}
Random Forest	0.8255	0.52	0.85	0.64	0.9266	0.874 (+/- 0.012)	{'max_depth': 13, 'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 63}
Support Vector Machine	0.8142	0.5	0.88	0.63	0.8975	0.879 (+/- 0.015)	{'C': 3.845401188473625, 'gamma': 0.09607143064099162}
XGBoost	0.8274	0.52	0.83	0.64	0.9236	0.875 (+/- 0.012)	{'learning_rate': 0.06396921323890797, 'max_depth': 9, 'n_estimators': 173}

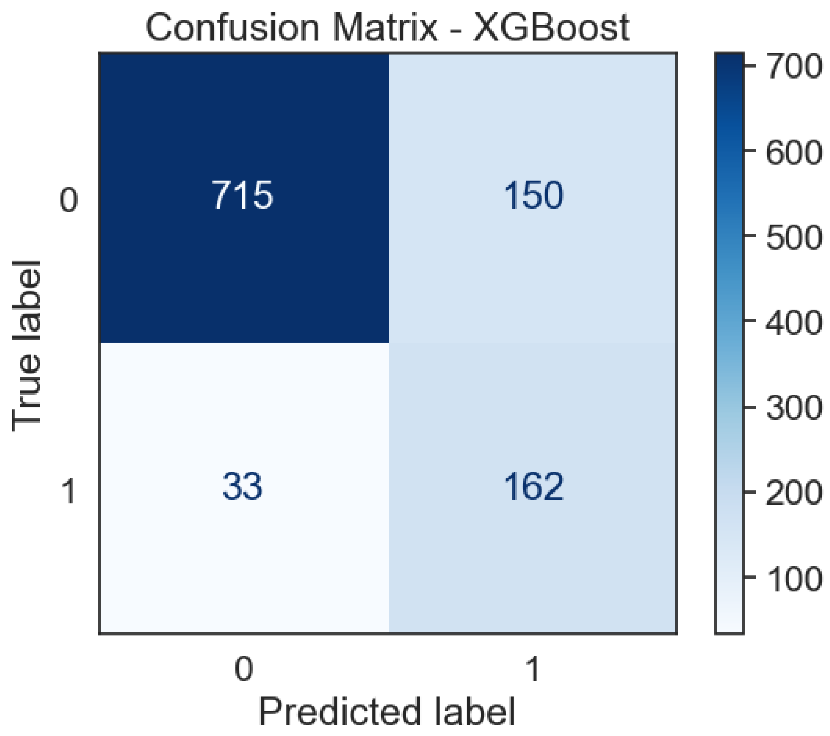
3 SVM: after Randomized Search

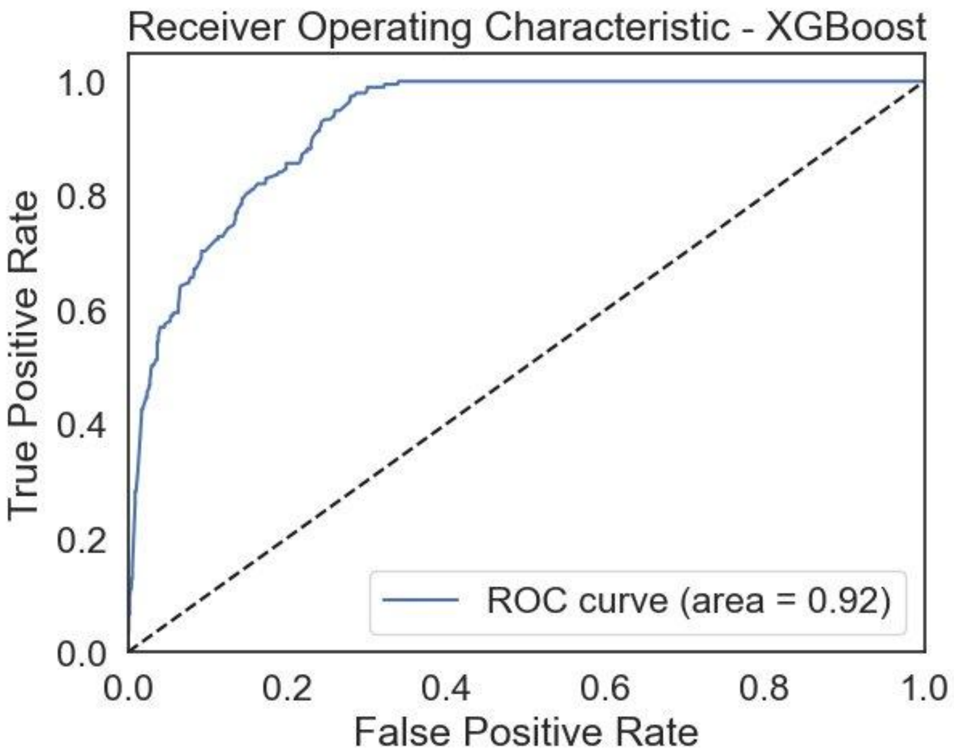




Model	Accuracy	Precision	Recall	F1-Score	AUC Score	Cross-validation Score	Best Parameters
Logistic Regression	0.7566	0.42	0.81	0.55	0.8246	0.795 (+/- 0.012)	{'C': 0.10778765841014329, 'penalty': 'l2'}
Decision Tree	0.8264	0.52	0.83	0.64	0.9203	0.869 (+/- 0.012)	{'max_depth': 17, 'min_samples_leaf': 7, 'min_samples_split': 8}
Random Forest	0.8255	0.52	0.85	0.64	0.9266	0.874 (+/- 0.012)	{'max_depth': 13, 'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 63}
Support Vector Machine	0.8142	0.5	0.88	0.63	0.8975	0.879 (+/- 0.015)	{'C': 3.845401188473625, 'gamma': 0.09607143064099162}
XGBoost	0.8274	0.52	0.83	0.64	0.9236	0.875 (+/- 0.012)	{'learning_rate': 0.06396921323890797, 'max_depth': 9, 'n_estimators': 173}

3 XGBoost: after Randomized Search



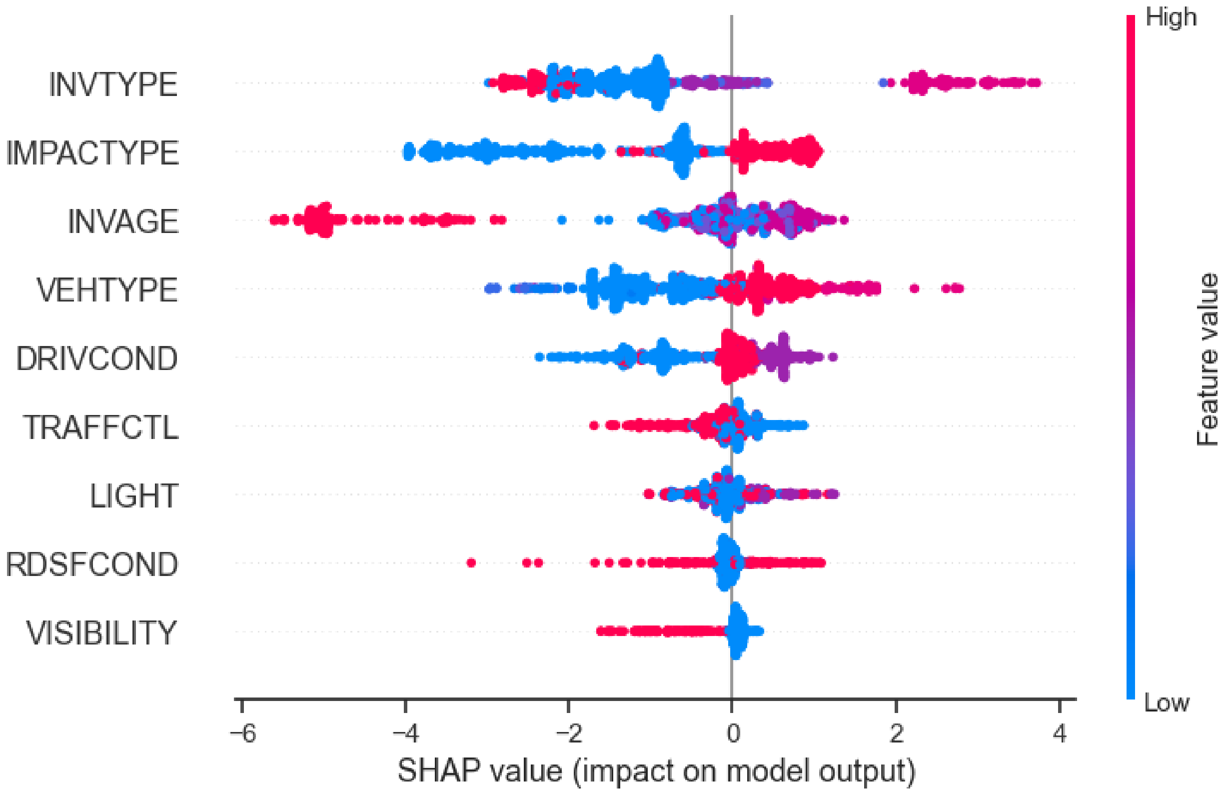


Model	Accuracy	Precision	Recall	F1-Score	AUC Score	Cross-validation Score	Best Parameters
Logistic Regression	0.7566	0.42	0.81	0.55	0.8246	0.795 (+/- 0.012)	{'C': 0.10778765841014329, 'penalty': 'l2'}
Decision Tree	0.8264	0.52	0.83	0.64	0.9203	0.869 (+/- 0.012)	{'max_depth': 17, 'min_samples_leaf': 7, 'min_samples_split': 8}
Random Forest	0.8255	0.52	0.85	0.64	0.9266	0.874 (+/- 0.012)	{'max_depth': 13, 'min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 63}
Support Vector Machine	0.8142	0.5	0.88	0.63	0.8975	0.879 (+/- 0.015)	{'C': 3.845401188473625, 'gamma': 0.09607143064099162}
XGBoost	0.8274	0.52	0.83	0.64	0.9236	0.875 (+/- 0.012)	{'learning_rate': 0.06396921323890797, 'max_depth': 9, 'n_estimators': 173}

4 SHAP: Feature Importance Visualization

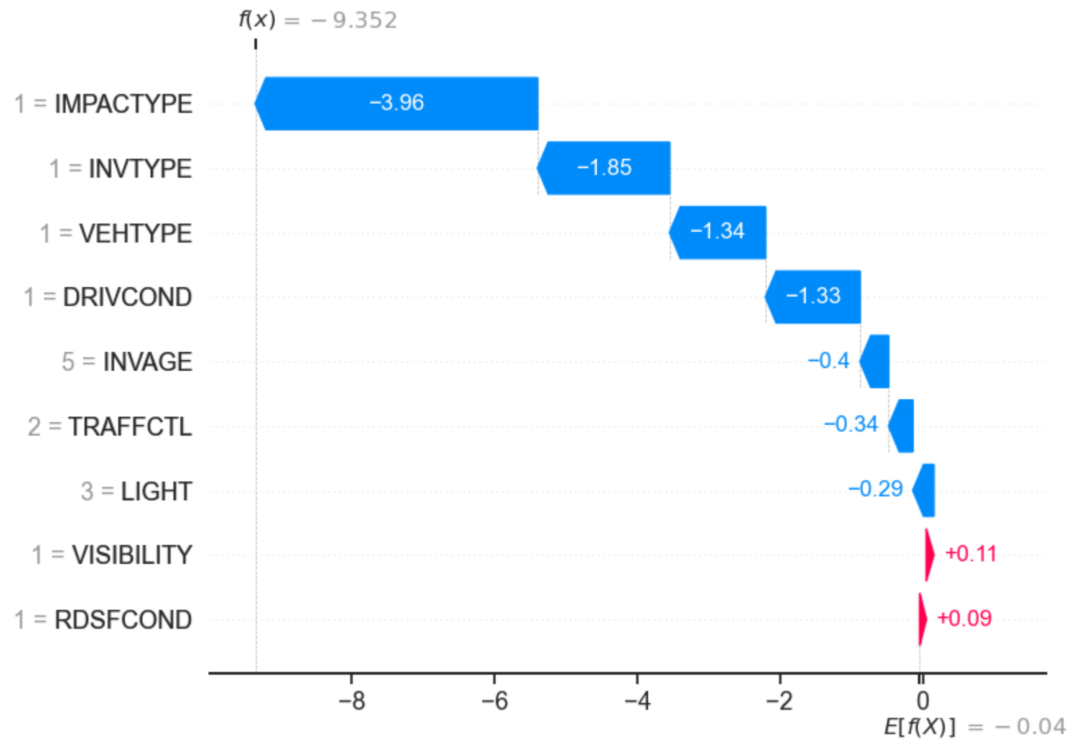
One reason we use GBMs(Gradient Boosted Machines):
Random Forests work best when the goal is prediction performance for our result, but they are not ideal **if we want to understand how features impact the target.**

SHAP beeswarm plot



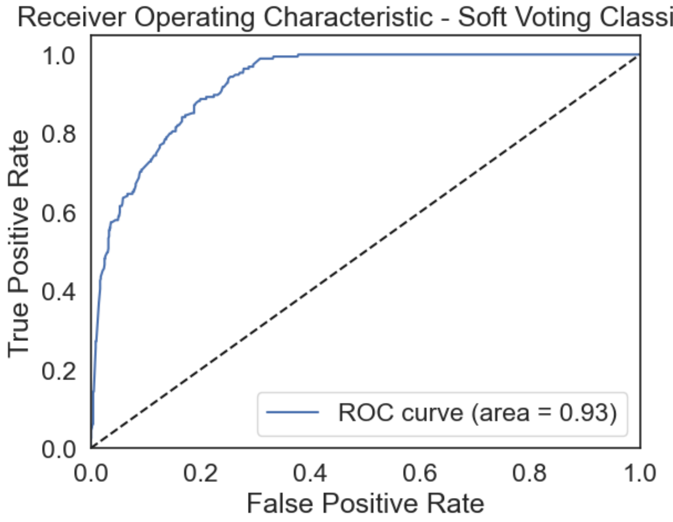
SHAP waterfall plot

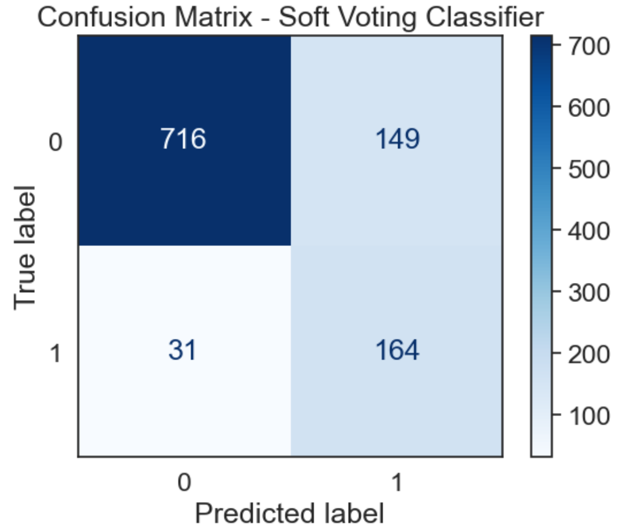
```python
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[1])
```



4 Result: Ensemble with Soft Voting

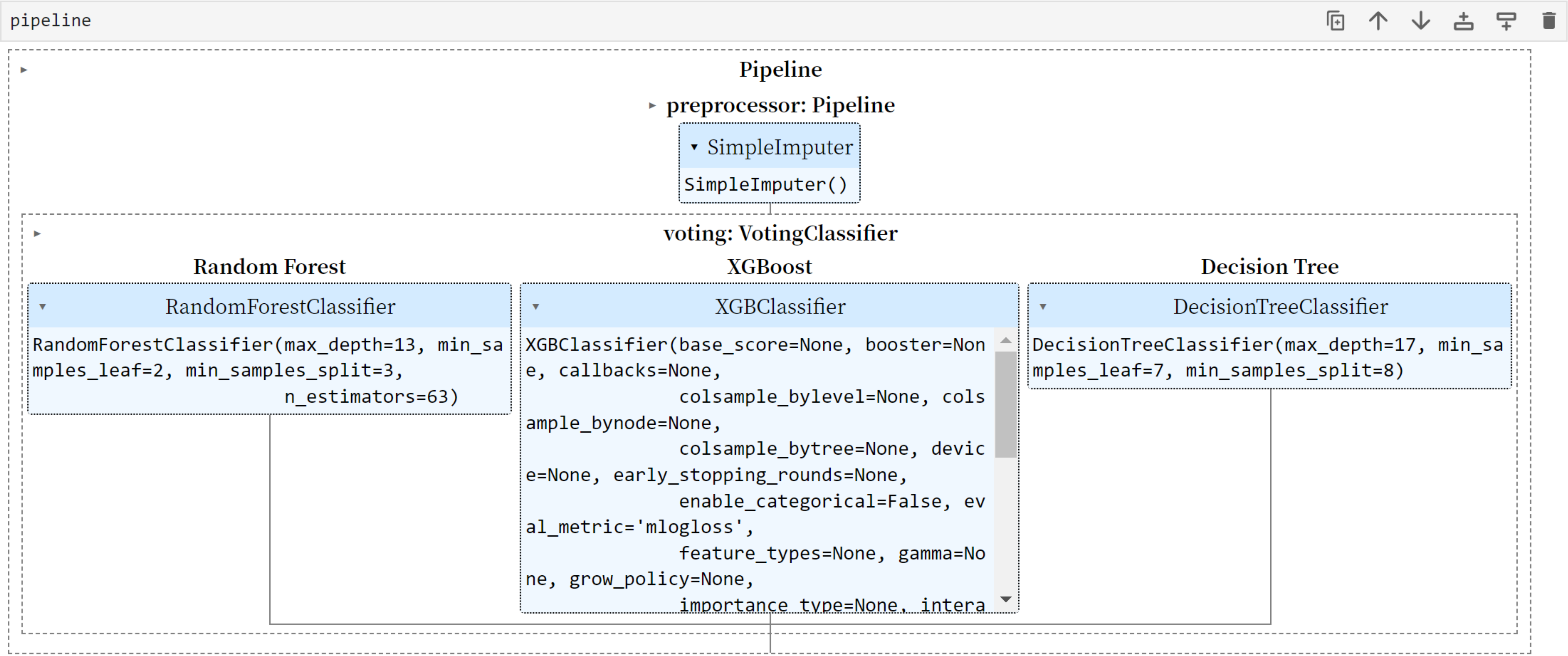
```python
from sklearn.ensemble import VotingClassifier

# Create and train Soft Voting Classifier with Decision Tree included
soft_voting_model = VotingClassifier(estimators=[
    ('Random Forest', RandomForestClassifier(
        n_estimators=63,
        max_depth=13,
        min_samples_leaf=2,
        min_samples_split=3
    )),
    ('XGBoost', xgb.XGBClassifier(
        use_label_encoder=False,
        eval_metric='mlogloss',
        learning_rate=0.035877998160001694,
        max_depth=9,
        n_estimators=181
    )),
    ('Decision Tree', DecisionTreeClassifier(
        max_depth=17,
        min_samples_leaf=7,
        min_samples_split=8
    ))
], voting='soft')
```





Model	Accuracy	Precision	Recall	F1-Score	AUC Score
Ensemble with softing voting	0.8302	0.52	0.84	0.65	0.926

5 Model Deployment: Pipeline



5 Model Deployment: Plotly and Dash

5 Model Deployment: Video Demo or Live Demo

Predict fatality in incidents

Traffic Control Type:

No Control

Vehicle Type:

Small Vehicles

Driver Condition:

Normal

Involved Person Age Group:

Infants and Young Children (0 to 9)

Visibility:

Clear

Light Condition:

Artificial Light

Road Surface Condition:

Wet

Involved Person Type:

Pedestrians

Impact Type:

Vehicle-to-Vehicle Collisions

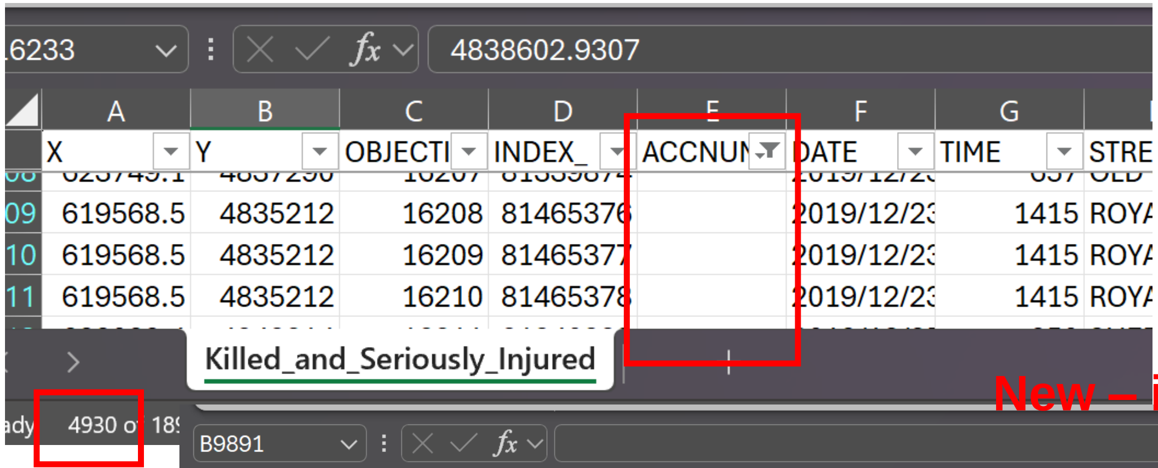
Submit

The probability of a fatality in this incident is 0.4

6 Conclusion: limitation and Improvement

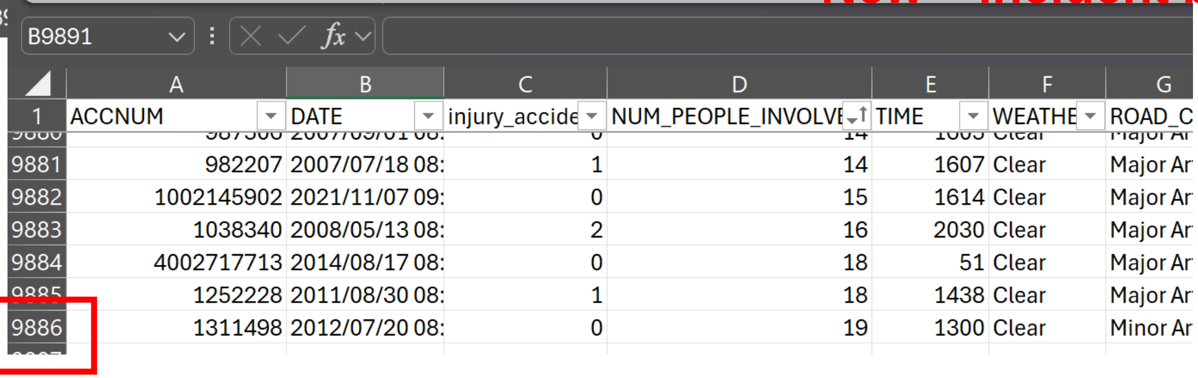
- **Transitioning from person-based data to incident-based data**
- **ACCNUM – fill Null value**

Original – person based

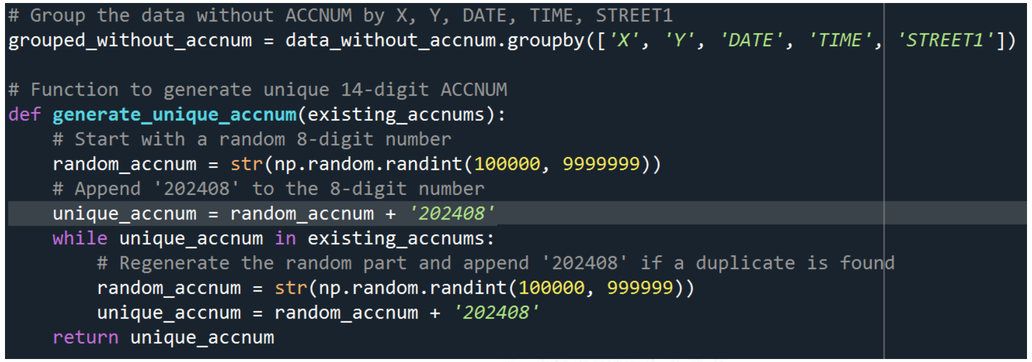


X	Y	OBJECTI	INDEX_	ACCNUM	DATE	TIME	STRE
619568.5	4835212	16208	81465376		2019/12/23	1415	ROYA
619568.5	4835212	16209	81465377		2019/12/23	1415	ROYA
619568.5	4835212	16210	81465378		2019/12/23	1415	ROYA

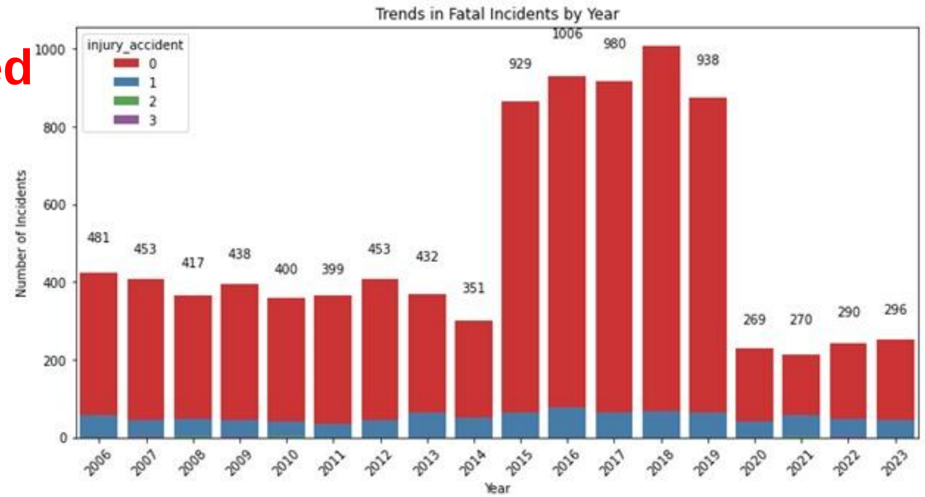
New – incident based



	ACCNUM	DATE	injury_accide	NUM_PEOPLE_INVOLVE	TIME	WEATHE	ROAD_C
9881	982207	2007/07/18 08:	1	14	1607	Clear	Major Ar
9882	1002145902	2021/11/07 09:	0	15	1614	Clear	Major Ar
9883	1038340	2008/05/13 08:	2	16	2030	Clear	Major Ar
9884	4002717713	2014/08/17 08:	0	18	51	Clear	Major Ar
9885	1252228	2011/08/30 08:	1	18	1438	Clear	Major Ar
9886	1311498	2012/07/20 08:	0	19	1300	Clear	Minor Ar



```python
# Group the data without ACCNUM by X, Y, DATE, TIME, STREET1
grouped_without_accnum = data_without_accnum.groupby(['X', 'Y', 'DATE', 'TIME', 'STREET1'])

# Function to generate unique 14-digit ACCNUM
def generate_unique_accnum(existing_accnums):
    # Start with a random 8-digit number
    random_accnum = str(np.random.randint(100000, 9999999))
    # Append '202408' to the 8-digit number
    unique_accnum = random_accnum + '202408'
    while unique_accnum in existing_accnums:
        # Regenerate the random part and append '202408' if a duplicate is found
        random_accnum = str(np.random.randint(100000, 999999))
        unique_accnum = random_accnum + '202408'
    return unique_accnum
```



- **Setting the Fatal Criteria: ACCLASS & INJURY => ACCLASS**

Thank you!