

Kaggle Playground

Problem Statement / Real World Implementations

Examine the challenge of predicting the risk of road accidents from tabular data. Real-world implications include public safety, insurance risk analysis, and smart city traffic management. Accurate forecasting helps optimize emergency response and reduce accident rates

1. Importing Libraries

```
In [1]: # Core Data Science Libraries
import numpy as np
import pandas as pd
import warnings

# Visualization Libraries
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Scikit-Learn for Preprocessing and Modeling
from sklearn.model_selection import KFold, train_test_split
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Machine Learning Models
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import (
    RandomForestRegressor,
    ExtraTreesRegressor,
    AdaBoostRegressor,
    GradientBoostingRegressor,
    BaggingRegressor
)
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

# Hyperparameter Tuning
import optuna

# Notebook settings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
```

2. Loading Dataset

```
In [2]: # Define file paths
TRAIN_PATH = "/kaggle/input/playground-series-s5e10/train.csv"
TEST_PATH = "/kaggle/input/playground-series-s5e10/test.csv"
SUBMISSION_PATH = "/kaggle/input/playground-series-s5e10/sample_submission.csv"

# Load the datasets into pandas DataFrames
train_df = pd.read_csv(TRAIN_PATH)
test_df = pd.read_csv(TEST_PATH)
submission_df = pd.read_csv(SUBMISSION_PATH)
```

```
In [3]: print("Train shape:", train_df.shape)
print("Test shape:", test_df.shape)
```

Train shape: (517754, 14)
Test shape: (172585, 13)

```
In [4]: df=train_df
df.head(5)
```

```
Out[4]:
```

	id	road_type	num_lanes	curvature	speed_limit	lighting	weather	road_signs_present
0	0	urban	2	0.06	35	daylight	rainy	False
1	1	urban	4	0.99	35	daylight	clear	True
2	2	rural	4	0.63	70	dim	clear	False
3	3	highway	4	0.07	35	dim	rainy	True
4	4	rural	1	0.58	60	daylight	foggy	False

```
In [5]: print(df["road_type"].unique())
print(df["lighting"].unique())
print(df["weather"].unique())
print(df["time_of_day"].unique())
```

```
['urban' 'rural' 'highway']
['daylight' 'dim' 'night']
['rainy' 'clear' 'foggy']
['afternoon' 'evening' 'morning']
```

```
In [6]: df.isna().sum()
```

```
Out[6]: id          0
road_type         0
num_lanes         0
curvature         0
speed_limit       0
lighting          0
weather           0
road_signs_present 0
public_road       0
time_of_day       0
holiday           0
school_season     0
num_reported_accidents 0
accident_risk     0
dtype: int64
```

```
In [7]: df.head()
```

```
Out[7]:
```

	id	road_type	num_lanes	curvature	speed_limit	lighting	weather	road_signs_present
0	0	urban	2	0.06	35	daylight	rainy	False
1	1	urban	4	0.99	35	daylight	clear	True
2	2	rural	4	0.63	70	dim	clear	False
3	3	highway	4	0.07	35	dim	rainy	True
4	4	rural	1	0.58	60	daylight	foggy	False

4. EDA

```
In [13]: # Select only numeric columns for correlation matrix
numeric_df = train_df[numerical_cols + ['accident_risk']]
corr_matrix = numeric_df.corr()

# Create the interactive heatmap
fig = go.Figure(data=go.Heatmap(
    z=corr_matrix.values,
    x=corr_matrix.columns,
    y=corr_matrix.columns,
    colorscale='RdBu_r',
    zmin=-1, zmax=1,
    text=corr_matrix.round(2).values,
    texttemplate="%{text}",
    hoverongaps=False))

fig.update_layout(
    title='Correlation Heatmap of Numerical Features',
    width=800, height=800
)
fig.show()
```

3. Normalization of data

```
In [14]: def encode_features(df):  
          df_encoded = df.copy()  
  
          # Boolean to integer  
          for col in df_encoded.select_dtypes(include='bool').columns:
```

```

df_encoded[col] = df_encoded[col].astype(int)

# Categorical to integer
categorical_cols = df_encoded.select_dtypes(include='object').columns
if len(categorical_cols) > 0:
    encoder = OrdinalEncoder()
    df_encoded[categorical_cols] = encoder.fit_transform(df_encoded[categorical_cols])

return df_encoded

train_ids = train_df['id']
test_ids = test_df['id']

train_processed = encode_features(train_df.drop('id', axis=1))
test_processed = encode_features(test_df.drop('id', axis=1))

```

In [15]: df.head(5)

Out[15]:

	id	road_type	num_lanes	curvature	speed_limit	lighting	weather	road_signs_present
0	0	urban	2	0.06	35	daylight	rainy	False
1	1	urban	4	0.99	35	daylight	clear	True
2	2	rural	4	0.63	70	dim	clear	False
3	3	highway	4	0.07	35	dim	rainy	True
4	4	rural	1	0.58	60	daylight	foggy	False

In [16]:

```

# Exclude target column if present
features = train_processed.drop(columns=['accident_risk'], errors='ignore')

# 1. Check summary statistics
print("Summary Statistics:\n")
display(features.describe())

# 2. Check for large differences in scale
range_df = features.max() - features.min()
print("\nFeature Ranges:\n")
print(range_df.sort_values(ascending=False))

# 3. Visualize distribution of feature scales
plt.figure(figsize=(10, 6))
sns.boxplot(data=features, orient='h', fliersize=1)
plt.title("Feature Value Distributions (Check for Scale Differences)")
plt.show()

# 4. Correlation check
corr_matrix = features.corr()
high_range_features = range_df[range_df > range_df.mean()].index.tolist()
print(f"\nFeatures with significantly higher ranges: {high_range_features}")

# 5. Quick rule-based decision
if range_df.max() / range_df.min() > 10:
    print("\n✅ Feature scaling is likely necessary (large scale differences detected)")

```

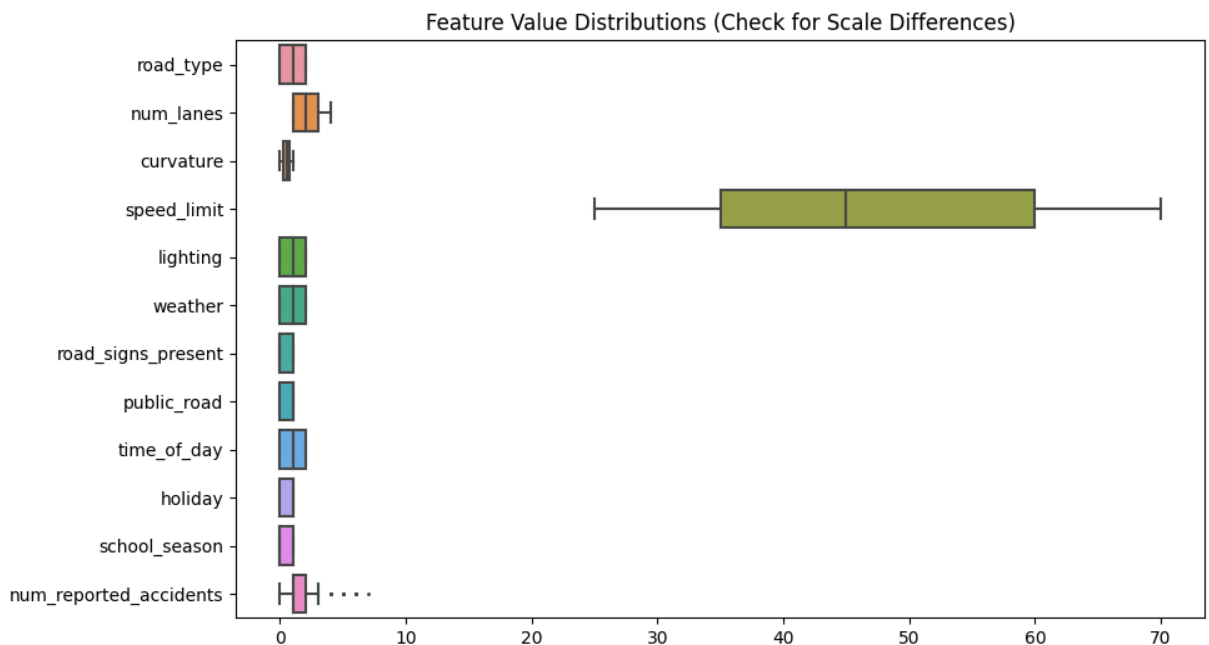
```
else:
    print("\n✗ Feature scaling might not be strictly necessary (features on similar scales)")
```

Summary Statistics:

	road_type	num_lanes	curvature	speed_limit	lighting	weather
count	517754.000000	517754.000000	517754.000000	517754.000000	517754.000000	517754.000000
mean	0.995540	2.491511	0.488719	46.112575	0.957312	0.957312
std	0.816326	1.120434	0.272563	15.788521	0.801956	0.801956
min	0.000000	1.000000	0.000000	25.000000	0.000000	0.000000
25%	0.000000	1.000000	0.260000	35.000000	0.000000	0.000000
50%	1.000000	2.000000	0.510000	45.000000	1.000000	1.000000
75%	2.000000	3.000000	0.710000	60.000000	2.000000	2.000000
max	2.000000	4.000000	1.000000	70.000000	2.000000	2.000000

Feature Ranges:

```
speed_limit          45.0
num_reported_accidents 7.0
num_lanes             3.0
road_type             2.0
time_of_day           2.0
lighting              2.0
weather               2.0
curvature             1.0
public_road           1.0
road_signs_present    1.0
holiday               1.0
school_season         1.0
dtype: float64
```



Features with significantly higher ranges: ['speed_limit', 'num_reported_accidents']

✅ Feature scaling is likely necessary (large scale differences detected).

Train test split

```
In [17]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler, Power
import numpy as np

# Use encoded data for model training
X = train_processed.drop("accident_risk", axis=1)
y = train_processed["accident_risk"]

# Ensure all columns are numeric
X = X.select_dtypes(include=[np.number])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Choose scaling method
selected_method = 'Standard Scaling'

# Apply the best scaling method
if selected_method == 'Min-Max Scaling':
    scaler = MinMaxScaler()
elif selected_method == 'Standard Scaling':
    scaler = StandardScaler()
elif selected_method == 'Robust Scaling':
    scaler = RobustScaler()
elif selected_method == 'Power Transformation':
    scaler = PowerTransformer(method='yeo-johnson')
else:
    scaler = None # Log or Decimal handled separately

# Perform scaling
if scaler is not None:
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
elif selected_method == 'Log Transformation':
    X_train_scaled = np.log1p(X_train.clip(lower=1e-6))
    X_test_scaled = np.log1p(X_test.clip(lower=1e-6))
elif selected_method == 'Decimal Scaling':
    X_train_scaled = X_train / 100.0
    X_test_scaled = X_test / 100.0
else:
    X_train_scaled = X_train
    X_test_scaled = X_test
```

```
In [18]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Initialize models
models = [
```

```

DecisionTreeRegressor(),
RandomForestRegressor(),
XGBRegressor(),
AdaBoostRegressor(),
KNeighborsRegressor(),
GradientBoostingRegressor(),
LGBMRegressor(),
BaggingRegressor(),
ExtraTreesRegressor()
]

print("🔍 Evaluating Models...\n")
mse_scores = []

for model in models:
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(mse)
    print(f"{model.__class__.__name__:<30} MSE: {mse:.5f}")

# Select best model
best_model_default = models[np.argmin(mse_scores)]
print("\n✅ Best Model Based on MSE:", best_model_default.__class__.__name__)

```

🔍 Evaluating Models...

```

DecisionTreeRegressor      MSE: 0.00691
RandomForestRegressor      MSE: 0.00355
XGBRegressor               MSE: 0.00317
AdaBoostRegressor          MSE: 0.00680
KNeighborsRegressor        MSE: 0.00453
GradientBoostingRegressor  MSE: 0.00325
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.008339 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 205
[LightGBM] [Info] Number of data points in the train set: 414203, number of used features: 12
[LightGBM] [Info] Start training from score 0.352605
LGBMRegressor              MSE: 0.00318
BaggingRegressor           MSE: 0.00386
ExtraTreesRegressor        MSE: 0.00389

```

✅ Best Model Based on MSE: XGBRegressor

In [19]:

```


# Evaluate final model
y_pred = best_model_default.predict(X_test_scaled)

mse_default = mean_squared_error(y_test, y_pred)
mae_default = mean_absolute_error(y_test, y_pred)
r2_default = r2_score(y_test, y_pred)

print("\n📊 Final Model Evaluation:")
print(f"Mean Squared Error : {mse_default:.5f}")

```

```
print(f"Mean Absolute Error: {mae_default:.5f}")
print(f"R² Score           : {r2_default:.5f}")
```

 Final Model Evaluation:
Mean Squared Error : 0.00317
Mean Absolute Error: 0.04370
R² Score : 0.88522

Selecting best model and Generating Submission

```
In [20]: print("\n🚀 Retraining the best model on full training data...")

# Prepare full training features and target
X_full = train_processed.drop(columns=['accident_risk'], errors='ignore')
y_full = train_processed['accident_risk']

# Ensure all columns are numeric
X_full = X_full.select_dtypes(include=[np.number])

# Scale full data using the same scaler
if scaler is not None:
    X_full_scaled = scaler.fit_transform(X_full)
else:
    X_full_scaled = X_full

# Retrain best model on the full scaled dataset
best_model_default.fit(X_full_scaled, y_full)

print(f"✅ Model retrained successfully: {best_model_default.__class__.__name__}\n")

🚀 Retraining the best model on full training data...
✅ Model retrained successfully: XGBRegressor
```

```
In [21]: # Keep IDs for submission if available
if 'id' in test_df.columns:
    test_ids = test_df['id']
else:
    test_ids = range(len(test_df)) # create sequential IDs if missing

# Encode test data (using your encode_features function)
test_processed = encode_features(test_df.drop('id', axis=1, errors='ignore'))

# Ensure numeric columns only
X_submission = test_processed.select_dtypes(include=[np.number])

# Scale using the same scaler
if scaler is not None:
    X_submission_scaled = scaler.transform(X_submission)
else:
    X_submission_scaled = X_submission
```

```
In [22]: print("🤖 Generating predictions using the best model...")
submission_preds = best_model_default.predict(X_submission_scaled)
```

```
# Optional: clip predictions to valid range [0, 1]
submission_preds = np.clip(submission_preds, 0, 1)
```

👉 Generating predictions using the best model...

```
In [23]: submission = pd.DataFrame({
        'id': test_ids,
        'accident_risk': submission_preds
    })

submission.to_csv('submission.csv', index=False)

print("\n✅ Submission file 'submission.csv' generated successfully!")
display(submission.head())
```

✅ Submission file 'submission.csv' generated successfully!

	id	accident_risk
0	517754	0.293325
1	517755	0.120547
2	517756	0.186391
3	517757	0.306890
4	517758	0.408361

```
In [24]: plt.figure(figsize=(8, 5))
sns.histplot(submission['accident_risk'], bins=30, kde=True)
plt.title('Distribution of Predicted Accident Risk')
plt.xlabel('Accident Risk')
plt.ylabel('Frequency')
plt.show()
```

Distribution of Predicted Accident Risk

