Cost Prediction for Logistic Company

INCLASS REGRESSION PROBLEM FOR AML-1413 FALL 2023

BY TEAM GAMMA B

TEAM MEMBERS

AKASH(

AKHIL

JAGADEESH

KAMESWARA

KAPIL

SATHWICK

STEPS INVOLVED

-Data Validation & cleansing
-Data preprocessing
-Exploratory Data Analysis

Data Validation and Cleansing

Data Validation & Cleansing

- Dimensions of the data
- The Train dataset (38999 Rows and 12 Columns)
- The Test dataset(802 Rows and 11 Columns)
- Converting date to date-time format
- Checking missing values and its percentage

Data Validation and Cleaning

In [3]:
```
# Let's Look The Dimensions Of The Data:
print(f'The Train dataset Contain {train.shape[0]} Rows and {train.shape[1]} Columns')
```
```
The Train dataset Contain 38999 Rows and 12 Columns
```

In [5]:
```
# Let's Look The Dimensions Of The Data:
print(f'The Test dataset Contain {test.shape[0]} Rows and {test.shape[1]} Columns')
```
```
The Test dataset Contain 802 Rows and 11 Columns
```

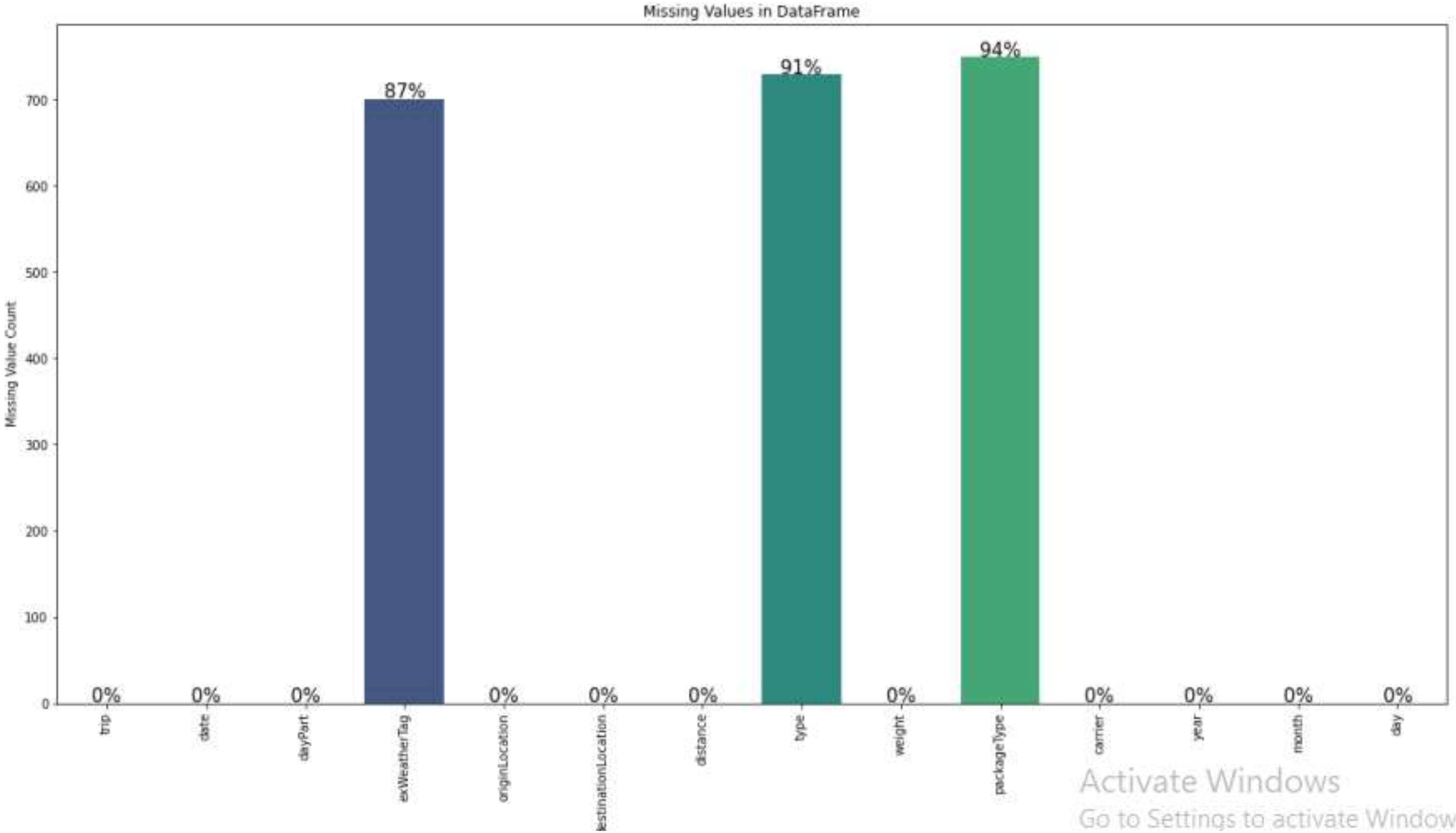
In [6]:
```
#Check Data Types
train.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38999 entries, 0 to 38998
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   trip                 38999 non-null  object 
 1   date                 38999 non-null  object 
 2   dayPart              38999 non-null  object 
 3   exWeatherTag         4882 non-null   object 
 4   originLocation       38999 non-null  object 
 5   destinationLocation  38999 non-null  object 
 6   distance             38999 non-null  int64  
 7   type                 3748 non-null   object 
 8   weight               38999 non-null  int64  
 9   packageType          2500 non-null   object 
 10  carrier              38999 non-null  object 
 11  cost                 38999 non-null  float64
dtypes: float64(1), int64(2), object(9)
```

Converting date into datetime Format

```
## Converting date to datetime format and extracting the temporal features
train['date'] = pd.to_datetime(train['date'], format='%Y-%m-%d')
train['year'] = train['date'].dt.year
train['month'] = train['date'].dt.month
train['day'] = train['date'].dt.day


test['date'] = pd.to_datetime(test['date'], format='%Y-%m-%d')
test['year'] = test['date'].dt.year
test['month'] = test['date'].dt.month
test['day'] = test['date'].dt.day
```

Check for missing values in the data

```python
## Checking missing values and visualizing it

missing_counts = test.isnull().sum()
present_values=test.notnull().sum()
unique_value_counts = test.nunique()
total_count = len(test)
missing_percentage= (missing_counts / total_count) * 100
present_percentage= (present_values / total_count) * 100
unique_percentage= (unique_value_counts / total_count) * 100

# Plotting the missing values % in the dataset
plt.figure(figsize=(20, 10))
ax=sns.barplot(x=missing_counts.index, y=missing_counts.values, palette='viridis')

# Add labels and title
plt.title('Missing Values in DataFrame')
plt.xlabel('Columns')
plt.ylabel('Missing Value Count')
plt.xticks(rotation=90)

for p, label in zip(ax.patches, missing_percentage):
    ax.annotate(f'{label:.0f}%', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=15, color='black', xytext=(0, 5), textcoords='offs

plt.show()
```



Missing Values in DataFrame
Missing Value Count
0
100
200
300
400
500
600
700
87%
91%
94%
0%
trip
date
dayPart
exWeatherTag
originLocation
estinationLocation
distance
type
weight
packageType
carrier
year
month
day

Imputing features having high percentage of missing values

- Type, packageType and exWeatherTag have high missing %, so let's impute the missing values as 'Not_available' category since they seem to be important information.

In [10]:

```
# Sample of the unique elements of the features
Unique_values_list=[]
for column in train.columns:
    unique_values = train[column].unique().tolist()
    num_unique=train[column].nunique()
    if num_unique==1:
      Unique_values_list.append(column)
    print("{} Unique values in '{}': {} ".format(num_unique,column,unique_values[:5]))
print(Unique_values_list)
```

```
38999 Unique values in 'trip': ['t52712528', 't29859381', 't25702332', 't27713405', 't49439220']
1074 Unique values in 'date': [1504656000000000000, 1508544000000000000, 1500768000000000000, 15086304000000000
00, 1576022400000000000]
2 Unique values in 'dayPart': ['night', 'day']
2 Unique values in 'exWeatherTag': [nan, 'snow', 'heat']
9 Unique values in 'originLocation': ['S4', 'S8', 'S9', 'S6', 'S7']
9 Unique values in 'destinationLocation': ['D7', 'D1', 'D5', 'D4', 'D2']
17 Unique values in 'distance': [2200, 1800, 2800, 3200, 2000]
1 Unique values in 'type': ['expedited', nan]
499 Unique values in 'weight': [50, 12, 1, 43, 3]
1 Unique values in 'packageType': [nan, 'TT']
4 Unique values in 'carrier': ['D', 'B', 'C', 'A']
3665 Unique values in 'cost': [68.41315193, 36.45064919, 9.05793866, 57.32008718, 77.2637771]
3 Unique values in 'year': [2017, 2019, 2018]
12 Unique values in 'month': [9, 10, 7, 12, 6]
30 Unique values in 'day': [6, 21, 15, 22, 11]
['type', 'packageType']
```

Type, packageType and exWeatherTag have high missing %, so let's impute the missing values as 'Not_available' category since they seem to be important information.

Feature Engineering

In [14]:

```
train = train.apply(lambda x: x.fillna('NA'))
test = test.apply(lambda x: x.fillna('NA'))
```

Importing holidays data set

Since it is a time series dataset, lets add date related features such as holiday_or_not, weekday_or_not

```
In [19]: # using holidays library to get the holidays
         import holidays

         holiday_list = list()
         print("--- Canada ---")
         for date in holidays.Canada(years=[2014,2015,2016,2017,2018,2019], observed=True).items():
             print(str(date[0]), date[1])
             holiday_list.append([date[0], date[1]])
         Holiday_dataset=pd.DataFrame(holiday_list,columns=['Date','Holiday'])

         Holiday_dataset['Date'] = pd.to_datetime(Holiday_dataset['Date'])

         # Create 'day_Holiday_or_not' column in train and test DataFrames
         train['day_Holiday_or_not'] = train['date'].isin(Holiday_dataset['Date']).astype(int)
         test['day_Holiday_or_not'] = test['date'].isin(Holiday_dataset['Date']).astype(int)
         train['is_weekday'] = train['date'].apply(lambda x: 1 if x.weekday() < 5 else 0)
         test['is_weekday'] = test['date'].apply(lambda x: 1 if x.weekday() < 5 else 0)
         train['weekday']=train['date'].dt.day_name()
         test['weekday']=test['date'].dt.day_name()
```

```
--- Canada ---
2016-01-01 New Year's Day
2016-03-25 Good Friday
2016-07-01 Canada Day
2016-09-05 Labour Day
2016-12-25 Christmas Day
2016-12-26 Christmas Day (Observed)
2017-01-01 New Year's Day
2017-01-02 New Year's Day (Observed)
2017-04-14 Good Friday
2017-07-01 Canada Day
2017-09-04 Labour Day
2017-12-25 Christmas Day
2018-01-01 New Year's Day
2018-03-30 Good Friday
2018-07-01 Canada Day
2018-09-03 Labour Day
```

Data Preprocessing

Data Preprocessing

- Splitting into train and test(X_train, X_test, Y_train,Y_test)
- Normalizing the variables using standard scalar

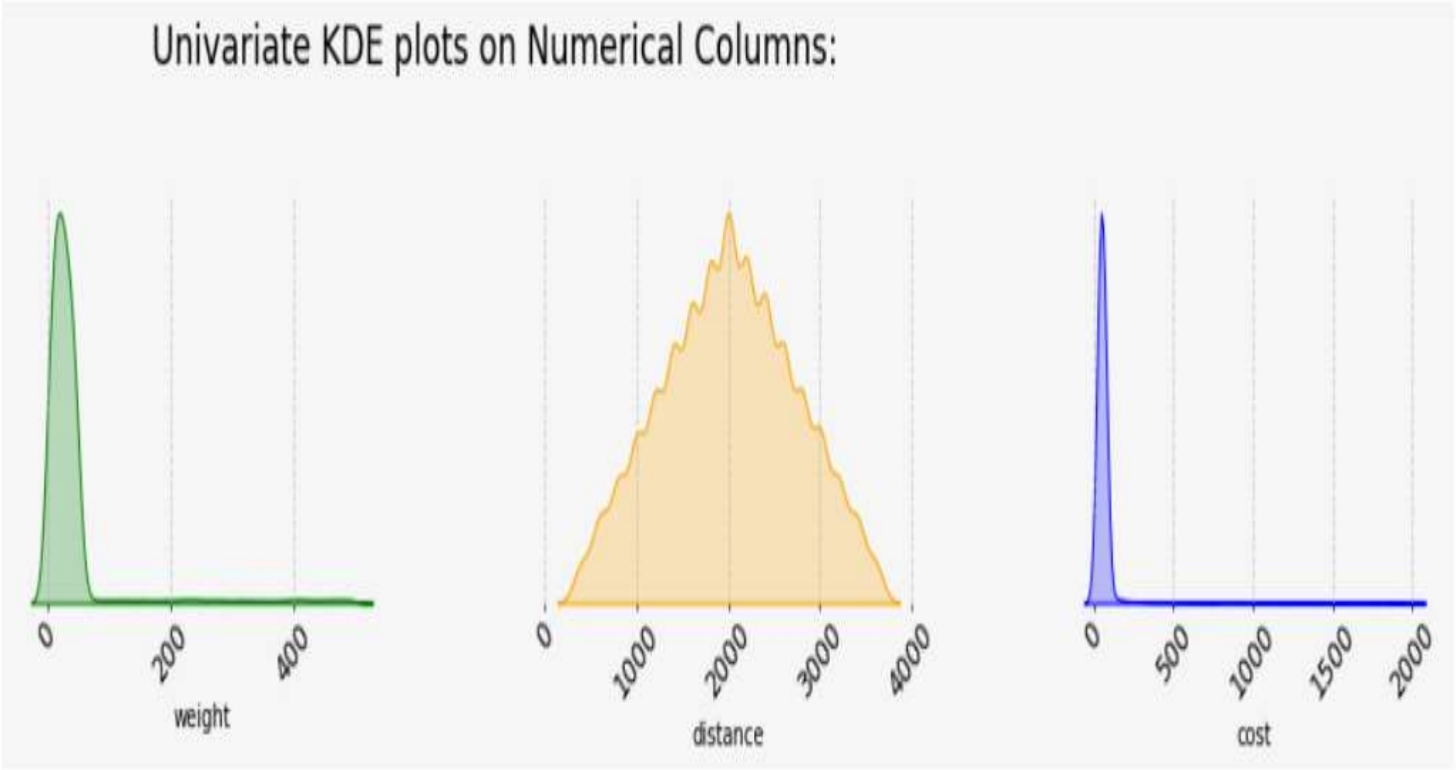
```
val_trip=test.pop('trip')
test.drop(columns=['date'],inplace=True)
train.drop(columns=['trip','date'],inplace=True)

Y=train.pop('cost')
X=train.copy()

#Splitting train and test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

numerical_columns = X_train.select_dtypes(include=['int64', 'float64']).columns
# Normalizing the variables
pipeline = Pipeline([
    ('std_scalar', StandardScaler())
])

X_train[numerical_columns] = pipeline.fit_transform(X_train[numerical_columns])
X_test[numerical_columns] = pipeline.transform(X_test[numerical_columns])
test[numerical_columns]=pipeline.transform(test[numerical_columns])
```

EDA

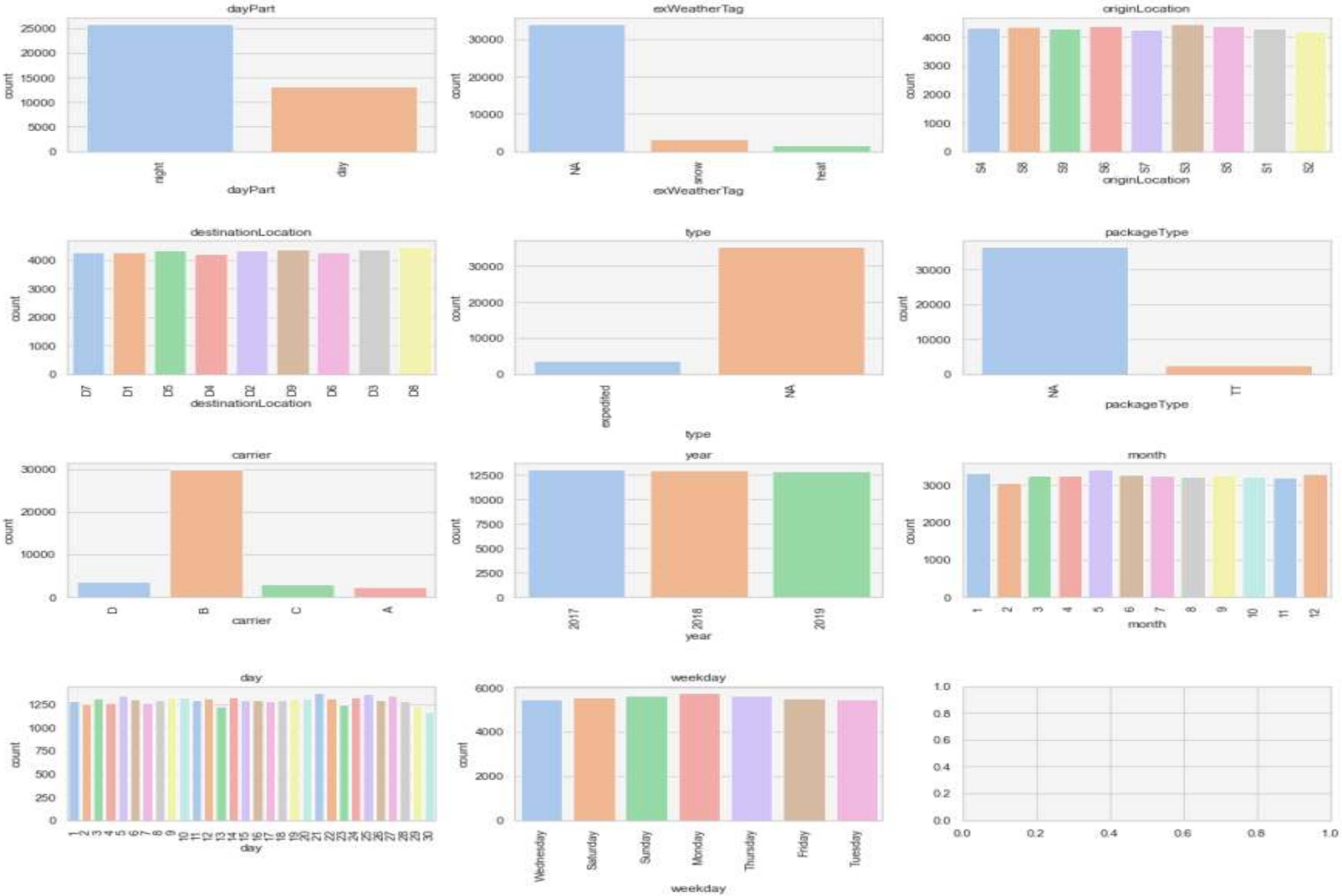
UNIVARIATE ANALYSIS

BIVARIATE ANALYSIS

Univariate Analysis



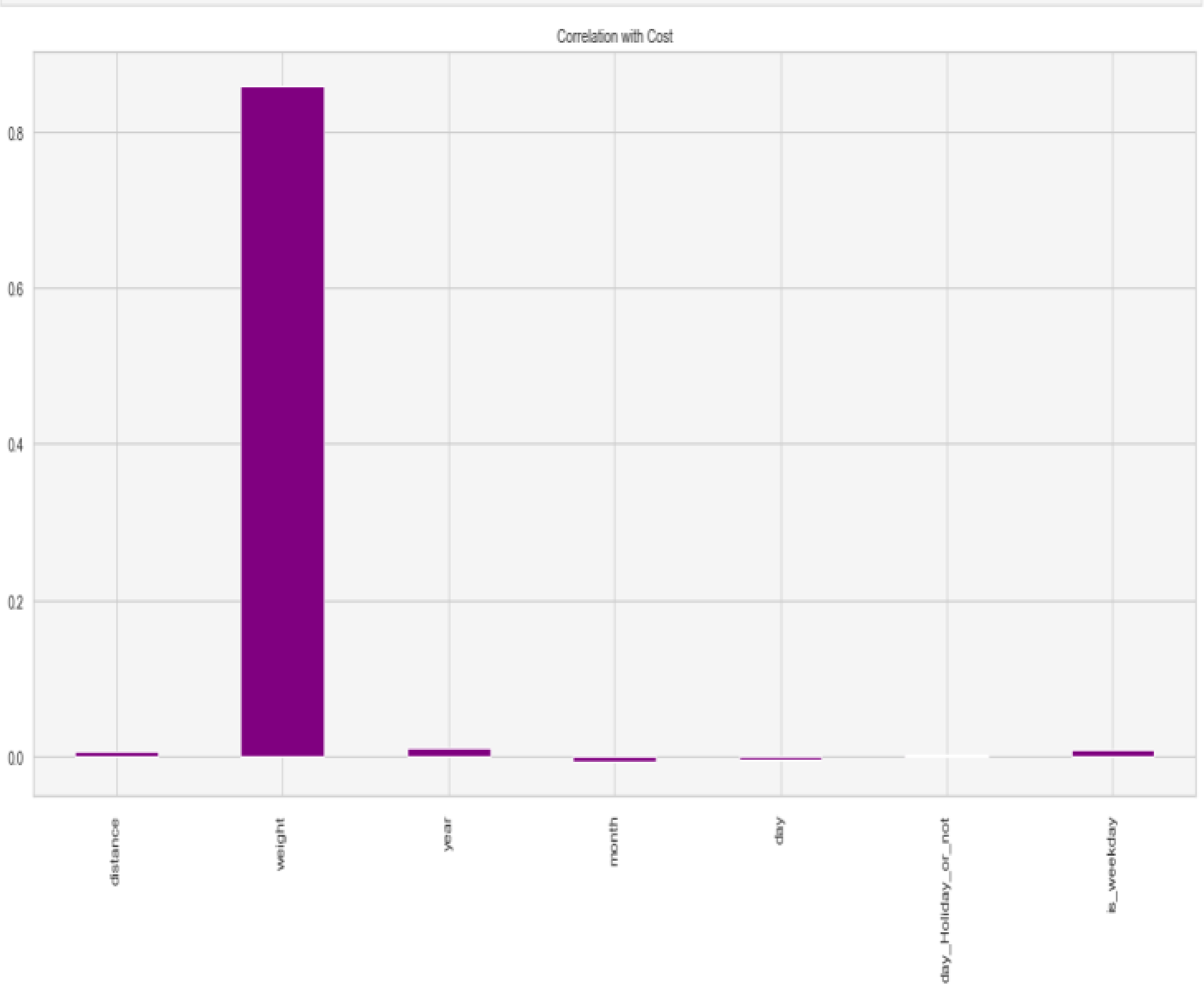
PLOTTING THE UNIVARIATE KDE PLOTS ON NUMERICAL COLUMNS

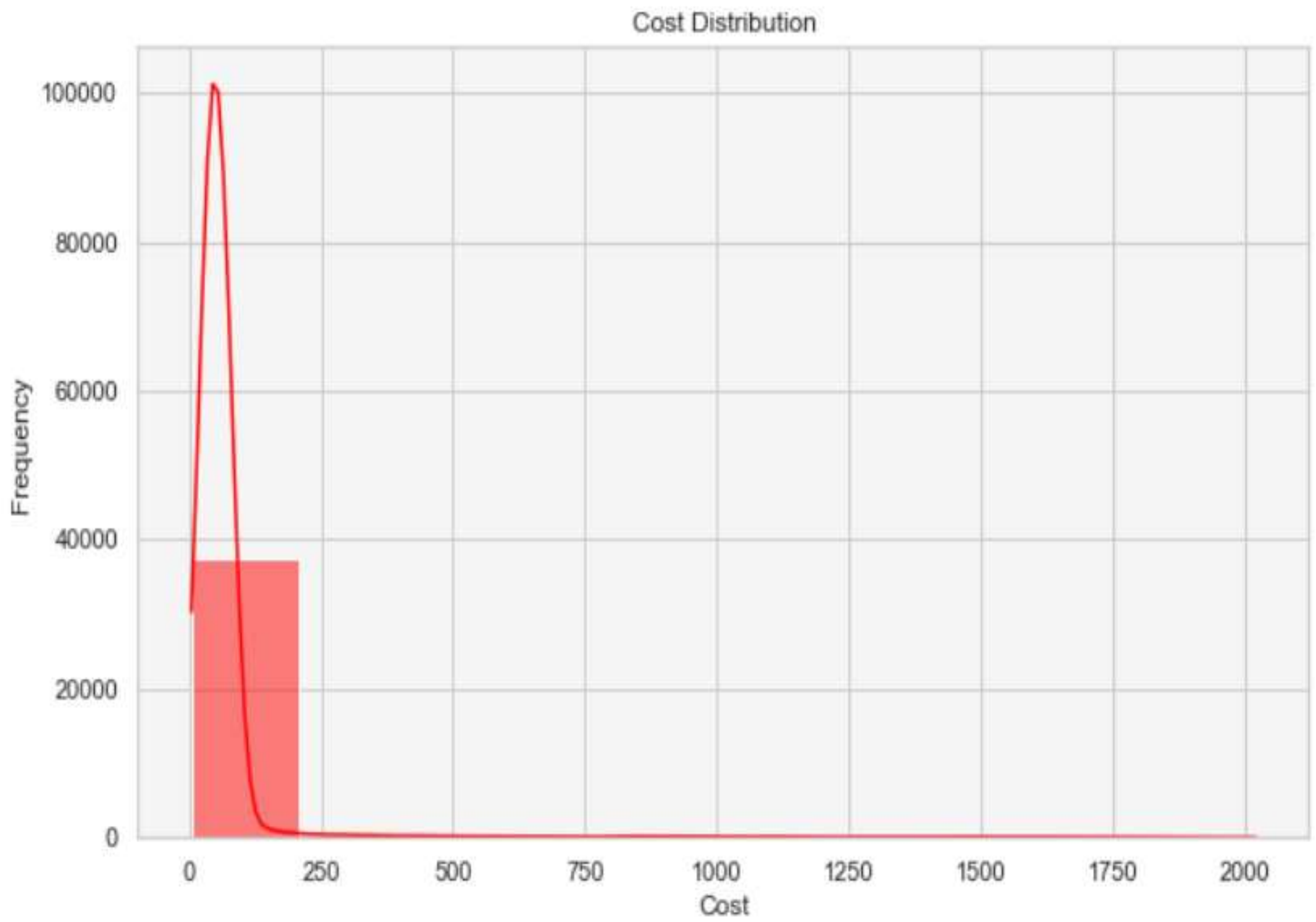

dayPart
count
night
day
dayPart
exWeatherTag
count
NA
snow
heat
exWeatherTag
originLocation
count
S4
S8
S9
S6
S7
S3
S5
S1
S2
originLocation
destinationLocation
count
D7
D1
D5
D4
D2
D9
D6
D3
D8
destinationLocation
type
count
expedited
NA
type
packageType
count
NA
TT
packageType
carrier
count
D
B
C
A
carrier
year
count
2017
2018
2019
year
month
count
month
day
count
day
weekday
count
Wednesday
Saturday
Sunday
Monday
Thursday
Friday
Tuesday
weekday

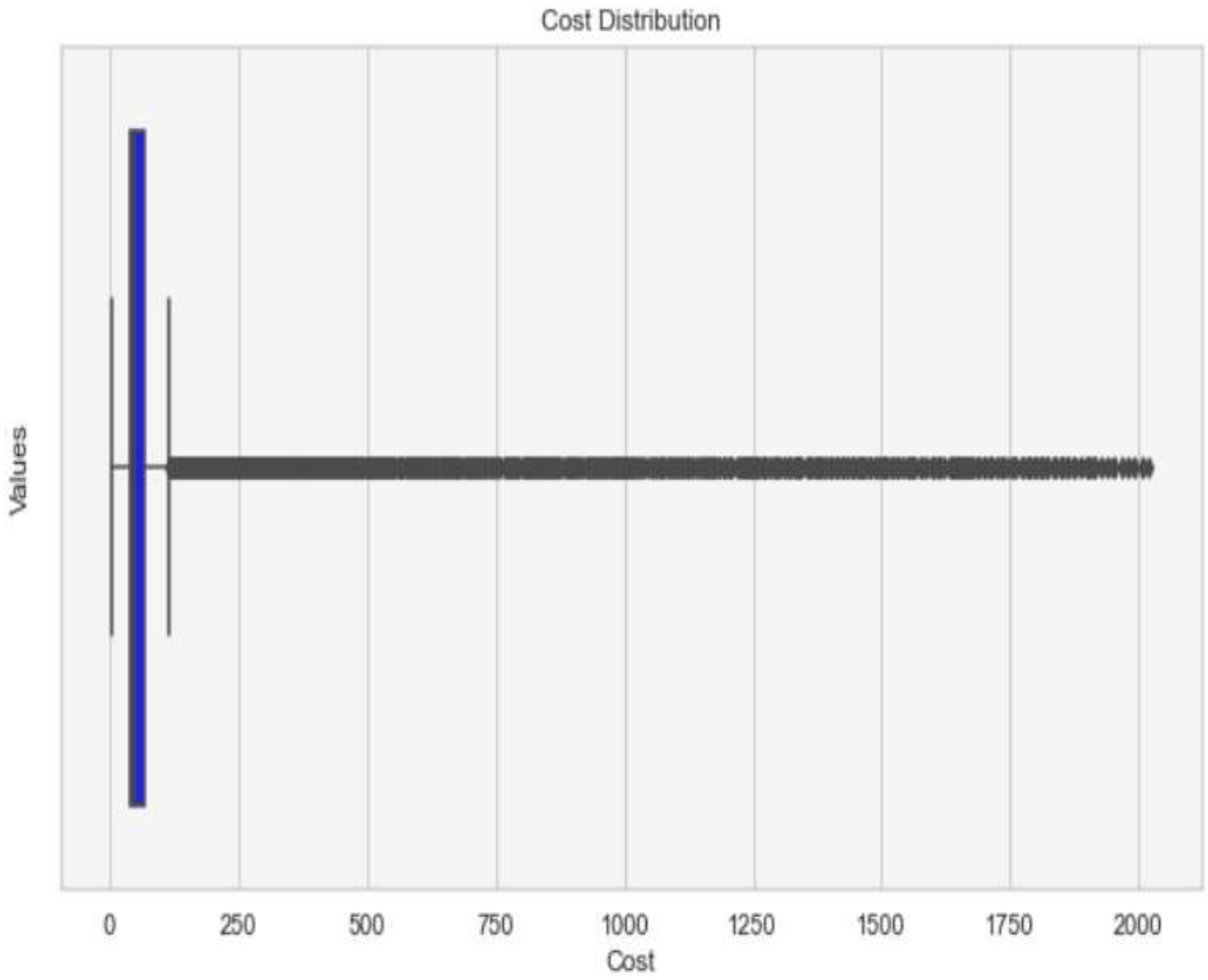
Correlation heatmap

Correlation plot with the target variable cost

Weight is highly correlated with the target variable

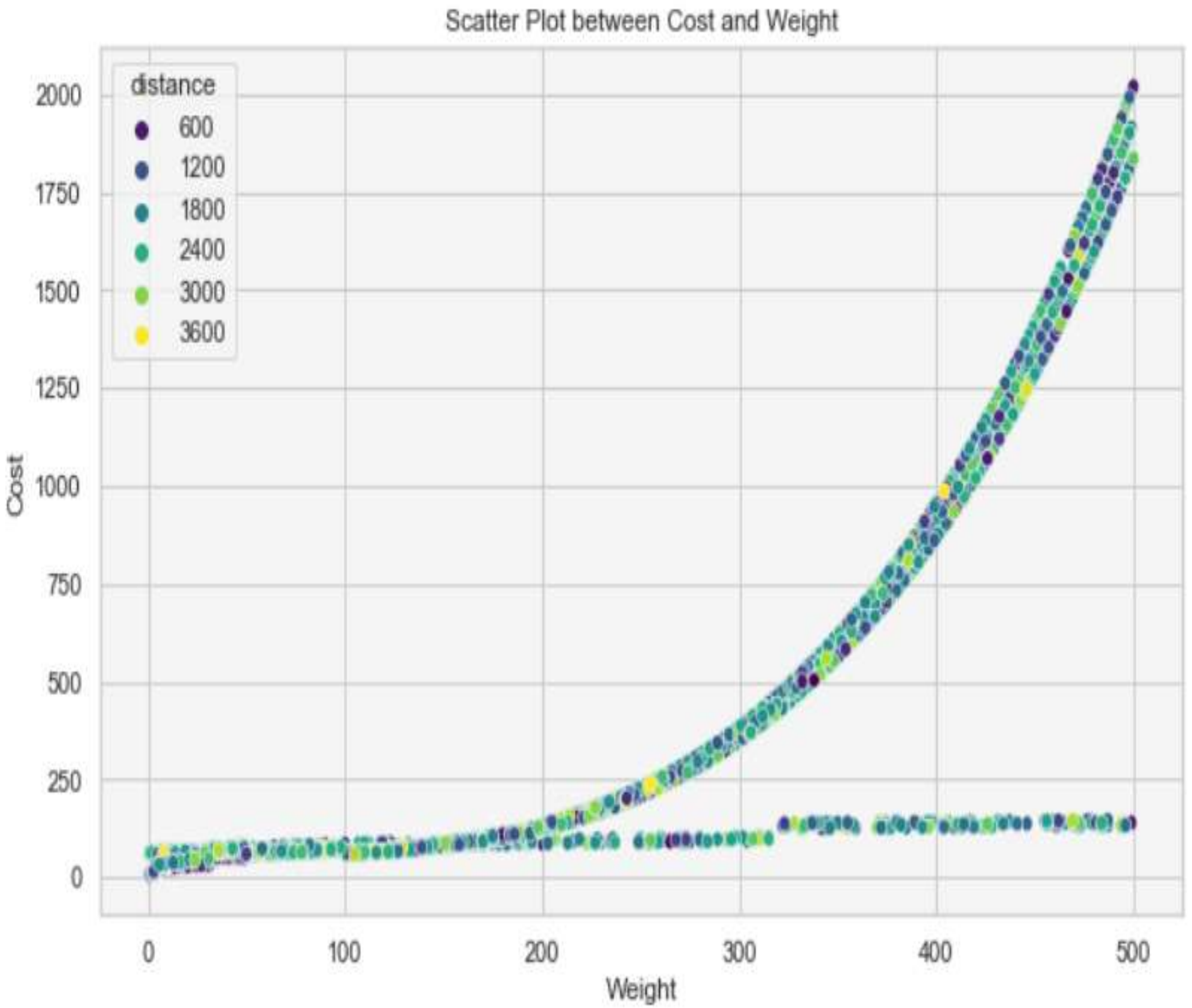


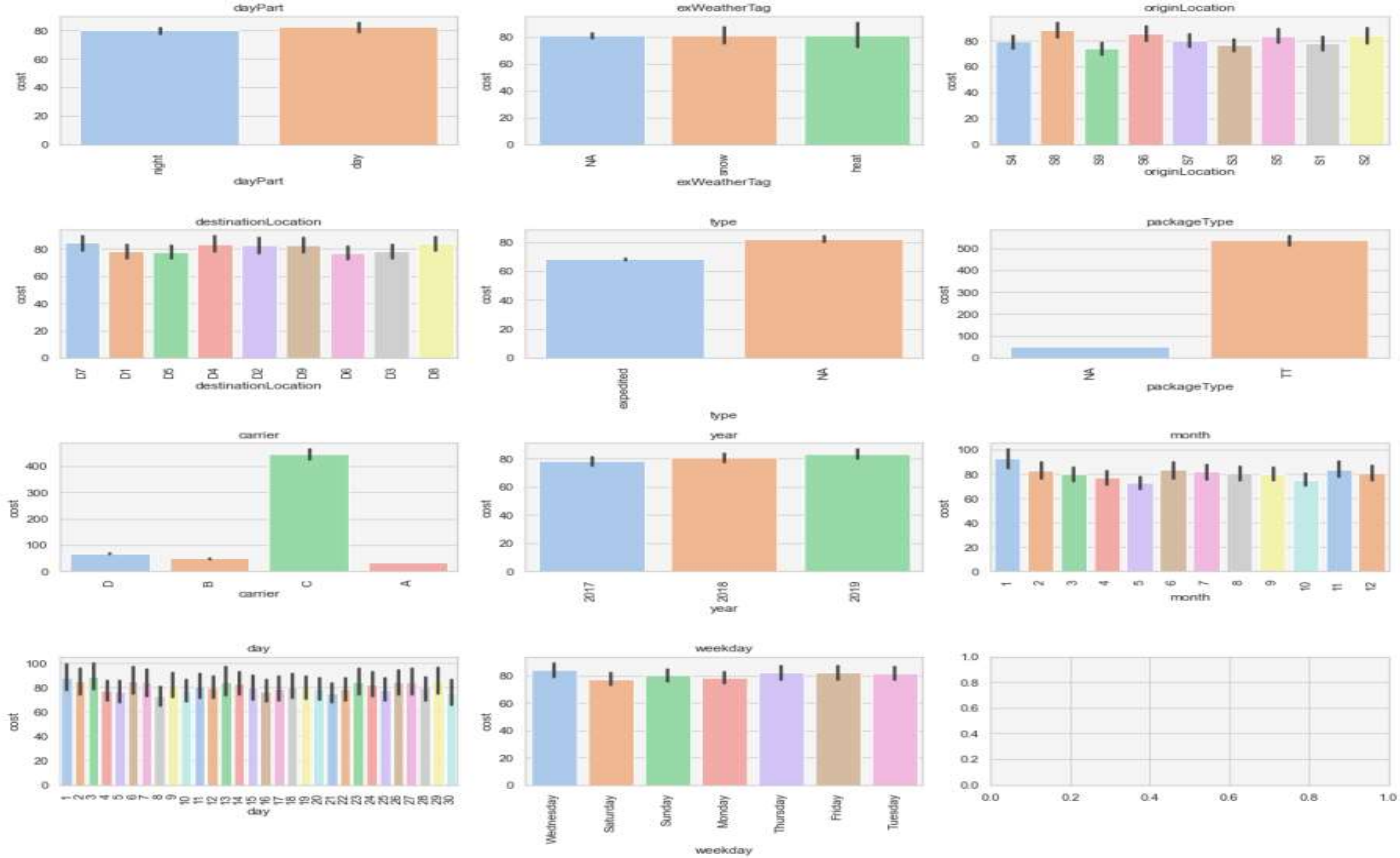

Cost Distribution
100000
80000
60000
40000
20000
0
Frequency
0
250
500
750
1000
1250
1500
1750
2000
Cost


Cost Distribution
Values
0
250
500
750
1000
1250
1500
1750
2000
Cost

Bivariate analysis

Performed on cost on weight variables




dayPart
exWeatherTag
originLocation
destinationLocation
type
packageType
carrier
year
month
day
weekday
cost
night
day
NA
snow
heat
S4
S8
S9
S6
S7
S3
S5
S1
S2
D7
D1
D5
D4
D2
D9
D6
D3
D8
expedited
NA
NA
TT
D
B
C
A
2017
2018
2019
Wednesday
Saturday
Sunday
Monday
Thursday
Friday
Tuesday

Observations

LOCATIONS AND AGE ARE EQUALLY DISTRIBUTED.

AS WEIGHT INCREASES THE COST INCREASES.

MOST DELIVERIES ARE DONE DURING THE NIGHT

. WEIGHT IMPACTS THE COST MOST AND IT HAS HIGH CORRELATION WITH COST.

COST DOESN'T CHANGE BASED ON WEATHER AND DAYPART CONDITIONS.

COST IS HIGH IF THE PACKAGETYPE IS TT OR IF ITS CARRIED BY CARRIER C.

EXPEDITED PACKAGES HAVE A SLIGHTLY LOWER COST.

COST SLIGHTLY INCREASES BY YEAR.

AS THE YEAR STARTS THE COST IS HIGH AND IT DECREASES TILL MONTH 5 AND IT PICKS UP AFTER 6TH MONTH. OCTOBER HAS A SLIGHT DIP BUT IT PICKS UP AFTER THAT.

Modelling

Modelling

Feature encoding: Let's encode one hot encoding of the train, test and val together and later split them.

```
combined_df = pd.concat([X_train, X_test, test], axis=0)

# Perform one-hot encoding on the combined DataFrame
combined_df_encoded = pd.get_dummies(combined_df, columns=train.select_dtypes(exclude="number").columns.to_list

# Split the combined DataFrame back into train, test, and val
train_encoded = combined_df_encoded.iloc[:len(X_train)]
test_encoded = combined_df_encoded.iloc[len(X_train):len(X_train) + len(X_test)]
val_encoded = combined_df_encoded.iloc[len(X_train) + len(X_test):]
```

Training the dataset

- Linear Regression
- Ridge Regression
- XG Boost
- Cat Boost

```
# The below function trains the dataset for Linear Regression,Ridge regression, Xgboost and Catboost models and
def train_models(X_train, X_test, y_train, y_test):
    model_dict = {
        "linear": LinearRegression(),
        "Ridge": Ridge(alpha=0.2),
        ##"KNN": KNeighborsRegressor(n_jobs=-1, n_neighbors=4),
        "XGB": XGBRegressor(random_state=42),
       ## "light": LGBMRegressor(random_state=42),
        "Cat": CatBoostRegressor(random_state=42, loss_function='RMSE', verbose=False)
    }
    list1=[]
    dict1={}


    for model_name, model in model_dict.items() :
        model.fit(X_train, y_train)
        pred = model.predict(X_test)
        num_predictors = X_train.shape[1]

        n = X_train.shape[0]

        adjusted_r_squared = 1 - (1 - r2_score(y_test, pred)) * (n - 1) / (n - num_predictors - 1)

        print(f"Training loss for model {model_name}   MSE : {mean_squared_error(y_test, pred, squared=False),]
        dict1[model_name]=[mean_squared_error(y_test, pred),sqrt(mean_squared_error(y_test, pred)),r2_score(y_t
                           mean_absolute_error(y_test, pred),model.score(X_test,y_test)*100,adjusted_r_squared]
    return dict1
```

Training loss for the models

MODEL	MSE	RMSE	R-SQUARE	MAE	ACCURACY	ADJUSTED R SQUARE
linear	81.61	81.61	0.80	41.40	0.80	0.80
Ridge	81.61	81.61	0.80	41.40	0.80	0.80
XGB	1.71	1.71	0.99	0.34	0.99	0.99
Cat	1.58	1.58	0.99	0.39	0.99	0.99

```
# Running the train_models function
dict1=train_models(train_encoded,test_encoded, y_train, y_test)
```

```
Training loss for model linear   MSE : (81.6144663137134,),RMSE : (81.6144663137134,), R-Square : (0.8098722881943007,), MAE : (41.40249987247436,) , Accuracy : 0.8098722881943007 ,Adjusted R-Square : 0.8095583503881415
Training loss for model Ridge   MSE : (81.61736842557802,),RMSE : (81.61736842557802,), R-Square : (0.8098587665302301,), MAE : (41.4006551439411,) , Accuracy : 0.8098587665302301 ,Adjusted R-Square : 0.8095448063971755
Training loss for model XGB   MSE : (1.7149978499386376,),RMSE : (1.7149978499386376,), R-Square : (0.9999160465996152,), MAE : (0.3463397212158473,) , Accuracy : 0.9999160465996152 ,Adjusted R-Square : 0.9999159079762336

Training loss for model Cat   MSE : (1.58649023994922,),RMSE : (1.58649023994922,), R-Square : (0.9999281567549223,), MAE : (0.39570604624943556,) , Accuracy : 0.9999281567549223 ,Adjusted R-Square : 0.999928038127761
```

Initiating Catboost

- Since Catboost is our best model, we'll use gridsearch for hyperparameter tuning to find our best parameters.

In [104…

```
# Instantiate CatBoostRegressor
cbr = CatBoostRegressor()

# Create a comprehensive grid with various hyperparameters
grid = {
    'depth': [6,9],
    'learning_rate': [ 0.1],
    'iterations': [100, 500],
    'subsample': [0.5, 0.9],
    'random_seed': [42]
}

# Define RMSE as the scoring metric
rmse_scorer = make_scorer(lambda y_true, y_pred: np.sqrt(np.mean((y_true - y_pred) ** 2)), greater_is_better=Fa

# Instantiate GridSearchCV for CatBoostRegressor with the comprehensive grid and RMSE as the scoring metric
gscv = GridSearchCV(estimator=cbr, param_grid=grid, scoring=rmse_scorer, cv=5)

# Fit the grid search on the training data
gscv.fit(train_encoded, y_train)

# Print the best hyperparameters from the grid search
print('Best Hyperparameters:', gscv.best_params_)

# Use the best model from grid search for predictions
best_model = gscv.best_estimator_
y_pred = best_model.predict(test_encoded)

# Calculate evaluation metrics on the test data
rmse = sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the evaluation metrics
print('The RMSE score of the CatBoost is', rmse)
print('The MAE score of the CatBoost is', mae)
print('The R2 score of the CatBoost is', r2)
print('The accuracy score of the CatBoost is', best_model.score(test_encoded, y_test) * 100)
```

```
482:	learn: 1.1024763	total: 2.11s	remaining: 74.3ms
483:	learn: 1.0994269	total: 2.11s	remaining: 69.9ms
484:	learn: 1.0986041	total: 2.12s	remaining: 65.5ms
485:	learn: 1.0949944	total: 2.12s	remaining: 61.1ms
486:	learn: 1.0947183	total: 2.13s	remaining: 56.8ms
487:	learn: 1.0929279	total: 2.13s	remaining: 52.4ms
488:	learn: 1.0909668	total: 2.13s	remaining: 48ms
489:	learn: 1.0884928	total: 2.14s	remaining: 43.6ms
490:	learn: 1.0873845	total: 2.14s	remaining: 39.3ms
491:	learn: 1.0862656	total: 2.15s	remaining: 34.9ms
492:	learn: 1.0852044	total: 2.15s	remaining: 30.5ms
493:	learn: 1.0834287	total: 2.15s	remaining: 26.2ms
494:	learn: 1.0809067	total: 2.16s	remaining: 21.8ms
495:	learn: 1.0789189	total: 2.16s	remaining: 17.5ms
496:	learn: 1.0748498	total: 2.17s	remaining: 13.1ms
497:	learn: 1.0736250	total: 2.17s	remaining: 8.72ms
498:	learn: 1.0726284	total: 2.18s	remaining: 4.36ms
499:	learn: 1.0720685	total: 2.18s	remaining: 0us
Best Hyperparameters: {'depth': 6, 'iterations': 500, 'learning_rate': 0.1, 'random_seed': 42, 'subsample': 0.
5}
The RMSE score of the CatBoost is 1.8319448087674535
The MAE score of the CatBoost is 0.5125643984315946
The R2 score of the CatBoost is 0.9999042065306653
The accuracy score of the CatBoost is 99.99042065306652
```

In [105…

```
print('Best Hyperparameters:', gscv.best_params_)
```

```
Best Hyperparameters: {'depth': 6, 'iterations': 500, 'learning_rate': 0.1, 'random_seed': 42, 'subsample': 0.
5}
```

Let's use adaboost to improve on the previous catboost.

8]:

```
from sklearn.ensemble import AdaBoostRegressor

ada = AdaBoostRegressor(base_estimator = model,n_estimators=200,learning_rate=0.1)

ada.fit(train_encoded, y_train)
y_pred = ada.predict(test_encoded)

rmse = sqrt(mean_squared_error(y_test, y_pred))

mae = mean_absolute_error(y_test, y_pred)
r2=r2_score(y_test, y_pred)

print('The RMSE score of the catboost is ',rmse)
print('The MAE score of the catboost ',mae)
print('The R2 score of the catboost is ',r2)
print('The accuracy score of the catboost is ',ada.score(test_encoded,y_test)*100)
```

```
The RMSE score of the catboost is  0.6795211189527792
The MAE score of the catboost  0.1251449807349938
The R2 score of the catboost is  0.9999868199502696
The accuracy score of the catboost is  99.99868199502696
```

Ada boost

This is the best model on our test set and had an rmse of 0.6 on val set also.

```
val_set=pd.DataFrame(val_trip)
val_set['cost']=ada.predict(val_encoded)
val_set.to_csv('Submission6.csv')
```

Now that we know our best parameters and model, let's use the entire available data to train and predict on the validation set.

Combining x_train and x_test

Ada improved the RMSE and after a submitting it to Kaggle we had the highest personal best of 0.4 public score and 0.3 private score

```
# Combining x_train and x_test
combined_x_train = pd.concat([train_encoded, test_encoded], axis=0)
combined_y_train = pd.concat([y_train, y_test], axis=0)

ada.fit(combined_x_train,combined_y_train)
y_pred = ada.predict(test_encoded)

rmse = sqrt(mean_squared_error(y_test, y_pred))

mae = mean_absolute_error(y_test, y_pred)
r2=r2_score(y_test, y_pred)

print('The RMSE score of the catboost is ',rmse)
print('The MAE score of the catboost ',mae)
print('The R2 score of the catboost is ',r2)
print('The accuracy score of the catboost is ',ada.score(test_encoded,y_test)*100)
```

```
The RMSE score of the catboost is  0.0647618394141029
The MAE score of the catboost  0.037675440082020006
The R2 score of the catboost is  0.9999998802847485
The accuracy score of the catboost is  99.99998802847485
```

```
val_set=pd.DataFrame(val_trip)
val_set['cost']=ada.predict(val_encoded)
val_set.to_csv('Submission7.csv')
```

ANN using Tensorflow for Regression

LET'S USE A NEURAL NETWORK WITH THREE LAYERS AND RELU AS THE ACTIVATION FUNCTION

In [45]:
```
## Let's use a neural network with three layers and relu as the activation function
nn_model = tf.keras.Sequential([
    tf.keras.layers.Dense(units = 64, activation = tf.nn.relu, input_shape = [train_encoded.shape[1]]),
    tf.keras.layers.Dense(units = 64, activation = tf.nn.relu),
    tf.keras.layers.Dense(units = 1)
    ])
```

In [51]:
```
nn_model.compile(loss = 'mse', optimizer = tf.keras.optimizers.RMSprop(0.001), metrics = ['mae','mse'])
```

In [52]:
```
nn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                2944
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 7,169
Trainable params: 7,169
Non-trainable params: 0
_________________________________________________________________
```

In [53]:
```
history = nn_model.fit(train_encoded, y_train, epochs = 500, validation_data=(test_encoded, y_test))
```

```
Epoch 495/500
854/854 [==============================] - 4s 4ms/step - loss: 5.1839 - mae: 0.9507 - mse: 5.1839 - val_loss: 1
3.0082 - val_mae: 1.1555 - val_mse: 13.0082
Epoch 496/500
854/854 [==============================] - 4s 5ms/step - loss: 5.2333 - mae: 0.9378 - mse: 5.2333 - val_loss:
 7.2455 - val_mae: 1.4560 - val_mse: 7.2455
Epoch 497/500
854/854 [==============================] - 4s 4ms/step - loss: 5.2347 - mae: 0.9386 - mse: 5.2347 - val_loss:
 6.7140 - val_mae: 1.1064 - val_mse: 6.7140
Epoch 498/500
854/854 [==============================] - 4s 4ms/step - loss: 5.1578 - mae: 0.9328 - mse: 5.1578 - val_loss:
 4.4918 - val_mae: 0.8519 - val_mse: 4.4918
Epoch 499/500
854/854 [==============================] - 4s 4ms/step - loss: 5.2732 - mae: 0.9337 - mse: 5.2732 - val_loss: 1
0.3131 - val_mae: 1.2639 - val_mse: 10.3131
Epoch 500/500
854/854 [==============================] - 4s 4ms/step - loss: 4.9170 - mae: 0.9308 - mse: 4.9170 - val_loss:
 5.1875 - val_mae: 0.8679 - val_mse: 5.1875
```

In [54]:

```
hist = pd.DataFrame(history.history)
hist.tail()
```

Out[54]:

	loss	mae	mse	val_loss	val_mae	val_mse
495	5.233289	0.937784	5.233289	7.245465	1.456009	7.245465
496	5.234665	0.938590	5.234665	6.713998	1.106439	6.713998
497	5.157828	0.932813	5.157828	4.491841	0.851876	4.491841
498	5.273247	0.933667	5.273247	10.313062	1.263892	10.313062
499	4.916996	0.930770	4.916996	5.187454	0.867891	5.187454

Printing the scores of Neural Network results

The RMSE score is 2.27759863217814

The MAE score 0.8678907700156109

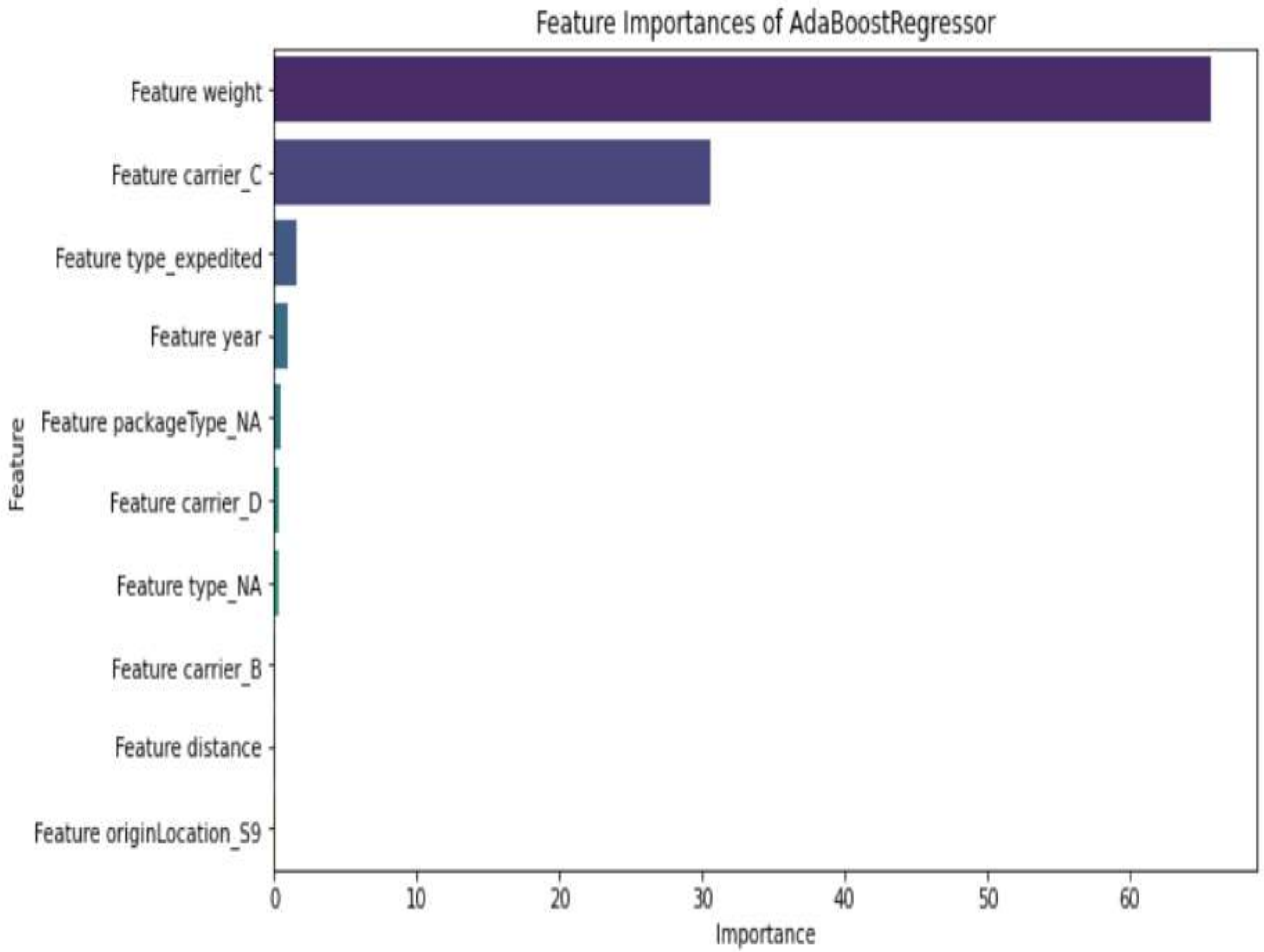
The R2 score is 0.9998519305313268

```python
## Printing the NN results
y_pred = nn_model.predict(test_encoded)
rmse = sqrt(mean_squared_error(y_test, y_pred))

mae = mean_absolute_error(y_test, y_pred)
r2=r2_score(y_test, y_pred)

print('The RMSE score of the neural network is ',rmse)
print('The MAE score of the neural network ',mae)
print('The R2 score of the neural network is ',r2)
```

```
The RMSE score of the neural network is  2.27759863217814
The MAE score of the neural network  0.8678907700156109
The R2 score of the neural network is  0.9998519305313268
```

Feature Importance

- As we found from the EDA weight and Carrier C and expedited are the best predictors of the cost.



Conclusion

- We chose catboost and adaboost as our final models, we used base models such as Linear regression, Ridge Regression, Xgboost and Catboost.
- The model was evaluated based on RMSE,MAE,R-Square and adjusted R-square.

Score in Kaggle

- Our competition kaggle score was 1.2164 and our late submitted score improved to 0.4 RMSE on the validation set

Submission and Description	Private Score	Public Score	Selected
Submission7.csv Complete (after deadline) · KapileshAp · now	0.4282	0.3472	☐
Submission6.csv Complete (after deadline) · KapileshAp · 1h ago	0.6502	0.49565	☐
Submission6.csv Complete · KapileshAp · 9h ago	1.22271	1.12164	☑

References

- https://www.kaggle.com/competitions/cost-prediction-for-logistic-company-fall2023/overview

ANY QUESTIONS?

Thank you