UK Car Accident Classifier

Jason Han
Dana You
Jason Dai
Abdumijit Dolkun

Table of Contents

Project Overview

Goal: use of supervised learning to predict and prevent traffic accidents based on certain patterns identified in the environment, driver, and vehicle

Why Machine Learning?

At a glance, our guiding process:

- Search for dataset
- Clean each subset of data and select columns
- Create a combined dataframe
- Feature engineering
- Experiment with different classifier methods
- Optimization

All About the Dataset

Background

- Kaggle: https://www.kaggle.com/datasets/benoit72/uk-accidents-10-years-history-with-many-variables
- “UK Accidents over a 10 year period with multiple variables”
  - From 2005-2014
- Stats relate only to personal injury accidents that are:
  - 1. On public roads
  - 2. Reported to the British police
  - 3. Recorded using the STATS19 accident reporting form

Structure

- DF #1: Accidents
  - Details about circumstances of accidents from
- DF #2: Vehicles
  - Details about the vehicle(s) involved in recorded accidents
- DF #3: Casualties
  - Details about those injured or harmed in the collision
- xls File: Lookup Tables
  - Information about each variable in the 3 dataframes

Lookup Tables

Feature Examples
Sex of Driver
Weather Conditions
Casualty Severity

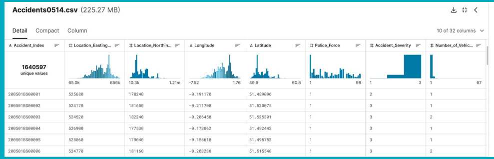
code	label
1	Male
2	Female
3	Not known
-1	Data missing or out of range

code	label
1	Fatal
2	Serious
3	Slight

code	label
1	Fine no high winds
2	Raining no high winds
3	Snowing no high winds
4	Fine + high winds
5	Raining + high winds
6	Snowing + high winds
7	Fog or mist
8	Other
9	Unknown
-1	Data missing or out of range

Process of Merging

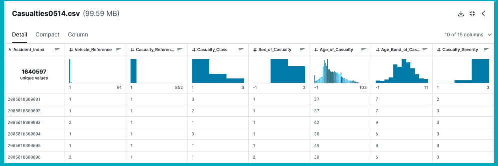
STEP 1



Accidents0514.csv (225.27 MB)

Detail Compact Column — 10 of 32 columns

Accident_Index	Location_Easting...	Location_Northin...	Longitude	Latitude	Police_Force	Accident_Severity	Number_of_Vehic...
1640597 unique values	65.0k – 656k	10.3k – 1.21m	-7.52 – 1.76	49.9 – 60.8	1 – 98	1 – 3	1 – 67
200501BS00001	525680	178240	-0.191170	51.489096	1	2	1
200501BS00002	524170	181650	-0.211708	51.520075	1	3	1
200501BS00003	524520	182240	-0.206458	51.525301	1	3	2
200501BS00004	526900	177530	-0.173862	51.482442	1	3	1
200501BS00005	528060	179040	-0.156618	51.495752	1	3	1
200501BS00006	524770	181160	-0.203238	51.515540	1	3	2



Casualties0514.csv (99.59 MB)

Detail Compact Column — 10 of 15 columns

Accident_Index	Vehicle_Reference	Casualty_Referen...	Casualty_Class	Sex_of_Casualty	Age_of_Casualty	Age_Band_of_Cas...	Casualty_Severity
1640597 unique values	1 – 91	1 – 852	1 – 3	-1 – 2	-1 – 103	-1 – 11	1 – 3
200501BS00001	1	1	3	1	37	7	2
200501BS00002	1	1	2	1	37	7	3
200501BS00003	2	1	1	1	62	9	3
200501BS00004	1	1	3	1	30	6	3
200501BS00005	1	1	1	1	49	8	3
200501BS00006	2	1	1	2	30	6	3

Vehicles0514.csv (189.65 MB)

Detail Compact Column — 10 of 22 columns

Accident_Index	Vehicle_Reference	Vehicle_Type	Towing_and_Arti...	Vehicle_Manoeuvre	Vehicle_Location...	Junction_Location	Skidding_and_Ov...
1640597 unique values	1 – 91	-1 – 98	-1 – 5	-1 – 18	-1 – 9	-1 – 8	-1 – 5
200501BS00001	1	9	0	18	0	0	0
200501BS00002	1	11	0	4	0	3	0
200501BS00003	1	11	0	17	0	0	0
200501BS00003	2	9	0	2	0	0	0
200501BS00004	1	9	0	18	0	0	0
200501BS00005	1	3	0	18	0	0	1

Three Separate Dataframes

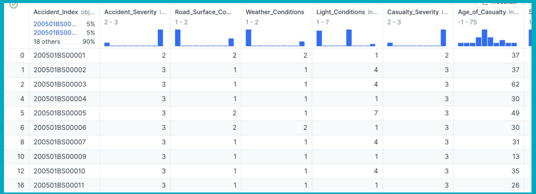
Accident, Casualty, Vehicle

STEP 2

```python
merged_df = pd.merge(df_accident_reduced, df_casualty_reduced, on='Accident_Index', how='inner')
merged_df = pd.merge(merged_df, df_vehicle_reduced, on='Accident_Index', how='inner')
```

Merge Function

STEP 3



	Accident_Index obj...	Accident_Severity i...	Road_Surface_Co...	Weather_Conditions	Light_Conditions in...	Casualty_Severity i...	Age_of_Casualty in...
	200501BS00... 5%; 200501BS00... 5%; 18 others 90%	2 - 3	1 - 2	1 - 2	1 - 7	2 - 3	-1 - 75
0	200501BS00001	2	2	2	1	2	37
1	200501BS00002	3	1	1	4	3	37
2	200501BS00003	3	1	1	4	3	62
4	200501BS00004	3	1	1	1	3	30
5	200501BS00005	3	2	1	7	3	49
6	200501BS00006	3	2	2	1	3	30
8	200501BS00007	3	1	1	4	3	31
10	200501BS00009	3	1	1	1	3	13
12	200501BS00010	3	1	1	4	3	35
16	200501BS00011	3	1	1	1	3	26

One Combined Dataframe

Duplicate indices removed

Key Stakeholders

01

LAW ENFORCEMENT

Identify accident-prone areas for preventative action

02

INSURANCE COMPANIES

Assess risk more accurately to incentivize safer driving

03

TRANSPORTATION SAFETY

Improve road infrastructure and traffic safety

04

CITY PLANNING

Generate more efficient road and city plans

Identifying Categories and Features

All Possible Features

Breakdown

Environment

Includes features concerning: weather, road, & light conditions

Vehicle

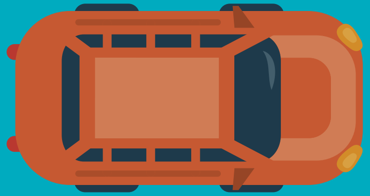
Includes features concerning: engine capacity & age of vehicle

Driver

Includes features concerning: driver sex & age

Feature Engineering in Road Accident Analysis

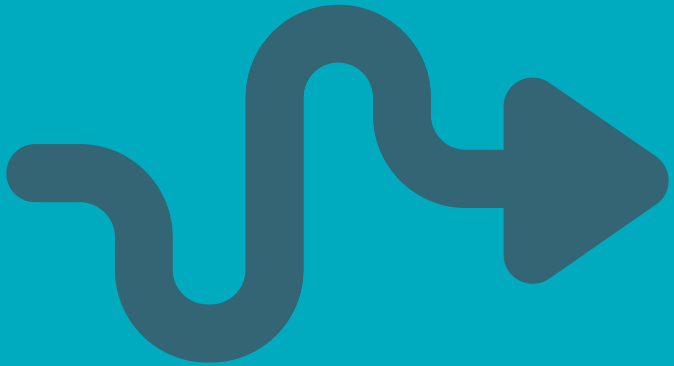
STEP 1



Objective

Enhancing data quality and predictive power.

STEP 2



Process

Cleaning and transforming raw data into meaningful features

STEP 3

Significance

Improving model accuracy and interpretability.

Cleaning and Encoding Categorical Variables

01 **Focused on three key features:**

Road_Surface_Conditions, Weather_Conditions, Light_Conditions.

02 **Invalid entries removal**

(-1 and 9) considered as data entry errors.

03 **One-hot encoding**

Applied one-hot encoding to transform these categorical variables into a format suitable for modeling.

03 **Summary**

Preserves the informational value of categorical variables while making them usable for the ML models.

Code

```python
# Road surface conditions cleaning
rd_null = merged_df['Road_Surface_Conditions'].isnull().sum()
print(rd_null)

cleaned_df = merged_df.copy()
n = -1
cleaned_df = cleaned_df[cleaned_df['Road_Surface_Conditions'] != n]

# Weather conditions cleaning (removing -1 and 9)
w_null = merged_df['Weather_Conditions'].isnull().sum()
print(w_null)
m = 9
cleaned_df = cleaned_df[cleaned_df['Weather_Conditions'] != m]
cleaned_df = cleaned_df[cleaned_df['Weather_Conditions'] != n]

# Light conditions cleaning
l_null = merged_df['Light_Conditions'].isnull().sum()
print(l_null)
cleaned_df = cleaned_df[cleaned_df['Light_Conditions'] != n]

# All 3 are categorical and they have no nulls or -1
# One hot encoding for all
cleaned_df = pd.get_dummies(cleaned_df, columns=['Road_Surface_Conditions','Weather_Conditions','Light_Conditions'])
cleaned_df
```

Feature Transformation - Age and Sex

'Age_of_Driver'

- Converted into age groups to capture generational effects.
- Removal of -1 values to maintain data integrity.

'Sex_of_Driver'

- Encoded as binary variables to simplify the model's understanding.
- Removal of (-1 and 3) values to maintain data integrity.

- **Grouping Ages:** Categorizing ages reveals how different age groups exhibit distinct driving behaviors and accident risks.
- **Encoding Gender:** Binary gender encoding aids in analyzing the impact of gender on driving habits and accident rates.

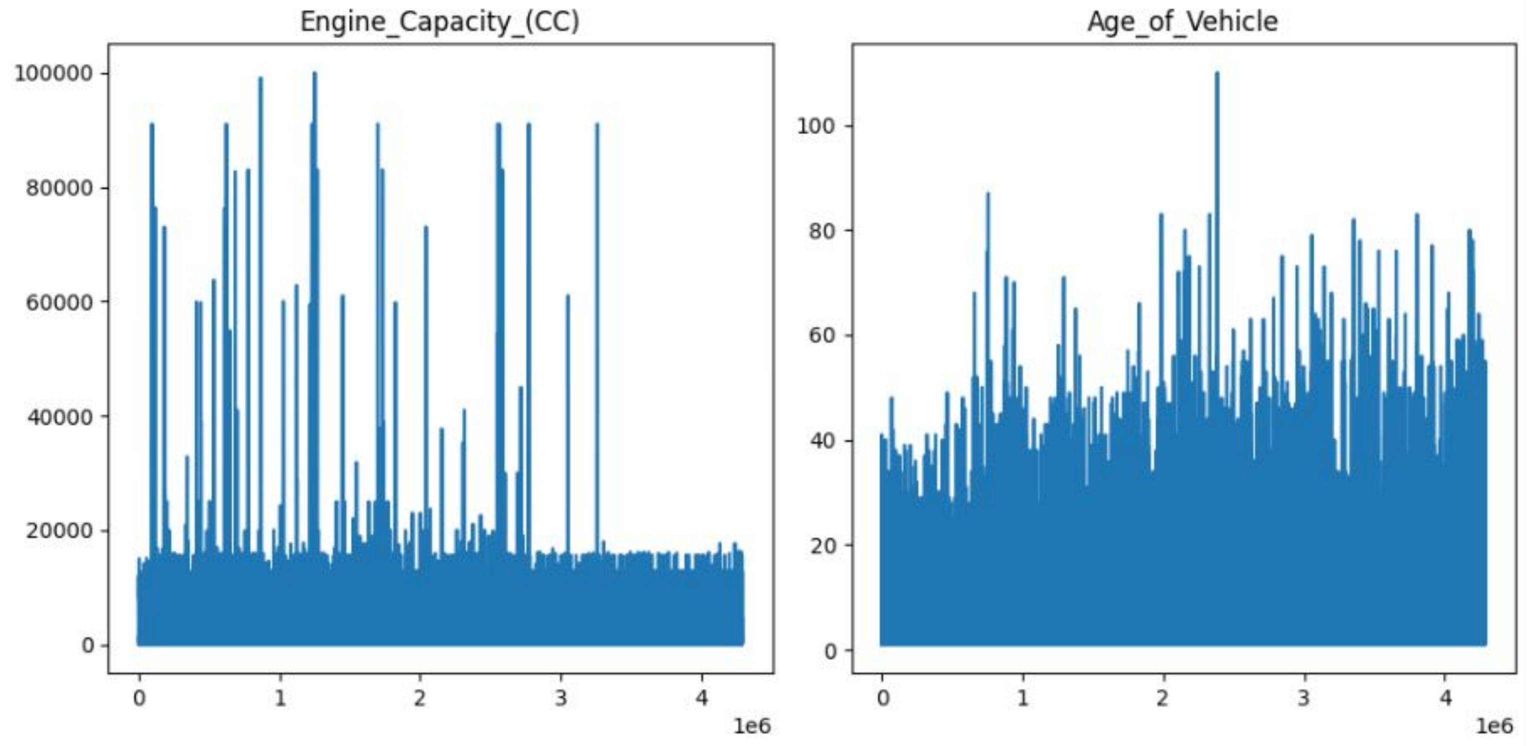
Code

```python
# Age_of_Driver Cleaning
# Check for -1 values and then remove them
age_drv_null = cleaned_df['Age_of_Driver'].isnull().sum()
age_drv_minus_one = (cleaned_df['Age_of_Driver'] == -1).sum()
print('Age_of_Driver - Null:', age_drv_null)
print('Age_of_Driver - -1:', age_drv_minus_one)

cleaned_df = cleaned_df[cleaned_df['Age_of_Driver'] != -1]

#one-hot encoding age_of_driver
bins=[0, 12, 18, 35, 50, float('inf')]
labels = ['1', '2', '3', '4', '5']
cleaned_df['Driver_Age_Group'] = pd.cut(cleaned_df['Age_of_Driver'],  bins=bins, labels=labels, include_lowest=True, r

# Sex_of_Driver Cleaning
# Check for -1 values and then remove them
sex_drv_null = cleaned_df['Sex_of_Driver'].isnull().sum()
sex_drv_minus_one = (cleaned_df['Sex_of_Driver'] == -1).sum()
cleaned_df = cleaned_df[cleaned_df['Sex_of_Driver'] != 3]
print('Sex_of_Driver - Null:', sex_drv_null)
print('Sex_of_Driver - -1:', sex_drv_minus_one)

cleaned_df = cleaned_df[cleaned_df['Sex_of_Driver'] != -1]

# Assuming 'Sex_of_Driver' is a categorical variable that should be one-hot encoded
cleaned_df = pd.get_dummies(cleaned_df, columns=['Sex_of_Driver'])
cleaned_df = cleaned_df.drop(columns=['Age_of_Driver'])
cleaned_df
```

Handling Vehicle and Casualty Features

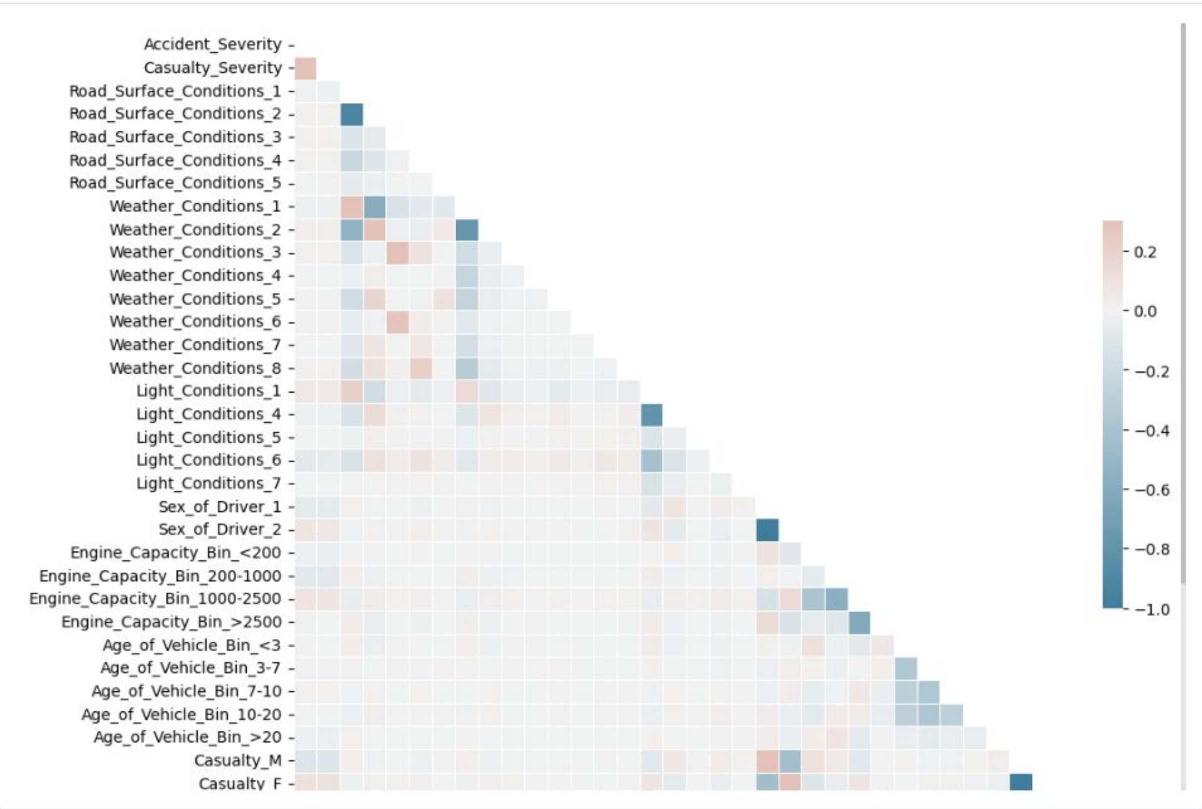
Binning and Encoding Vehicle & Casualty Data		
01	‘Engine_Capacity_(CC)’ and ‘Age_of_Vehicle’	Binned into categories based on capacity and age for better model interpretation.
02	‘Age_of_Casualty’	Grouped into age brackets, similar to ‘Age_of_Driver’
03	‘Sex_of_Casualty’	One-hot encoded to align with other sex-related features.
04	Summary	Binning continuous variables like engine capacity and vehicle age simplifies analysis by grouping data into interpretable categories, enhancing insights into trends and patterns.

Visual



Engine_Capacity_(CC)
100000
80000
60000
40000
20000
0
0
1
2
3
4
1e6
Age_of_Vehicle
100
80
60
40
20
0
0
1
2
3
4
1e6

Correlation Graph



Data Modeling

Steps of Data Modeling

01 — Splitting Data — Training & Test

02 — Model Methods — Explore Modeling

03 — Optimization — Finalize Model

Split & Start

```python
# Splitting dataset into train and test split
X = cleaned_df.drop(['Accident_Severity', 'Casualty_Severity'], axis=1)
y = cleaned_df['Accident_Severity']

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

K-Nearest-Neighbors

- Instance base
- Handles numerical & categorical
- Classifies based on similarity
- CV: 82.84%
- Test: 83.16%

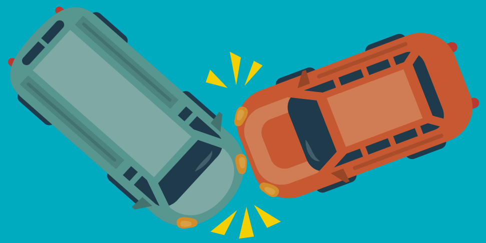
```python
k_fold = KFold(n_splits=10, shuffle=True, random_state=42)

knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier(n_neighbors=5))
])
knn_pipeline.fit(X_train, y_train)

knn_cv_scores = cross_val_score(knn_pipeline, X_train, y_train, cv=k_fold, scoring='accuracy')
avg_cv_accuracy = np.mean(knn_cv_scores)

y_pred = knn_pipeline.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

print(f"Average KNN Cross-Validation Accuracy: {avg_cv_accuracy * 100:.2f}%")
print(f"KNN Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Average KNN Cross-Validation Accuracy: 82.84%
KNN Test Accuracy: 83.16%
```

Decision Tree & Random Forest

```python
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

dt_cv_scores = cross_val_score(decision_tree, X_train, y_train, cv=k_fold, scoring='accuracy')
avg_cv_accuracy = np.mean(dt_cv_scores)

y_pred = decision_tree.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

print(f"Average Decisoin Tree Cross-Validation Accuracy: {avg_cv_accuracy * 100:.2f}%")
print(f"Decisoin Tree Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Average Decisoin Tree Cross-Validation Accuracy: 84.48%
Decisoin Tree Test Accuracy: 84.42%
```



- Easy to interpret
- Handles high-dimensional data
- Model complex interactions between features
- CV: 84.48% & 84.53%
- Test: 84.42% & 84.45%

```python
rforest = RandomForestClassifier(n_estimators=100, random_state=42)
rforest.fit(X_train, y_train)

rf_cv_scores = cross_val_score(rforest, X_train, y_train, cv=k_fold, scoring='accuracy')
avg_cv_accuracy = np.mean(rf_cv_scores)

y_pred = rforest.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

print(f"Average Random Forest Cross-Validation Accuracy: {avg_cv_accuracy * 100:.2f}%")
print(f"Random Forest Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Average Random Forest Cross-Validation Accuracy: 84.53%
Random Forest Test Accuracy: 84.45%
```

Logistic Regression

- Good for binary & multiclass
- Handles numerical & categorical features
- Easy Interpretation
- CV: 84.56%
- Tets: 84.49%

Logistic Regression

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

lreg = LogisticRegression(max_iter=1000)
lreg.fit(X_train_scaled, y_train)

lreg_cv_scores = cross_val_score(lreg, X_train_scaled, y_train, cv=k_fold, scoring='accuracy')
avg_cv_accuracy = np.mean(lreg_cv_scores)

y_pred = lreg.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred)

print(f"Average Logistic Regression Cross-Validation Accuracy: {avg_cv_accuracy * 100:.2f}%")
print(f"Logistic Regression Test Accuracy: {test_accuracy * 100:.2f}%")
```

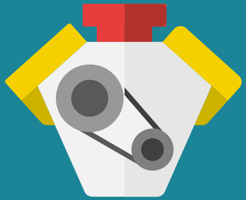
```
Average Logistic Regression Cross-Validation Accuracy: 84.56%
Logistic Regression Test Accuracy: 84.49%
```

Neural Network

- Captures Complex Patterns
- Model non-linear relationships
- Needs preprocessing
- Tuning hyperparameters
- CV: 84.56%
- Test: 84.49%

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

nn = MLPClassifier(
    hidden_layer_sizes=(30),
    activation='logistic',
    solver='lbfgs',
    random_state=42
)
nn.fit(X_train_scaled, y_train)

nn_cv_scores = cross_val_score(nn, X_train_scaled, y_train, cv=k_fold, scoring='accuracy', error_score='raise')
avg_cv_accuracy = np.mean(nn_cv_scores)

y_pred = nn.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred)

print(f"Average Neural Networks Cross-Validation Accuracy: {avg_cv_accuracy * 100:.2f}%")
print(f"Neural Networks Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Average Neural Networks Cross-Validation Accuracy: 84.56%
Neural Networks Test Accuracy: 84.49%
```



Ensemble - Best of both worlds

Log-Reg
Random Forest

CV: 84.53%
Test: 84.49%

```
ensemble = VotingClassifier(
    estimators=[
        ('lreg', lreg),
        ('rf', rforest)
        ],
    voting='hard')
ensemble.fit(X_train, y_train)

ensemble_cv_scores = cross_val_score(ensemble, X_train, y_train, cv=k_fold, scoring='accuracy')
avg_cv_accuracy = np.mean(ensemble_cv_scores)

y_pred = nn.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

print(f"Average Ensemble Cross-Validation Accuracy: {avg_cv_accuracy * 100:.2f}%")
print(f"Ensemble Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Average Ensemble Cross-Validation Accuracy: 84.53%
Ensemble Test Accuracy: 84.49%
```

Data Results

A | **Logistic Regression**

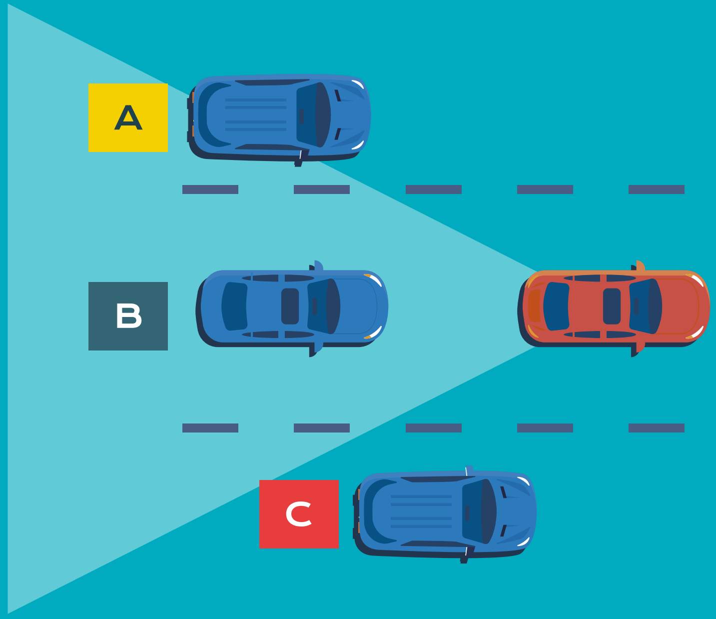
Displayed an accuracy of 84.56%

B | **K-Nearest Neighbors**

Displayed an accuracy of 83.75%

C | **Ensemble Methods**

Displayed an accuracy of 84.52%

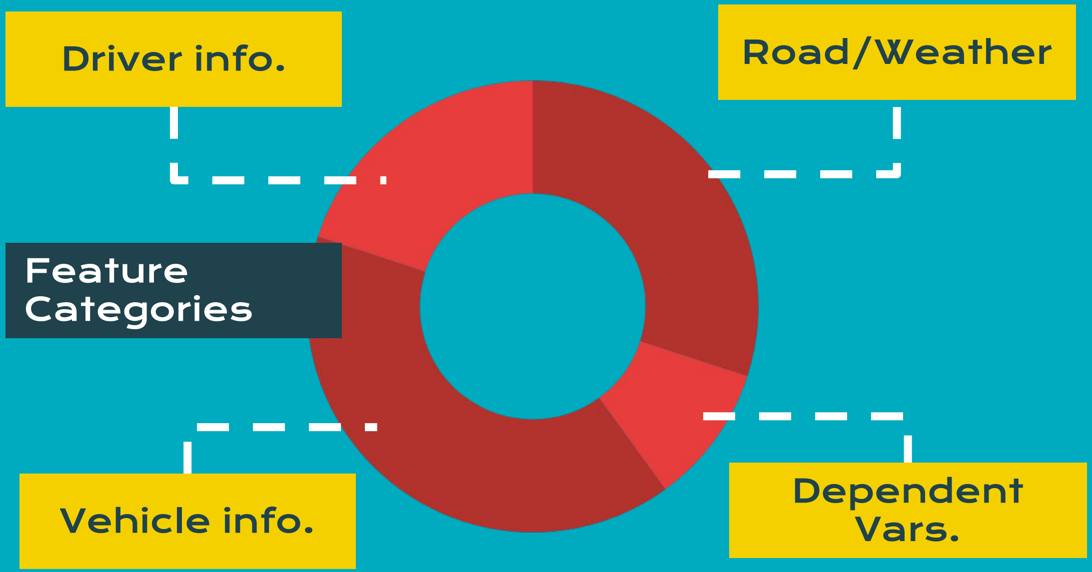


Data Results (cont.)

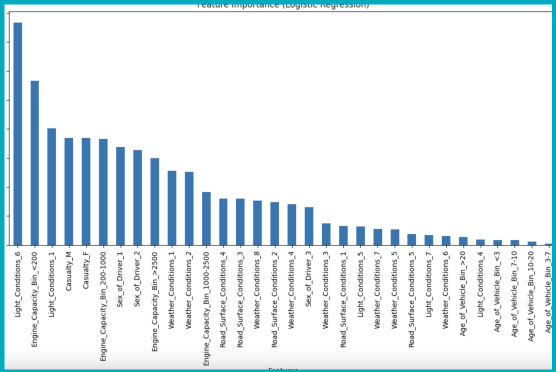
- By selecting specific categories of features, we aimed to narrow down main impacting factors

- We noticed that the light condition feature had the highest feature coefficient when examining by significance

- Combining the coefficients in their categories, the Driver info. Category had the highest significance



Conclusions



THEORY

Driver information likely indicates a greater correlation with accidents over other categories

DATA

Though light conditions has the highest single influence, driver info has an overall higher combined coefficient

IMPLICATIONS

It's likely that a combination of the weather and driver have the greatest correlation

NOTE

There are still many ways to calculate significance and impact

Real World Application and Readiness

Accuracy

Accuracy is not high enough for accident prediction yet

Category Selection

Category structure requires a wider range of features for each

Dependency Analysis

Current classifier doesn't illustrate the impact of dependant features

Though the model sheds light on factors that cause accidents, it is not yet ready for real world application

Possible improvements and recommendations

Data	Classifier	Cleaning	Analysis
Selective but denser feature categories	Use of model and combination	Handle feature specific nulls more accurately	Incorporate further metrics to analyze influence
More intensive dataset	Implementation of further optimizations	Incorporate error analysis	Contextualize dependent features

Thank you!

Any questions?