Road Accidents Analysis & Severity Prediction

Importing necessary Libraries

In [4]:
```
%%time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
import requests
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

```
CPU times: total: 3.38 s
Wall time: 8.49 s
```

Loading Dataset

In [5]:
```
data = pd.read_excel('RTAdata.xlsx')
data.head()
```

Out[5]:

	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_driver_relation	Driving_experience	Type_of_vehicle	Owner_of_vehicle
0	17:02:00	Monday	18-30	Male	Above high school	Employee	1-2yr	Automobile	Owner
1	17:02:00	Monday	31-50	Male	Junior high school	Employee	Above 10yr	Public (> 45 seats)	Owner
2	17:02:00	Monday	18-30	Male	Junior high school	Employee	1-2yr	Lorry (41-100Q)	Owner
3	01:06:00	Sunday	18-30	Male	Junior high school	Employee	5-10yr	Public (> 45 seats)	Governmental
4	01:06:00	Sunday	18-30	Male	Junior high school	Employee	2-5yr	NaN	Owner

5 rows × 32 columns

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12316 entries, 0 to 12315
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Time                     12316 non-null  object
 1   Day_of_week              12316 non-null  object
 2   Age_band_of_driver       12316 non-null  object
 3   Sex_of_driver            12316 non-null  object
 4   Educational_level        11575 non-null  object
```

Cleaning Dataset

In [7]:
```python
# Remove rows with "Unknown" values
df_cleaned = data[~data.apply(lambda row: row.astype(str).str.contains('Unknown').any(), axis=1)]
cleaned_file_path = 'cleaned_dataset.xlsx'
df_cleaned.to_excel(cleaned_file_path, index=False)
print(f"Cleaned data saved to {cleaned_file_path}")
```

```
Cleaned data saved to cleaned_dataset.xlsx
```

In [8]:
```python
data1 = pd.read_excel('cleaned_dataset.xlsx')
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7417 entries, 0 to 7416
Data columns (total 32 columns):
```

In [10]:
```python
data1.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
Number_of_vehicles_involved	7417.0	2.036807	0.668984	1.0	2.0	2.0	2.0	7.0
Number_of_casualties	7417.0	1.586086	1.042470	1.0	1.0	1.0	2.0	8.0
Casualty_severity	4675.0	2.898610	0.313694	1.0	3.0	3.0	3.0	3.0

In [11]:
```python
#skewness
selected_columns1=["Number_of_vehicles_involved","Number_of_casualties"]
skewness = data1[selected_columns1].skew()
print(skewness)
```

```
Number_of_vehicles_involved     1.260581
Number_of_casualties            2.227987
dtype: float64
```

In [12]:
```python
for i in data1.columns:
    if data1[i].dtypes== object:
        print(i)
        print(data1[i].unique())
        print(data1[i].nunique())
        print()S
```

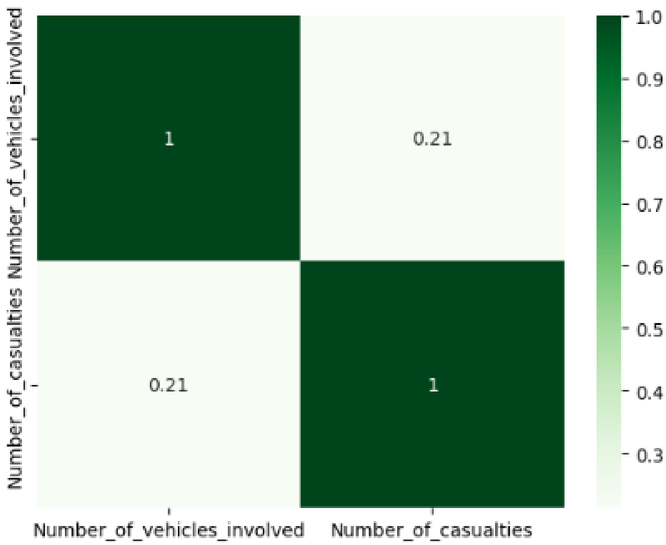
```
Day_of_week
['Monday' 'Sunday' 'Wednesday' 'Friday' 'Saturday' 'Thursday' 'Tuesday']
7

Age_band_of_driver
['18-30' '31-50' 'Under 18' 'Over 51']
4

Sex_of_driver
['Male' 'Female']
2

Educational_level
['Above high school' 'Junior high school' 'Elementary school' nan
 'High school' 'Illiterate' 'Writing & reading']
6

Vehicle_driver_relation
['Employee' 'Owner' nan 'Other']
3

Driving_experience
['1-2yr' 'Above 10yr' '5-10yr' '2-5yr' 'No Licence' 'Below 1yr' nan
 'unknown']
7

Type_of_vehicle
['Automobile' 'Public (> 45 seats)' 'Lorry (41-100Q)' nan 'Lorry (11-40Q)'
 'Public (13-45 seats)' 'Long lorry' 'Public (12 seats)' 'Taxi'
 'Pick up upto 10Q' 'Stationwagen' 'Other' 'Ridden horse' 'Motorcycle'
 'Turbo' 'Bicycle' 'Special vehicle' 'Bajaj']
17
```

By Nischay Gautam

Correlation

In [15]:
```python
#for Numerical Variables
correlation = data[selected_columns1].corr()
sns.heatmap(correlation,annot = True, cmap = 'Greens')
```

Out[15]: <Axes: >



Cramer's V for Categorical Variables

In [73]:
```python
#for categorical variables
from scipy.stats import chi2_contingency
selected_columns2=["Age_band_of_casualty","Sex_of_casualty","Type_of_vehicle","Road_surface_type","Road_surface_conditions",
                   "Light_conditions","Type_of_collision","Accident_severity"]
data2 = data1[selected_columns2]

# Function to calculate Cramér's V
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    r, k = confusion_matrix.shape
    return np.sqrt(chi2 / (n * (min(r, k) - 1)))

# Calculate Cramér's V for each variable with respect to the target variable
target_variable = 'Accident_severity'
results = {}
for variable in data2.columns:
    if variable != target_variable:
        contingency_table = pd.crosstab(data2[variable], data2[target_variable])
        cramers_v_value = cramers_v(contingency_table)
        results[variable] = cramers_v_value

# Display the results
for variable, value in results.items():
    print(f"Cramér's V for {variable} with respect to {target_variable}: {value}")
```

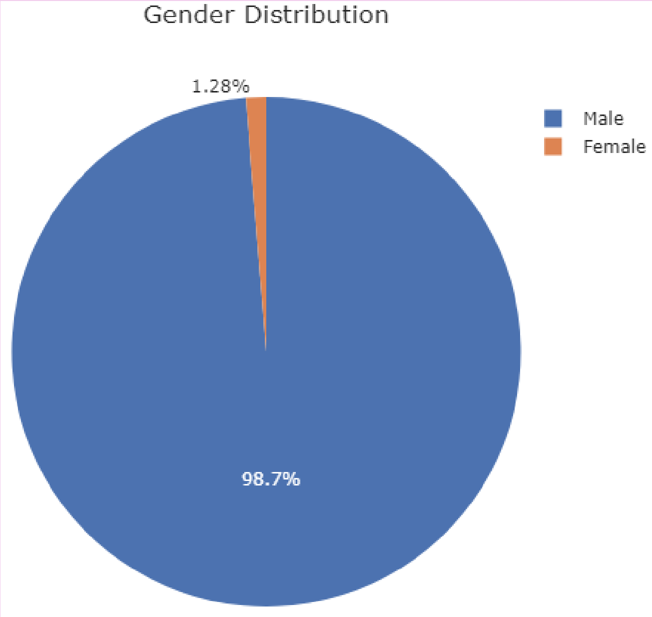
```
Cramér's V for Age_band_of_casualty with respect to Accident_severity: 0.04100781145890263
Cramér's V for Sex_of_casualty with respect to Accident_severity: 0.016888876631200308
Cramér's V for Type_of_vehicle with respect to Accident_severity: 0.060166950245805886
Cramér's V for Road_surface_type with respect to Accident_severity: 0.022444251595571993
Cramér's V for Road_surface_conditions with respect to Accident_severity: 0.018750916611646087
Cramér's V for Light_conditions with respect to Accident_severity: 0.0554158069505708
Cramér's V for Type_of_collision with respect to Accident_severity: 0.035515244289020614
```

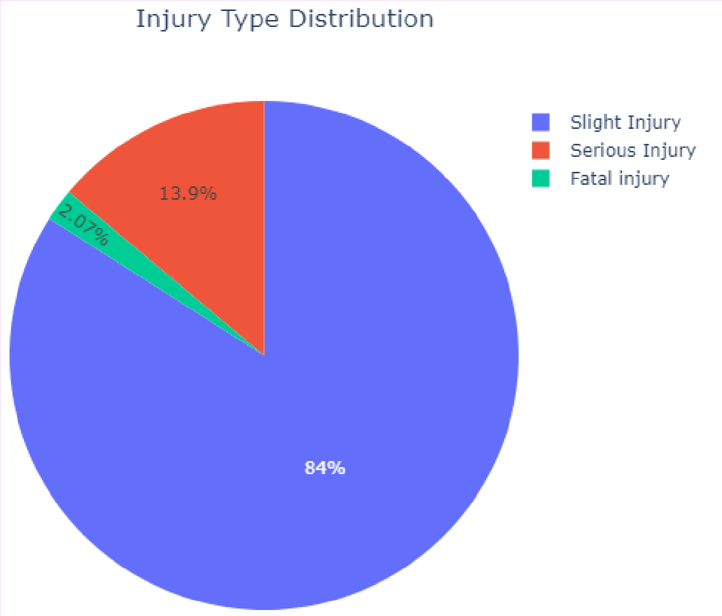
By Nischay Gautam

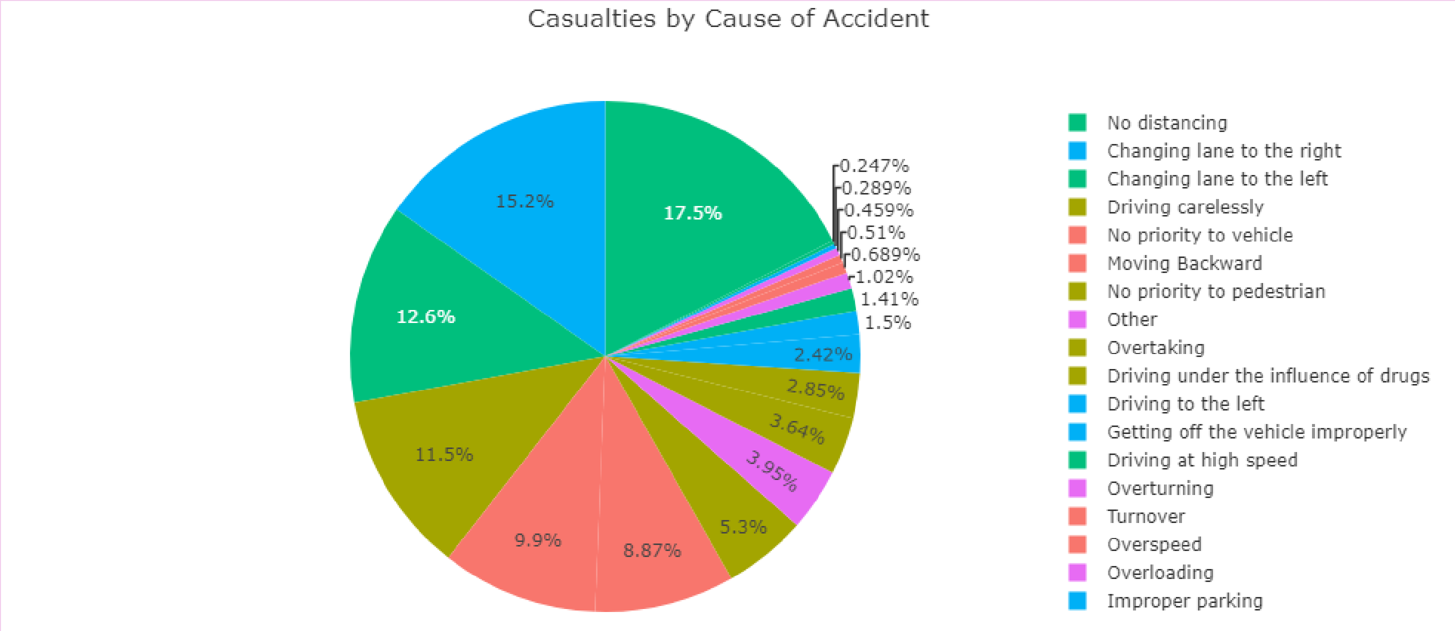
Exploratory Data Analysis (EDA)

Pie Charts for Gender Distribution, Injury Type and Cause of Accidents

In [18]:
```
#PIE Charts
fig1=px.pie(data1, data1['Sex_of_driver'], data1['Number_of_casualties'],title='Gender Distribution' ,template='seaborn')
fig2=px.pie(data1, data1['Accident_severity'], data1['Number_of_casualties'],title='Injury Type Distribution',template='plotly')
fig1.show()
fig2.show()
fig3=px.pie(data1,data1['Cause_of_accident'],data1['Number_of_casualties'],
         color='Cause_of_accident',template='ggplot2',title='Casualties by Cause of Accident')
fig3.show()
```



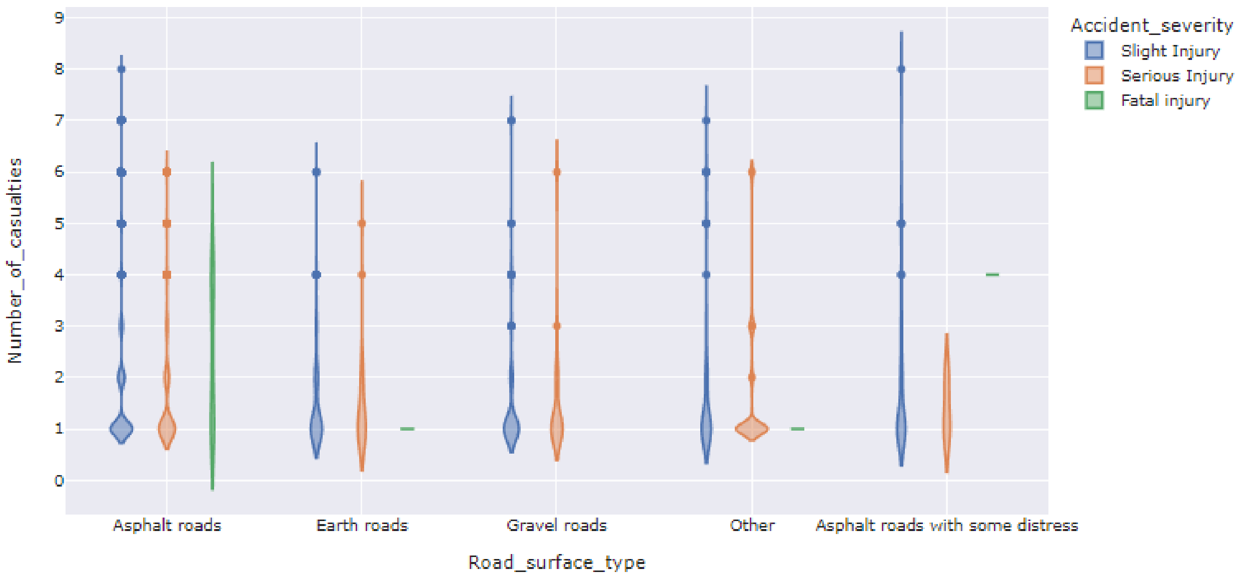




By Nischay Gautam

Violin plot for Casualties by Road-surface type

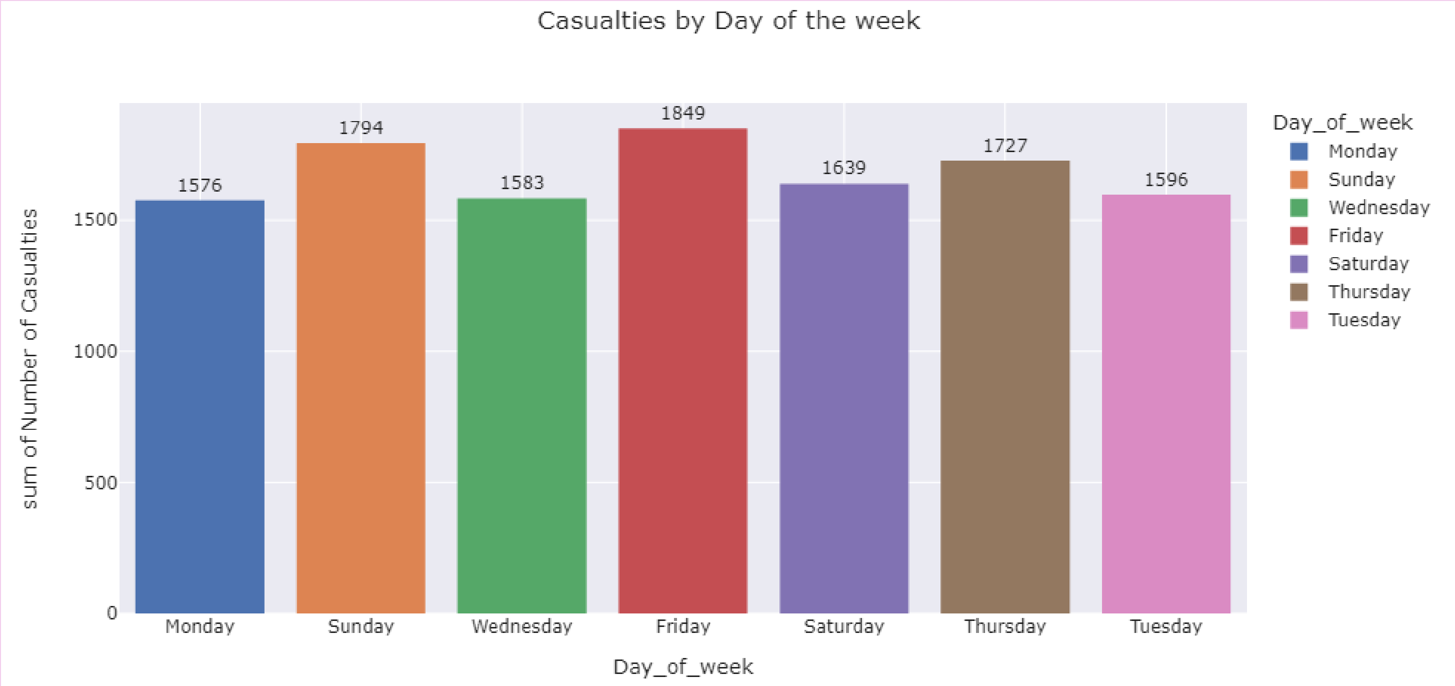
In [19]:
```
fig4=px.violin(data1,data1['Road_surface_type'], data1['Number_of_casualties'],color='Accident_severity',
               template='seaborn')
fig4.show()
```

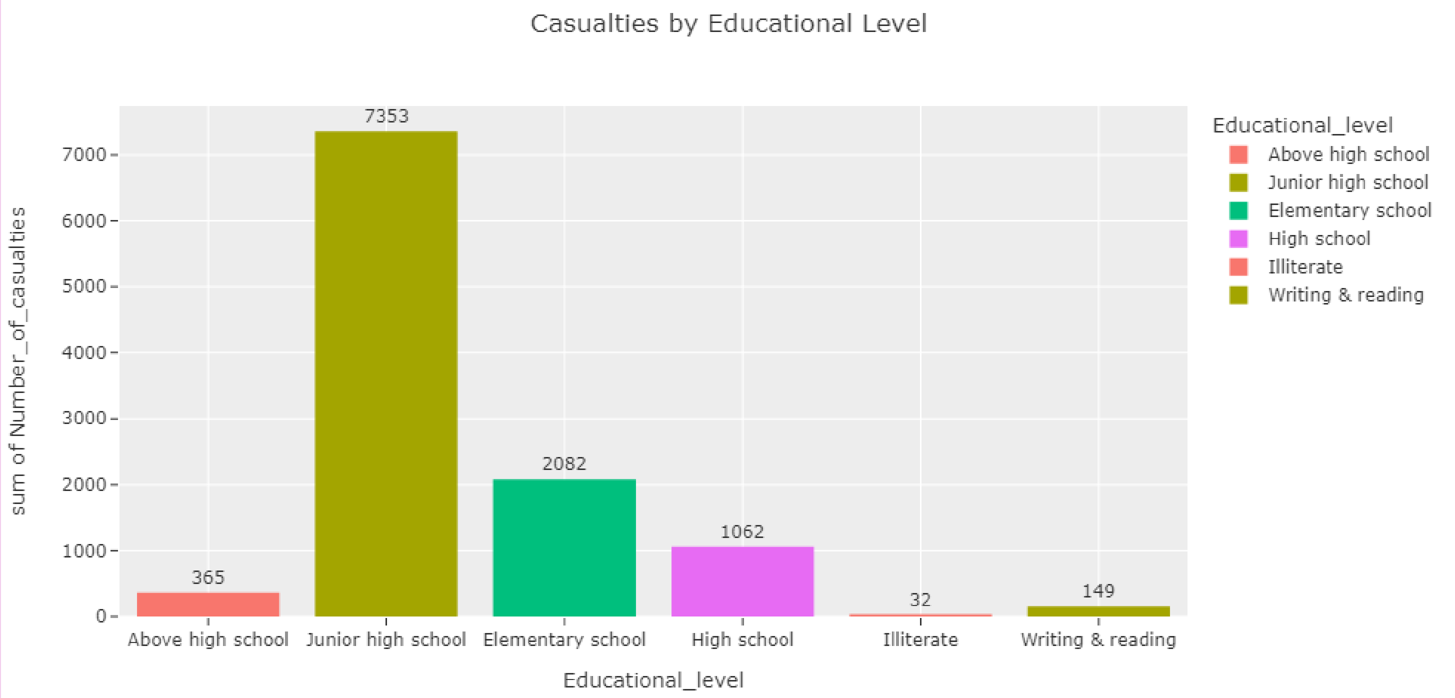


Bar Charts for Casualties by Day of the week, Education-level

In [20]:
```
fig5=px.histogram(data1,data1['Day_of_week'],data1['Number_of_casualties'],
       color='Day_of_week',template='seaborn',title='Casualties by Day of the week',
             labels={'Number_of_casualties': 'Number of Casualties'})
fig5.update_traces(texttemplate='%{y}', textposition='outside')

fig6=px.histogram(data1,data1['Educational_level'],data1['Number_of_casualties'],
       color='Educational_level',template='ggplot2',title='Casualties by Educational Level')
fig6.update_traces(texttemplate='%{y}', textposition='outside')
fig5.show()
fig6.show()
```

By Nischay Gautam

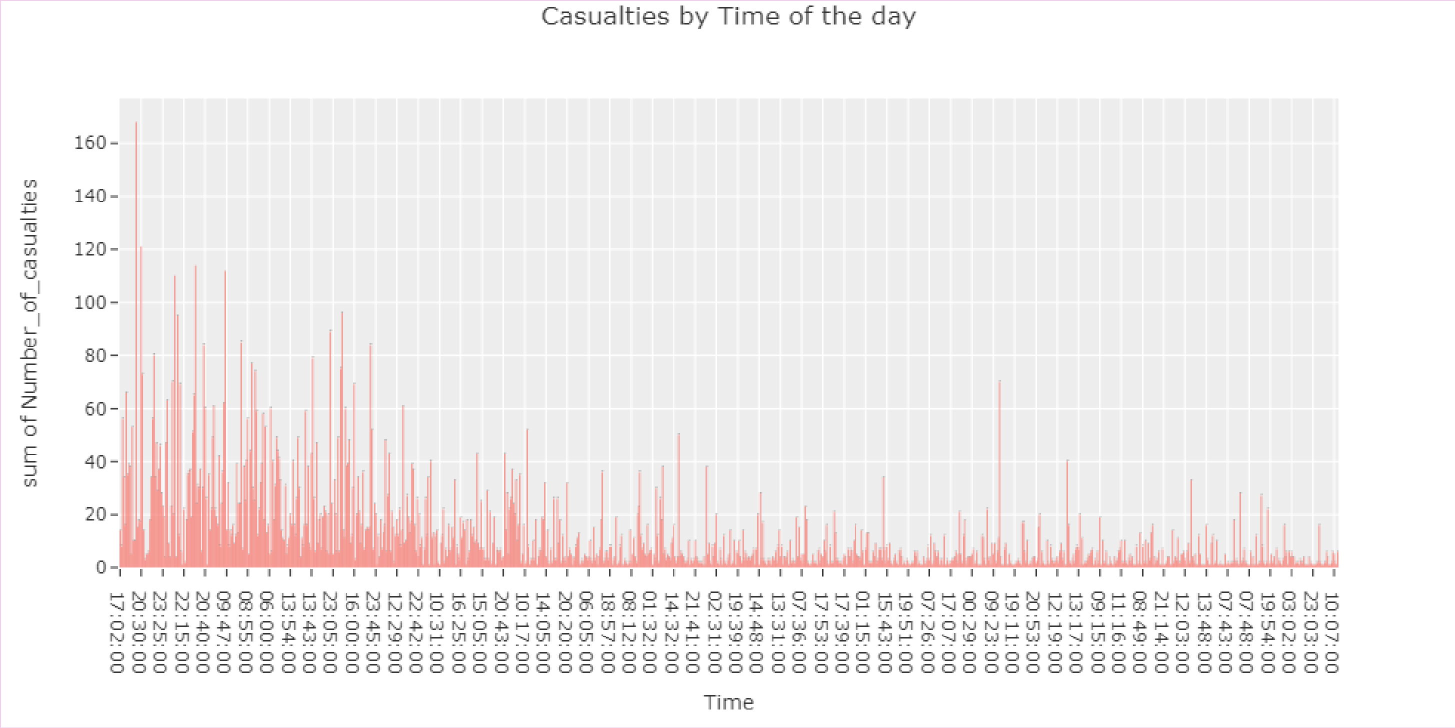




Bar diagram for Casualties by Time of the day

In [21]:
```
fig7=px.histogram(data1,data1['Time'],data1['Number_of_casualties']
       ,template='ggplot2',title='Casualties by Time of the day')
fig7.update_traces(texttemplate='%{y}', textposition='outside')
fig7.show()
```

By Nischay Gautam



Bar diagram above doesn’t help us , so we will create 3-hour interval groups.

By Nischay Gautam

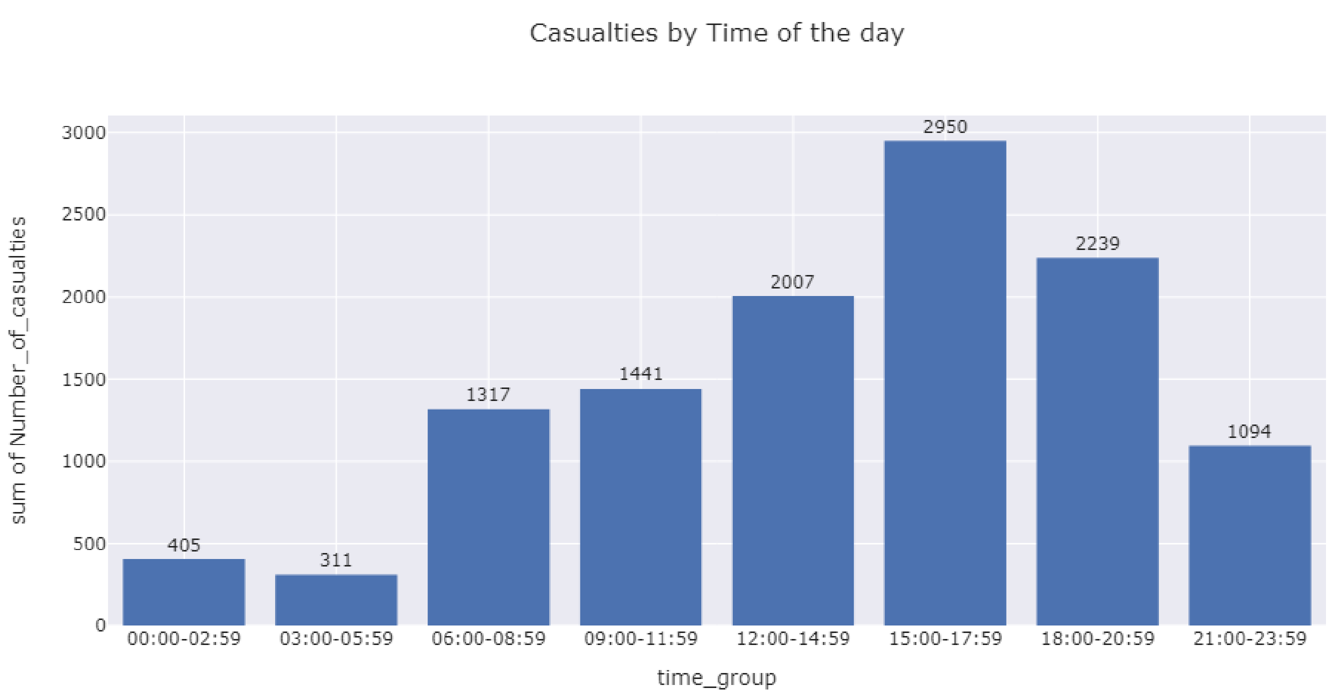
Bar diagram above doesnt help us , so we will create 3-hour interval groups

In [11]:
```python
print(data1['Time'].dtypes)
# Convert time to 3-hour interval groups
bins = pd.to_timedelta(['00:00:00', '03:00:00', '06:00:00', '09:00:00',
                        '12:00:00', '15:00:00', '18:00:00', '21:00:00', '23:59:59'])
labels = ['00:00-02:59', '03:00-05:59', '06:00-08:59', '09:00-11:59',
          '12:00-14:59', '15:00-17:59', '18:00-20:59', '21:00-23:59']
data1['time_group'] = pd.cut(pd.to_datetime(data1['Time'],
                                             format='%H:%M:%S').dt.time.apply(lambda x: pd.to_timedelta(str(x))),
                             bins=bins, labels=labels)
print(data1[['Time', 'time_group']])
```

```
object
         Time   time_group
0     17:02:00  15:00-17:59
1     17:02:00  15:00-17:59
2     17:02:00  15:00-17:59
3     01:06:00  00:00-02:59
4     01:06:00  00:00-02:59
...        ...          ...
```

Bar diagram for Casualties by Time of the day

In [ ]:
```python
# Define the order of time groups
time_group_order = ['00:00-02:59', '03:00-05:59', '06:00-08:59', '09:00-11:59',
          '12:00-14:59', '15:00-17:59', '18:00-20:59', '21:00-23:59']
fig8=px.histogram(data1,data1['time_group'],data1['Number_of_casualties']
       ,template='seaborn',title='Casualties by Time of the day',category_orders={'time_group': time_group_order})
fig8.update_traces(texttemplate='%{y}', textposition='outside')
fig8.show()
```



By Nischay Gautam

Data Pre-processing

In [13]:
```
#making a duplicate dataset
data3=data1.copy(deep=True)
data3.head()
```

Out[13]:

sualty	Age_band_of_casualty	Casualty_severity	Work_of_casuality	Fitness_of_casuality	Pedestrian_movement	Cause_of_accident	Accident_severity	time_group
NaN	NaN	NaN	NaN	NaN	Not a Pedestrian	Moving Backward	Slight Injury	15:00-17:59
NaN	NaN	NaN	NaN	NaN	Not a Pedestrian	Overtaking	Slight Injury	15:00-17:59
Male	31-50	3.0	Driver	NaN	Not a Pedestrian	Changing lane to the left	Serious Injury	15:00-17:59
emale	18-30	3.0	Driver	Normal	Not a Pedestrian	Changing lane to the right	Slight Injury	00:00-02:59
NaN	NaN	NaN	NaN	NaN	Not a Pedestrian	Overtaking	Slight Injury	00:00-02:59

Selecting required columns

In [33]:
```
columns_for_model=["time_group","Accident_severity","Day_of_week","Type_of_vehicle","Road_surface_type","Type_of_collision","Casu
                    "Sex_of_casualty"]
data_for_model=data3[columns_for_model]
data_for_model.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7417 entries, 0 to 7416
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   time_group         7417 non-null   category
 1   Accident_severity  7417 non-null   object
 2   Day_of_week        7417 non-null   object
 3   Type_of_vehicle    6831 non-null   object
 4   Road_surface_type  7314 non-null   object
 5   Type_of_collision  7326 non-null   object
 6   Casualty_class     4675 non-null   object
 7   Sex_of_casualty    4675 non-null   object
dtypes: category(1), object(7)
```

In [34]:
```
data_for_model.head()
```

Out[34]:

	time_group	Accident_severity	Day_of_week	Type_of_vehicle	Road_surface_type	Type_of_collision	Casualty_class	Sex_of_casualty
0	15:00-17:59	Slight Injury	Monday	Automobile	Asphalt roads	Collision with roadside-parked vehicles	NaN	NaN
1	15:00-17:59	Slight Injury	Monday	Public (> 45 seats)	Asphalt roads	Vehicle with vehicle collision	NaN	NaN
2	15:00-17:59	Serious Injury	Monday	Lorry (41-100Q)	Asphalt roads	Collision with roadside objects	Driver or rider	Male
3	00:00-02:59	Slight Injury	Sunday	Public (> 45 seats)	Earth roads	Vehicle with vehicle collision	Pedestrian	Female
4	00:00-02:59	Slight Injury	Sunday	NaN	Asphalt roads	Vehicle with vehicle collision	NaN	NaN

Since data type is object, we need to convert them into nominal type before appling any machine learning technique.

In [35]:
```
#converting object type values to nominal values.
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in data_for_model.columns:
    if data_for_model[col].dtype == object or data_for_model[col].dtype.name == 'category':
        data_for_model[col] = le.fit_transform(data_for_model[col])
```

By Nischay Gautam

In [37]:
```
data_RF=data_for_model.copy(deep=True)
data_RF.head()
data_RF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7417 entries, 0 to 7416
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   time_group         7417 non-null   int32
 1   Accident_severity  7417 non-null   int32
 2   Day_of_week        7417 non-null   int32
 3   Type_of_vehicle    7417 non-null   int32
 4   Road_surface_type  7417 non-null   int32
 5   Type_of_collision  7417 non-null   int32
 6   Casualty_class     7417 non-null   int32
 7   Sex_of_casualty    7417 non-null   int32
dtypes: int32(8)
```

Random-Forest Classifier

In [57]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score,recall_score,f1_score,precision_score

x1=data_RF.drop('Accident_severity',axis=1)
y1=data_RF['Accident_severity']
xtrain, xtest, ytrain, ytest = train_test_split(x1,y1,test_size=0.30)
rf = RandomForestClassifier()
rf.fit(xtrain,ytrain)

ypred=rf.predict(xtest)

print('confusion matrix :',confusion_matrix(ytest,ypred))
print('classification report:',classification_report(ytest,ypred))
print('accuracy :',round(accuracy_score(ytest,ypred),2))
print('precision :',round(precision_score(ytest,ypred,average='weighted'),2))
print('recall :',round(recall_score(ytest,ypred,average='weighted'),2))
print('f1 :',round(f1_score(ytest,ypred,average='weighted'),2))
print()
```

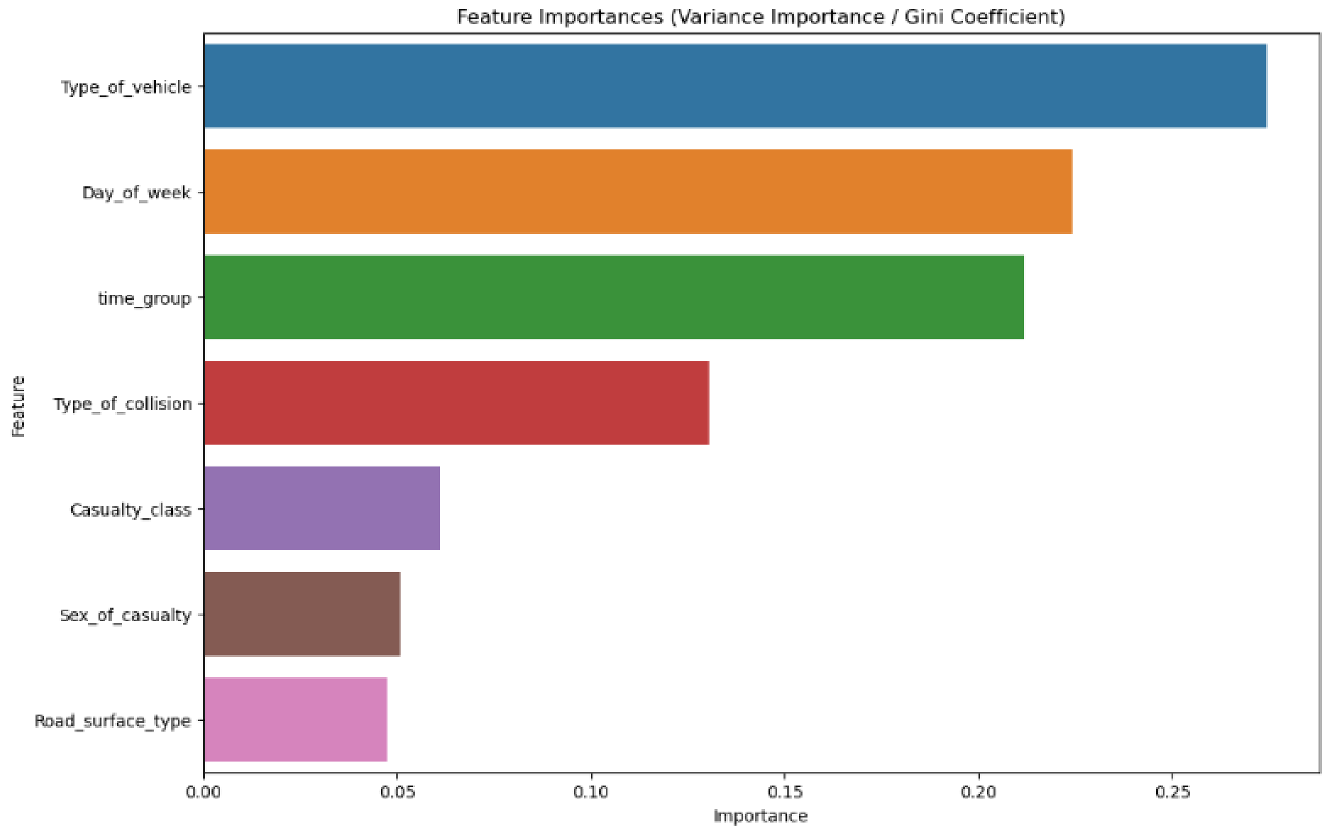
```
confusion matrix : [[   2    1   23]
 [   1   14  294]
 [   1   83 1807]]
classification report:               precision    recall  f1-score   support

           0       0.50      0.08      0.13        26
           1       0.14      0.05      0.07       309
           2       0.85      0.96      0.90      1891

    accuracy                           0.82      2226
   macro avg       0.50      0.36      0.37      2226
weighted avg       0.75      0.82      0.78      2226

accuracy : 0.82
precision : 0.75
recall : 0.82
f1 : 0.78
```

By Nischay Gautam

In [58]:

```
#variance importance and Gini coefficient
feature_importances = rf.feature_importances_
features = x1.columns
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
})

importance_df = importance_df.sort_values(by='Importance', ascending=False)

print('Feature Importances (Variance Importance / Gini Coefficient):')
print(importance_df)

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importances (Variance Importance / Gini Coefficient)')
plt.show()
```

```
Feature Importances (Variance Importance / Gini Coefficient):
            Feature  Importance
2   Type_of_vehicle    0.274236
1       Day_of_week    0.224294
0        time_group    0.211873
4 Type_of_collision    0.130441
5     Casualty_class    0.061067
6    Sex_of_casualty    0.050696
3 Road_surface_type    0.047393
```



By Nischay Gautam

K-Nearest Neighbour

In [41]:
```
data_KNN=data_for_model.copy(deep=True)
data_KNN.head()
data_KNN.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7417 entries, 0 to 7416
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   time_group        7417 non-null   int32
 1   Accident_severity 7417 non-null   int32
 2   Day_of_week       7417 non-null   int32
 3   Type_of_vehicle   7417 non-null   int32
 4   Road_surface_type 7417 non-null   int32
 5   Type_of_collision 7417 non-null   int32
 6   Casualty_class    7417 non-null   int32
 7   Sex_of_casualty   7417 non-null   int32
dtypes: int32(8)
memory usage: 231.9 KB
```

In [61]:
```
from sklearn.neighbors import KNeighborsClassifier
X2 = data_KNN.drop('Accident_severity', axis=1)
y2 = data_KNN['Accident_severity']

X_train X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

In [53]:
```
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)

print('Classification Report:')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8140161725067385
Confusion Matrix:
[[   0    1   13]
 [   0    5  222]
 [   4   36 1203]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        14
           1       0.12      0.02      0.04       227
           2       0.84      0.97      0.90      1243

    accuracy                           0.81      1484
   macro avg       0.32      0.33      0.31      1484
weighted avg       0.72      0.81      0.76      1484
```

By Nischay Gautam

Permutation Importance

In [55]:
```
from sklearn.inspection import permutation_importance
perm_importance = permutation_importance(knn, X_test, y_test, n_repeats=30, random_state=42)

features = data_KNN.drop('Accident_severity', axis=1).columns
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': perm_importance.importances_mean,
    'Standard Deviation': perm_importance.importances_std})

importance_df = importance_df.sort_values(by='Importance', ascending=False)

print('Permutation Feature Importances:')
print(importance_df)
```

```
Permutation Feature Importances:
             Feature  Importance  Standard Deviation
4  Type_of_collision    0.002583            0.003127
5      Casualty_class    0.000022            0.003758
0          time_group   -0.000517            0.003012
6     Sex_of_casualty   -0.000674            0.002723
1         Day_of_week   -0.002156            0.004174
2     Type_of_vehicle   -0.002673            0.004492
3   Road_surface_type   -0.003077            0.001711
```

XGBOOST(eXtreme Gradient Boosting)

In [62]:
```
data_XGB=data_for_model.copy(deep=True)
data_XGB.head()
data_XGB.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7417 entries, 0 to 7416
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   time_group         7417 non-null   int32
 1   Accident_severity  7417 non-null   int32
 2   Day_of_week        7417 non-null   int32
 3   Type_of_vehicle    7417 non-null   int32
 4   Road_surface_type  7417 non-null   int32
 5   Type_of_collision  7417 non-null   int32
 6   Casualty_class     7417 non-null   int32
 7   Sex_of_casualty    7417 non-null   int32
dtypes: int32(8)
memory usage: 231.9 KB
```

In [84]:
```python
import xgboost as xgb
X3 = data_XGB.drop('Accident_severity', axis=1)
y3 = data_XGB['Accident_severity']

X_train, X_test, y_train, y_test = train_test_split(X3, y3, test_size=0.2)

dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

params = {'objective': 'multi:softmax','num_class': 3,'eval_metric': 'mlogloss','eta': 0.3,'max_depth':
          6,'min_child_weight': 1,
          'subsample': 1,'colsample_bytree': 1}
```

In [85]:
```python
num_rounds = 100

evals = [(dtrain, 'train'), (dtest, 'eval')]
model = xgb.train(params, dtrain, num_boost_round=num_rounds, evals=evals)

y_pred = model.predict(dtest)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

print('Classification Report:')
print(classification_report(y_test, y_pred))
```

```
[0]	train-mlogloss:0.86080	eval-mlogloss:0.86569
[1]	train-mlogloss:0.72372	eval-mlogloss:0.73336
[2]	train-mlogloss:0.63806	eval-mlogloss:0.65146
[3]	train-mlogloss:0.58258	eval-mlogloss:0.59957
[4]	train-mlogloss:0.54459	eval-mlogloss:0.56653
[5]	train-mlogloss:0.51835	eval-mlogloss:0.54410
[6]	train-mlogloss:0.49906	eval-mlogloss:0.52861
[7]	train-mlogloss:0.48518	eval-mlogloss:0.51847
[8]	train-mlogloss:0.47384	eval-mlogloss:0.51271
[9]	train-mlogloss:0.46532	eval-mlogloss:0.50776
[10]	train-mlogloss:0.45840	eval-mlogloss:0.50476
[11]	train-mlogloss:0.45302	eval-mlogloss:0.50200
[12]	train-mlogloss:0.44710	eval-mlogloss:0.50073
[13]	train-mlogloss:0.44243	eval-mlogloss:0.49962
```

By Nischay Gautam

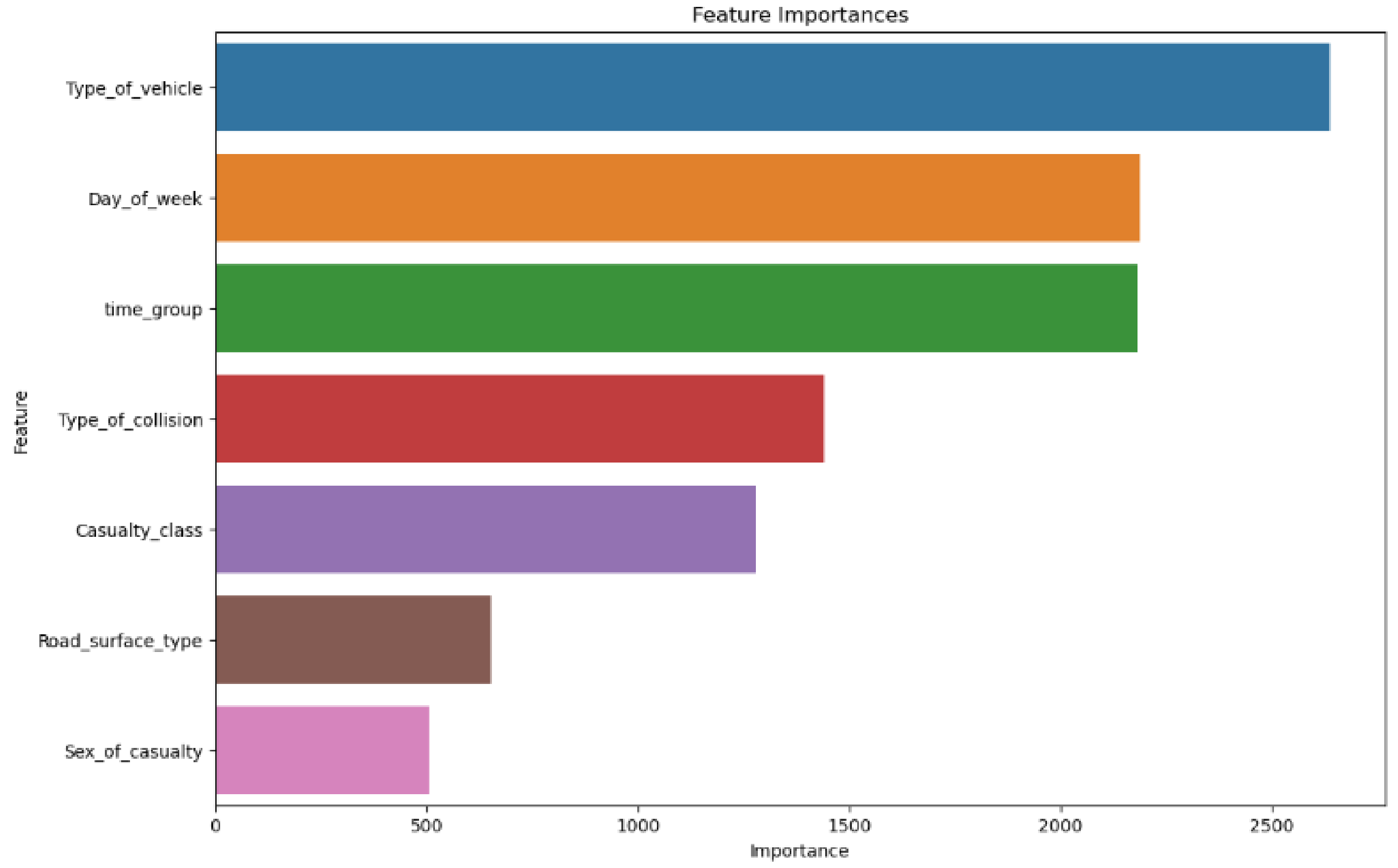
```
[74]	train-mlogloss:0.32925	eval-mlogloss:0.53290
[75]	train-mlogloss:0.32815	eval-mlogloss:0.53410
[76]	train-mlogloss:0.32704	eval-mlogloss:0.53475
[77]	train-mlogloss:0.32613	eval-mlogloss:0.53505
[78]	train-mlogloss:0.32550	eval-mlogloss:0.53565
[79]	train-mlogloss:0.32463	eval-mlogloss:0.53643
[80]	train-mlogloss:0.32377	eval-mlogloss:0.53721
[81]	train-mlogloss:0.32279	eval-mlogloss:0.53733
[82]	train-mlogloss:0.32206	eval-mlogloss:0.53752
[83]	train-mlogloss:0.32132	eval-mlogloss:0.53803
[84]	train-mlogloss:0.32026	eval-mlogloss:0.53924
[85]	train-mlogloss:0.31947	eval-mlogloss:0.54055
[86]	train-mlogloss:0.31894	eval-mlogloss:0.54092
[87]	train-mlogloss:0.31798	eval-mlogloss:0.54206
[88]	train-mlogloss:0.31711	eval-mlogloss:0.54241
[89]	train-mlogloss:0.31605	eval-mlogloss:0.54246
[90]	train-mlogloss:0.31554	eval-mlogloss:0.54292
[91]	train-mlogloss:0.31504	eval-mlogloss:0.54383
[92]	train-mlogloss:0.31425	eval-mlogloss:0.54453
[93]	train-mlogloss:0.31357	eval-mlogloss:0.54524
[94]	train-mlogloss:0.31305	eval-mlogloss:0.54540
[95]	train-mlogloss:0.31254	eval-mlogloss:0.54617
[96]	train-mlogloss:0.31211	eval-mlogloss:0.54659
[97]	train-mlogloss:0.31154	eval-mlogloss:0.54674
[98]	train-mlogloss:0.31009	eval-mlogloss:0.54709
[99]	train-mlogloss:0.30893	eval-mlogloss:0.54801
Accuracy: 0.8268194070080862
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.09      0.15        23
           1       0.17      0.02      0.03       220
           2       0.84      0.98      0.91      1241

    accuracy                           0.83      1484
   macro avg       0.56      0.36      0.36      1484
weighted avg       0.74      0.83      0.76      1484
```

By Nischay Gautam

Feature Importance

In [86]:
```
importance = model.get_score(importance_type='weight')
importance_df = pd.DataFrame({
    'Feature': list(importance.keys()),
    'Importance': list(importance.values())
})

# Sort the DataFrame by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

print('Feature Importances:')
print(importance_df)

# Plot feature importances
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importances')
plt.show()
```

```
Feature Importances:
             Feature  Importance
2    Type_of_vehicle      2636.0
1        Day_of_week      2187.0
0         time_group      2182.0
4  Type_of_collision      1440.0
5     Casualty_class      1281.0
3  Road_surface_type       652.0
6    Sex_of_casualty       505.0
```



By Nischay Gautam