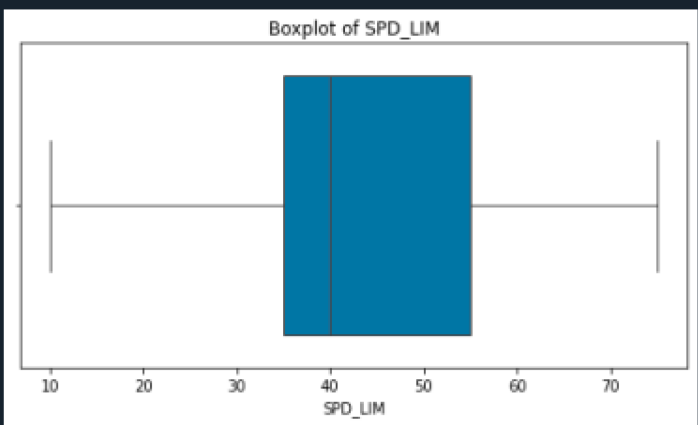
Part1.

Descriptive Statistic:

Index	RushHour	WRK_ZONE	WKDY	INT_HWY	LGTCON_day	LEVEL	SPD_LIM	;UR_COND_dr	TRAF_two_way	EATHER_adver
count	600	600	600	600	600	600	600	600	600	600
mean	0.458333	0.0116667	0.795	0.146667	0.73	0.23	43.2917	0.785	0.583333	0.161667
std	0.498677	0.10747	0.404038	0.495545	0.44433	0.421184	12.2695	0.411165	0.493418	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

Missing Value:

```
In [74]: print(df.isnull().sum())
RushHour          0
WRK_ZONE          0
WKDY              0
INT_HWY           0
LGTCON_day        0
LEVEL             0
SPD_LIM           0
SUR_COND_dry      0
TRAF_two_way      0
WEATHER_adverse   0
MAX_SEV           0
dtype: int64
```



There is no missing value, so there is no need for special handling. The values of those numeric predictors are mainly binary, so there is no outlier for those value. Regarding to the speed limit, there is no outlier based on the graph.

Categorical variables conversion:

Convert with label coder.

0 represents fatal, 1 represents no-injury and 2 represents non-fatal.

```
fatal -> 0
no-injury -> 1
non-fatal -> 2
```

y -

	0
0	1
1	2
2	1
3	1
4	2
5	2
6	1
7	2
8	1
9	2
10	1

Data Partition:

```
In [77]: 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)
    ...: 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, 9)
Validation set size: (90, 9)
Test set size: (90, 9)
```

Part2.

Feature Selection:

I'm doing a quick training with randomforest model to calculate the importance of each predictor. Based on the calculation and the domain knowledge, I will remove below three predictors. WEATHER_adverse, INT_HWY and WRK_ZONE.

Architecture of MLP:

There are 7 features in the input layer, for hidden layer I will try different combinations to test out which provides a better accuracy and a more balance result between training and validation. There are 3 classes in the output layer which are the three accident severity levels.

I'm testing two activation function, one is Relu + Softmax and the other one is Tanh. Tanh is quite suitable for classification tasks and ReLU mitigates the vanishing gradient issues. Both of them provide a non-linear transformation compared to no activation function.

I used the standard scaler to minimize the impact of numeric predictor speed_limit.

```
from sklearn.model_selection import train_test_split
X = df.drop(['WEATHER_adverse', 'INT_HWY', 'WRK_ZONE', 'MAX_SEV'], axis=1)
y = df['MAX_SEV']

scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
le = LabelEncoder()
y = le.fit_transform(df['MAX_SEV'])
for i, label in enumerate(le.classes_):
    print(f"{label} -> {i}")
```

```python
#part2
architectures = [
    (4,),
    (4,4),
    (8, 8),
    (16, 16),
    (16, 8, 4),
    (32, 16, 8),
    (32,32,32),
    (64,64,32),
    (64,64,64,16),
    (64,64,64,64)
]
results_relu = []
results_tanh = []


for hidden_layers in architectures:
    # ReLu
    clf_relu = MLPClassifier(
        hidden_layer_sizes=hidden_layers,
        activation='relu',
        solver='adam',
        max_iter=1000,
        random_state=1
    )


    clf_tanh = MLPClassifier(
        hidden_layer_sizes=hidden_layers,
        activation='tanh',
        solver='adam',
        max_iter=1000,
        random_state=1
    )


    clf_relu.fit(X_train, y_train)
    clf_tanh.fit(X_train, y_train)
```

Model Training and relevant metrics:

```
   ...: results_df = pd.DataFrame(results_relu + results_tanh)
   ...: print(results_df)
       architecture  train_accuracy  ...  validation_loss  activation
0              (4,)        0.614286  ...         0.791952        ReLU
1            (4, 4)        0.576190  ...         0.788952        ReLU
2            (8, 8)        0.592857  ...         0.798181        ReLU
3          (16, 16)        0.638095  ...         0.793165        ReLU
4        (16, 8, 4)        0.630952  ...         0.923062        ReLU
5       (32, 16, 8)        0.683333  ...         0.883875        ReLU
6      (32, 32, 32)        0.657143  ...         0.795234        ReLU
7      (64, 64, 32)        0.685714  ...         1.050265        ReLU
8  (64, 64, 64, 16)        0.719048  ...         1.036063        ReLU
9  (64, 64, 64, 64)        0.695238  ...         0.899650        ReLU
10             (4,)        0.569048  ...         0.748638        Tanh
11           (4, 4)        0.588095  ...         0.735209        Tanh
12           (8, 8)        0.607143  ...         0.758582        Tanh
13         (16, 16)        0.614286  ...         0.739183        Tanh
14       (16, 8, 4)        0.630952  ...         0.798584        Tanh
15      (32, 16, 8)        0.614286  ...         0.739115        Tanh
16     (32, 32, 32)        0.635714  ...         0.750778        Tanh
17     (64, 64, 32)        0.657143  ...         0.784518        Tanh
18 (64, 64, 64, 16)        0.616667  ...         0.751704        Tanh
19 (64, 64, 64, 64)        0.607143  ...         0.769120        Tanh
```

Attached is the training accuracy and validation_loss by using different activation function and different hidden layers. According to the statistics, I will use (32,16,8) for ReLU as it provides a good accuracy 0.683 and a logical validation loss 0.88. Also I will try (32,16,8) for Tanh with a 0.614 accuracy and 0.739 validation loss in the later training process also to compare which activation function is a better solution for this dataset.

Model with early stop:

```python
best_model_relu = MLPClassifier(
    hidden_layer_sizes=(32,16,8),
    activation='relu',
    solver='adam',
    max_iter=1000,
    random_state=1,
    early_stopping=True,
    validation_fraction=0.1,
    learning_rate_init=0.02,
    n_iter_no_change=20
)

best_model_tanh = MLPClassifier(
    hidden_layer_sizes=(32,16,8),
    activation='tanh',
    solver='adam',
    max_iter=1000,
    random_state=1,
    early_stopping=True,
    validation_fraction=0.1,
    learning_rate_init=0.02,
    n_iter_no_change=20
)
print("\nReLU Model Training:")
best_model_relu.fit(X_train, y_train)
print(f"Number of iterations: {best_model_relu.n_iter_}")
print(f"Training set accuracy: {best_model_relu.score(X_train, y_train):.4f}")
print(f"Validation set accuracy: {best_model_relu.score(X_validation, y_validation):.4f}")
print(f"Test set accuracy: {best_model_relu.score(X_test, y_test):.4f}")

# Train Tanh model and print information
print("\nTanh Model Training:")
best_model_tanh.fit(X_train, y_train)
print(f"Number of iterations: {best_model_tanh.n_iter_}")
print(f"Training set accuracy: {best_model_tanh.score(X_train, y_train):.4f}")
print(f"Validation set accuracy: {best_model_tanh.score(X_validation, y_validation):.4f}")
print(f"Test set accuracy: {best_model_tanh.score(X_test, y_test):.4f}")
```

After introduce early stop in neural network training, I set the n_iter_no_change to 20 and learning rate as 2%. By compared the result of these two model, I decided to use the activation function Tanh for further study.

```
ReLU Model Training:
Number of iterations: 31
Training set accuracy: 0.6143
Validation set accuracy: 0.5556
Test set accuracy: 0.4556

Tanh Model Training:
Number of iterations: 62
Training set accuracy: 0.6738
Validation set accuracy: 0.5778
Test set accuracy: 0.5111
```

The training accuracy is higher than the ReLU one and ReLU seems have some issue with overfitting.

To give us more confidence that we have found a global minimum point rather than a local minimum. I initialized few more training with different random state, as check the accuracy of them and the small value of standard deviation, we are more likely getting the global optimum.

Diagnostic checks:

The diagnostic analysis of our neural network model revealed several important insights about its performance. The model achieved an accuracy of 52.22% on the test set, slightly outperforming its validation set accuracy of 42.22%. Detailed class-wise metrics exposed significant imbalances in the model's predictive capabilities. Most notably, the model completely failed to identify Class 0 instances (0% sensitivity and precision, but 100% specificity), indicating a severe class imbalance issue. Class 1 showed better performance with test set precision of 61.76% and sensitivity of 42.86%, demonstrating moderate reliability but a high false negative rate. Class 2 exhibited the highest sensitivity (66.67%) but lower precision (46.43%) and specificity (41.18%), suggesting a bias towards over-prediction of this class. The disparity in performance across classes, particularly the complete failure in Class 0 prediction, indicates that future improvements should focus on addressing class imbalance, possibly through techniques such as resampling or adjusting class weights in the model architecture.

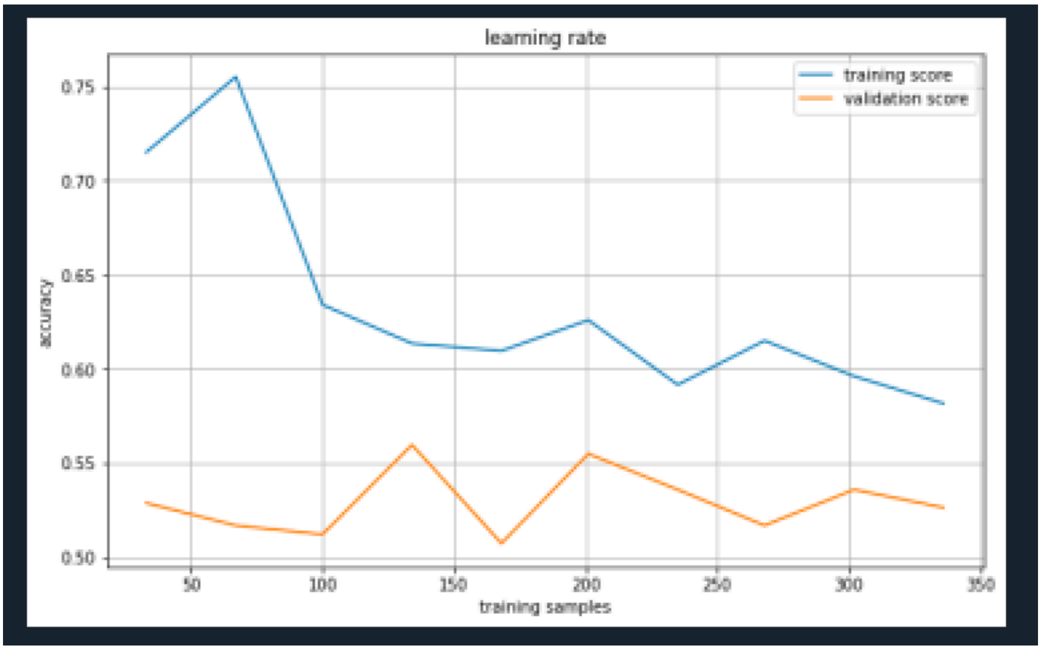
```
Validation Set Diagnostic Metrics:
Accuracy: 0.4222
Error Rate: 0.5778

Detailed metrics for each class:

Class 0:
Precision: 0.0000
Sensitivity (Recall): 0.0000
Specificity: 1.0000

Class 1:
Precision: 0.3889
Sensitivity (Recall): 0.3182
Specificity: 0.5217

Class 2:
Precision: 0.4444
Sensitivity (Recall): 0.5333
Specificity: 0.3333

Test Set Diagnostic Metrics:
Accuracy: 0.5222
Error Rate: 0.4778

Detailed metrics for each class:

Class 0:
Precision: 0.0000
Sensitivity (Recall): 0.0000
Specificity: 1.0000

Class 1:
Precision: 0.6176
Sensitivity (Recall): 0.4286
Specificity: 0.6829

Class 2:
Precision: 0.4643
Sensitivity (Recall): 0.6667
Specificity: 0.4118
```

F1score:

```
In [97]: f1_val = f1_score(y_validation, y_val_pred, average='weighted')
    ...: f1_test = f1_score(y_test, y_test_pred, average='weighted')
    ...: print(f"\F1score:")
    ...: print(f"validation F1: {f1_val:.4f}")
    ...: print(f"Test F1: {f1_test:.4f}")
\F1score:
validation F1: 0.4135
Test F1: 0.5127
```

The diagnostic analysis revealed moderate model performance with a test set accuracy of 52.22% and F1 score of 0.5127, showing improvement from the validation set (accuracy: 42.22%, F1: 0.4135). Class-wise analysis exposed significant imbalances: Class 0 showed complete failure in detection (0% sensitivity), while Class 1 achieved the highest precision (61.76%) and Class 2 demonstrated the highest sensitivity (66.67%). These results indicate a clear need for addressing class imbalance issues to improve overall model performance.

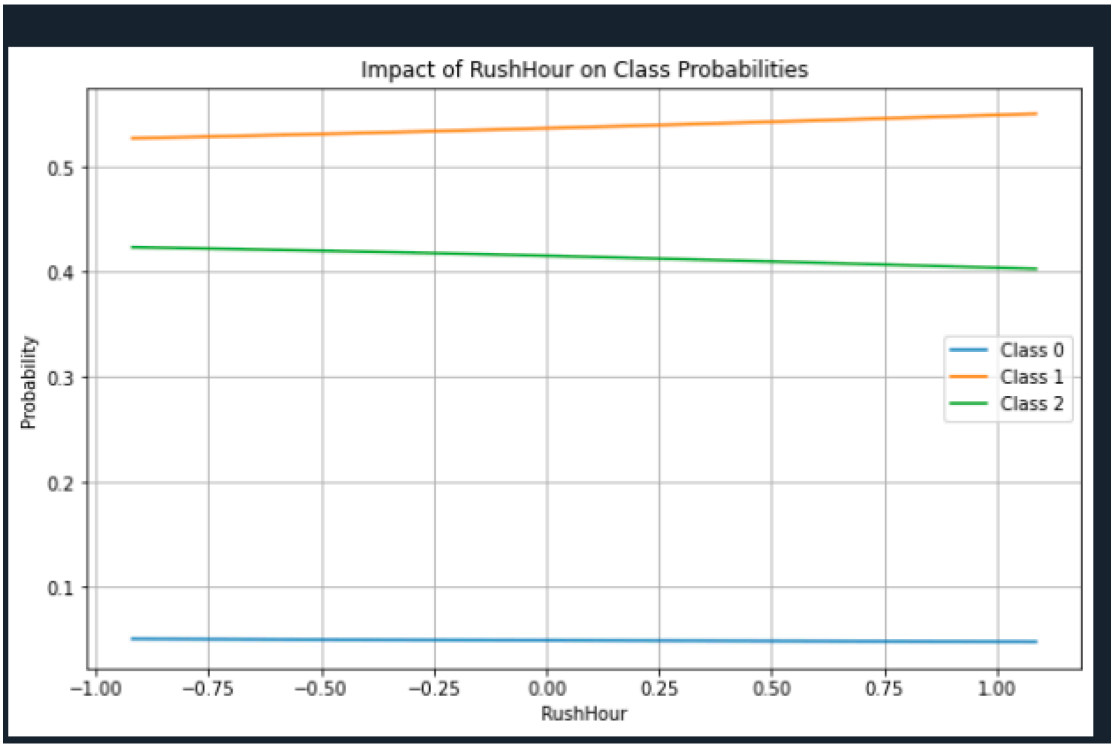
Learning Curves:

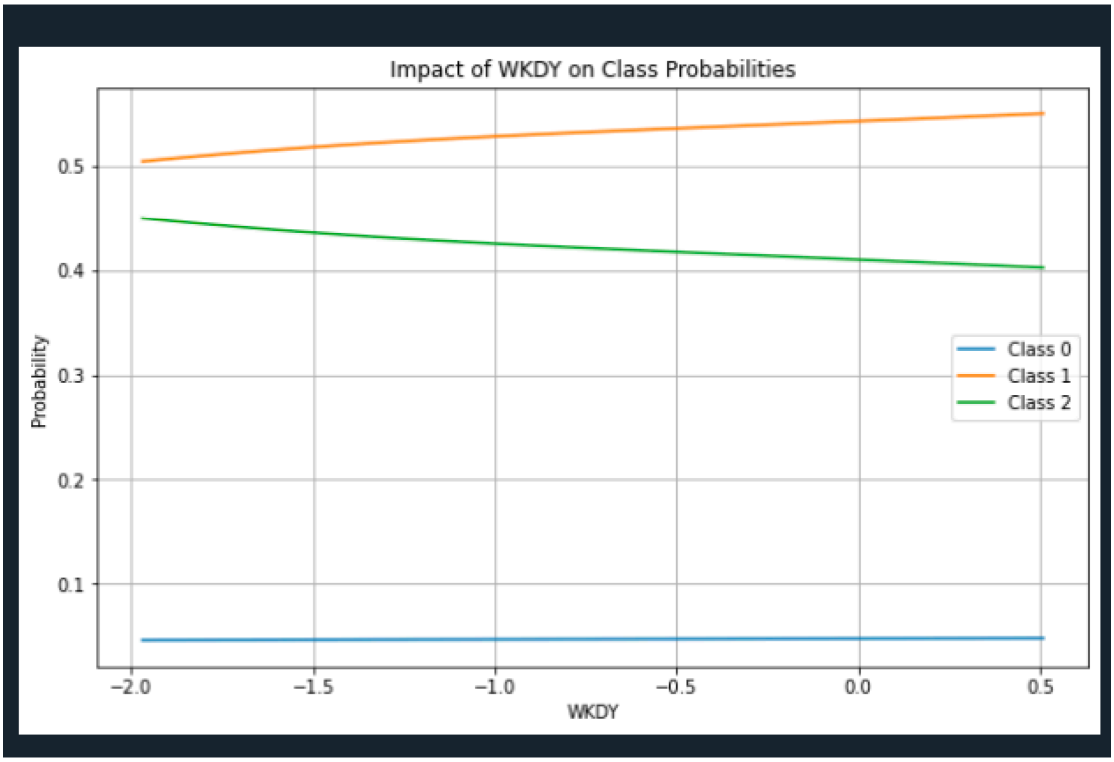


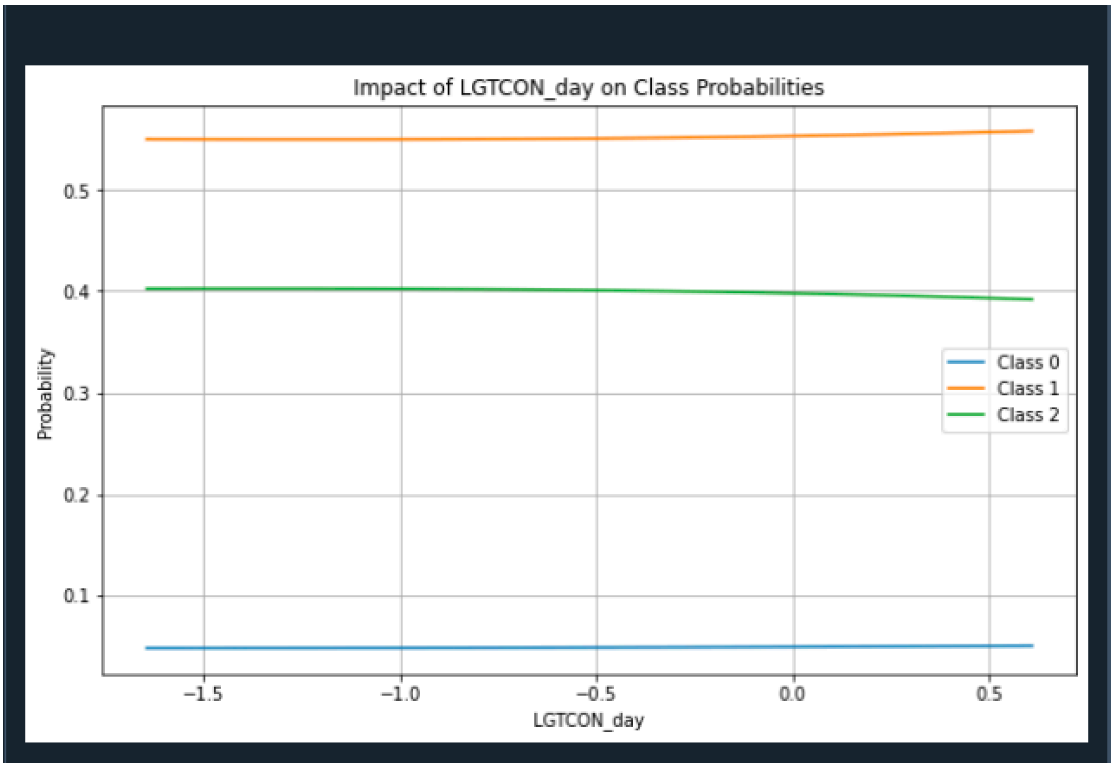

```
    warnings.warn(
\overfit check:
training accuracy: 0.5786
test accuracy: 0.5222
difference: 0.0563
```

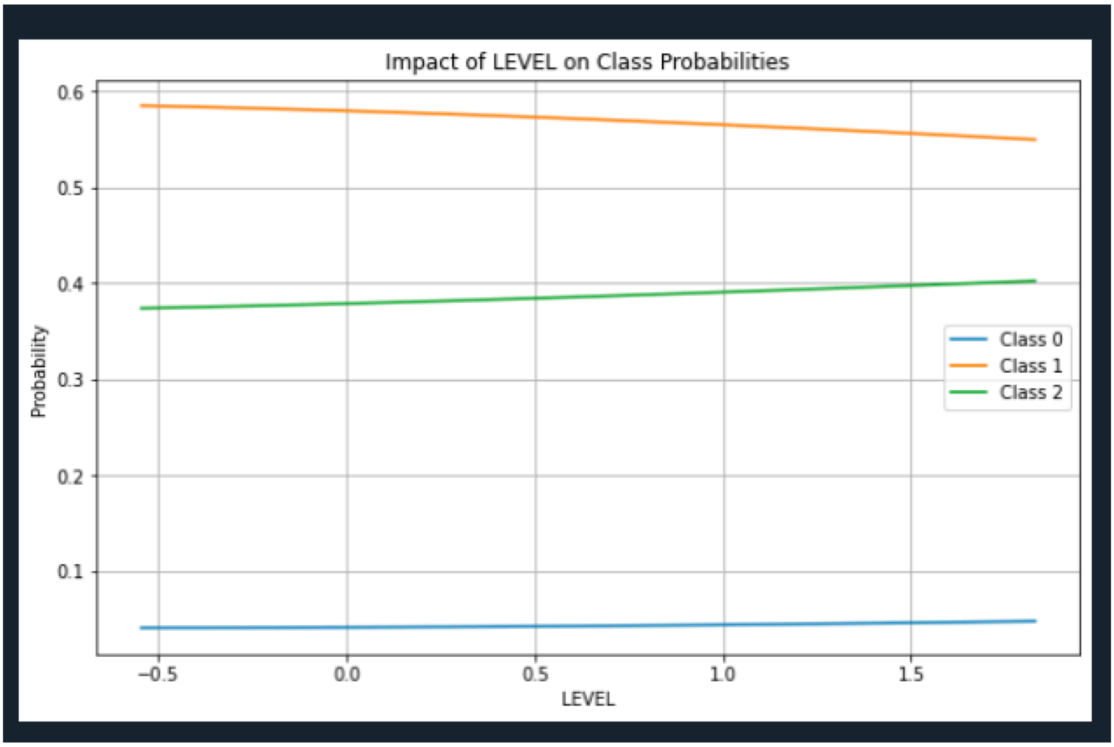
The learning curve analysis shows evidence of slight overfitting, with a training accuracy of 0.5786 compared to a test accuracy of 0.5222 (difference of 0.0563). The learning curve plot demonstrates a consistent gap between training and validation scores, with the training score maintaining higher values than the validation score. However, both curves flatten out as training samples increase, suggesting that while overfitting is present, it's relatively mild. The model might benefit from additional regularization techniques, but the parallel nature of the curves indicates that the model has reached a stable learning state.

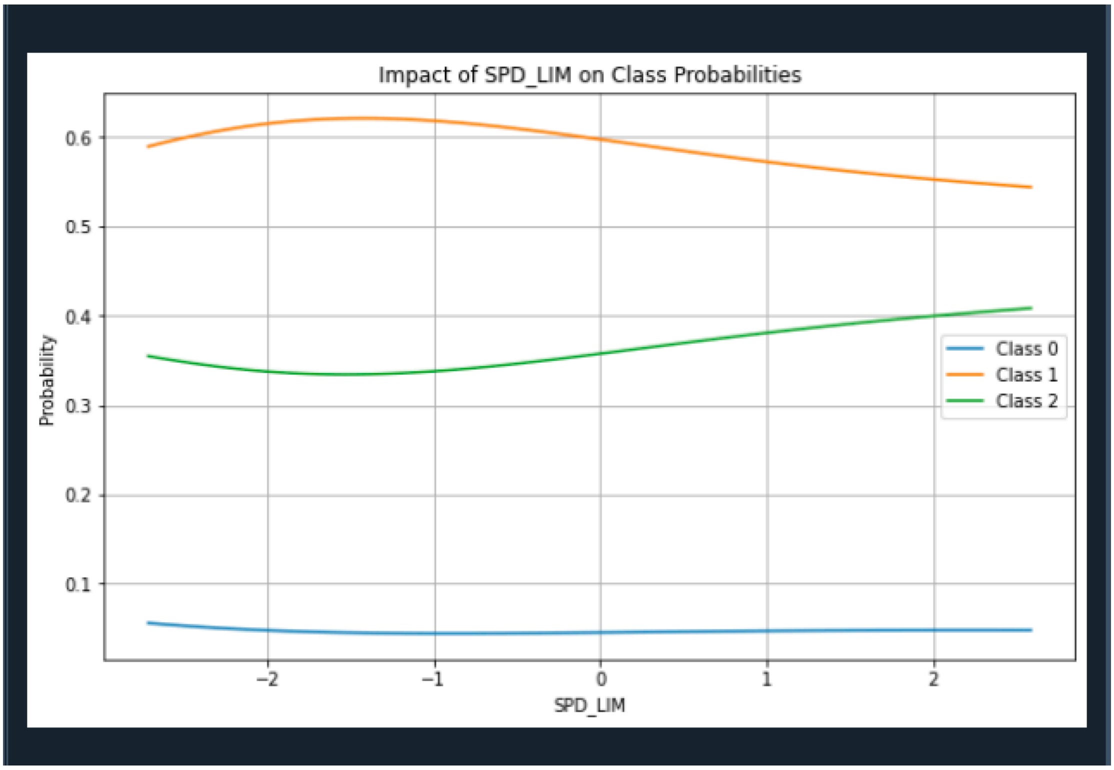
Model Interpretation:

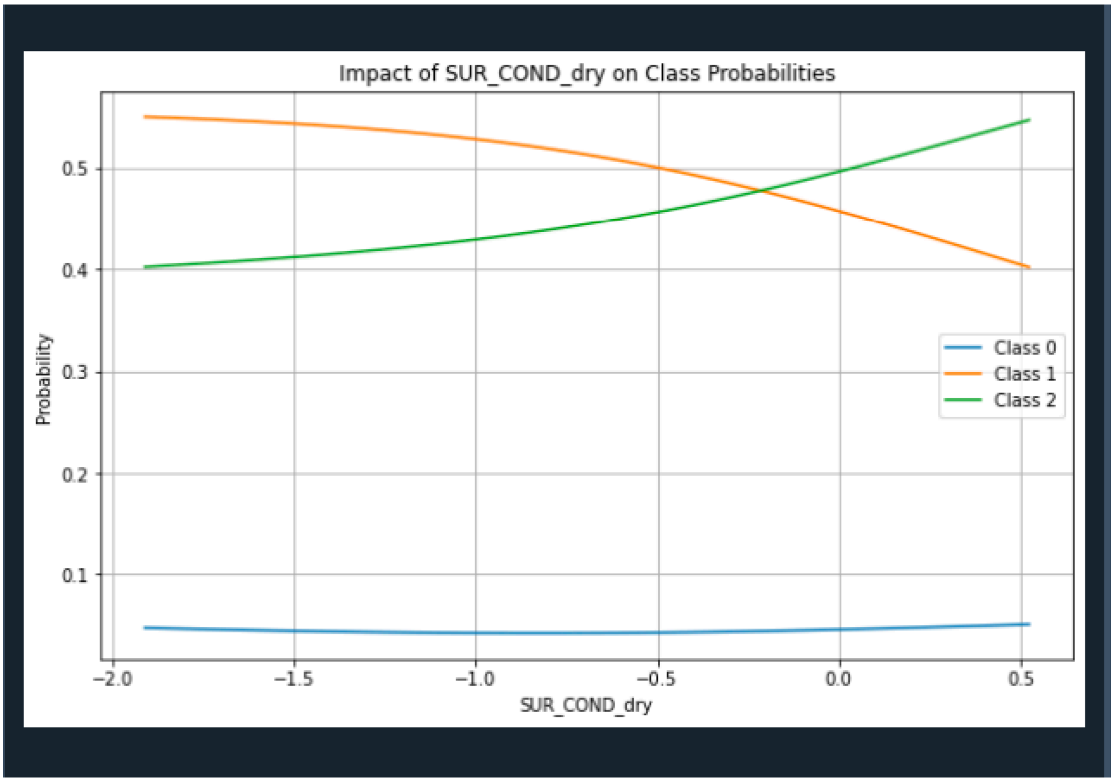


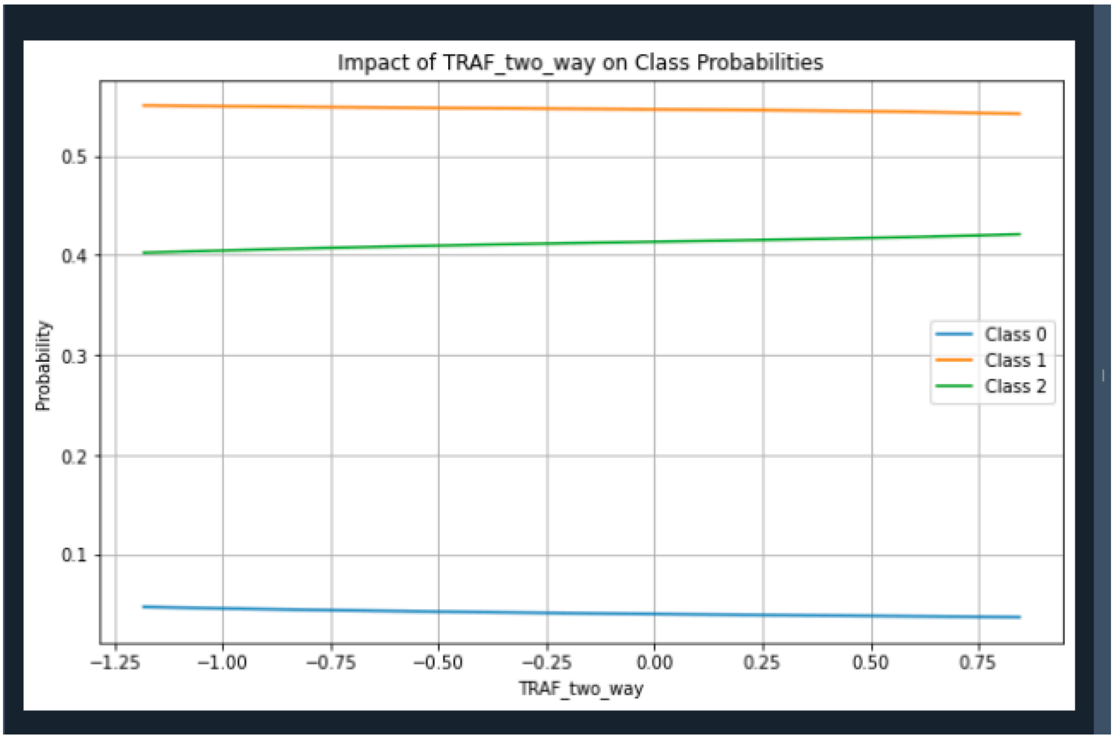



Impact of LGTCON_day on Class Probabilities
Probability
0.1
0.2
0.3
0.4
0.5
−1.5
−1.0
−0.5
0.0
0.5
LGTCON_day
Class 0
Class 1
Class 2


Impact of LEVEL on Class Probabilities
Probability
0.1
0.2
0.3
0.4
0.5
0.6
−0.5
0.0
0.5
1.0
1.5
LEVEL
Class 0
Class 1
Class 2


Impact of SPD_LIM on Class Probabilities
Probability
0.6
0.5
0.4
0.3
0.2
0.1
−2
−1
0
1
2
SPD_LIM
Class 0
Class 1
Class 2


Impact of SUR_COND_dry on Class Probabilities
Probability
0.5
0.4
0.3
0.2
0.1
−2.0
−1.5
−1.0
−0.5
0.0
0.5
SUR_COND_dry
Class 0
Class 1
Class 2



Neural network simulation reveals varying impacts of predictors on accident severity classification:

Most Influential Variables:

1. Surface Condition (SUR_COND_dry):
    - Strongest predictor overall
    - Clear inverse relationship between Class 1 and 2
    - Higher values significantly increase Class 2 probability (40% to 55%)
    - Critical threshold at median value where probabilities cross

2. Speed Limit (SPD_LIM):
    - Non-linear relationship with Class 1
    - Peak effect at moderate speeds
    - Class 1 probability peaks at ~62%
    - Class 2 shows inverse relationship at higher speeds

Moderately Influential Variables:

3. Level of Road (LEVEL):
    - Inverse relationship between Class 1 and 2
    - Higher levels decrease Class 1 probability (58% to 55%)
    - Gradual increase in Class 2 probability with level

4. Weekday (WKDY):
    - Weekdays show higher Class 1 probability

- Weekend periods favor Class 2 outcomes
- ~5% variation in probabilities

Least Influential Variables:

5. Rush Hour
6. Lighting Conditions (LGTCON_day)
7. Traffic Way Type (TRAF_two_way)
   - Minimal impact on predictions
   - Stable probabilities across values
   - Limited predictive value

```
fatal -> 0
no-injury -> 1
non-fatal -> 2
```

Summary:

Introduction & Methodology:

The study implemented a neural network model for accident severity classification, utilizing environmental and road condition variables with tanh activation function. Data preprocessing included normalization and train-test splitting.

Results & Performance:

The model achieved a test accuracy of 52.22% and F1 score of 0.5127, with slight overfitting indicated by a 5.63% gap between training and test accuracy. Neural network simulation revealed Surface Condition and Speed Limit as the most influential predictors, while Rush Hour and Lighting Conditions showed minimal impact.

Implications & Future Steps:

Model performance suggests room for improvement, particularly in predicting Class 0 accidents. Future work should focus on addressing class imbalance, feature engineering, and exploring alternative model architectures. Findings indicate that focusing on surface conditions and speed management could be key to accident prevention strategies.