```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import warnings
warnings.filterwarnings("ignore")
```

```
dataset = pd.read_csv(r"C:\Users\afroz\OneDrive\Desktop\smartbridge\traffic volume.csv")
```

```
dataset.head()
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
0	NaN	288.28	0.0	0.0	Clouds	02-10-2012	09:00:00	5545
1	NaN	289.36	0.0	0.0	Clouds	02-10-2012	10:00:00	4516
2	NaN	289.58	0.0	0.0	Clouds	02-10-2012	11:00:00	4767
3	NaN	290.13	0.0	0.0	Clouds	02-10-2012	12:00:00	5026
4	NaN	291.14	0.0	0.0	Clouds	02-10-2012	13:00:00	4918

```
dataset.iloc[35764]
```

```
holiday                 NaN
temp                 292.95
rain                    0.0
snow                    0.0
weather               Clear
date             28-07-2017
Time               23:00:00
traffic_volume         2488
Name: 35764, dtype: object
```

```
dataset.shape
```

```
(48204, 8)
```

```
dataset['holiday'].value_counts()
```

```
holiday
Labor Day                    7
Christmas Day                6
Thanksgiving Day             6
Martin Luther King Jr Day    6
New Years Day                6
Veterans Day                 5
Columbus Day                 5
Memorial Day                 5
Washingtons Birthday         5
State Fair                   5
Independence Day             5
Name: count, dtype: int64
```

```
dataset['holiday'] = dataset['holiday'].fillna('None')
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48204 entries, 0 to 48203
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   holiday         48204 non-null  object
 1   temp            48151 non-null  float64
 2   rain            48202 non-null  float64
 3   snow            48192 non-null  float64
```

```
 4   weather         48155 non-null  object
 5   date            48204 non-null  object
 6   Time            48204 non-null  object
 7   traffic_volume  48204 non-null  int64
dtypes: float64(3), int64(1), object(4)
memory usage: 2.9+ MB
```

Handling Missing values

```
dataset.isna().sum()
```

```
holiday           0
temp             53
rain              2
snow             12
weather          49
date              0
Time              0
traffic_volume    0
dtype: int64
```

```
#Dropping Holiday columns
```

```
# dataset.drop(columns='holiday',inplace=True)
```

```
dataset.head()
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
0	None	288.28	0.0	0.0	Clouds	02-10-2012	09:00:00	5545
1	None	289.36	0.0	0.0	Clouds	02-10-2012	10:00:00	4516
2	None	289.58	0.0	0.0	Clouds	02-10-2012	11:00:00	4767
3	None	290.13	0.0	0.0	Clouds	02-10-2012	12:00:00	5026
4	None	291.14	0.0	0.0	Clouds	02-10-2012	13:00:00	4918

```
dataset['weather'].mode()
```

```
0    Clouds
Name: weather, dtype: object
```

```
dataset['weather'].value_counts()
```

```
weather
Clouds          15144
Clear           13383
Mist             5942
Rain             5665
Snow             2875
Drizzle          1818
Haze             1359
Thunderstorm     1033
Fog               912
Smoke              20
Squall              4
Name: count, dtype: int64
```

```
dataset['weather'].fillna('Clouds',inplace=True)
```

```
dataset.sample()
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
22856	None	276.23	0.42	0.0	Drizzle	28-04-2016	07:00:00	6154

```
num_col=['temp','rain','snow']
```

```
for i in num_col:
    dataset[i]=dataset[i].fillna(dataset[i].mean())
```

```
dataset.isna().sum()
```

```
holiday           0
temp              0
rain              0
snow              0
weather           0
date              0
Time              0
traffic_volume    0
dtype: int64
```

```
dataset.describe()
```

	temp	rain	snow	traffic_volume
count	48204.000000	48204.000000	48204.000000	48204.000000
mean	281.205351	0.334278	0.000222	3259.818355
std	13.336338	44.789133	0.008168	1986.860670
min	0.000000	0.000000	0.000000	0.000000
25%	272.180000	0.000000	0.000000	1193.000000
50%	282.429000	0.000000	0.000000	3380.000000
75%	291.800000	0.000000	0.000000	4933.000000
max	310.070000	9831.300000	0.510000	7280.000000

Label Encoding weather and holiday Columns

This code applies Label Encoding to the weather and holiday columns in the dataset. Each unique category in these columns is converted into an integer value.

weather: Converts weather types like Clear, Clouds, etc., into numbers.

holiday: Converts holiday labels (like None, New Year's Day, etc.) into numbers.

Helps prepare categorical features for machine learning models.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
dataset['weather'] = le.fit_transform(dataset['weather'])
dataset['holiday']=le.fit_transform(dataset['holiday'])
```

```
dataset.head(10)
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
0	7	288.28	0.0	0.0	1	02-10-2012	09:00:00	5545
1	7	289.36	0.0	0.0	1	02-10-2012	10:00:00	4516
2	7	289.58	0.0	0.0	1	02-10-2012	11:00:00	4767
3	7	290.13	0.0	0.0	1	02-10-2012	12:00:00	5026
4	7	291.14	0.0	0.0	1	02-10-2012	13:00:00	4918
5	7	291.72	0.0	0.0	0	02-10-2012	14:00:00	5181
6	7	293.17	0.0	0.0	0	02-10-2012	15:00:00	5584
7	7	293.86	0.0	0.0	0	02-10-2012	16:00:00	6015
8	7	294.14	0.0	0.0	1	02-10-2012	17:00:00	5791
9	7	293.10	0.0	0.0	1	02-10-2012	18:00:00	4770

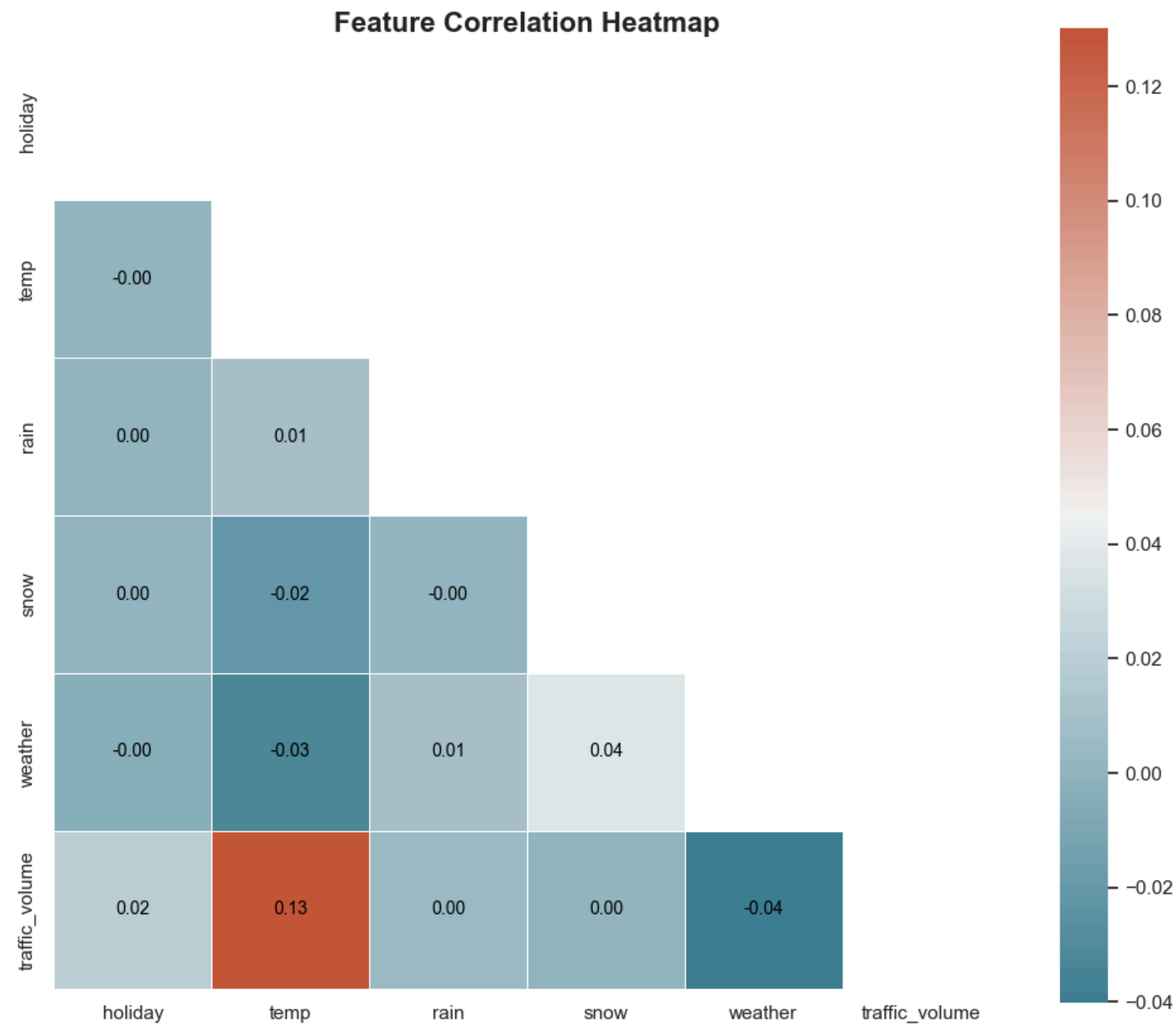
Data Visualization

```
numeric_data = dataset.select_dtypes(include='number')
numeric_data.corr()
```

	holiday	temp	rain	snow	weather	traffic_volume
holiday	1.000000	-0.000472	0.000066	0.000432	-0.004328	0.018676
temp	-0.000472	1.000000	0.009070	-0.019758	-0.033559	0.130034
rain	0.000066	0.009070	1.000000	-0.000090	0.009542	0.004714
snow	0.000432	-0.019758	-0.000090	1.000000	0.036662	0.000735
weather	-0.004328	-0.033559	0.009542	0.036662	1.000000	-0.040035
traffic_volume	0.018676	0.130034	0.004714	0.000735	-0.040035	1.000000

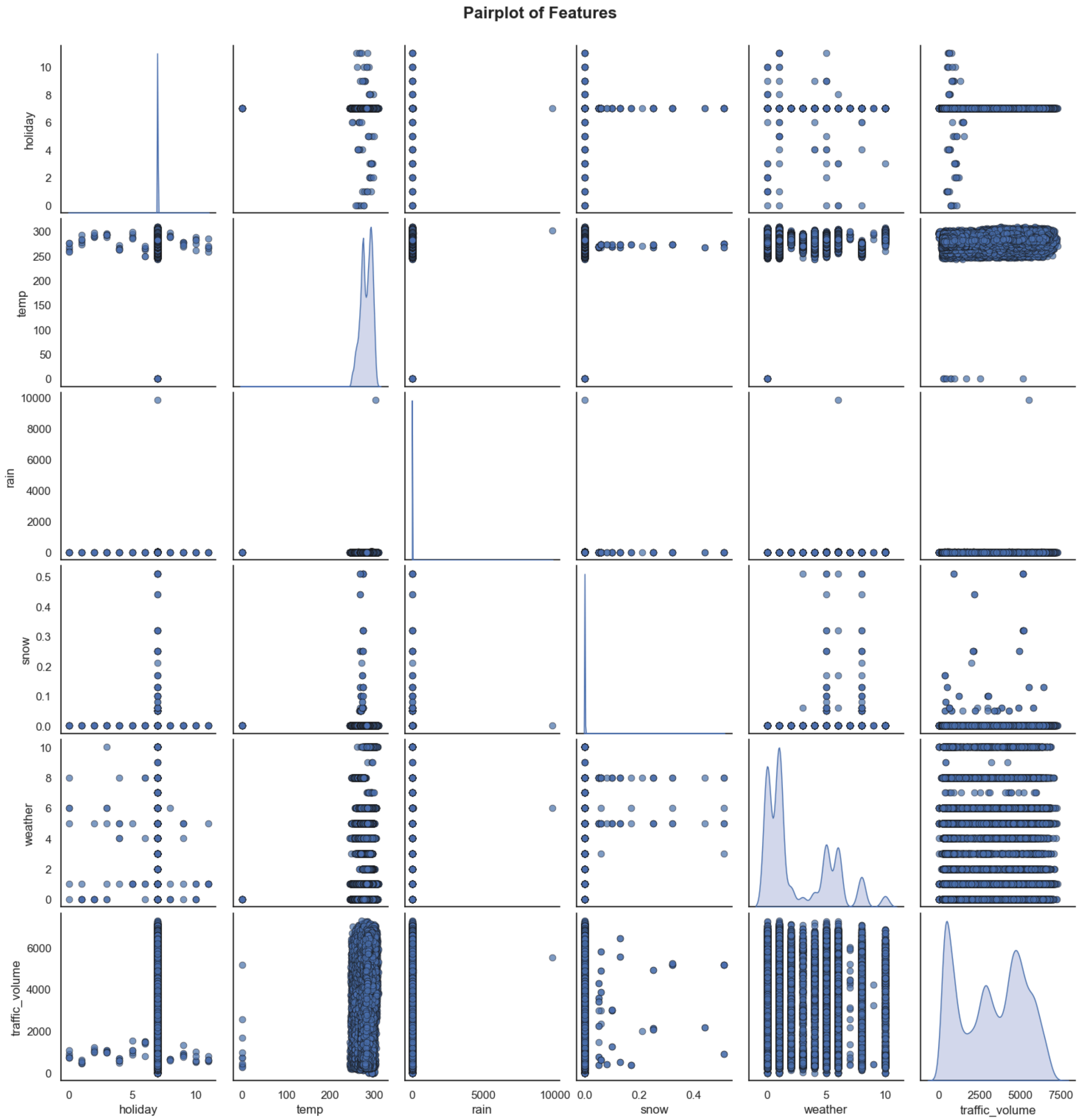
correltion

```
# Set the plot size and style
plt.figure(figsize=(10, 10))
sns.set(style="white")
numeric_data = dataset.select_dtypes(include='number')
mask = np.triu(np.ones_like(numeric_data.corr(), dtype=bool))
# Create a custom diverging colormap
cmap = sns.diverging_palette(220, 20, as_cmap=True)

# Draw the heatmap
sns.heatmap(numeric_data.corr(),
            mask=mask,
            cmap=cmap,
            annot=True,      # Show correlation values
            fmt=".2f",       # 2 decimal places
            square=True,
            linewidths=.5,
            cbar_kws={"shrink": 0.8},
            annot_kws={"size": 10, "color": "black"})

# Title
plt.title("Feature Correlation Heatmap", fontsize=16, fontweight='bold')

plt.tight_layout()
plt.show()
```



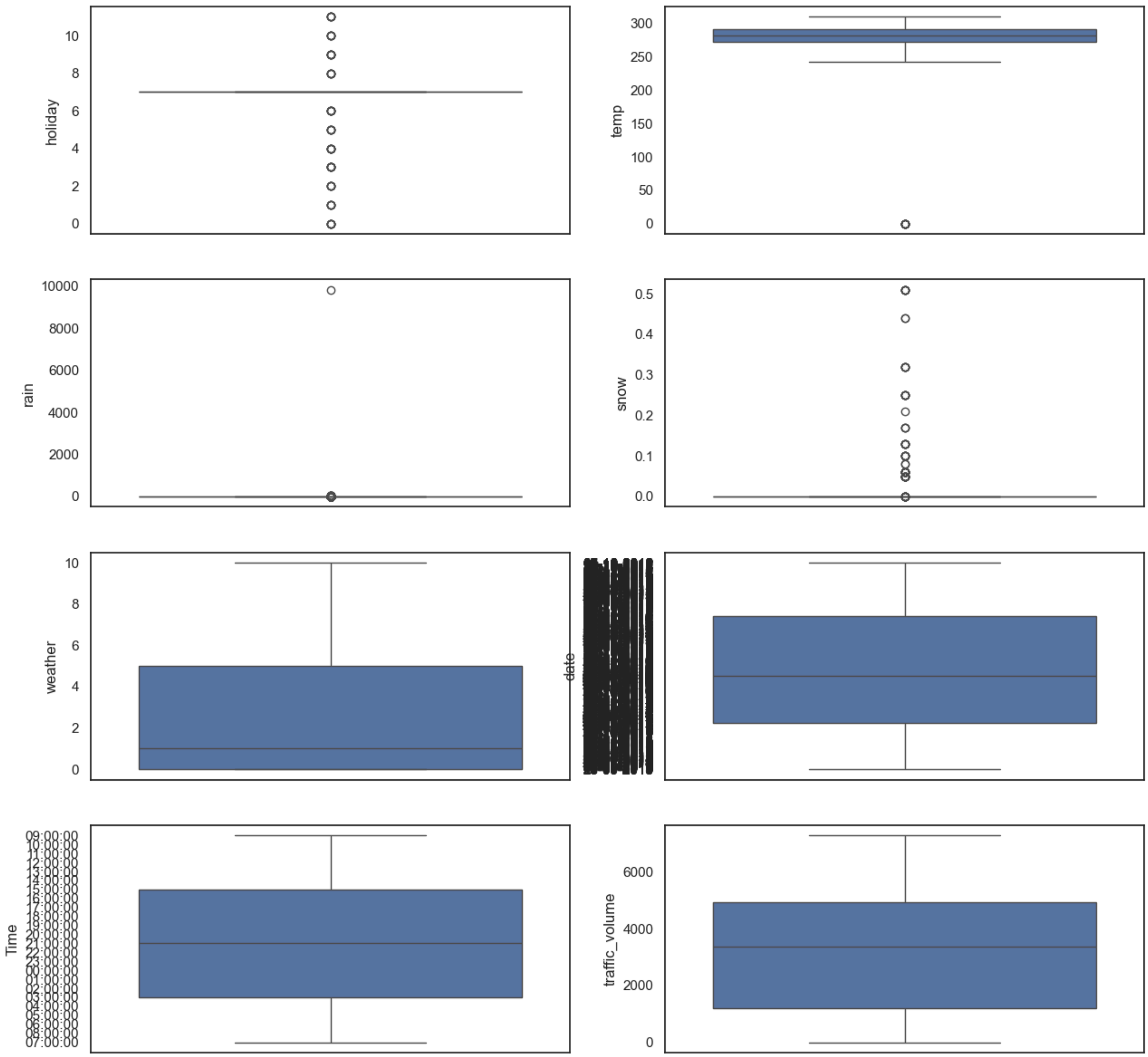
Pair Plot:

```
# Create pairplot
sns.pairplot(dataset,
             diag_kind='kde',         # Smooth KDE on diagonal

             plot_kws={'alpha': 0.7, 's': 40, 'edgecolor': 'k'},  # Scatter settings
             height=2.5)              # Control size of plots

plt.suptitle("Pairplot of Features", fontsize=16, fontweight='bold', y=1.02)
plt.show()
```


Pairplot of Features
holiday
temp
rain
snow
weather
traffic_volume
0
5
10
100
200
300
5000
10000
0.0
0.1
0.2
0.3
0.4
0.5
2
4
6
8
50
150
250
2000
4000
6000
8000
2500
7500

Boxplot

```
ig, axes = plt.subplots(4, 2, figsize=(15, 15))
axes = axes.flatten()
i=0
for c in dataset:
    sns.boxplot(dataset[c] ,ax=axes[i])
    i+=1
```




```
Start coding or generate with AI.
```

```
#splitting the date column into year,month,day
dataset[['day','month','year']] = dataset['date'].str.split('-',expand=True)


#splitting the date column into hours minutes seconds
dataset[['hours', 'minutes', 'seconds']] = dataset['Time'].str.split(":",expand=True)


# Dropping date Time columns
dataset.drop(columns=['date','Time'],inplace=True)
```

```
dataset.head()
```

	holiday	temp	rain	snow	weather	traffic_volume	day	month	year	hours	minutes	seconds
0	7	288.28	0.0	0.0	1	5545	02	10	2012	09	00	00
1	7	289.36	0.0	0.0	1	4516	02	10	2012	10	00	00
2	7	289.58	0.0	0.0	1	4767	02	10	2012	11	00	00
3	7	290.13	0.0	0.0	1	5026	02	10	2012	12	00	00
4	7	291.14	0.0	0.0	1	4918	02	10	2012	13	00	00

```
#Splitting the Dataset into Dependent and Independent variable
x = dataset.drop('traffic_volume',axis=1)
y = dataset['traffic_volume']
```

```
x.head()
```

	holiday	temp	rain	snow	weather	day	month	year	hours	minutes	seconds
0	7	288.28	0.0	0.0	1	02	10	2012	09	00	00
1	7	289.36	0.0	0.0	1	02	10	2012	10	00	00
2	7	289.58	0.0	0.0	1	02	10	2012	11	00	00
3	7	290.13	0.0	0.0	1	02	10	2012	12	00	00
4	7	291.14	0.0	0.0	1	02	10	2012	13	00	00

```
# column_names = ['holiday', 'temp', 'rain', 'snow', 'weather', 'day', 'month', 'year',
#                          'hours', 'minutes', 'seconds']
```

```
x.shape,y.shape
```

```
((48204, 11), (48204,))
```

```
y.head()
```

```
0    5545
1    4516
2    4767
3    5026
4    4918
Name: traffic_volume, dtype: int64
```

Start coding or generate with AI.

Feature Scaling

```
name=x.columns
name
```

```
Index(['holiday', 'temp', 'rain', 'snow', 'weather', 'day', 'month', 'year',
       'hours', 'minutes', 'seconds'],
      dtype='object')
```

```
from sklearn.preprocessing import scale
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x = scaler.fit_transform(x)
x = pd.DataFrame(x, columns=name)
```

```
x.head()
```

	holiday	temp	rain	snow	weather	day	month	year	hours	minutes	seconds
0	0.015856	0.530485	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.345548	0.0	0.0
1	0.015856	0.611467	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.201459	0.0	0.0
2	0.015856	0.627964	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.057371	0.0	0.0
3	0.015856	0.669205	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	0.086718	0.0	0.0
4	0.015856	0.744939	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	0.230807	0.0	0.0

Splitting the Data into Train and Test

```
# for i in range(1,100):
#     x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=i)
#     model=RandomForestRegressor()
#     model.fit(x_train,y_train)
#     # print(f'{i}-- {classification_report(y_test, y_pred)}')
#     print(f'{i}--{r2_score(y_train,train3)}')
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

Hyper Tunning

```
# from sklearn.ensemble import RandomForestRegressor
# from sklearn.model_selection import GridSearchCV

# # Define parameter grid
# param_grid = {
#     'n_estimators': [100, 200],
#     'max_depth': [10, 20, None],
#     'min_samples_split': [2, 5],
#     'min_samples_leaf': [1, 2]
# }

# # Initialize the model
# rf = RandomForestRegressor(random_state=42)

# # Grid Search with cross-validation
# grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
#                            cv=3, scoring='r2', n_jobs=-1, verbose=1)

# grid_search.fit(x_train, y_train)

# # Best model
# best_rf = grid_search.best_estimator_
# print("Best Parameters:", grid_search.best_params_)
```

```
# y_pred_rf=best_rf.predict(x_train)
# r2_=r2_score(y_train,y_pred_rf)
# print(r2_)
```

```
# y_pred_rf=best_rf.predict(x_test)
# r2_=r2_score(y_test,y_pred_rf)
# print(r2_)
```

Traing and Testing and Model

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR


lr = LinearRegression()
Dtree = DecisionTreeRegressor()
Rand = RandomForestRegressor()
svr = SVR()
XGB = xgboost.XGBRFRegressor()


# Fit the models with x_tain and y_train
lr.fit(x_train,y_train)
Dtree.fit(x_train,y_train)
Rand.fit(x_train,y_train)
svr.fit(x_train,y_train)
XGB.fit(x_train,y_train)
```

```
XGBRFRegressor

XGBRFRegressor(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bytree=None, device=None,
               early_stopping_rounds=None, enable_categorical=False,
               eval_metric=None, feature_types=None, feature_weights=None,
               gamma=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               multi_strategy=None, n_estimators=None, n_jobs=None,
               num_parallel_tree=None, objective='reg:squarederror',
```

```
x_test.shape
```

```
(9641, 11)
```

```
#predict the y_tain value and caculate the accuracy
train1 = lr.predict(x_train)
train2 = Dtree.predict(x_train)
train3 = Rand.predict(x_train)
train4 = svr.predict(x_train)
train5 = XGB.predict(x_train)
```

```
#x
```

	holiday	temp	rain	snow	weather	day	month	year	hours	minutes	seconds
0	0.015856	0.530485	-0.007463	-0.027235	-0.566452	-1.574903	1.027580	-1.855294	-0.345548	0.0	0.0
1	0.015856	0.611467	-0.007463	-0.027235	-0.566452	-1.574903	1.027580	-1.855294	-0.201459	0.0	0.0
2	0.015856	0.627964	-0.007463	-0.027235	-0.566452	-1.574903	1.027580	-1.855294	-0.057371	0.0	0.0
3	0.015856	0.669205	-0.007463	-0.027235	-0.566452	-1.574903	1.027580	-1.855294	0.086718	0.0	0.0
4	0.015856	0.744939	-0.007463	-0.027235	-0.566452	-1.574903	1.027580	-1.855294	0.230807	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
48199	0.015856	0.168313	-0.007463	-0.027235	-0.566452	1.635058	0.733478	1.313958	1.095340	0.0	0.0
48200	0.015856	0.116574	-0.007463	-0.027235	-0.566452	1.635058	0.733478	1.313958	1.239428	0.0	0.0
48201	0.015856	0.114324	-0.007463	-0.027235	2.666935	1.635058	0.733478	1.313958	1.383517	0.0	0.0
48202	0.015856	0.066334	-0.007463	-0.027235	-0.566452	1.635058	0.733478	1.313958	1.527606	0.0	0.0
48203	0.015856	0.068584	-0.007463	-0.027235	-0.566452	1.635058	0.733478	1.313958	1.671695	0.0	0.0

48204 rows × 11 columns

```
#dataset.iloc[35764]
```

```
holiday             7
temp           292.95
```

```
rain                0.0
snow                0.0
weather               0
traffic_volume     2488
day                  28
month                07
year               2017
hours                23
minutes              00
seconds              00
Name: 35764, dtype: object
```

```
x_train.head()
```

	holiday	temp	rain	snow	weather	day	month	year	hours	minutes	seconds
35764	0.015856	0.880659	-0.007463	-0.027235	-0.925717	1.405775	0.145275	0.785749	1.671695	0.0	0.0
31011	0.015856	-0.602747	-0.007463	-0.027235	-0.925717	-0.199205	-1.325232	0.785749	-0.201459	0.0	0.0
28019	0.015856	0.070833	-0.007463	-0.027235	-0.925717	-0.428488	1.321682	0.257541	0.086718	0.0	0.0
33195	0.015856	-0.177363	-0.007463	-0.027235	1.229874	1.635058	-0.737029	0.785749	-0.201459	0.0	0.0
22348	0.015856	-1.005935	-0.007463	-0.027235	-0.925717	-0.772412	-0.737029	0.257541	-1.498258	0.0	0.0

```
y_train
```

```
35764    2488
31011    4395
28019    4513
33195    3489
22348     751
         ...
11284    5761
44732    4799
38158    5139
860      2057
15795    4385
Name: traffic_volume, Length: 38563, dtype: int64
```

Model Evaluation for X_train data

```
# Predict and evaluate using R², MAE, MSE
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
print("Linear regression", r2_score(y_train, train1))
print("Decision Tree Regressor", r2_score(y_train, train2))
print("Random Forest Regressor",r2_score(y_train, train3))
print("SVR",r2_score(y_train, train4))
print("XGB",r2_score(y_train, train5))
```

```
Linear regression 0.13225712655997102
Decision Tree Regressor 1.0
Random Forest Regressor 0.9779024664396229
SVR 0.24187500547247331
XGB 0.7786684036254883
```

```
print("Linear regression", mean_squared_error(y_train, train1))
print("Decision Tree Regressor", mean_squared_error(y_train, train2))
print("Random Forest Regressor",mean_squared_error(y_train, train3))
print("SVR",mean_squared_error(y_train, train4))
print("XGB",mean_squared_error(y_train, train5))
```

```
Linear regression 3424093.4137532604
Decision Tree Regressor 0.0
Random Forest Regressor 87196.35901395639
SVR 2991543.7856290694
XGB 873369.25
```

```
print("Linear regression", mean_absolute_error(y_train, train1))
print("Decision Tree Regressor", mean_absolute_error(y_train, train2))
print("Random Forest Regressor",mean_absolute_error(y_train, train3))
print("SVR",mean_absolute_error(y_train, train4))
print("XGB",mean_absolute_error(y_train, train5))
```

```
Linear regression 1640.2171288638979
Decision Tree Regressor 0.0
Random Forest Regressor 185.25322511215415
SVR 1509.2879969637522
XGB 639.703857421875
```

```
print("Linear regression", np.sqrt(mean_squared_error(y_train, train1)))
print("Decision Tree Regressor", np.sqrt(mean_squared_error(y_train, train2)))
print("Random Forest Regressor",np.sqrt(mean_squared_error(y_train, train3)))
print("SVR",np.sqrt(mean_squared_error(y_train, train4)))
print("XGB",np.sqrt(mean_squared_error(y_train, train5)))
```

```
Linear regression 1850.430602252692
Decision Tree Regressor 0.0
Random Forest Regressor 295.29029617303104
SVR 1729.6079861139256
XGB 934.5422676369432
```

Model Evaluation for X_test data

```
# predict the y_test value and caculate the accuracy
test1 = lr.predict(x_test)
test2 = Dtree.predict(x_test)
test3 = Rand.predict(x_test)
test4 = svr.predict(x_test)
test5 = XGB.predict(x_test)
```

```
print("Linear regression",r2_score(y_test, test1))
print("Decision Tree Regressor",r2_score(y_test, test2))
print("Random Forest Regression",r2_score(y_test, test3))
print("SVR",r2_score(y_test, test4))
print("XGB",r2_score(y_test, test5))
```

```
Linear regression 0.1389494912742577
Decision Tree Regressor 0.7169825008411866
Random Forest Regression 0.8430752236706374
SVR 0.2448235297268001
XGB 0.7832188606262207
```

```
print("Linear regression", mean_squared_error(y_test, test1))
print("Decision Tree Regressor", mean_squared_error(y_test, test2))
print("Random Forest Regressor",mean_squared_error(y_test, test3))
print("SVR",mean_squared_error(y_test, test4))
print("XGB",mean_squared_error(y_test, test5))
```

```
Linear regression 3404174.861301238
Decision Tree Regressor 1118913.520381703
Random Forest Regressor 620404.2309739654
SVR 2985600.414724397
XGB 857047.125
```

```
print("Linear regression", mean_absolute_error(y_test, test1))
print("Decision Tree Regressor", mean_absolute_error(y_test, test2))
print("Random Forest Regressor",mean_absolute_error(y_test, test3))
print("SVR",mean_absolute_error(y_test, test4))
print("XGB",mean_absolute_error(y_test, test5))
```

```
Linear regression 1638.7989252319232
Decision Tree Regressor 555.1361891919926
Random Forest Regressor 496.8264121979048
SVR 1510.833149986356
XGB 632.3110961914062
```

```
print("Linear regression", np.sqrt(mean_squared_error(y_test, test1)))
print("Decision Tree Regressor", np.sqrt(mean_squared_error(y_test, test2)) )
print("Random Forest Regressor",np.sqrt(mean_squared_error(y_test, test3)) )
print("SVR",np.sqrt(mean_squared_error(y_test, test4)) )
print("XGB",np.sqrt(mean_squared_error(y_test, test5)))
```

```
Linear regression 1845.0406123717814
Decision Tree Regressor 1057.7870865073476
Random Forest Regressor 787.6574325009352
SVR 1727.889005325399
XGB 925.7683970626779
```

```
x.head()
```

	holiday	temp	rain	snow	weather	day	month	year	hours	minutes	seconds
0	0.015856	0.530485	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.345548	0.0	0.0
1	0.015856	0.611467	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.201459	0.0	0.0
2	0.015856	0.627964	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.057371	0.0	0.0
3	0.015856	0.669205	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	0.086718	0.0	0.0
4	0.015856	0.744939	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	0.230807	0.0	0.0

```
dataset.head()
```

	holiday	temp	rain	snow	weather	traffic_volume	day	month	year	hours	minutes	seconds
0	7	288.28	0.0	0.0	1	5545	02	10	2012	09	00	00
1	7	289.36	0.0	0.0	1	4516	02	10	2012	10	00	00
2	7	289.58	0.0	0.0	1	4767	02	10	2012	11	00	00
3	7	290.13	0.0	0.0	1	5026	02	10	2012	12	00	00
4	7	291.14	0.0	0.0	1	4918	02	10	2012	13	00	00

```
Rand.predict([[7, 289.28, 0.0, 0.0, 1, 2, 10, 2012, 10, 0, 0]])
```

```
array([1600.02])
```

Random Forest is very Less when compared with other models, so Saving the Random forest model and Deploying.

In Random Forest Best result:

✅ No overfitting: A small gap (97% → 84%) means the model isn't memorizing, it's learning.

✅ Good generalization: Predicts traffic volume on unseen data quite accurately.

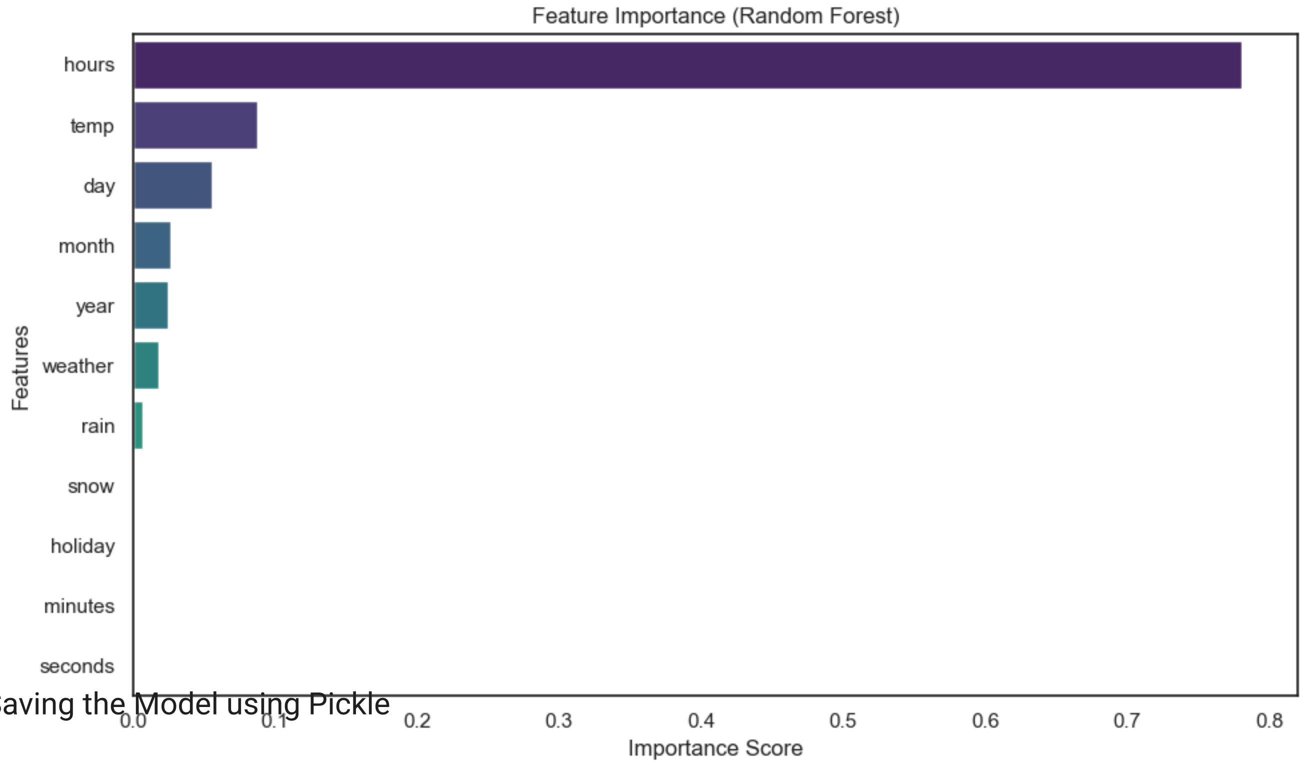
✅ Feature processing + model choice worked well.

Feature importances

```
Rand.feature_importances_
```

```
array([1.19496933e-05, 8.71820215e-02, 6.91457975e-03, 5.97033853e-05,
       1.77991664e-02, 5.57887871e-02, 2.66752344e-02, 2.47915670e-02,
       7.80776991e-01, 0.00000000e+00, 0.00000000e+00])
```

```
# Get feature importances
importances = Rand.feature_importances_
feature_names = x.columns
feature_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_df = feature_df.sort_values(by='Importance', ascending=False)



# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(data=feature_df, x='Importance', y='Feature', palette='viridis')
plt.title('Feature Importance (Random Forest)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.tight_layout()
plt.show()
```



Saving the Model using Pickle

```
import pickle


# # Assume 'model' is your trained model
# filename = 'finalized_model.pkl'

# # Save the model to disk
# with open(filename, 'wb') as file:
#     pickle.dump(Rand, file)



# with open(filename, 'rb') as file:
#     loaded_model = pickle.load(file)

# # Now you can use it to make predictions
# predictions = loaded_model.predict(x_test)
```