```
from google.colab import files
import pandas as pd

# Upload the Excel file
uploaded = files.upload()

# Read the Excel file into a Pandas DataFrame
df = pd.read_excel(next(iter(uploaded)))
```

```
Choose Files 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 UpdatedData.xlsx to UpdatedData.xlsx
```

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

```
# prompt: print column names

print(df.columns.tolist())
```

```
['Date', 'ContractorCode', 'Contractor Name', 'Contractor Mobile', 'Depot', 'Dealer Code', 'Dealer Name', 'Dealer Mobile', 'Call Typ
```

```
# Check for 'Total Volume' column

# Extract month from 'Date' column
df['Month'] = pd.to_datetime(df['Date'], errors='coerce').dt.month



# Aggregate data by month to compute total volume for each month
df_monthly = df.groupby('Month')['Total Volume'].sum().reset_index()

# Display the aggregated data
df_monthly_head = df_monthly.head()
print(df_monthly_head)
```

```
   Month  Total Volume
0      1        1281.4
1      2        1257.5
2      3        1587.0
3      4        1484.4
4      5        1774.0
```

```
# Creating a synthetic dataset for months 6 to 12
import numpy as np
np.random.seed(42)  # For reproducibility
remaining_months = np.arange(6, 13)  # Months from June to December
total_volume_remaining = np.random.randint(1000, 5000, size=7)  # Random total volumes between 1000 and 5000

# Creating DataFrame for remaining months
remaining_data = pd.DataFrame({'Month': remaining_months, 'Total Volume': total_volume_remaining})

# Display the created dataset for remaining months
print(remaining_data)
```

```
   Month  Total Volume
0      6          4174
1      7          4507
2      8          1860
3      9          2294
4     10          2130
5     11          2095
6     12          4772
```

```
import pandas as pd

# Create the first DataFrame
data1 = {'Month': [6, 7, 8, 9, 10, 11, 12],
         'Total Volume': [4174, 4507, 1860, 2294, 2130, 2095, 4772]}
df1 = pd.DataFrame(data1)

# Create the second DataFrame
data2 = {'Month': [1, 2, 3, 4, 5],
         'Total Volume': [1281.4, 1257.5, 1587.0, 1484.4, 1774.0]}
df2 = pd.DataFrame(data2)

# Concatenate the DataFrames vertically
result = pd.concat([df2, df1], axis=0, ignore_index=True)

# Sort the DataFrame based on the 'Month' column in ascending order
result.sort_values(by='Month', inplace=True)

# Reset index
result.reset_index(drop=True, inplace=True)

# Print the result
print(result)
```

```
    Month  Total Volume
0       1        1281.4
1       2        1257.5
2       3        1587.0
3       4        1484.4
4       5        1774.0
5       6        4174.0
6       7        4507.0
7       8        1860.0
8       9        2294.0
9      10        2130.0
10     11        2095.0
11     12        4772.0
```

```
import pandas as pd

# Load the dataset
data = {'Month': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
        'Total Volume': [1281.4, 1257.5, 1587.0, 1484.4, 1774.0, 4174.0, 4507.0, 1860.0, 2294.0, 2130.0, 2095.0, 4772.0]}
df = pd.DataFrame(data)

# No missing values, outliers, or categorical features to handle, so no preprocessing required for this dataset
```

```
# Evaluate the model using Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred=)
print("Mean Absolute Error (MAE):", mae)
```

```
Mean Absolute Error (MAE): 715.8821666666664
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error

# Load the dataset
data = {'Month': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
        'Total Volume': [1281.4, 1257.5, 1587.0, 1484.4, 1774.0, 4174.0, 4507.0, 1860.0, 2294.0, 2130.0, 2095.0, 4772.0]}
df = pd.DataFrame(data)

# Split the dataset into features (X) and target variable (y)
X = df[['Month']]
y = df['Total Volume']

# Scale the features and target variable
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
y_scaled = scaler.fit_transform(y.values.reshape(-1, 1))

# Split the scaled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Define the Random Forest regressor
rf_model = RandomForestRegressor(random_state=42)

# Define the hyperparameters to tune
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30]
}

# Perform grid search cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train, y_train)

# Get the best model from grid search
best_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Inverse scaling the predictions
# Evaluate the model using Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

# Displaying predictions
print('Predictions for Test Set:')
for month, pred in zip(X_test, y_pred):
    print(f'Month {month}: Predicted Total Volume = {pred:.2f}')
```

```
Mean Absolute Error (MAE): 715.8821666666664
Predictions for Test Set:
Month [1.30357228]: Predicted Total Volume = 3863.49
Month [1.01388955]: Predicted Total Volume = 2385.30
Month [-1.59325501]: Predicted Total Volume = 1405.25
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error

# Load the dataset
data = {'Month': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
        'Total Volume': [1281.4, 1257.5, 1587.0, 1484.4, 1774.0, 4174.0, 4507.0, 1860.0, 2294.0, 2130.0, 2095.0, 4772.0]}
df = pd.DataFrame(data)

# Split the dataset into features (X) and target variable (y)
X = df[['Month']]
y = df['Total Volume']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the Gradient Boosting regressor
gb_model = GradientBoostingRegressor(random_state=42)

# Define the hyperparameters to tune
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.1, 0.05, 0.01],
    'max_depth': [3, 4, 5]
}

# Perform grid search cross-validation to find the best hyperparameters
grid_search = GridSearchCV(gb_model, param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train, y_train)

# Get the best model from grid search
best_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Evaluate the model using Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)
```

```
Mean Absolute Error (MAE): 954.9666666544894
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_absolute_error

# Load the dataset
data = {'Month': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
        'Total Volume': [1281.4, 1257.5, 1587.0, 1484.4, 1774.0, 4174.0, 4507.0, 1860.0, 2294.0, 2130.0, 2095.0, 4772.0]}
df = pd.DataFrame(data)

# Split the dataset into features (X) and target variable (y)
X = df[['Month']]
y = df['Total Volume']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define a pipeline with PolynomialFeatures and GradientBoostingRegressor
pipeline = make_pipeline(PolynomialFeatures(degree=2), StandardScaler(), GradientBoostingRegressor(random_state=42))

# Define the hyperparameters to tune
param_grid = {
    'gradientboostingregressor__n_estimators': [100, 200, 300,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500,1600,1700,1800,1900,
    'gradientboostingregressor__learning_rate': [0.1, 0.05, 0.01,0.05,0.001,0.0005,0.00003,0.0009],
    'gradientboostingregressor__max_depth': [3, 4, 5,6,7,8,9,10]
}

# Perform grid search cross-validation to find the best hyperparameters
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train, y_train)

# Get the best model from grid search
best_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Evaluate the model using Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)



# Feature Engineering
# Seasonality: Create binary variables indicating different seasons
df['Winter'] = df['Month'].isin([12, 1, 2]).astype(int)
df['Spring'] = df['Month'].isin([3, 4, 5]).astype(int)
df['Summer'] = df['Month'].isin([6, 7, 8]).astype(int)
df['Fall'] = df['Month'].isin([9, 10, 11]).astype(int)

# Trend: Encode a linear trend based on the 'Month' variable
df['Trend'] = df['Month']

print(df)
```

```
    Month  Total Volume  Winter  Spring  Summer  Fall  Trend
0       1        1281.4       1       0       0     0      1
1       2        1257.5       1       0       0     0      2
2       3        1587.0       0       1       0     0      3
3       4        1484.4       0       1       0     0      4
4       5        1774.0       0       1       0     0      5
5       6        4174.0       0       0       1     0      6
6       7        4507.0       0       0       1     0      7
7       8        1860.0       0       0       1     0      8
8       9        2294.0       0       0       0     1      9
9      10        2130.0       0       0       0     1     10
10     11        2095.0       0       0       0     1     11
11     12        4772.0       1       0       0     0     12
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_absolute_error

# Load the dataset
data = {'Month': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
        'Total Volume': [1281.4, 1257.5, 1587.0, 1484.4, 1774.0, 4174.0, 4507.0, 1860.0, 2294.0, 2130.0, 2095.0, 4772.0]}
df = pd.DataFrame(data)

# Split the dataset into features (X) and target variable (y)
X = df[['Month']]
y = df['Total Volume']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Selection and Hyperparameter Tuning
# Linear Regression
linear_model = make_pipeline(StandardScaler(), LinearRegression())
linear_model.fit(X_train, y_train)

# Random Forest Regression
rf_model = RandomForestRegressor(random_state=42)
rf_param_grid = {'n_estimators': [100, 200, 300],
                 'max_depth': [None, 10, 20]}
rf_grid_search = GridSearchCV(rf_model, rf_param_grid, cv=5)
rf_grid_search.fit(X_train, y_train)



x_remaining = result[['Month']]  # Predictor
y_remaining = result['Total Volume']  # Response


from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression

# Assuming x_remaining and y_remaining are the feature and target variable DataFrames

# Scale the features and target variable
scaler = StandardScaler()
x_remaining_scaled = scaler.fit_transform(x_remaining)
y_remaining_scaled = scaler.fit_transform(y_remaining.values.reshape(-1, 1))

# Feature Engineering: Example using PolynomialFeatures
poly_features = PolynomialFeatures(degree=2)
x_remaining_poly = poly_features.fit_transform(x_remaining_scaled)

# Fitting the linear regression model to the remaining months
model_remaining = LinearRegression()
model_remaining.fit(x_remaining_poly, y_remaining_scaled)

# Making predictions for all months in the dataset
predictions_remaining_scaled = model_remaining.predict(x_remaining_poly)

# Inverse scaling the predictions
predictions_remaining = scaler.inverse_transform(predictions_remaining_scaled)

# Displaying predictions
print('Predictions for Remaining Months:')
for month, pred in zip(x_remaining['Month'], predictions_remaining):
    print(f'Month {month}: Predicted Total Volume = {pred[0]:.2f}')
```

```
Predictions for Remaining Months:
Month 1: Predicted Total Volume = 1146.42
Month 2: Predicted Total Volume = 1468.88
Month 3: Predicted Total Volume = 1764.86
Month 4: Predicted Total Volume = 2034.38
Month 5: Predicted Total Volume = 2277.44
Month 6: Predicted Total Volume = 2494.02
Month 7: Predicted Total Volume = 2684.14
Month 8: Predicted Total Volume = 2847.80
Month 9: Predicted Total Volume = 2984.98
Month 10: Predicted Total Volume = 3095.70
Month 11: Predicted Total Volume = 3179.95
Month 12: Predicted Total Volume = 3237.73
```

```
from sklearn.metrics import mean_absolute_error

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_remaining, predictions_remaining)
print('Mean Absolute Error (MAE):', mae)
```

```
Mean Absolute Error (MAE): 862.0135364635363
```

```
# prompt: Using dataframe df: diagnostic steps and code for the given dataset df

# Check the shape of the dataframe
df.shape

# Check the data types of each column
df.info()

# Check for missing values
df.isnull().sum()

# Check the summary statistics of the numerical columns
df.describe()

# Check the unique values for each categorical column
for col in df.select_dtypes(include='object'):
    print(f"Unique values for column '{col}':")
    print(df[col].unique())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Columns: 155 entries, Date to Zone
dtypes: datetime64[ns](1), float64(24), int64(116), object(14)
memory usage: 1.2+ MB
Unique values for column 'Contractor Name':
['VINEET KUM' 'BIJOY BARI' 'UTTAM KHUT' 'OM PRAKASH' 'PRADIPBHAI'
 'SUDARSHANB' 'GHANSHYAM ' 'MHO SADIQ ' 'RAJKUMAR D' 'BHARATKUMA'
 'NAKKA SUDH' 'VILAS LAXA' 'TAPAS DHAR' 'MURALI N. ' 'GIRI VISHW'
 'SHAMBHUNAT' 'AKBAR KHAN' 'SUNDAR LAL' 'MORLA SIVA' 'BASANT LAL'
 'RAHUL GHOS' 'MAYURBHAI ' 'KISHAN SUJ' 'ANAND. MS ' 'ARVIND KUM'
 'ABHISHEK S' 'IRSHAD ANS' 'SHEKH AZAH' 'CHANDAN PH' 'SURESH KUM'
 'AMARJEET B' 'YASMIN BAN' 'MITHLESH K' 'CHIRAYU LA' 'MANOJ KUMA'
 'ANIL PRAKA' 'VIJAY SO H' 'JEETU' 'SHAIK MANS' 'MOHD AFSAA' 'RAMKISHAN'
 'RAMKUMAR R' 'MUSTAK S/O' 'SHEKHAR YA' 'SUNIL KURA' 'JIARUL RAH'
 'VIJAY UTTA' 'CG 2016 - ' 'AMARNATH S' 'RABI NARAY' 'PREMANAND '
 'CHITRASEN ' 'VIKESH MAH' 'MAFIJUL MO' 'RAGHVENDRA' 'SONU (2528'
 'NAUSHAD' 'LOKESH KUM' 'MD IMRAN A' 'SANTOSH PA' 'RAMPARVESH'
 'RATAN HALD' 'RINKU' 'NABI HUSSA' 'VAIBHAV RA' 'AMAN KUMAR' 'ISMAIL GUL'
 'BIJOY ACHA' 'JAKIR GAFA' 'MOHARSINGH' 'NAJAM KHAN' 'AMIT DARJI'
 'RAM SAJIWA' 'ABDUL KADI' 'DINANATHBH' 'JAYDEB IND' 'MADHUSUDAN'
 'JAI PRAKAS' 'RAJNATH GU' 'JOSEPH GOW' 'NEPAL SIKD' 'RAKESH YAD'
 'RAMNAND VI' 'SOYEB ALAM' 'UPPARI VEN' 'RAMASHANKE' 'VIKAS MARU'
 'BISWAJIT S' 'PRAMOD RAO' 'JITENDRAKU' 'KRUSHNA CH' 'LOKENDRASI'
 'SANTANU SE' 'MD NABI RA' 'AMJAD MALI' 'AMIT SADHU' 'NASER MOLL'
 'AKHILESH N' 'PALASH BIS' 'BAKULBHAI ' 'KESHAV DEV' 'MR MOHD FA'
 'LAXMIDHAR ' 'MANZOOR AH' 'MAHADEB SA' 'MR JAGDISH' 'DEEPAK AHI'
 'TAPAN RAI' 'SAINUL ABI' 'TASAWWAR K' 'SOMESH PUR' 'SHITAL MAI'
 'NITYANAND ' 'MOHAMMED F' 'IKRAR AHME' 'SUNIL KUMA' 'PRASANTA K'
 'MR PRAKASH' 'SANJAY SIN' 'SUBROTA HA' 'BABU LAL C' 'BHAGVANTLA'
 'SATYA RANJ' 'BIJU B (25' 'NEERAJ' 'VINOD BHUP' 'MAHESH MAN' 'MR. AKASH'
 'TAPAN KUMA' 'KAMAL PATR' 'MR. SANDIP' 'AJOY BISWA' 'SALMAN KHA'
 'PATLE SHIV' 'RABINDRA R' 'ABHIMAN BH' 'JUSTIN NON' 'ASHOK KUMA'
 'GANGARAM Y' 'SANTOSHKUM' 'LALPATI SH' 'BABU DAS (' 'SK SELIM ('
 'RINGANIA A' 'NARAYAN BA' 'MR. MUKESH' 'SHAIKH IMR' 'SARFARAJ H'
 'KAMDEV RAV' 'NASIR' 'NAFIS AHIM' 'MEET JAGDI' 'BUDHIRAM M' 'JOBIN T J'
 'DASHRATH D' 'KARTICK MA' 'AXAYKUMAR ' 'BHALCHANDR' 'MR. SARMAN'
 'RANJITH (2' 'RAJENDRA K' 'EJAJKHAN S' 'AAKASH LAL' 'VIJAY SING'
 'JAMSHEER M' 'SIDDIQUE (' 'VADLA MAHE' 'MR MD SADD' 'BHAGWAT SH'
 'SATELU TIT' 'MALIK KHAN' 'DHARMENDRA' 'MANOJ T S' 'CHHOTU KUM'
 'DINESH CHA' 'MD ABDHUL ' 'ADARSH DIP' 'KULDEEP' 'BAPI MONDA'
 'RAVINDRA K' 'SANDEEP RA' 'BAIDUL RAH' 'VIRENDRA K' 'VIRENDRA R'
 'GANESH SAN' 'DULLOLA KR' 'SANTOSH MA' 'SHAJIKUMAR' 'SANJIB GHO'
 'RAMSINGH M' 'MR. DINESH' 'SURYABHAN ' 'GOBINDA MA' 'SEKH AHAMM'
 'RAJA RAM (' 'SIRAJUDDIN' 'TALIB (252' 'NATHU LAL ' 'ANOOP KUMA'
 'SUKHWINDER' 'NANDESHWAR' 'ARJUN KUMA' 'MR NITOO T' 'SURESHBHAI'
 'AJEET KUMA' 'MIZANUR RA' 'LALGI PAL ' 'SANJAY KUM' 'ROHAN CHAN'
 'UMESH KHUS' 'HEMANT KUM' 'RAMDARASH ' 'SANJIT RAN' 'MR. RAVIND'
 'LALIT KUMA' 'RAMMILAN R' 'CHANDAN CH' 'PRABIR ADH' 'AINUL SEKH'
 'GAYAPRASAD' 'JAIMANAGAL' 'NEPAL (252' 'SK HALIM (' 'RAJESH JIY'
 'BHAIRA RAM' 'NAJMUS SAH' 'BHAGWAN SI' 'PAPPU (252' 'MOHAN KUMA'
 'RAMVIJAY S' 'GOKUL BISW' 'LAXMAN NIS' 'KAMLESH' 'MUKESH KUM'
 'VINODKUMAR' 'MALAY SANT' 'SAJEDUL SE' 'HAPIJUL SA' 'MR JITENDR'
 'SUDHAKARAN' 'MR. SK HAY' 'SUNIL CHAU' 'MANJOOR AN' 'GAJANAN AM'
 'KONKATI GO' 'KUNDAN OJH' 'SREEJITH P' 'RAMDEO PRA' 'VISHNUBHAI'
 'RANDERA TO' 'NISAR AHAM' 'SANJIB MON' 'DEENANATH ' 'SAMME LAL '
 'PAWAN KUMA' 'NANDALAL S' 'VASIM AHAM' 'MR MAKARDH' 'MR Mohit'
 'SUKENDRA S' 'MOHAN (252' 'RUTVIKKUMA' 'JAMAL N K ' 'SALIMMALIK'
```

```
# Calculate the mode for each categorical column
for col in df.select_dtypes(include='object'):
    print(f"Mode for column '{col}':")
    print(df[col].mode())

# Calculate the mean and standard deviation for each numerical column
for col in df.select_dtypes(include='number'):
    print(f"Mean for column '{col}':")
    print(df[col].mean())
    print(f"Standard deviation for column '{col}':")
    print(df[col].std())
```

```
count_of_transactions_by_contractor_grouby_unique_paints=df.groupby("Contractor Name")["XTVmbianceFinishes_vol","JUHUXVTFinishes_vol",":
```

```
<ipython-input-18-2312d2c2778c>:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be depr
  count_of_transactions_by_contractor_grouby_unique_paints=df.groupby("Contractor Name")["XTVmbianceFinishes_vol","JUHUXVTFinishes_v
```

```
count_of_transactions_by_contractor_grouby_unique_paints
```

Contractor Name	XTVmbianceFinishes_vol	JUHUXVTFinishes_vol	JUHUXVTDIAMONDGLO_vol	DNAmbianceVelvetTouch_vol	JUHUXVTPearlGlo_vol	JUH
1. BHARATB	1	1	1	1	1	
AAKASH LAL	2	2	2	2	2	
AARIF	1	1	1	1	1	
ABDEALI KU	1	1	1	1	1	
ABDUL GAFF	1	1	1	1	1	
...	...	...	...	...	...	
YASMIN BAN	1	1	1	1	1	
YOGENDRA K	1	1	1	1	1	
YOGENDRA Y	1	1	1	1	1	
YOGESHA RA	1	1	1	1	1	
ZAFER ALI	1	1	1	1	1	

923 rows × 58 columns

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X = df[ [
     "JUHUXVTFinishes_vol", "JUHUXVTDIAMONDGLO_vol", "DNAmbianceVelvetTouch_vol",
    "JUHUXVTPearlGlo_vol", "JUHUXVTPlatGlo_vol", "JUHUXSuperclean3IN1_vol", "JUHUXSuperCoverSheen_vol",
    "JUHUXSuperCover_vol", "JUHUXWSPowerflexx_vol", "JUHUXWeathershieldMax_vol", "JUHUXWeathershieldTile_vol",
    "JUHUXWSFlash_vol", "JUHUXWeathershield_vol", "JUHUXAFuatechFlexibleWaterproo_vol", "JUHUXWSSignature_vol",
    "JUHUXAFuatechWaterproofBasecoa_vol", "DNAFuatechRoofCoat_vol", "JUHUXAFuatechWaterblock2K_vol",
    "JUHUXAFuatechCrackFiller_vol", "DNAFuatechInterior1K_vol", "DNAFuatechPTC_vol", "JUHUXPromiseExterior_vol",
    "JUHUXPromiseInterior_vol", "Faolin1KPUClrTopCoat_vol", "Faolin2KPUClrSealer_vol", "Faolin2KPUCLRTopCoat_vol",
    "Faolin2KPUOPQPmrSfr_vol", "Faolin2KPUOPQTopCoat_vol", "Faolin2KPUThinner_vol", "FaolinMelamineSealer_vol",
    "FaolinMelamineThinner_vol", "FaolinMelamineTopCoat_vol", "FaolinNCClrTopCoat_vol", "FaolinNCOPQTopCoat_vol",
    "FaolinNCSealer_vol", "JUHUXWSAlkaliBloc_vol", "ICIDuwelEAP_vol", "JUHUXWBCementPrimer_vol", "JUHUXSuperClean_vol",
    "DNAFuatechAdditive_vol", "FaolinNCThinner_vol", "DNAFuatechExtBasecoat_vol", "PromiseSheenExterior_vol",
    "PromiseSheenInterior_vol", "JUHUXPromisePrimer_vol", "ICIDuwelIAP_vol", "DNBetterLivingAirClean_vol",
    "FaolinEpoxyInsulator_vol", "DNFaolinLUXURIOPUSEALER_vol", "DNFaolinLUXURIOPUGLOSS_vol", "DNFaolinLUXURIOPUMATT_vol",
    "WaterBasedSatin_vol", "LustreFinish_vol", "JUHUXFLOORPLUS_vol", "DNAFuatechPUCoat_Vol", "JUHUXWeathershieldClear_vol",
    "JUHUXWeathershieldProtectRainproof_vol", "JUHUXGuardian_vol"]]
y = df['XTVmbianceFinishes_vol']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the testing set
predictions = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)

# If needed, you can also print the coefficients and intercept of the model
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
[ 1.04727573e-03  1.56247659e-03  1.00661857e-03  1.62129096e-03
  1.01985078e-03  1.89535361e-02 -1.36863566e-03  1.32958158e-03
  3.13050488e-03 -3.51169626e-02  5.90647398e-03  1.00661857e-03
  1.57341268e-03 -9.19459618e-03 -5.92956630e-04  1.44351742e-03
  1.31289104e-03 -6.96872004e-04 -2.51018965e-03  1.31289104e-03
  3.68492835e-03  1.00661857e-03  1.74896638e-03 -1.45290070e-02
  1.43517215e-03 -3.99507348e-03  1.10170990e-03  3.86238296e-03
 -7.21263509e-04  1.26575020e-03  1.70600420e-03 -1.35957731e-03
  1.87664179e-03  1.56749992e-03 -2.12012633e-02 -7.21263509e-04
  1.36926060e-03  2.33025569e-02  1.10658820e-03 -7.21263509e-04
 -7.06892469e-04  1.45553600e-03  1.10658820e-03 -8.33979772e-04
  1.50097019e-03  1.44351742e-03  1.33716593e-03  5.00263525e-03
  6.24532693e-03  1.54522515e-03 -4.57800796e-03  4.82248313e-04
 -3.76484796e-05  1.72457597e-03 -2.18847189e-03  1.57341268e-03
  1.11687861e-03  8.68909904e-04  1.54522515e-03  1.50857470e-03
  5.64107432e-04  5.38280707e-04  1.32958158e-03  1.50854081e-03
 -3.97545586e-03  1.10170990e-03 -3.98765161e-03 -1.03339546e-02
  1.44351742e-03  1.32958158e-03  1.54520359e-03  3.82769099e-04
 -5.92956630e-04 -7.21263509e-04 -2.92413675e-03  1.54522515e-03
  1.49472237e-03 -7.21263509e-04  4.93966812e-02 -1.99466456e-02
  3.31392908e-04  1.67408535e-03  1.48859892e-03  4.86388100e-04
 -2.99998027e-03 -7.59355825e-04  4.47722431e-02  5.90647398e-03
  1.35338225e-03  2.35729311e-04  1.62129096e-03  7.71799081e-04
  1.54086224e-03  1.11687861e-03 -4.48608254e-03  1.48859892e-03
  2.69452613e-03  1.07044708e-03  7.82655445e-04  1.30948425e-03
  1.11687861e-03  3.04159035e-04  1.30652972e-03  1.65983879e-03
  7.06879821e-04  1.60768655e-03 -3.08447602e-02 -5.92956630e-04
  1.95765843e-03  3.25171800e-03  1.56247659e-03 -6.96872004e-04
 -7.21263509e-04 -2.90574800e-03  1.38237686e-03  1.41974459e-03
  1.10658820e-03  1.89535361e-02 -4.93391833e-04 -7.21263509e-04
  1.54522515e-03 -3.69438061e-03  1.02554947e-02  7.79868206e-05
  1.21488483e-03  1.31289104e-03 -6.18877972e-04  3.17608243e-04
  1.64216649e-03  1.10170990e-03  1.33716593e-03 -7.21263509e-04
  2.57401210e-03  1.87664179e-03 -3.09824689e-02  8.03244198e-02
  1.16462617e-03  4.30290626e-04 -9.17159328e-04  1.62129096e-03
  1.37343195e-03 -2.05775636e-02  4.36153072e-03  1.65791985e-03
  1.14432725e-03  1.43484722e-03  7.54908055e-02  1.00661857e-03
  1.43517215e-03  3.02835575e-04  3.99302375e-04 -8.35199348e-04
 -6.30319669e-04  2.40458563e-03  4.30290626e-04 -6.45419986e-04
  1.32958158e-03  1.29486156e-03  7.25055300e-04  1.59262247e-03
  1.00661857e-03  1.44351742e-03  3.19057745e-03 -1.68276809e-03
  7.82590834e-04  1.30948425e-03 -7.21263509e-04 -5.92956630e-04
```

```
  1.33716593e-03 -1.96963307e-03  1.32958158e-03  1.72457597e-03
  1.41974459e-03 -6.45419986e-04  1.33716593e-03  1.72457597e-03
 -7.21263509e-04  1.41974459e-03  4.43591171e-02  1.11687861e-03
 -6.96872004e-04  5.90647398e-03 -1.49190708e-02  1.86005321e-02
  2.15708978e-03  1.50854081e-03 -3.81857146e-03  1.28203592e-03
  1.56247659e-03  1.44351742e-03 -1.86062189e-03 -2.72218935e-04
  4.18094874e-04 -7.21263509e-04 -2.99998027e-03  9.37530750e-04
 -7.21263509e-04  8.68909904e-04  1.01985258e-02  1.95931964e-03]
Mean Squared Error: 0.004940848907859593
Coefficients: [ 2.88629769e-19  4.34902072e-03  6.45100293e-18 -6.88543352e-05
 -6.27308841e-05 -6.13030174e-05  3.52365706e-18 -1.12716263e-04
 -8.10055336e-04 -5.37602472e-04 -9.36481286e-04 -3.90312782e-18
 -1.13935838e-04 -1.31295847e-05  5.42101086e-18  1.26273595e-04
 -2.55088764e-05 -6.30566229e-04 -7.29257642e-04 -1.76502022e-04
  4.33680869e-18  5.02332978e-06 -1.22281106e-05  3.25260652e-19
 -3.89363314e-04  5.42101086e-19  4.33680869e-19 -2.16840434e-19
```

```
rmse = mean_squared_error(y_test,predictions,squared=False)
print("Root Mean Squared Error:", rmse)
```

```
Root Mean Squared Error: 0.07029117233237467
```

```
X_single = X_test.iloc[0].values.reshape(1, -1)

# Make prediction on the single data point
pred = model.predict(X_single)

print("Prediction:", pred)
```

```
Prediction: [0.00104728]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
```

Start coding or generate with AI.

```
paintcolumns=["XTVmbianceFinishes_vol","JUHUXVTFinishes_vol","JUHUXVTDIAMONDGLO_vol","DNAmbianceVelvetTouch_vol","JUHUXVTPearlGlo_vol",'
Sum_of_unique_paint_groupby_contractorname=df.groupby("Contractor Name")[paintcolumns].sum()
Sum_of_unique_paint_groupby_contractorname[paintcolumns].sum()
```

```
JUHUXVTFinishes_vol                        0.0
JUHUXVTDIAMONDGLO_vol                    439.0
DNAmbianceVelvetTouch_vol                  0.0
JUHUXVTPearlGlo_vol                      903.6
JUHUXVTPlatGlo_vol                       123.0
JUHUXSuperclean3IN1_vol                  116.0
JUHUXSuperCoverSheen_vol                   0.0
JUHUXSuperCover_vol                     1016.0
JUHUXWSPowerflexx_vol                    642.0
JUHUXWeathershieldMax_vol               1622.0
JUHUXWeathershieldTile_vol                7.0
JUHUXWSFlash_vol                          0.0
JUHUXWeathershield_vol                 3589.0
JUHUXAFuatechFlexibleWaterproo_vol     1331.0
JUHUXWSSignature_vol                      0.0
JUHUXAFuatechWaterproofBasecoa_vol       81.0
JUHUXAFuatechWaterblock2K_vol             6.0
JUHUXAFuatechCrackFiller_vol             32.3
DNAFuatechInterior1K_vol                104.0
DNAFuatechPTC_vol                         0.0
JUHUXPromiseExterior_vol                1035.0
JUHUXPromiseInterior_vol                 896.0
Faolin1KPUClrTopCoat_vol                  0.0
Faolin2KPUClrSealer_vol                   4.0
Faolin2KPUCLRTopCoat_vol                  0.0
Faolin2KPUOPQPmrSfr_vol                   0.0
Faolin2KPUOPQTopCoat_vol                  0.0
Faolin2KPUThinner_vol                     0.0
FaolinMelamineSealer_vol                  0.0
FaolinMelamineThinner_vol                 0.0
FaolinMelamineTopCoat_vol                 0.0
FaolinNCClrTopCoat_vol                    0.0
FaolinNCOPQTopCoat_vol                    0.0
FaolinNCSealer_vol                        0.0
JUHUXWSAlkaliBloc_vol                   540.0
ICIDuwelEAP_vol                         245.0
JUHUXWBCementPrimer_vol                 112.0
JUHUXSuperClean_vol                     969.0
DNAFuatechAdditive_vol                    6.0
FaolinNCThinner_vol                       0.0
DNAFuatechExtBasecoat_vol               537.0
PromiseSheenExterior_vol                867.0
PromiseSheenInterior_vol                878.0
JUHUXPromisePrimer_vol                    0.0
ICIDuwelIAP_vol                           0.0
```

```
FaolinEpoxyInsulator_vol                    0.0
DNFaolinLUXURIOPUSEALER_vol                 0.0
DNFaolinLUXURIOPUGLOSS_vol                 10.0
DNFaolinLUXURIOPUMATT_vol                  19.0
WaterBasedSatin_vol                       251.0
LustreFinish_vol                            0.0
JUHUXFLOORPLUS_vol                         34.0
DNAFuatechPUCoat_Vol                        0.0
JUHUXWeathershieldClear_vol                 0.0
JUHUXWeathershieldProtectRainproof_vol    299.0
JUHUXGuardian_vol                           0.0
dtype: float64
```

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Assuming df is your DataFrame containing the data
# Replace df with your actual DataFrame
data = df[[ "JUHUXVTFinishes_vol", "JUHUXVTDIAMONDGLO_vol", "DNAmbianceVelvetTouch_vol",
    "JUHUXVTPearlGlo_vol", "JUHUXVTPlatGlo_vol", "JUHUXSuperclean3IN1_vol", "JUHUXSuperCoverSheen_vol",
    "JUHUXSuperCover_vol", "JUHUXWSPowerflexx_vol", "JUHUXWeathershieldMax_vol", "JUHUXWeathershieldTile_vol",
    "JUHUXWSFlash_vol", "JUHUXWeathershield_vol", "JUHUXAFuatechFlexibleWaterproo_vol", "JUHUXWSSignature_vol",
    "JUHUXAFuatechWaterproofBasecoa_vol", "DNAFuatechRoofCoat_vol", "JUHUXAFuatechWaterblock2K_vol",
    "JUHUXAFuatechCrackFiller_vol", "DNAFuatechInterior1K_vol", "DNAFuatechPTC_vol", "JUHUXPromiseExterior_vol",
    "JUHUXPromiseInterior_vol", "Faolin1KPUClrTopCoat_vol", "Faolin2KPUClrSealer_vol", "Faolin2KPUCLRTopCoat_vol",
    "Faolin2KPUOPQPmrSfr_vol", "Faolin2KPUOPQTopCoat_vol", "Faolin2KPUThinner_vol", "FaolinMelamineSealer_vol",
    "FaolinMelamineThinner_vol", "FaolinMelamineTopCoat_vol", "FaolinNCClrTopCoat_vol", "FaolinNCOPQTopCoat_vol",
    "FaolinNCSealer_vol", "JUHUXWSAlkaliBloc_vol", "ICIDuwelEAP_vol", "JUHUXWBCementPrimer_vol", "JUHUXSuperClean_vol",
    "DNAFuatechAdditive_vol", "FaolinNCThinner_vol", "DNAFuatechExtBasecoat_vol", "PromiseSheenExterior_vol",
    "PromiseSheenInterior_vol", "JUHUXPromisePrimer_vol", "ICIDuwelIAP_vol", "DNBetterLivingAirClean_vol",
    "FaolinEpoxyInsulator_vol", "DNFaolinLUXURIOPUSEALER_vol", "DNFaolinLUXURIOPUGLOSS_vol", "DNFaolinLUXURIOPUMATT_vol",
    "WaterBasedSatin_vol", "LustreFinish_vol", "JUHUXFLOORPLUS_vol", "DNAFuatechPUCoat_Vol", "JUHUXWeathershieldClear_vol",
    "JUHUXWeathershieldProtectRainproof_vol", "JUHUXGuardian_vol"]]

# Initialize the KMeans model with desired number of clusters
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the model to your data
kmeans.fit(data)

# Get the cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_

# Add cluster labels to the DataFrame
df['Cluster'] = labels

# Visualize the clusters (assuming you have 2 features)
plt.scatter(data['XTVmbianceFinishes_vol'], data[], c=labels, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.title('KMeans Clustering')
plt.show()

# You can further analyze and interpret the clusters based on your problem domain and requirements
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change frc
  warnings.warn(
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

                                 4 frames

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'XTVmbianceFinishes_vol'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'XTVmbianceFinishes_vol'
```

```
paintcolumns=["XTVmbianceFinishes_vol","JUHUXVTFinishes_vol","JUHUXVTDIAMONDGLO_vol","DNAmbianceVelvetTouch_vol","JUHUXVTPearlGlo_vol","
volume=[2.0, 0.0, 439.0, 0.0, 903.6, 123.0, 116.0, 0.0, 1016.0, 642.0, 1622.0, 7.0, 0.0, 3589.0, 1331.0, 0.0, 81.0, 6.0, 32.3, 104.0, 0
```

```
paintdf=pd.Dataframe(['paint type':paintcolumns,'volume':volume])
```

```
  File "<ipython-input-5-caf8a6cb498b>", line 1
    paintdf=pd.Dataframe(['paint type':paintcolumns,'volume':volume])
                                       ^
SyntaxError: invalid syntax
```

```
import pandas as pd

# Given data
paint_columns = ["XTVmbianceFinishes_vol","JUHUXVTFinishes_vol","JUHUXVTDIAMONDGLO_vol","DNAmbianceVelvetTouch_vol","JUHUXVTPearlGlo_vol
volume = [2.0, 0.0, 439.0, 0.0, 903.6, 123.0, 116.0, 0.0, 1016.0, 642.0, 1622.0, 7.0,0.0, 3589.0, 1331.0, 0.0, 81.0, 6.0, 32.3, 104.0, 0

# Create DataFrame
df = pd.DataFrame({'PaintColumns': paint_columns, 'Volume': volume})

# Display DataFrame
df
```

	PaintColumns	Volume
0	XTVmbianceFinishes_vol	2.0
1	JUHUXVTFinishes_vol	0.0
2	JUHUXVTDIAMONDGLO_vol	439.0
3	DNAmbianceVelvetTouch_vol	0.0
4	JUHUXVTPearlGlo_vol	903.6
5	JUHUXVTPlatGlo_vol	123.0
6	JUHUXSuperclean3IN1_vol	116.0
7	JUHUXSuperCoverSheen_vol	0.0
8	JUHUXSuperCover_vol	1016.0
9	JUHUXWSPowerflexx_vol	642.0
10	JUHUXWeathershieldMax_vol	1622.0
11	JUHUXWeathershieldTile_vol	7.0
12	JUHUXWSFlash_vol	0.0
13	JUHUXWeathershield_vol	3589.0
14	JUHUXAFuatechFlexibleWaterproo_vol	1331.0
15	JUHUXWSSignature_vol	0.0
16	JUHUXAFuatechWaterproofBasecoa_vol	81.0
17	JUHUXAFuatechWaterblock2K_vol	6.0
18	JUHUXAFuatechCrackFiller_vol	32.3
19	DNAFuatechInterior1K_vol	104.0
20	DNAFuatechPTC_vol	0.0
21	JUHUXPromiseExterior_vol	1035.0
22	JUHUXPromiseInterior_vol	896.0
23	Faolin1KPUClrTopCoat_vol	0.0
24	Faolin2KPUClrSealer_vol	4.0
25	Faolin2KPUCLRTopCoat_vol	0.0
26	Faolin2KPUOPQPmrSfr_vol	0.0
27	Faolin2KPUOPQTopCoat_vol	0.0
28	Faolin2KPUThinner_vol	0.0
29	FaolinMelamineSealer_vol	0.0
30	FaolinMelamineThinner vol	0.0