This is to predict accident outcomes (injury vs no injury) using classification algorithms. In this analysis, I will use different algorithms including Logistic Regression, Decision Tree, Random Forest and K Nearest Neighbor to model the data and access the performances.

Part 1. Initial Data Inspection

```
#part1
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score
df = pd.read_csv('/Users/yizhen/Desktop/6105hw/assignment3/accidents-Assignment_3.csv')
df.shape
df.head()
df.dtypes
```

In the data inspection part, we load the data and check the data frame shape, display the first few rows and check the data types. In this data frame, we have 600 rows and 11 columns.

```
In [155]: df.shape
Out[155]: (600, 11)

In [156]: df.head()
Out[156]:
   RushHour  WRK_ZONE  WKDY  ...  TRAF_two_way  WEATHER_adverse     MAX_SEV
0       1.0       0.0   1.0  ...           0.0                1  no-injury
1       1.0       0.0   1.0  ...           1.0                0     injury
2       1.0       0.0   NaN  ...           0.0                1  no-injury
3       1.0       0.0   1.0  ...           0.0                1  no-injury
4       1.0       0.0   1.0  ...           0.0                1     injury

[5 rows x 11 columns]

In [157]: df.dtypes
Out[157]:
RushHour           float64
WRK_ZONE           float64
WKDY               float64
INT_HWY            float64
LGTCON_day           int64
LEVEL              float64
SPD_LIM            float64
SUR_COND_dry       float64
TRAF_two_way       float64
WEATHER_adverse      int64
MAX_SEV             object
dtype: object
```

The descriptive statistics and correlation matrics are as below. I added the mode number of all the variables into the matrix also.

```
#descriptive value + matrix
summary_stats = df.describe()
mode_data = df.mode().iloc[0]
summary_stats.loc['mode'] = mode_data
```

Index	RushHour	WRK_ZONE	WKDY	INT_HWY	LGTCON_day	LEVEL	SPD_LIM	SUR_COND_dry	TRAF_two_way	WEATHER_adverse
count	578	599	599	599	600	599	599	598	599	600
mean	0.458478	0.0116861	0.796327	0.146912	0.73	0.230384	43.2638	0.784281	0.582638	0.161667
std	0.498704	0.107559	0.403065	0.495923	0.44433	0.421431	12.2607	0.411665	0.493536	0.368452
min	0	0	0	0	0	0	10	0	0	0
25%	0	0	1	0	0	0	35	1	0	0
50%	0	0	1	0	1	0	40	1	1	0
75%	1	0	1	0	1	0	55	1	1	0
max	1	1	1	9	1	1	75	1	1	1
mode	0	0	1	0	1	0	35	1	1	0

Part 2.Data Preprocessing

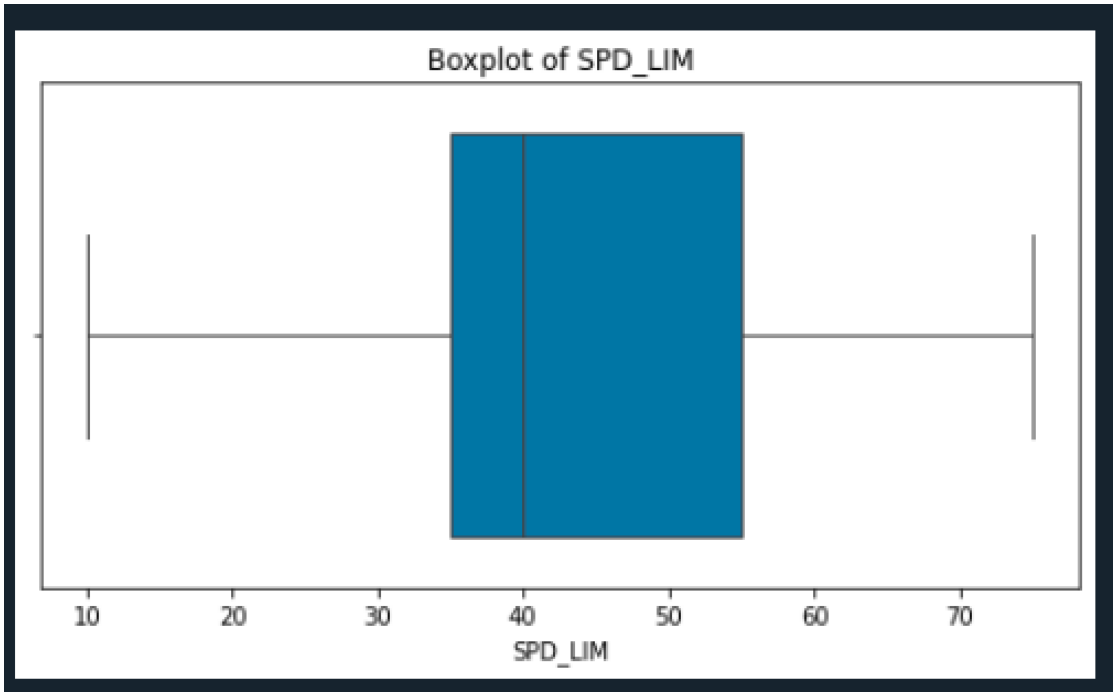
a. Data Processing

```
print(df.isnull().sum())
# use mode value for na value
for column in df.columns:
    if df[column].isna().sum() > 0:
        if column == 'SPD_LIM':
            median_value = df[column].median()
            df[column].fillna(median_value, inplace=True)
        else:
            mode_value = df[column].mode()[0]
            df[column].fillna(mode_value, inplace=True)
# %matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

outlier_cols = ['SPD_LIM']
for col in outlier_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()

#No Outlier
X = df.drop(columns='MAX_SEV')
Y = df['MAX_SEV']
Y = pd.get_dummies(Y, drop_first=True, dtype=int)
```

In the Data processing part, I check whether the dataset have null value first.

```
In [161]: print(df.isnull().sum())
RushHour            22
WRK_ZONE             1
WKDY                 1
INT_HWY              1
LGTCON_day           0
LEVEL                1
SPD_LIM              1
SUR_COND_dry         2
TRAF_two_way         1
WEATHER_adverse      0
MAX_SEV              0
dtype: int64
```

There are some missing values as shown above, I observed that only SPD_LIM is numeric value, so I decided to use the median value to replace the null value and use mode value to replace the others. As there is only one numeric column, I only plot the box plot of SPD_LIM. There is no outlier detected by using the box plot.



Regarding to the categorical variables, only MAX_SEV is a categorical variable, I only need to convert this into numeric format.

Index	no-injury
0	1
1	0
2	1
3	1
4	0
5	0
6	1
7	0
8	1
9	0
10	1
11	0

b. Data Partitioning

I split the data into 3 parts, 70% for training, 15% for validation and 15% for testing.

```
from sklearn.model_selection import train_test_split


X_train, X_temp, Y_train, Y_temp = train_test_split(X, Y, test_size=0.30, random_state=1)
X_validation, X_test, Y_validation, Y_test = train_test_split(X_temp, Y_temp, test_size=0.50, random_state=1)
```

```
In [165]: print("Training set size:", X_train.shape)
     ...: print("Validation set size:", X_validation.shape)
     ...: print("Test set size:", X_test.shape)
Training set size: (420, 10)
Validation set size: (90, 10)
Test set size: (90, 10)
```

Part3. Model Building

In the model building, I selected all the predictors including rush hour, work zone, intersection on a highway, lighting condition, level of accident, speed limit, surface condition, traffic lane, and weather condition. I think all of them are possible indicators for accidents.

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
logit_reg = LogisticRegression(penalty="l2", C=1e42, solver='liblinear')
logit_reg.fit(X_train, Y_train)
print('intercept ', logit_reg.intercept_[0])
print(pd.DataFrame({'coeff': logit_reg.coef_[0]},  index=X.columns).transpose())

coeff = pd.DataFrame({'coeff': logit_reg.coef_[0]},  index=X.columns).transpose()

import statsmodels.api as sm

# Add a constant to the features for the intercept term
X_train_sm = sm.add_constant(X_train)

# Fit the model using statsmodels to get p-values
logit_model = sm.Logit(Y_train, X_train_sm)
result = logit_model.fit()

# Print the summary which includes p-values
print(result.summary())
```

```
         Iterations 5
                          Logit Regression Results
==============================================================================
Dep. Variable:              no-injury   No. Observations:                  420
Model:                          Logit   Df Residuals:                      409
Method:                           MLE   Df Model:                           10
Date:                Tue, 15 Oct 2024   Pseudo R-squ.:                0.01965
Time:                        22:28:15   Log-Likelihood:               -284.84
converged:                       True   LL-Null:                      -290.55
Covariance Type:            nonrobust   LLR p-value:                   0.3258
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1599      0.696      0.230      0.818      -1.204       1.523
RushHour      -0.0083      0.206     -0.040      0.968      -0.412       0.395
WRK_ZONE       1.1081      1.169      0.948      0.343      -1.183       3.400
WKDY           0.4461      0.248      1.797      0.072      -0.040       0.933
INT_HWY        0.3002      0.378      0.793      0.428      -0.441       1.042
LGTCON_day     0.1259      0.238      0.529      0.597      -0.340       0.592
LEVEL         -0.0754      0.237     -0.318      0.750      -0.539       0.389
SPD_LIM       -0.0037      0.010     -0.356      0.722      -0.024       0.017
SUR_COND_dry  -0.6378      0.431     -1.481      0.139      -1.482       0.206
TRAF_two_way  -0.0976      0.232     -0.421      0.673      -0.552       0.357
WEATHER_adverse -0.1187    0.469     -0.253      0.800      -1.037       0.800
==============================================================================
```

We can see from the summary, the Pseudo R-square value is super low as 1.97% which indicates that the logistic regression model cannot describe this problem. The P value of all the indicators are quite high which also shows that logistic regression is not suitable for this problem.

Decision Tree:

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier,  DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV

param_grid = {
   'max_depth': [10, 20, 30, 40],
   'min_samples_split': [20, 40, 60, 80, 100],
   'min_impurity_decrease': [0, 0.0005, 0.001, 0.005, 0.01],
}


# Which values are best?
# n_jobs=-1 will utilize all available CPUs
#cv means cross validation
gridSearch = GridSearchCV(DecisionTreeClassifier(random_state=1),
    param_grid, cv=5, n_jobs=-1)
gridSearch.fit(X_train, Y_train)
print('Initial score: ', gridSearch.best_score_)
print('Initial parameters: ', gridSearch.best_params_)

#to score base on f1
# gridSearch = GridSearchCV(DecisionTreeClassifier(random_state=1),
#                           param_grid,
#                           cv=5,
#                           scoring='f1',
#                           n_jobs=-1)
# gridSearch.fit(X_train, Y_train)

# print('Best score: ', gridSearch.best_score_)
# print('Best parameters: ', gridSearch.best_params_)

#decision tree
param_grid = {
'max_depth': list(range(2, 16)), # 14 values
'min_samples_split': list(range(35, 48)),
'min_impurity_decrease': [0, 0.001, 0.0011], # 3 values
}
gridSearch = GridSearchCV(DecisionTreeClassifier(random_state=1),
    param_grid, cv=5, n_jobs=-1)
gridSearch.fit(X_train, Y_train)
print('Improved score: ', gridSearch.best_score_)
print('Improved parameters: ', gridSearch.best_params_)
bestClassTree = gridSearch.best_estimator_
```

In the Decision Tree model, I used grid search to tune the model parameters. Grid search is based on cross validations, by using it I tune the parameter values of max depth, minimum samples split and minimum impurity decrease of decision trees. I found that below parameter value will give a higher accuracy.

```
Improved score:  0.5619047619047619
Improved parameters:  {'max_depth': 10, 'min_impurity_decrease': 0, 'min_samples_split':
41}
```

Random Forest:

```
#random forest
param_grid_rf = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
}


grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=1), param_grid_rf, cv=5, n_jobs=-1)
grid_search_rf.fit(X_train, Y_train)

print('Random Forest Initial Score:', grid_search_rf.best_score_)
print('Random Forest Initial Parameters:', grid_search_rf.best_params_)


param_grid_rf = {
    'n_estimators': [100, 130, 160, 180],
    'max_depth': list(range(8, 15)),
    'min_samples_split': list(range(8, 14)),
}


grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=1), param_grid_rf, cv=5, n_jobs=-1)
grid_search_rf.fit(X_train, Y_train)

print('Random Forest Best Score:', grid_search_rf.best_score_)
print('Random Forest Best Parameters:', grid_search_rf.best_params_)
```

In the Random Forest model, I also use grid search to tune the value of tree numbers, max depth and minimum samples split. The parameter values are as below.

```
Random Forest Best Score: 0.5428571428571429
Random Forest Best Parameters: {'max_depth': 8, 'min_samples_split': 13, 'n_estimators':
100}
```

K Nearest Neighbors

```
#KNN
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
scaler = StandardScaler()
train_X_scaled = scaler.fit_transform(X_train)
valid_X_scaled = scaler.transform(X_validation)
test_X_scaled = scaler.transform(X_test)

results = []
for k in range(1, 15):
   knn = KNeighborsClassifier(n_neighbors=k).fit(train_X_scaled,Y_train)
   results.append({
        'k': k,
        'accuracy': accuracy_score(Y_validation,
             knn.predict(valid_X_scaled))
})
print(results)
#k = 7, with highest score 0.6111
```

In KNN model, I scaled the training data set to ensure same unit impact of each variable. To tune the nearest neighbor count, I let it running for all 1 to 15 and calculate the accuracy. After calculation, I found when k = 7, we have the highest accuracy.

Inde	Type	Size	Value
0	dict	2	{'k':1, 'accuracy':0.5666666666666667}
1	dict	2	{'k':2, 'accuracy':0.5555555555555556}
2	dict	2	{'k':3, 'accuracy':0.5222222222222223}
3	dict	2	{'k':4, 'accuracy':0.5555555555555556}
4	dict	2	{'k':5, 'accuracy':0.5222222222222223}
5	dict	2	{'k':6, 'accuracy':0.5333333333333333}
6	dict	2	{'k':7, 'accuracy':0.6111111111111112}
7	dict	2	{'k':8, 'accuracy':0.5444444444444444}
8	dict	2	{'k':9, 'accuracy':0.5111111111111111}
9	dict	2	{'k':10, 'accuracy':0.5222222222222223}
10	dict	2	{'k':11, 'accuracy':0.5333333333333333}
11	dict	2	{'k':12, 'accuracy':0.5777777777777777}
12	dict	2	{'k':13, 'accuracy':0.5555555555555556}

Part 4. Model Diagnostics

In the model diagnostics part, I calculated the accuracy, confusion matrix, precision, sensitivity and specificity value of all the models.

```python
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

def evaluate_model(model, X, Y, dataset_name):
    Y_pred = model.predict(X)
    cm = confusion_matrix(Y, Y_pred)
    accuracy = accuracy_score(Y, Y_pred)
    precision = precision_score(Y, Y_pred)
    recall = recall_score(Y, Y_pred)
    specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])

    print(f"=== {dataset_name} Metrics ===")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall (Sensitivity): {recall:.2f}")
    print(f"Specificity: {specificity:.2f}")
    print("\n")

# 在验证集和测试集上评估每个模型
print("Logistic Regression:")
evaluate_model(logit_reg, X_validation, Y_validation, "Validation")

print("Decision Tree:")
evaluate_model(gridSearch.best_estimator_, X_validation, Y_validation, "Validation")

print("Random Forest:")
evaluate_model(grid_search_rf.best_estimator_, X_validation, Y_validation, "Validation")

print("KNN:")
knn_best = KNeighborsClassifier(n_neighbors=7).fit(train_X_scaled, Y_train)
evaluate_model(knn_best, valid_X_scaled, Y_validation, "Validation")
```

```
Logistic Regression:
=== Validation Metrics ===
Confusion Matrix:
[[46  0]
 [44  0]]
Accuracy: 0.51
Precision: 0.00
Recall (Sensitivity): 0.00
Specificity: 1.00


Decision Tree:
=== Validation Metrics ===
Confusion Matrix:
[[32 14]
 [21 23]]
Accuracy: 0.61
Precision: 0.62
Recall (Sensitivity): 0.52
Specificity: 0.70


Random Forest:
=== Validation Metrics ===
Confusion Matrix:
[[32 14]
 [18 26]]
Accuracy: 0.64
Precision: 0.65
Recall (Sensitivity): 0.59
Specificity: 0.70


KNN:
=== Validation Metrics ===
Confusion Matrix:
[[31 15]
 [20 24]]
Accuracy: 0.61
Precision: 0.62
Recall (Sensitivity): 0.55
Specificity: 0.67
```

Logistic Regression:
As per checking the Logistic Regression metrics, it tends to predict all samples as negative and fails to recognize positive cases. This may be due to data imbalance, weak relationships between features and the target variable, or underfitting.

Decision Tree:
Accuracy (0.61): The overall prediction accuracy increased to 61%.
Precision (0.62): Among the samples predicted as positive, 62% are actual positives.
Recall (0.52): 52% of the actual positive cases were correctly predicted.
Specificity (0.70): 70% of the negative cases were correctly predicted.
Conclusion: The decision tree model achieved a relatively balanced performance in recognizing both positive and negative cases, but there is still room for improvement.

Random Forest:
Accuracy (0.64): Accuracy improved to 64%.
Precision (0.65): 65% of the samples predicted as positive are actual positives.
Recall (0.59): 59% of the actual positive cases were correctly predicted.
Specificity (0.70): 70% of the negative cases were correctly predicted.
Conclusion: The random forest model outperforms the decision tree, especially in recall and accuracy, indicating that the ensemble method effectively improves the model's generalization ability.

KNN:
Accuracy (0.61): Accuracy is similar to the decision tree.
Precision (0.62): 62% of the samples predicted as positive are actual positives.
Recall (0.55): 55% of the actual positive cases were correctly predicted.
Specificity (0.67): 67% of the negative cases were correctly predicted.
Conclusion: The KNN model's performance is similar to the decision tree but slightly lower than the random forest.

From the metrics we can conclude that Random Forest performs best, possibly because it integrates multiple decision trees, reducing overfitting and improving generalization. The logistic regression model may not capture the complex relationships in the data and there is underfitting happening.

Multicollinearity

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data["feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(len(X_train.columns))]
print(vif_data)
```

	feature	VIF
0	RushHour	1.904334
1	WRK_ZONE	1.028205
2	WKDY	4.211103
3	INT_HWY	1.851290
4	LGTCON_day	4.034661
5	LEVEL	1.338247
6	SPD_LIM	11.555729
7	SUR_COND_dry	10.574302
8	TRAF_two_way	2.711433
9	WEATHER_adverse	2.877831

Seems there is multicollinearity happening in the dataset. SPD_LIM and SUR_COND_dry VIF value are quite high.

Part5. Model Interpretation

Logistic Regression:

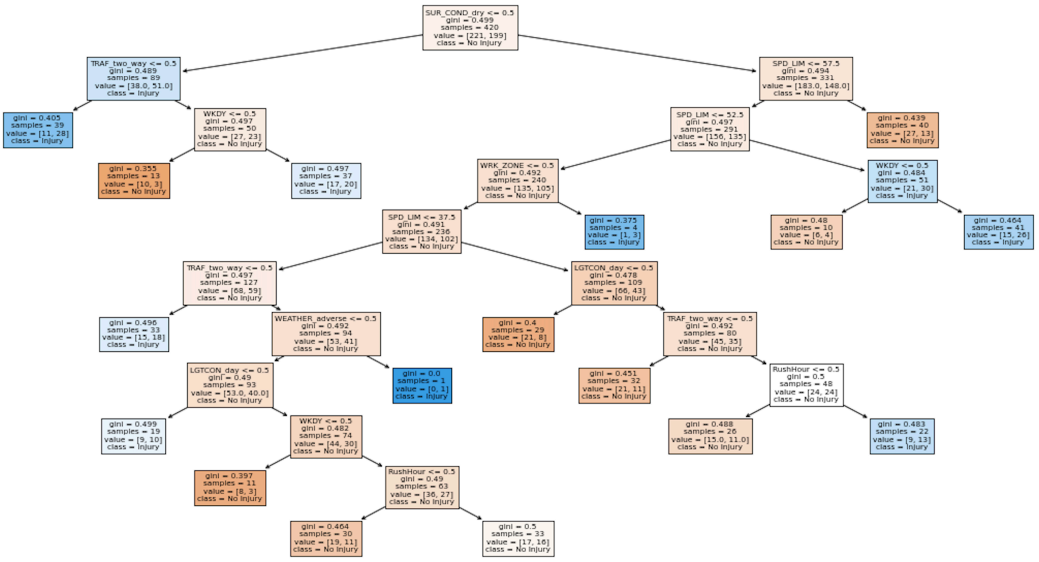
```
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:              no-injury   No. Observations:                  420
Model:                          Logit   Df Residuals:                      409
Method:                           MLE   Df Model:                           10
Date:                Tue, 15 Oct 2024   Pseudo R-squ.:                0.01965
Time:                        22:28:15   Log-Likelihood:                -284.84
converged:                       True   LL-Null:                       -290.55
Covariance Type:            nonrobust   LLR p-value:                    0.3258
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1599      0.696      0.230      0.818      -1.204       1.523
RushHour      -0.0083      0.206     -0.040      0.968      -0.412       0.395
WRK_ZONE       1.1081      1.169      0.948      0.343      -1.183       3.400
WKDY           0.4461      0.248      1.797      0.072      -0.040       0.933
INT_HWY        0.3002      0.378      0.793      0.428      -0.441       1.042
LGTCON_day     0.1259      0.238      0.529      0.597      -0.340       0.592
LEVEL         -0.0754      0.237     -0.318      0.750      -0.539       0.389
SPD_LIM       -0.0037      0.010     -0.356      0.722      -0.024       0.017
SUR_COND_dry  -0.6378      0.431     -1.481      0.139      -1.482       0.206
TRAF_two_way  -0.0976      0.232     -0.421      0.673      -0.552       0.357
WEATHER_adverse -0.1187    0.469     -0.253      0.800      -1.037       0.800
==============================================================================
```

I'm going to talk about WKDY only as it has a relatively low p value.

The coefficient implies that accidents on weekdays are more likely to result in no injury. The p-value (p = 0.072) is marginally significant, suggesting a potential meaningful effect that may warrant further investigation.

Decision Tree:

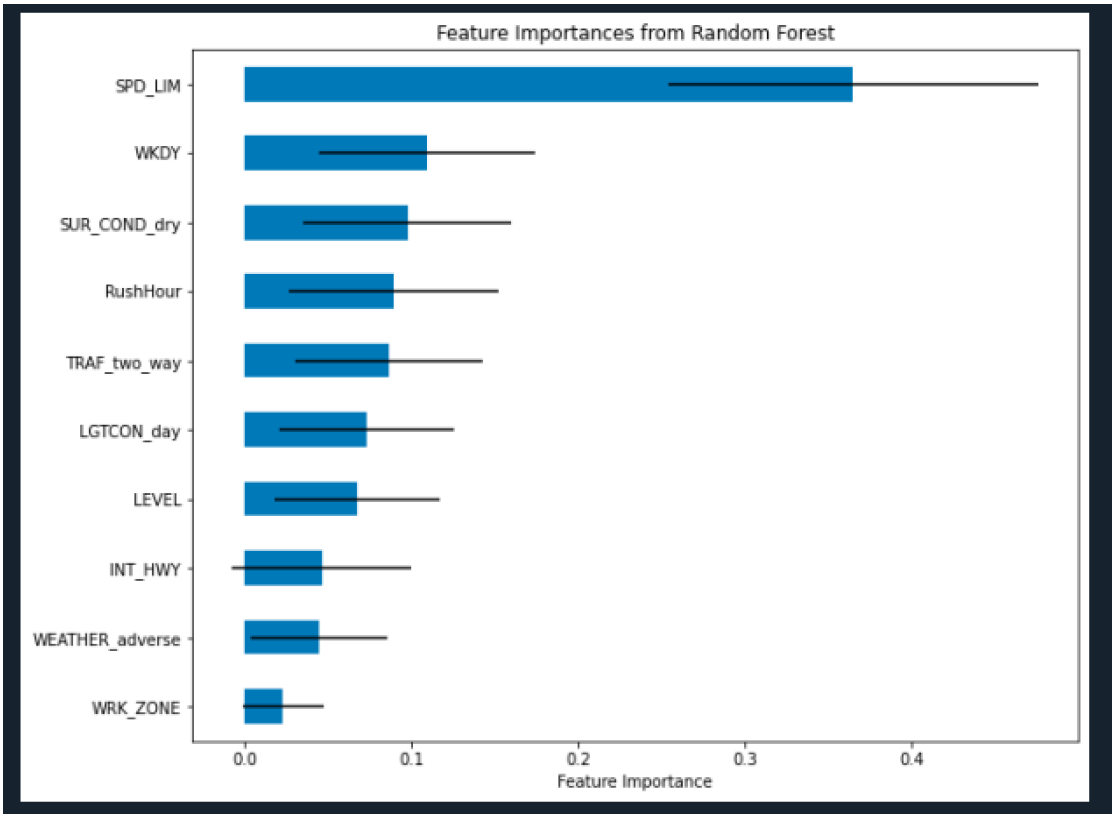
```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
tree.plot_tree(bestClassTree,
               feature_names=X_train.columns,
               class_names=['No Injury', 'Injury'],
               filled=True)
plt.show()
```



Gini Impurity indicates how mixed the node is regarding the target classes (Injury vs. No-Injury).
Lower Gini: The node is purer (more homogeneous), indicating most samples in that node belong to the same class.
Higher Gini: The node contains a more diverse mix of both outcomes.
We can see that Speed, surface conditions, and weather play a crucial role in predicting injuries.

Random Forest:

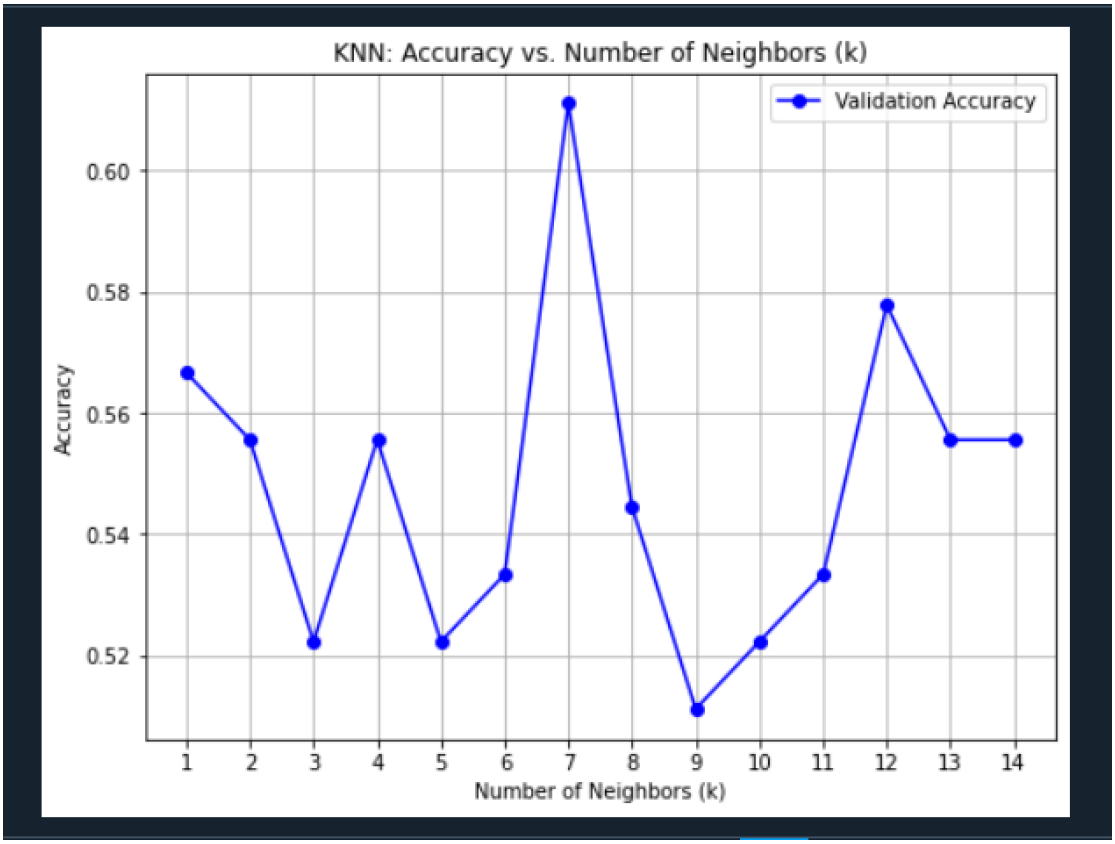
```
# Get feature importances from the best model
importances = best_rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in best_rf.estimators_], axis=0)

# Create a DataFrame to display feature importances
df1 = pd.DataFrame({
    'feature': X_train.columns,
    'importance': importances,
    'std': std
})

# Sort the DataFrame by importance
df1 = df1.sort_values('importance')

# Plot feature importances with error bars
ax = df1.plot(kind='barh', xerr='std', x='feature', legend=False, figsize=(10, 8))
ax.set_ylabel('')  # Remove y-axis label for clarity
ax.set_xlabel('Feature Importance')
plt.title('Feature Importances from Random Forest')
plt.show()
```



The importance plot also shows that SPD_LIM, WKDY and SUR_COND_DRY are the most important factors among all the predictors.

KNN:

```
k_values = [result['k'] for result in results]
accuracies = [result['accuracy'] for result in results]

# Plotting the accuracy vs k
plt.figure(figsize=(8, 6))
plt.plot(k_values, accuracies, marker='o', linestyle='-', color='b', label='Validation Accuracy')
plt.xticks(k_values)  # Ensures each k value is labeled on the x-axis
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('KNN: Accuracy vs. Number of Neighbors (k)')
plt.legend()
plt.grid(True)
plt.show()
```



We can see very clearly from the graph that when k=7, we can have the highest accuracy.

Simulation:

Since Decion Tree, Random Forest and KNN model are not linear, there is no coefficient we can refer. To understand the impact of each variable while keeping other variables with the mean value. I designed a function to check how the possibility will vary when changing the variable from min value to max value.

```python
def simulate_feature_effect_classification(model, X_train, feature_name, X_mean, feature_min

    X_sim_min = X_mean.copy()
    X_sim_max = X_mean.copy()


    X_sim_min[feature_name] = feature_min
    X_sim_max[feature_name] = feature_max


    if scaler:
        X_sim_min = scaler.transform([X_sim_min])
        X_sim_max = scaler.transform([X_sim_max])
    else:

        X_sim_min = np.array(X_sim_min).reshape(1, -1)
        X_sim_max = np.array(X_sim_max).reshape(1, -1)


    prob_min = model.predict_proba(X_sim_min)[0][1]
    prob_max = model.predict_proba(X_sim_max)[0][1]

    return prob_min, prob_max


X_mean_class = X_train.mean()


for feature_name in X_train.columns:
    feature_min = X_train[feature_name].min()
    feature_max = X_train[feature_name].max()


    #log_prob_min, log_prob_max = simulate_feature_effect_classification(logit_reg, X_train,
    #print(f"Logistic Regression: {feature_name} min={feature_min} -> prob={log_prob_min:.2f


    tree_prob_min, tree_prob_max = simulate_feature_effect_classification(bestClassTree, X_t
    print(f"Decision Tree: {feature_name} min={feature_min} -> prob={tree_prob_min:.2f}, max


    rf_prob_min, rf_prob_max = simulate_feature_effect_classification(best_rf, X_train, featu
    print(f"Random Forest: {feature_name} min={feature_min} -> prob={rf_prob_min:.2f}, max={


    knn_prob_min, knn_prob_max = simulate_feature_effect_classification(knn_best, train_X_sc
    print(f"KNN: {feature_name} min={feature_min} -> prob={knn_prob_min:.2f}, max={feature_m
```

```
...:     print(f"KNN: {feature_name} min={feature_min} -> prob={knn_prob_min:.2f}, max={feature_max} ->
prob={knn_prob_max:.2f}")
Decision Tree: RushHour min=0.0 -> prob=0.42, max=1.0 -> prob=0.59
Random Forest: RushHour min=0.0 -> prob=0.47, max=1.0 -> prob=0.54
KNN: RushHour min=0.0 -> prob=0.57, max=1.0 -> prob=0.57
Decision Tree: WRK_ZONE min=0.0 -> prob=0.42, max=1.0 -> prob=0.75
Random Forest: WRK_ZONE min=0.0 -> prob=0.47, max=1.0 -> prob=0.65
KNN: WRK_ZONE min=0.0 -> prob=0.43, max=1.0 -> prob=0.86
Decision Tree: WKDY min=0.0 -> prob=0.42, max=1.0 -> prob=0.42
Random Forest: WKDY min=0.0 -> prob=0.36, max=1.0 -> prob=0.47
KNN: WKDY min=0.0 -> prob=0.43, max=1.0 -> prob=0.57
Decision Tree: INT_HWY min=0.0 -> prob=0.42, max=1.0 -> prob=0.42
Random Forest: INT_HWY min=0.0 -> prob=0.47, max=1.0 -> prob=0.39
KNN: INT_HWY min=0.0 -> prob=0.57, max=1.0 -> prob=0.57
Decision Tree: LGTCON_day min=0 -> prob=0.28, max=1 -> prob=0.42
Random Forest: LGTCON_day min=0 -> prob=0.31, max=1 -> prob=0.47
KNN: LGTCON_day min=0 -> prob=0.14, max=1 -> prob=0.57
Decision Tree: LEVEL min=0.0 -> prob=0.42, max=1.0 -> prob=0.42
Random Forest: LEVEL min=0.0 -> prob=0.47, max=1.0 -> prob=0.53
KNN: LEVEL min=0.0 -> prob=0.57, max=1.0 -> prob=0.71
Decision Tree: SPD_LIM min=10.0 -> prob=0.37, max=75.0 -> prob=0.33
Random Forest: SPD_LIM min=10.0 -> prob=0.30, max=75.0 -> prob=0.59
KNN: SPD_LIM min=10.0 -> prob=0.43, max=75.0 -> prob=0.29
Decision Tree: SUR_COND_dry min=0.0 -> prob=0.54, max=1.0 -> prob=0.42
Random Forest: SUR_COND_dry min=0.0 -> prob=0.50, max=1.0 -> prob=0.47
KNN: SUR_COND_dry min=0.0 -> prob=0.43, max=1.0 -> prob=0.57
Decision Tree: TRAF_two_way min=0.0 -> prob=0.34, max=1.0 -> prob=0.42
Random Forest: TRAF_two_way min=0.0 -> prob=0.41, max=1.0 -> prob=0.47
KNN: TRAF_two_way min=0.0 -> prob=0.29, max=1.0 -> prob=0.57
Decision Tree: WEATHER_adverse min=0 -> prob=0.42, max=1 -> prob=0.42
Random Forest: WEATHER_adverse min=0 -> prob=0.47, max=1 -> prob=0.45
KNN: WEATHER_adverse min=0 -> prob=0.57, max=1 -> prob=0.43
```

From the output we can understand, being in the work_zone, the probability of no-injury is

higher than being in the non-work zone. From Random Forest model we can know that in highway area, the probability of no-injury is lower than in non-highway area.

Conclusion:

From our analysis, the Random Forest model emerged as the best performer with well-balanced accuracy and robust feature importance insights. Key predictors, such as speed limits, road conditions, and time-based factors (e.g., weekday or rush hour), were identified as significant contributors to accident outcomes. The KNN model achieved a peak accuracy at k = 7 but was outperformed by Random Forest in terms of overall predictive power and stability.

The findings emphasize that enforcing speed limits, improving road conditions, and monitoring high-risk periods (like weekends or adverse weather) could effectively reduce injury-prone accidents.

Regarding to the next steps, we can fine-tune the Random Forest with more hyperparameters like max_features. We can also experiment with ensemble methods (e.g., Gradient Boosting or XGBoost) for better performance. Definitely more data would be a great help for us to build the model.