

#<https://www.kaggle.com/datasets/divyeshardeshana/supply-chain-shipment-pricing-data> (DataSet Reference)

```
from google.colab import files
uploaded = files.upload()
```

No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving SCMS_Delivery_History_Dataset.csv to SCMS_Delivery_History_Dataset.csv

```
import pandas as pd

df = pd.read_csv("SCMS_Delivery_History_Dataset.csv",
                 sep="\t",
                 encoding="latin1",
                 on_bad_lines="skip")

df.head()
df.shape
```

(10324, 33)

as per Docotr Hajer recommendation to drop the IDs

```
id_cols = ["ID", "Project Code"]

df = df.drop(columns=id_cols, errors="ignore")
```

Handle Missing Values & Clean Data

Convert target column to numeric

```
df["Freight Cost (USD)"] = pd.to_numeric(df["Freight Cost (USD)"], errors="coerce")
df = df[df["Freight Cost (USD)"].notna()]
```

Numeric columns to clean

```
numeric_cols_fix = [
    "Line Item Quantity", "Line Item Value", "Pack Price", "Unit Price",
    "Line Item Insurance (USD)", "Weight (Kilograms)"
]
```

Convert numerics and drop rows with missing numeric values

```
for col in numeric_cols_fix:
    df[col] = pd.to_numeric(df[col], errors="coerce")
df = df.dropna(subset=numeric_cols_fix)
```

Fix categorical missing values

```
df["Shipment Mode"] = df["Shipment Mode"].fillna(df["Shipment Mode"].mode()[0])
```

```
df.shape
```

(5992, 33)

#Train ML Models + GridSearchCV

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

Split data

```
target = "Freight Cost (USD)"
X = df.drop(columns=[target])
y = df[target]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Preprocessing

```
numeric_cols = X_train.select_dtypes(include=["int64", "float64"]).columns
categorical_cols = X_train.select_dtypes(include=["object"]).columns
```

```
numeric_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
```

```
categorical_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])
```

```
preprocessor = ColumnTransformer([
    ("num", numeric_pipeline, numeric_cols),
    ("cat", categorical_pipeline, categorical_cols)
])
```

Models for GridSearchCV

```
models = {
    "RandomForest": {
        "model": RandomForestRegressor(),
        "params": {"model__n_estimators": [100], "model__max_depth": [10, None]}
    },
    "GradientBoosting": {
        "model": GradientBoostingRegressor(),
        "params": {"model__learning_rate": [0.1], "model__n_estimators": [100]}
    }
}
```

```
results = {}
```

```
for name, config in models.items():
```

```

pipe = Pipeline([("prep", preprocessor), ("model", config["model"])])
grid = GridSearchCV(pipe, config["params"], scoring="r2", cv=3, n_jobs=-1)
grid.fit(X_train, y_train)
preds = grid.predict(X_test)

results[name] = {
    "Best Params": grid.best_params_,
    "Test R2": r2_score(y_test, preds),
    "MAE": mean_absolute_error(y_test, preds),
    "MSE": mean_squared_error(y_test, preds)
}

results

```

```

{'RandomForest': {'Best Params': {'model__max_depth': None,
'model__n_estimators': 100},
'Test R2': 0.6635813084379232,
'MAE': 3851.491893974145,
'MSE': 67621974.47085758},
'GradientBoosting': {'Best Params': {'model__learning_rate': 0.1,
'model__n_estimators': 100},
'Test R2': 0.5623581112884458,
'MAE': 5142.061470157303,
'MSE': 87968383.94566375}}

```

```

#SelectKBest

from sklearn.feature_selection import SelectKBest, f_regression

X_train_processed = preprocessor.fit_transform(X_train)

k = 10
selector = SelectKBest(score_func=f_regression, k=k)
selector.fit(X_train_processed, y_train)

selected_indices = selector.get_support(indices=True)

selected_indices

```

```

/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:381: RuntimeWarning: invalid value encountered in sqrt
X_norms = np.sqrt(row_norms(X.T, squared=True) - n_samples * X_means**2)
array([[ 2,    3,    6,    7, 1935, 6424, 6456, 11553, 17074,
        17171]])

```

```

#RandomForest Feature Importance

rf_model = RandomForestRegressor()
rf_pipeline = Pipeline([("prep", preprocessor), ("model", rf_model)])
rf_pipeline.fit(X_train, y_train)

importances = rf_pipeline.named_steps["model"].feature_importances_

import numpy as np
top10_idx = np.argsort(importances)[-10:]
top10_idx

```

```

array([[10351,    7,    3,    2,    0, 10396,    4, 16942,    5,
        6]])

```

```

#Train Models Using Only Selected Features - Recreate ColumnSelector

from sklearn.base import BaseEstimator, TransformerMixin

class ColumnSelector(BaseEstimator, TransformerMixin):
    def __init__(self, indices):
        self.indices = indices
    def fit(self, X, y=None): return self
    def transform(self, X): return X[:, self.indices]

```

```

#Model A - SelectKBest Reduced Model

kbest_pipeline = Pipeline([
    ("prep", preprocessor),
    ("select", ColumnSelector(selected_indices)),
    ("model", RandomForestRegressor())
])

kbest_pipeline.fit(X_train, y_train)
preds_kbest = kbest_pipeline.predict(X_test)

results_kbest = {
    "R2": r2_score(y_test, preds_kbest),
    "MAE": mean_absolute_error(y_test, preds_kbest),
    "MSE": mean_squared_error(y_test, preds_kbest)
}

results_kbest

```

```

{'R2': 0.36781131918498855,
'MAE': 5926.1197846426685,
'MSE': 127073340.1772045}

```

```

# Model B - RF Feature-Selected Model

rf_fs_pipeline = Pipeline([
    ("prep", preprocessor),
    ("select", ColumnSelector(top10_idx)),
    ("model", RandomForestRegressor())
])

rf_fs_pipeline.fit(X_train, y_train)
preds_rf_fs = rf_fs_pipeline.predict(X_test)

results_rf_fs = {
    "R2": r2_score(y_test, preds_rf_fs),

```

```
"MAE": mean_absolute_error(y_test, preds_rf_fs),  
"MSE": mean_squared_error(y_test, preds_rf_fs)
```

```
results_rf_fs
```

```
{ 'R2': 0.5246152221397784, 'MAE': 5262.722498165138, 'MSE': 95554908.56656656 }
```