```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
import pandas as pd
import numpy as np
```

```
# Read in the data
shipment_pricing = '/content/gdrive/MyDrive/Supply_Chain_Shipment_Pricing_Dataset.csv'

shipment_pricing = pd.read_csv(shipment_pricing)
```

```
shipment_pricing.head()
```

	id	project code	pq #	po / so #	asn/dn #	country	managed by	fulfill via	vendor inco term	shipment mode	...	unit of measure (per pack)	line item quantity	line item value	pack price	unit price	manuf
0	1	100-CI-T01	Pre-PQ Process	SCMS-4	ASN-8	Côte d'Ivoire	PMO - US	Direct Drop	EXW	Air	...	30	19	551.0	29.00	0.97	Rar Cher
1	3	108-VN-T01	Pre-PQ Process	SCMS-13	ASN-85	Vietnam	PMO - US	Direct Drop	EXW	Air	...	240	1000	6200.0	6.20	0.03	Auro
2	4	100-CI-T01	Pre-PQ Process	SCMS-20	ASN-14	Côte d'Ivoire	PMO - US	Direct Drop	FCA	Air	...	100	500	40000.0	80.00	0.80	ABB' V
3	15	108-VN-T01	Pre-PQ Process	SCMS-78	ASN-50	Vietnam	PMO - US	Direct Drop	EXW	Air	...	60	31920	127360.8	3.99	0.07	Paon
4	16	108-VN-T01	Pre-PQ Process	SCMS-81	ASN-55	Vietnam	PMO - US	Direct Drop	EXW	Air	...	60	38000	121600.0	3.20	0.05	Auro

5 rows × 33 columns

```
# Feature names
shipment_pricing.columns
```

```
Index(['id', 'project code', 'pq #', 'po / so #', 'asn/dn #', 'country',
       'managed by', 'fulfill via', 'vendor inco term', 'shipment mode',
       'pq first sent to client date', 'po sent to vendor date',
       'scheduled delivery date', 'delivered to client date',
       'delivery recorded date', 'product group', 'sub classification',
       'vendor', 'item description', 'molecule/test type', 'brand', 'dosage',
       'dosage form', 'unit of measure (per pack)', 'line item quantity',
       'line item value', 'pack price', 'unit price', 'manufacturing site',
       'first line designation', 'weight (kilograms)', 'freight cost (usd)',
       'line item insurance (usd)'],
      dtype='object')
```

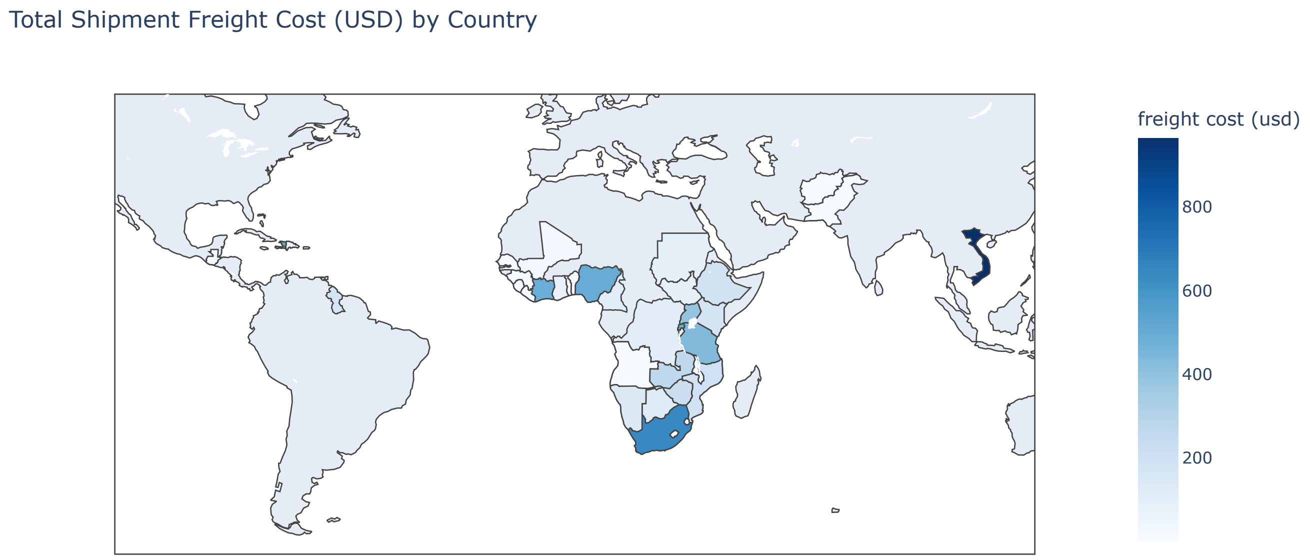
```
# Re-parse with the correct datetime format (m/d/y)
shipment_pricing['po sent to vendor date'] = pd.to_datetime(shipment_pricing['po sent to vendor date'], format='%m/%d/%Y')
shipment_pricing['delivered to client date'] = pd.to_datetime(shipment_pricing['delivered to client date'], format='mixed')
```

```
# Temporarily coerce to find bad rows
shipment_pricing['temp_sent'] = pd.to_datetime(shipment_pricing['po sent to vendor date'], errors='coerce')
shipment_pricing['temp_delivered'] = pd.to_datetime(shipment_pricing['delivered to client date'], errors='coerce')
# Remove invalid dates
shipment_pricing = shipment_pricing[shipment_pricing['po sent to vendor date'].notna() & shipment_pricing['delivered to client date'].notna(
```

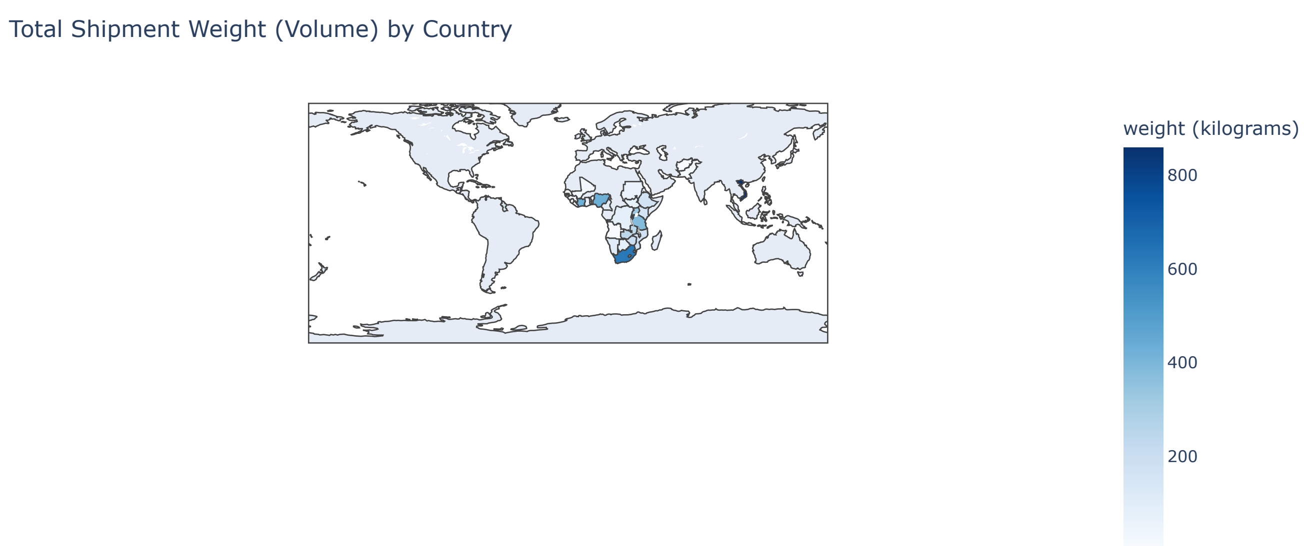
```
# Remove rows where parsing failed (bad strings)
shipment_pricing = shipment_pricing[shipment_pricing['temp_sent'].notna() & shipment_pricing['temp_delivered'].notna()]
# Drop temp columns
shipment_pricing.drop(columns=['temp_sent', 'temp_delivered'], inplace=True)
```

```
# Lead time column
df_filtered = shipment_pricing.copy()
df_filtered['lead_time'] = (df_filtered['delivered to client date'] - df_filtered['po sent to vendor date']).dt.days
```

```
# Remove rows with missing or negative lead times
df_filtered = df_filtered[df_filtered['lead_time'] > 0]


# Remove 'Truck' shipment mode (only one instance)
df_filtered = df_filtered[df_filtered['shipment mode'].str.lower() != 'truck']


# Remove specific unwanted text values from 'weight (kilograms)'
df_filtered = df_filtered[~df_filtered['weight (kilograms)'].astype(str).isin(['Freight Included in Commodity Cost'])]
df_filtered = df_filtered[~df_filtered['weight (kilograms)'].astype(str).str.startswith('See ASN')]


# Clean numeric columns
numeric_cols = ['freight cost (usd)', 'lead_time', 'weight (kilograms)']
# Make sure to work on a copy explicitly
df_filtered = df_filtered.copy()
for col in numeric_cols:
    # Keep only rows where the column is numeric (allowing decimals)
    df_filtered = df_filtered[df_filtered[col].apply(lambda x: str(x).replace('.', '', 1).isdigit())]
    # Safely convert to numeric
    df_filtered.loc[:, col] = pd.to_numeric(df_filtered[col], errors='coerce')
    # Log transform
    df_filtered.loc[:, col] = np.log1p(df_filtered[col])
# Drop missing values from important columns
df_filtered.dropna(subset=numeric_cols + ['shipment mode', 'vendor inco term', 'country', 'product group'], inplace=True)


# Add time-based features
df_filtered['month'] = df_filtered['po sent to vendor date'].dt.month
df_filtered['year'] = df_filtered['po sent to vendor date'].dt.year
df_filtered['quarter'] = df_filtered['po sent to vendor date'].dt.quarter


import plotly.express as px


# CHOROPLETHS


# Total Shipment Freight Cost (USD) by Country
# Group and sum by country
shipment_by_country = df_filtered.groupby('country')['freight cost (usd)'].sum().reset_index()
# Plot
fig_map = px.choropleth(
    shipment_by_country,
    locations='country',
    locationmode='country names',
    color='freight cost (usd)',
    color_continuous_scale='Blues',
    title='Total Shipment Freight Cost (USD) by Country')
fig_map.show()
```



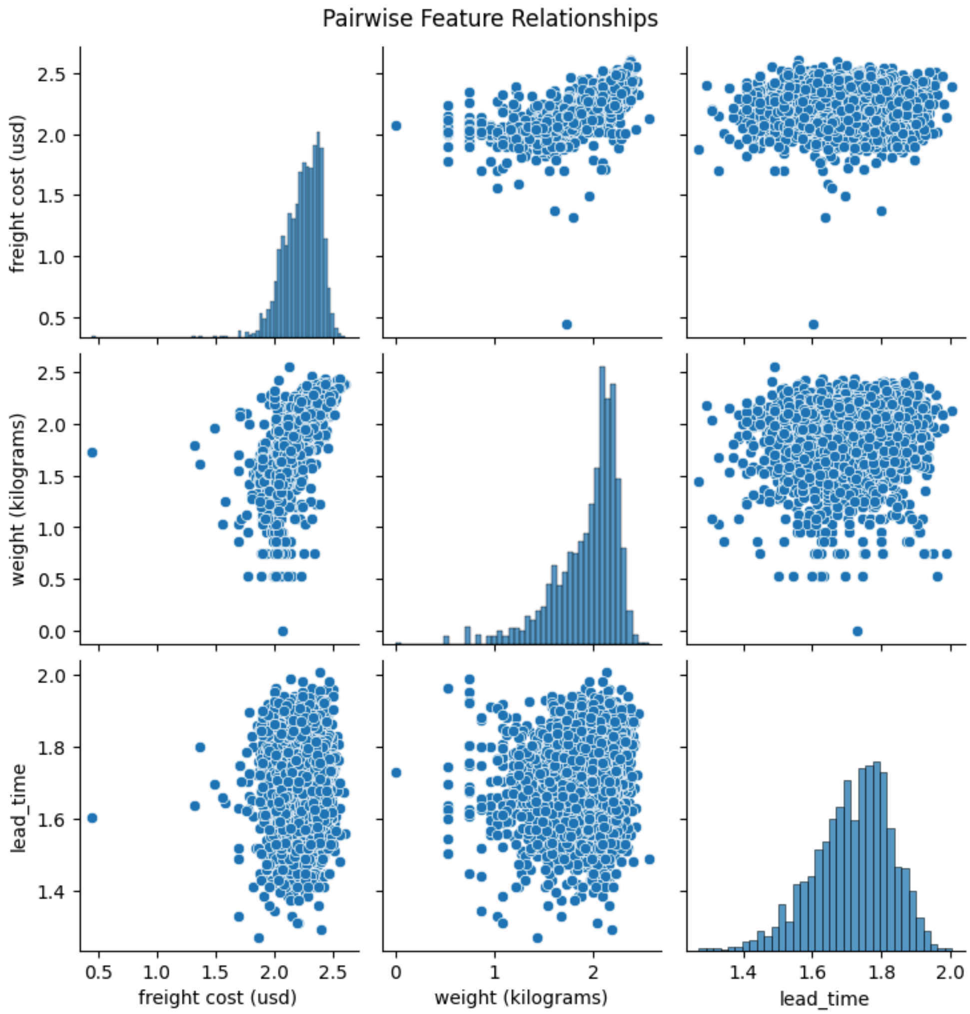

```
# Total Shipment Freight Weight (Kilograms) by Country
# Replace with the appropriate column if needed
df_filtered['weight (kilograms)'] = pd.to_numeric(df_filtered['weight (kilograms)'], errors='coerce')
# Group and sum by country
shipment_by_country = df_filtered.groupby('country')['weight (kilograms)'].sum().reset_index()
# Plot
fig_map = px.choropleth(
    shipment_by_country,
    locations='country',
    locationmode='country names',
    color='weight (kilograms)',
    color_continuous_scale='Blues',
    title='Total Shipment Weight (Volume) by Country')
fig_map.show()
```



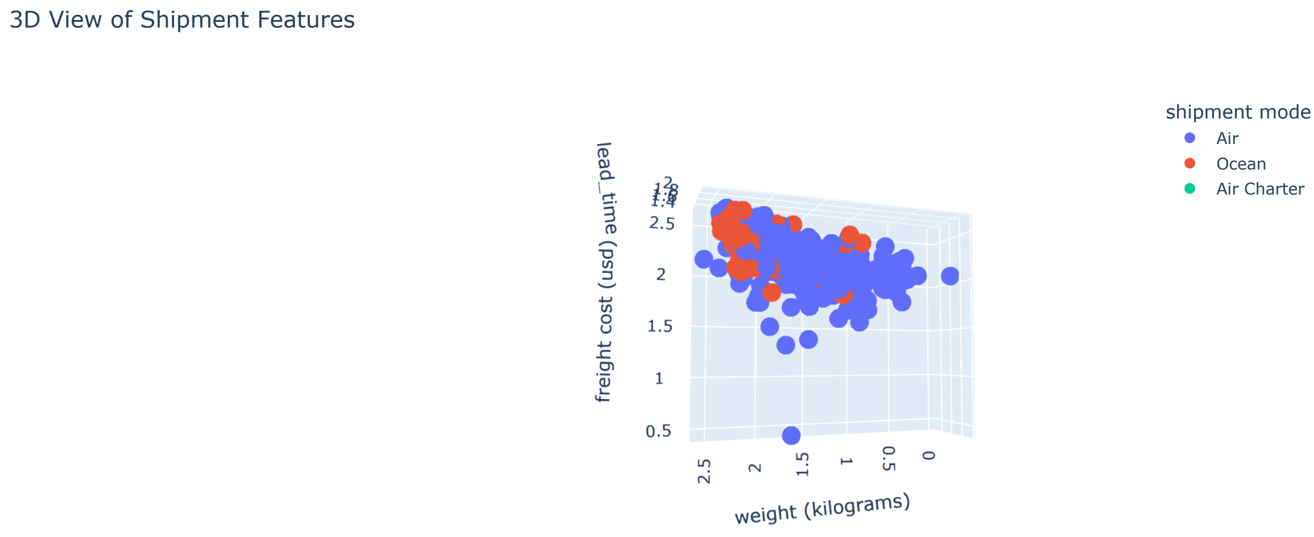

```
# PAIRWISE
```

```
# Pairwise Feature Relationships
```

```
sns.pairplot(df_filtered[numeric_cols])  # Choose top features
plt.suptitle("Pairwise Feature Relationships", y=1.02)
plt.show()
```

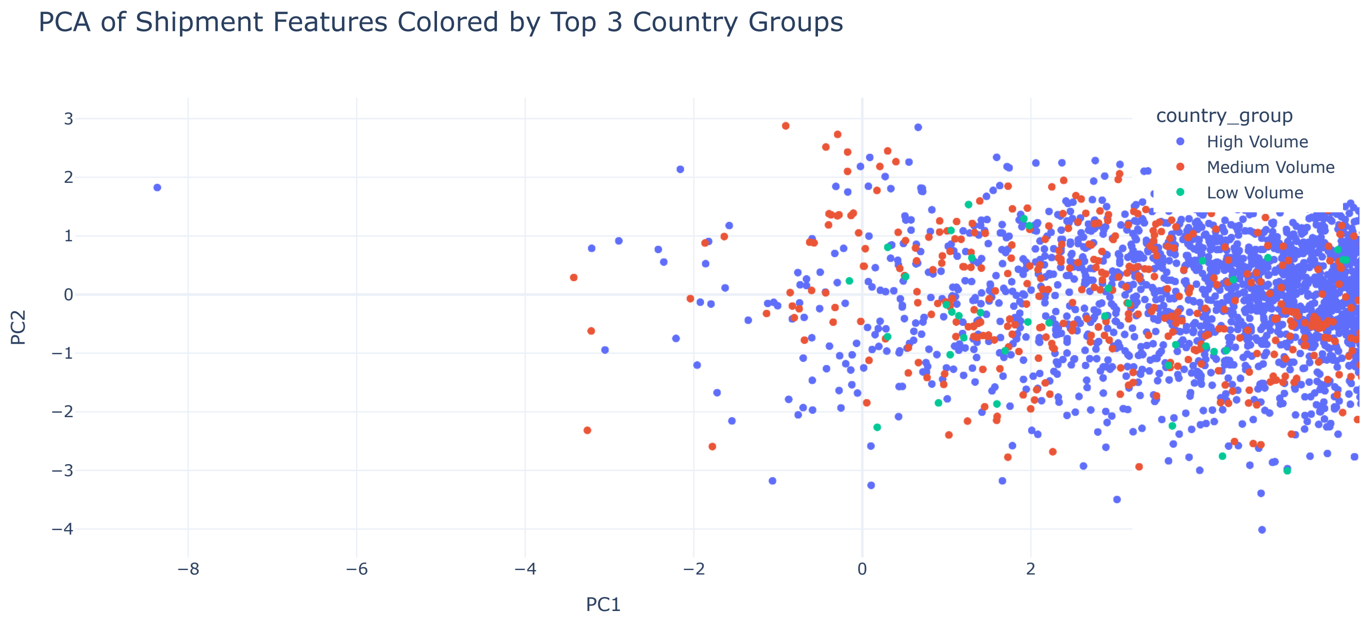



```
#cost, weight, and lead time:
fig = px.scatter_3d(df_filtered, x='weight (kilograms)', y='lead_time', z='freight cost (usd)',
                    color='shipment mode', title='3D View of Shipment Features')
fig.show()
```




```
import pandas as pd
import plotly.express as px
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


# PCA FOR FREQUENCY OF SHIPMENT TO A COUNTRY


# Count how many times each country appears
country_counts = df_filtered['country'].value_counts()
# Get thresholds
high_threshold = np.percentile(country_counts.values, 66)
low_threshold = np.percentile(country_counts.values, 33)
# Define a function to group countries
def group_country(country):
    count = country_counts.get(country, 0)
    if count >= high_threshold:
        return 'High Volume'
    elif count >= low_threshold:
        return 'Medium Volume'
    else:
        return 'Low Volume'
# Apply the group to the dataframe
df_filtered['country_group'] = df_filtered['country'].apply(group_country)


# Prepare features
numeric_features = ['freight cost (usd)', 'weight (kilograms)', 'lead_time']
df_pca = df_filtered.dropna(subset=numeric_features + ['country_group'])


# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_pca[numeric_features])


# Perform PCA
pca = PCA(n_components=2)
components = pca.fit_transform(X_scaled)
df_pca['PC1'] = components[:, 0]
df_pca['PC2'] = components[:, 1]


# Visualize, color by grouped country
fig = px.scatter(
    df_pca,
    x = 'PC1',
    y = 'PC2',
```

```
    color = 'country_group',
    title = 'PCA of Shipment Features Colored by Top 3 Country Groups',
    labels = {'country_grouped': 'Country Group'},
    template = 'plotly_white')
fig.update_layout(title_font_size=20)
fig.show()
```



What are the most important features that influence lead time?

> Gradient Boosting Regressor

```
# GRADIENT BOOSTING REGRESSOR


# ATTEMPT 1


import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PowerTransformer
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.svm import SVR
import seaborn as sns


# Feature Engineering
features = ['country', 'shipment mode', 'vendor inco term', 'product group', 'freight cost (usd)', 'weight (kilograms)', 'month', 'year', 'c
X = df_filtered[features]
y = df_filtered['lead_time']
# Categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()
# Preprocessing
preprocessor = ColumnTransformer(transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols),
        ('num', Pipeline(steps=[
            ('skew', PowerTransformer(method='yeo-johnson', standardize=False)),
            ('scale', StandardScaler())
```

```
        ]), numeric_cols)])
#preprocessor = ColumnTransformer(transformers=[('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)], remainder='passthrough')


# Regressors: Gradient Boosting, Random Forest, SVR
models = {
    'GradientBoosting': GradientBoostingRegressor(random_state=42),
    'RandomForest': RandomForestRegressor(random_state=42),
    'SVR': SVR()}
#model = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', GradientBoostingRegressor(random_state=42))])
results = {}
for name, regressor in models.items():
    pipe = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', regressor)])


# Cross-validation
scores = cross_val_score(pipe, X, y, cv=5, scoring='r2')
results[name] = {
    'mean_r2': np.mean(scores),
    'std_r2': np.std(scores)}
print(f"{name} R²: {scores.mean():.3f} ± {scores.std():.3f}")
```

```
SVR R²: -0.046 ± 0.058
```

```
# Hyperparameter tuning
param_grid = {
    'regressor__n_estimators': [100, 200],
    'regressor__max_depth': [3, 5, 7],
    'regressor__learning_rate': [0.01, 0.1]}
best_pipe = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', GradientBoostingRegressor(random_state=42))])
grid = GridSearchCV(best_pipe, param_grid, cv=5, scoring='r2', n_jobs=-1)
grid.fit(X, y)
print("Best Parameterss:", grid.best_params_)
print("Best CV R² Score:", grid.best_score_)
```

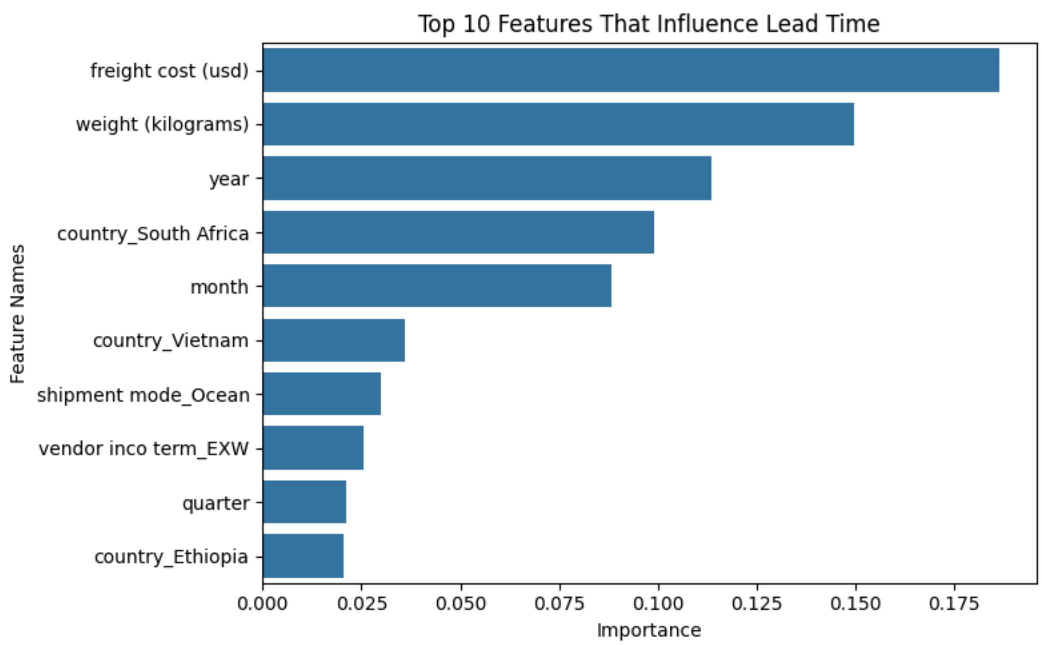
```
Best Parameterss: {'regressor__learning_rate': 0.1, 'regressor__max_depth': 7, 'regressor__n_estimators': 100}
Best CV R² Score: 0.23791485882337776
```

```
#Best Parameters: {'regressor__learning_rate': 0.1, 'regressor__max_depth': 7, 'regressor__n_estimators': 100}
#Best CV R² Score: 0.23791485882337776


# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
final_model = grid.best_estimator_
final_model.fit(X_train, y_train)
# Evals
y_pred = final_model.predict(X_test)
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R²:", r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.06877065144114564
Root Mean Squared Error: 0.09269457933708254
R²: 0.29022510588808836
```

```
# Mean Absolute Error: 0.06877065144114564
# Root Mean Squared Error: 0.09269457933708254
# R²: 0.29022510588808836


from sklearn.compose import make_column_selector as selector


# After model fitting:
preprocessor = final_model.named_steps['preprocessor']
reg = final_model.named_steps['regressor']


# Get feature names from OneHotEncoder (categorical)
ohe = preprocessor.named_transformers_['cat']
ohe_feature_names = ohe.get_feature_names_out(categorical_cols)


# Get passthrough (numerical and time) features based on remainder='passthrough'
# These are the columns that were not one-hot encoded
```

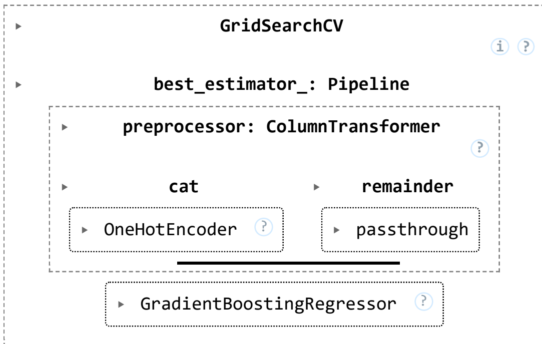
```
# Assume they are in the same order as the ones passed to the pipeline
passthrough_features = [col for col in X.columns if col not in categorical_cols]
# Combine
all_features = list(ohe_feature_names) + passthrough_features
# Check for alignment before creating Series
assert len(all_features) == len(reg.feature_importances_), f"Feature name count ({len(all_features)}) does not match importances ({len(reg.f

# Visualize feature importances
importances = pd.Series(reg.feature_importances_, index=all_features)
top = importances.sort_values(ascending=False).head(10)

plt.figure(figsize=(8, 5))
sns.barplot(x=top.values, y=top.index)
plt.title("Top 10 Features That Influence Lead Time")
plt.ylabel("Feature Names")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```




```
# ATTEMPT 2

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define features and target
categorical_cols = ['shipment mode', 'vendor inco term', 'country']
X = df_filtered[categorical_cols + ['weight (kilograms)', 'lead_time']]
y = df_filtered['freight cost (usd)']

# Preprocessing
encoder = ColumnTransformer(transformers=[('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)], remainder='passthrough')

# Gradient Boosting with Pipeline
pipeline = Pipeline([('preprocessor', encoder),('model', GradientBoostingRegressor(random_state=42))])

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

```
# Grid Search (optional for tuning)
param_grid = {
    'model__n_estimators': [100, 200],
    'model__learning_rate': [0.05, 0.1],
    'model__max_depth': [3, 5]}
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:


The format of the columns of the 'remainder' transformer in ColumnTransformer.transformers_ will change in version 1.7 to match the form
At the moment the remainder columns are stored as indices (of type int). With the same ColumnTransformer configuration, in the future th
To use the new behavior now and suppress this warning, use ColumnTransformer(force_int_remainder_cols=False).
```




```
# Evaluation
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Best Parameters:", grid_search.best_params_)
print(f"Mean Absolute Error: {mae:.2f}") # Ideal 0
print(f"Mean Squared Error: {mse:.2f}") #Ideal 0
print(f"Root Mean Squared Error: {rmse:.2f}") # Ideal 0
print(f"R² Score: {r2:.4f}") #Ideal 1
```

```
Best Parameters: {'model__learning_rate': 0.1, 'model__max_depth': 3, 'model__n_estimators': 100}
Mean Absolute Error: 0.05
Mean Squared Error: 0.00
Root Mean Squared Error: 0.07
R² Score: 0.7809
```

```
#Best Parameters: {'model__learning_rate': 0.1, 'model__max_depth': 3, 'model__n_estimators': 100}
#Mean Absolute Error: 0.05
#Mean Squared Error: 0.00
#Root Mean Squared Error: 0.07
#R² Score: 0.7809
```

```
import plotly.express as px
```

```
# Extract feature names
onehot = grid_search.best_estimator_.named_steps['preprocessor'].named_transformers_['cat']
onehot_features = onehot.get_feature_names_out()
```

```
# Feature names
numeric_features = ['weight (kilograms)', 'lead_time']
```

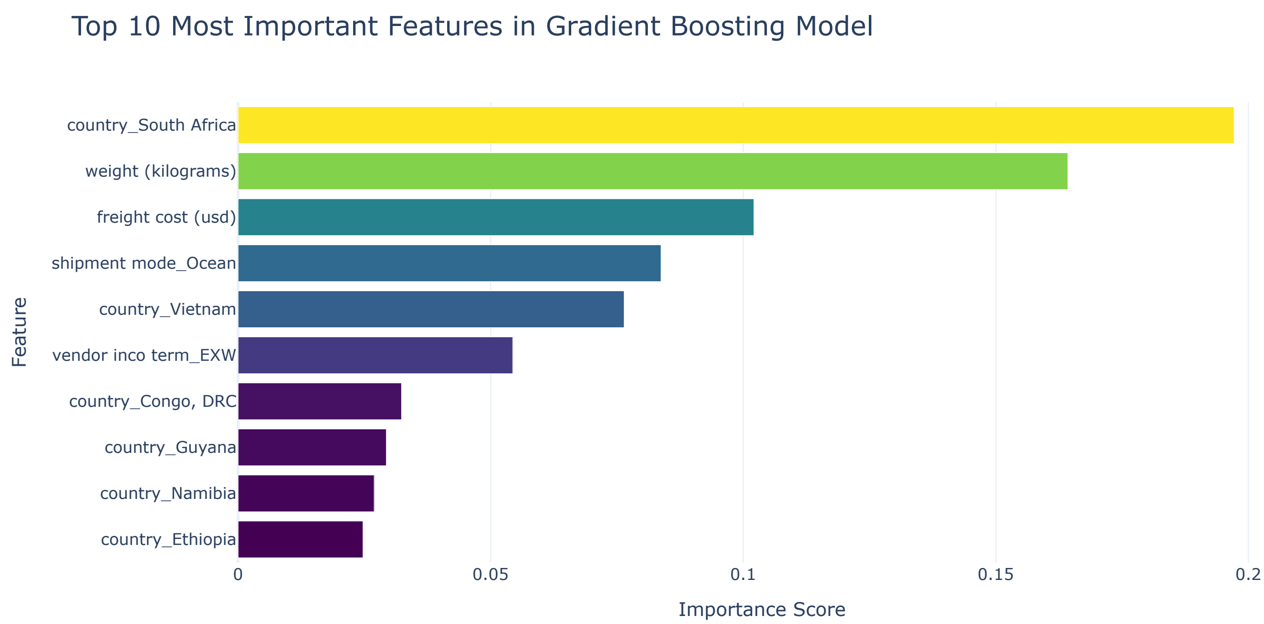
```
# Importances from Gradient Boosting
importances = grid_search.best_estimator_.named_steps['model'].feature_importances_
feat_importance_series = pd.Series(importances, index=all_features)
# Top 10
top_features = feat_importance_series.nlargest(10).sort_values()
```

```
print(feat_importance_series)
```

```
shipment mode_Air            0.000766
shipment mode_Air Charter    0.000000
shipment mode_Ocean          0.001866
vendor inco term_CIF         0.000375
vendor inco term_CIP         0.045856
vendor inco term_DAP         0.000000
vendor inco term_DDP         0.013322
vendor inco term_DDU         0.001255
vendor inco term_EXW         0.034343
vendor inco term_FCA         0.000339
country_Afghanistan          0.000230
country_Angola               0.000000
country_Benin                0.000000
country_Botswana             0.001505
country_Burundi              0.000000
country_Cameroon             0.000620
country_Congo, DRC           0.000239
country_Côte d'Ivoire        0.001518
country_Dominican Republic   0.000099
country_Ethiopia             0.004479
country_Ghana                0.000000
country_Guatemala            0.000752
country_Guinea               0.000000
country_Guyana               0.000463
country_Haiti                0.004388
country_Kenya                0.004495
country_Lesotho              0.000000
country_Liberia              0.000000
country_Malawi               0.000000
country_Mali                 0.000618
country_Mozambique           0.000269
country_Namibia              0.000617
country_Nigeria              0.002689
country_Pakistan             0.000000
country_Rwanda               0.006390
country_Senegal              0.000000
country_Sierra Leone         0.000000
country_South Africa         0.007191
country_South Sudan          0.000319
country_Sudan                0.000000
country_Swaziland            0.000000
country_Tanzania             0.000021
country_Togo                 0.000000
country_Uganda               0.007358
country_Vietnam              0.027062
country_Zambia               0.000684
country_Zimbabwe             0.002010
weight (kilograms)           0.816930
lead_time                    0.010932
dtype: float64
```

```
# Most important features to this model:
# weight (kilograms) – Over 81% of the model’s predictive power comes from shipment weight.
  # This makes sense since heavier shipments generally cost more to transport.
# vendor inco term_CIP – Incoterm (Carriage and Insurance Paid) strongly influences cost.
  # Suggests that terms where the vendor pays for more services (like insurance and transport) raise the freight cost.
# vendor inco term_EXW – EXW (Ex Works) means buyer bears almost all shipping costs — variation in this term significantly affects freight c
# country_Vietnam – Freight costs from Vietnam seem more variable or higher on average — strong enough to influence predictions.

# Least important features to this model:
# Many countries and shipment modes have near-zero importance (eg; shipment mode_Air Charter, vendor inco term_DAP, country_Benin, country_F
  # These values either occur infrequently or do not meaningfully change the cost.


# Visualize
fig = px.bar(
    top_features,
    orientation='h',
    labels={'value': 'Importance Score', 'index': 'Feature'},
    title='Top 10 Most Important Features in Gradient Boosting Model',
    color=top_features,
    color_continuous_scale='Viridis')
fig.update_layout(
    title_font_size=20,
    xaxis_title='Importance Score',
    yaxis_title='Feature',
    template='plotly_white',
    coloraxis_showscale=False)
```

```
fig.show()
all_features = np.concatenate([onehot_features, numeric_features])
```



How do ARV and HIV lab commodity prices vary across countries and over time?

> Time Series Analysis

```
# Time-Series
# Continuous time index, single target variable (unit price), time structure (years)
# Aggregate average unit price by year and country
df_time_series = df_filtered.groupby(['year', 'country'])['unit price'].mean().reset_index()
# Pivot table to make each country a column
df_time_series = df_time_series.pivot(index='year', columns='country', values='unit price')
# Fill missing values (optional: forward-fill)
df_time_series = df_time_series.fillna(method='ffill')
# Reset index for time series modeling
df_time_series.index = pd.to_datetime(df_time_series.index, format='%Y')
```

```
<ipython-input-110-e4383a62a50b>:8: FutureWarning:

DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
```

```
from statsmodels.tsa.arima.model import ARIMA
from prophet import Prophet
```

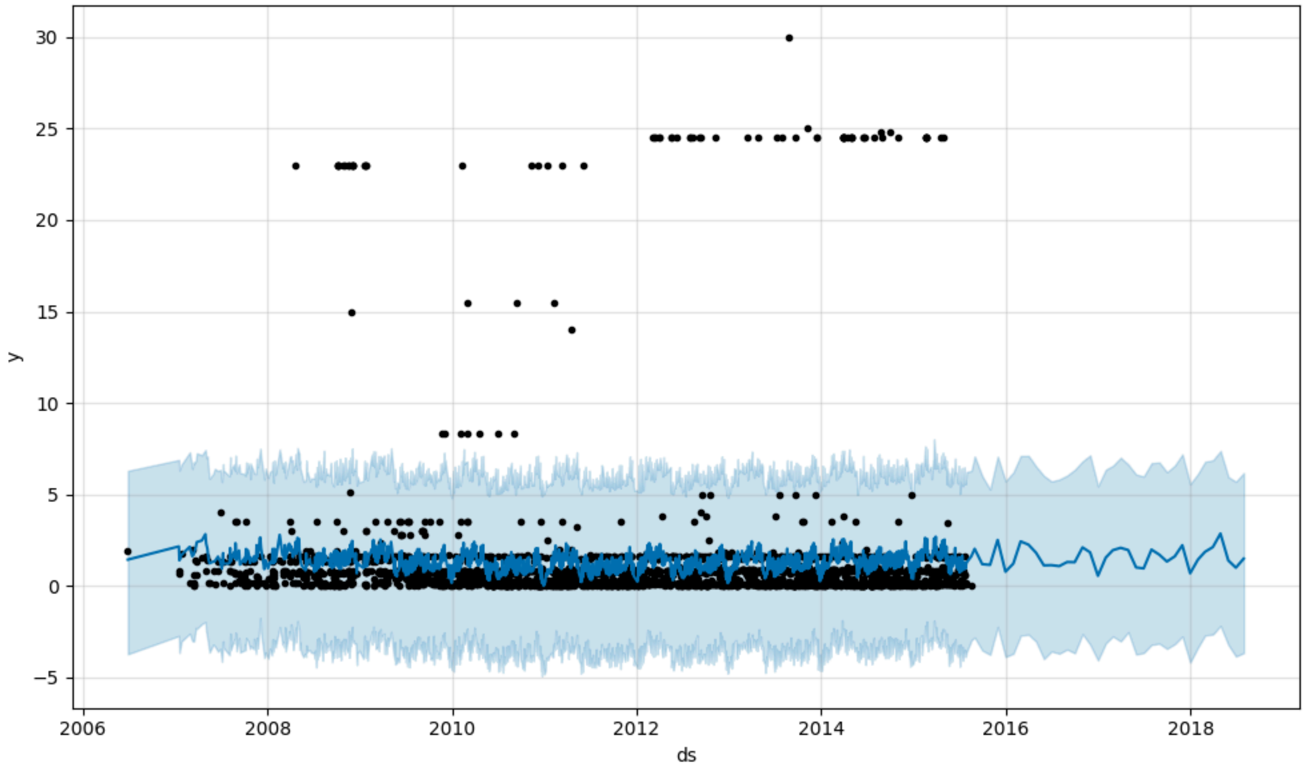
```
# Example: Forecast for a specific country )
country = 'South Africa'
series = df_time_series[country].dropna()
# Fit ARIMA model
model = ARIMA(series, order=(1,1,1))  # (p,d,q) order needs tuning
model_fit = model.fit()
# Forecast next 3 years
forecast = model_fit.forecast(steps=3)
print(forecast)
```

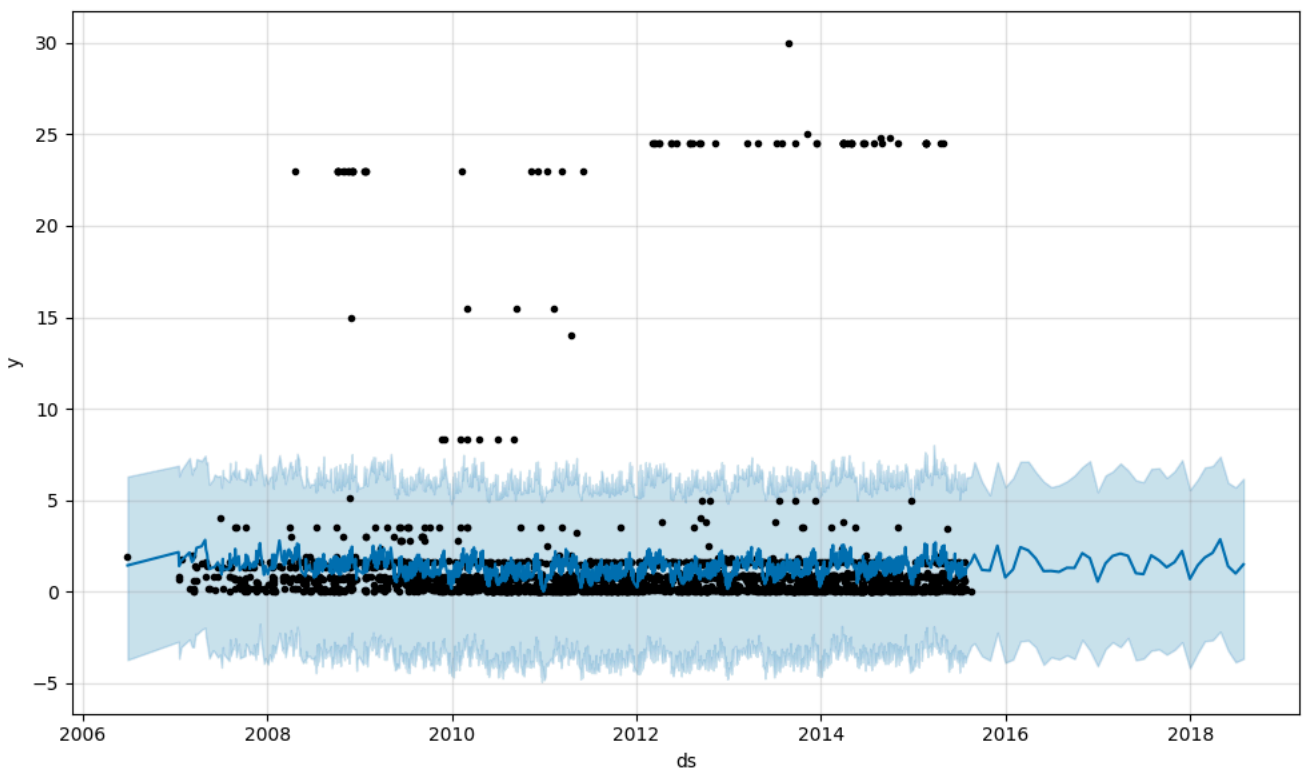
```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency YS-JAN will be used.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency YS-JAN will be used.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:
```

```
No frequency information was provided, so inferred frequency YS-JAN will be used.

2016-01-01    0.106364
2017-01-01    0.106364
2018-01-01    0.106364
Freq: YS-JAN, Name: predicted_mean, dtype: float64
```

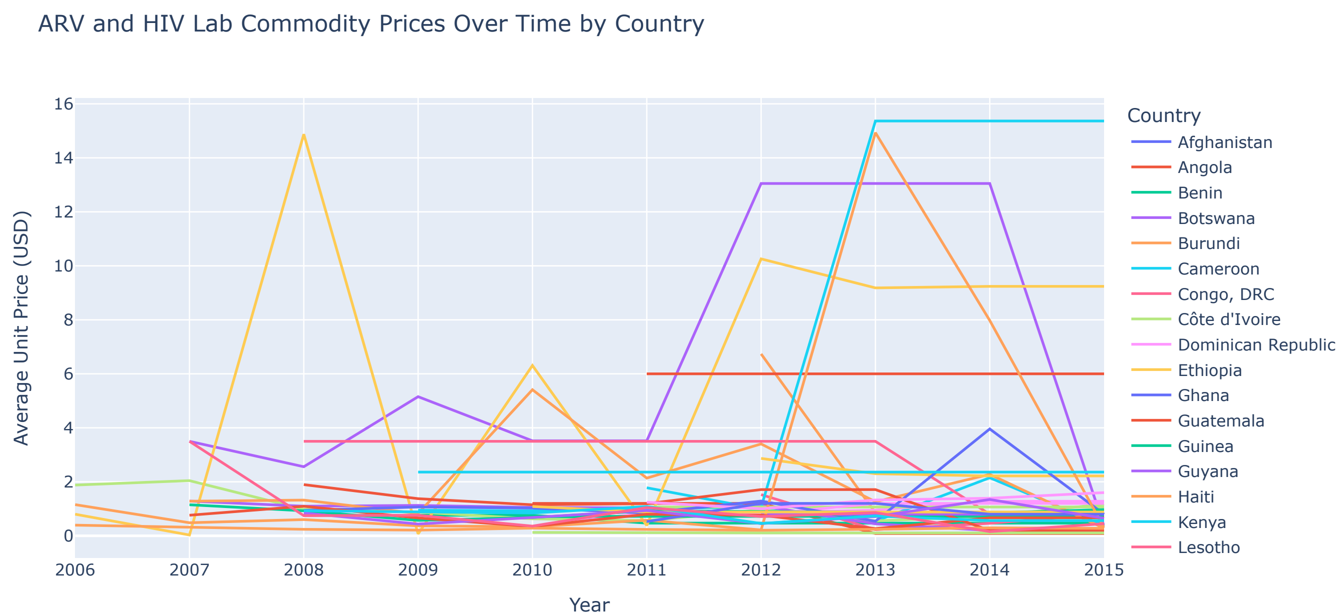
```
# For seasonality and external regressors
# Prepare data for Prophet
df_prophet = df_filtered[['delivered to client date', 'unit price']].dropna()
df_prophet.rename(columns={'delivered to client date': 'ds', 'unit price': 'y'}, inplace=True)
# Fit model
model = Prophet()
model.fit(df_prophet)
# Make future predictions
future = model.make_future_dataframe(periods=36, freq='M')  # Forecast next 3 years
forecast = model.predict(future)
# Visualize
model.plot(forecast)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp78u9cgf_/867gwrrs.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp78u9cgf_/b372btex.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=81768', '
08:34:17 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
08:34:17 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarning:

'M' is deprecated and will be removed in a future version, please use 'ME' instead.
```





```
# Forecasting price/year (rising/lowering)
# ARIMA accounts for past trends, so if prices were increasing before, it projects a continued increase
# Trend forecasting based on historical data
# Visualization shows (rising/lowering) trend over time
# Shows seasonal price fluctuations (e.g., prices drop in Q1 every year)
# Confidence interval (range of possible future values)
# If the model detects a periodic dip (e.g., price drops every 5 years), it adjusts accordingly???
# Doesn't handle external factors like inflation or policy changes


# Melt the pivoted DataFrame for long-format plotly compatibility
df_long = df_time_series.reset_index().melt(id_vars='year', var_name='country', value_name='unit_price')
df_long['year'] = df_long['year'].dt.year
fig = px.line(
    df_long,
    x='year',
    y='unit_price',
    color='country',
    title="ARV and HIV Lab Commodity Prices Over Time by Country",
    labels={'unit_price': 'Average Unit Price (USD)', 'year': 'Year'})
fig.update_layout(legend_title_text='Country', hovermode='x unified')
fig.show()
```



How can we optimize the supply chain expenses to reduce overall costs while maintaining efficient delivery?

> Optimization Algorithms

```
import pandas as pd
import numpy as np
from scipy.optimize import linprog


# Filter and clean dataset
df = shipment_pricing.copy()
# Convert dates
df['po sent to vendor date'] = pd.to_datetime(df['po sent to vendor date'], errors='coerce')
df['delivered to client date'] = pd.to_datetime(df['delivered to client date'], errors='coerce')
# Calculate lead time
df['lead_time'] = (df['delivered to client date'] - df['po sent to vendor date']).dt.days
# Clean numerical values
numeric_cols = ['freight cost (usd)', 'weight (kilograms)', 'lead_time']
df[numeric_cols] = df[numeric_cols].apply(pd.to_numeric, errors='coerce')
# Drop NaNs and only keep air and sea (if truck is rare)
df.dropna(subset=numeric_cols, inplace=True)
df = df[df['shipment mode'].isin(['Air', 'Sea'])]
```

```
# Cost vector: freight cost per shipment
c = df['freight cost (usd)'].values

# Maximum Constraints
A = [
    # Total weight <= max
    df['weight (kilograms)'].values,
    # Total lead time <= max avg * n
    df['lead_time'].values]
b = [
    # max weight
    100000,
    # max total lead time
    25 * len(df)]

# Minimum constraints
min_weight = 50000
# min weight ≥
A.append([-w for w in df['weight (kilograms)'].values])
b.append(-min_weight)

# Bounds: [0, 1] per shipment (fractional selection)
x_bounds = [(0, 1) for _ in range(len(df))]

# Solve
result = linprog(c=c, A_ub=A, b_ub=b, bounds=x_bounds, method='highs')

if result.success:
    df['selected'] = result.x
    total_cost = result.fun
    print(f"Optimization successful. Total optimized cost: ${total_cost:,.2f}")
    print(df[df['selected'] > 0.01][['country', 'freight cost (usd)', 'lead_time', 'weight (kilograms)', 'selected']].head())
else:
    print("Optimization failed:", result.message)
```

```
Optimization successful. Total optimized cost: $537.26
             country  freight cost (usd)  lead_time  weight (kilograms)  \
35            Rwanda                0.75         52                99.0
4923   Côte d'Ivoire            1664.12         30            154780.0

      selected
35      1.0000
4923    0.3224
```

```
# Cost vector: freight cost per shipment
c = df['freight cost (usd)'].values

# Weight constraint: total weight <= max_weight
max_weight = 100000
A_weight = [df['weight (kilograms)'].values]
b_weight = [max_weight]

# Lead time constraint: average lead time <= max_lead
max_avg_lead_time = 25
A_lead = [df['lead_time'].values]
b_lead = [max_avg_lead_time * len(df)]  # Total lead time for all shipments

# Minimum number of shipments constraint
A_min_shipments = [-np.ones(len(df))]  # Sum(x) ≥ 10 → -sum(x) ≤ -10
b_min_shipments = [-10]

# Combine constraints
A = A_weight + A_lead + A_min_shipments
b = b_weight + b_lead + b_min_shipments

# Set bounds for decision variables: each shipment is 0 (not selected) to 1 (selected fractionally)
x_bounds = [(0, 1) for _ in range(len(df))]

# Solve linear program
result = linprog(c=c, A_ub=A, b_ub=b, bounds=x_bounds, method='highs')

if result.success:
    df['selected'] = result.x
    total_cost = result.fun
    print(f"Optimization successful. Total optimized cost: ${total_cost:,.2f}")
    print(df[['country', 'shipment mode', 'freight cost (usd)', 'weight (kilograms)', 'lead_time', 'selected']].head(10))
```

```
else:
    print("Optimization failed:", result.message)
```

```
Optimization successful. Total optimized cost: $497.23
     country shipment mode  freight cost (usd)  weight (kilograms)  lead_time  \
13    Rwanda           Air            64179.42              7416.0         67
18   Vietnam           Air              807.47                34.0         89
19  Tanzania           Air              912.96               162.0         55
20   Nigeria           Air             2682.47               341.0         37
21   Nigeria           Air            15893.71              2278.0         33
23   Vietnam           Air             4193.49               941.0        103
24   Vietnam           Air             1767.38               117.0         54
25     Haiti           Air             3518.38               171.0         26
26     Haiti           Air             3097.85                60.0         30
30  Ethiopia           Air            12237.61              4228.0         48

    selected
13       0.0
18       0.0
19       0.0
20       0.0
21       0.0
23       0.0
24       0.0
25       0.0
26       0.0
30       0.0
```

Can we identify unusual patterns or outliers in pricing or shipment data that may indicate issues in the supply chain?

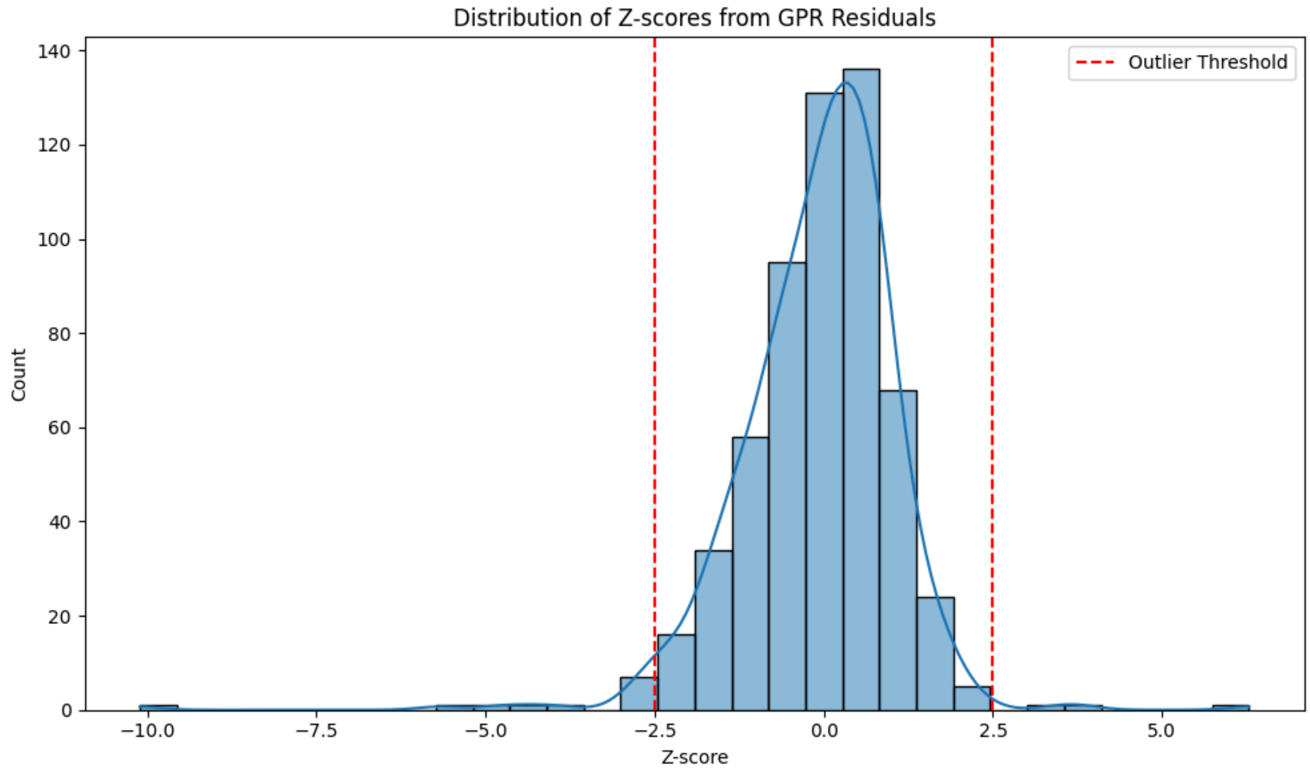
> Gaussian Processes, K-Nearest Neighbors (KNN)

```
# GAUSSIAN PROCESS


import pandas as pd
import numpy as np
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns


# ATTEMPT 1


# Define input features and target
features = ['country', 'shipment mode', 'vendor inco term', 'weight (kilograms)', 'freight cost (usd)', 'product group']
target = 'lead_time'
X = df_filtered[features]
y = df_filtered[target]


# Categorical and numerical separation
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numerical_cols = [col for col in X.columns if col not in categorical_cols]


# Preprocessing pipeline
preprocessor = ColumnTransformer(transformers=[('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_cols),('num'


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


# Define Gaussian Process with RBF kernel
kernel = C(1.0, (1e-3, 1e3)) * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
gpr = GaussianProcessRegressor(kernel=kernel, alpha=1e-2, normalize_y=True)
# Pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', gpr)])


# Fit model
pipeline.fit(X_train, y_train)
# Predict on test data with standard deviation
```

```
y_pred, y_std = pipeline.named_steps['regressor'].predict(pipeline.named_steps['preprocessor'].transform(X_test), return_std=True)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/gaussian_process/kernels.py:442: ConvergenceWarning:

The optimal value found for dimension 0 of parameter k2__length_scale is close to the specified lower bound 0.01. Decreasing the bound a
```

```
# Compute residuals
residuals = y_test - y_pred
z_scores = residuals / y_std


# Identify outliers: where |z| > threshold
threshold = 2.5
outliers = np.abs(z_scores) > threshold


# Add output for analysis
results = X_test.copy()
results['actual'] = y_test
results['predicted'] = y_pred
results['std'] = y_std
results['z_score'] = z_scores
results['outlier'] = outliers


# Visualize residuals
plt.figure(figsize=(10, 6))
sns.histplot(z_scores, bins=30, kde=True)
plt.axvline(threshold, color='red', linestyle='--', label='Outlier Threshold')
plt.axvline(-threshold, color='red', linestyle='--')
plt.title('Distribution of Z-scores from GPR Residuals')
plt.xlabel('Z-score')
plt.legend()
plt.tight_layout()
plt.show()
# Show top outliers
print("Top potential outliers based on GPR:")
print(results[results['outlier']].sort_values(by='z_score', key=np.abs, ascending=False).head(10))
```




```
Top potential outliers based on GPR:
          country shipment mode vendor inco term  weight (kilograms)  \
4539       Rwanda           Air              EXW            2.085795
4099  South Africa          Air              DDP            2.242650
4331      Vietnam           Air              EXW            2.123395
219      Ethiopia           Air              EXW            2.254311
5509        Haiti           Air              CIP            1.124748
547         Haiti           Air              EXW            1.439569
4573      Vietnam           Air              EXW            2.050961
2810      Nigeria           Air              EXW            2.182262
700       Namibia           Air              EXW            1.881679
417        Rwanda           Air              EXW            2.001290

      freight cost (usd) product group    actual  predicted       std  \
4539            2.463870          HRDT  1.660640   1.824710  0.016236
4099            2.278239           ARV  1.742137   1.690882  0.008155
4331            2.185798           ARV  1.770740   1.905309  0.023689
219             2.459575          HRDT  1.579009   1.603158  0.005021
5509            1.898118           ARV  1.831260   1.879558  0.011070
547             1.869461          HRDT  1.271150   1.717702  0.114142
4573            2.161990           ARV  1.870734   1.815727  0.014422
2810            2.421349          HRDT  1.780778   1.607889  0.049713
700             2.117667          HRDT  1.385227   1.717702  0.114142
417             2.267052          HRDT  1.397363   1.717516  0.114142

        z_score  outlier
4539 -10.105080     True
4099   6.284939     True
4331  -5.680725     True
219   -4.809850     True
5509  -4.362895     True
547   -3.912234     True
4573   3.814113     True
2810   3.477769     True
700   -2.912807     True
417   -2.804853     True
```

```
# Extremely low standard deviations = overconfidence
  # Some predictions have repeated uncertainly values/standard deviations
# Unrealistically large z-scores
```

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel, ConstantKernel as C
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
```

```
import numpy as np
import pandas as pd

# --- Preprocessing ---

categorical_cols = ['country', 'shipment mode', 'vendor inco term', 'product group']
numeric_cols = ['weight (kilograms)', 'freight cost (usd)']

X = df_filtered[categorical_cols + numeric_cols]
y = df_filtered['lead_time']

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

# Define preprocessing
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse=False), categorical_cols),
    ('num', StandardScaler(), numeric_cols)
])

# Define kernel with sensible bounds
kernel = C(1.0, (1e-2, 1e2)) * RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0)) + WhiteKernel(noise_level=1.0, noise_level_bounds=(1e

# Define GPR model
gpr = GaussianProcessRegressor(kernel=kernel, alpha=1e-4, normalize_y=True, random_state=42)

# Create pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', gpr)
])

# --- Fit model ---
pipeline.fit(X_train, y_train)

# --- Predict with std dev for uncertainty ---
X_test_transformed = pipeline.named_steps['preprocessor'].transform(X_test)
y_pred, y_std = pipeline.named_steps['regressor'].predict(X_test_transformed, return_std=True)

# --- Flag outliers based on z-score ---
z_scores = (y_test.values - y_pred) / y_std
outliers = np.abs(z_scores) > 2  # You can also try 2.5 or 3

# --- Create result DataFrame ---
results = X_test.copy()
results['actual'] = y_test.values
results['predicted'] = y_pred
results['std'] = y_std
results['z_score'] = z_scores
results['outlier'] = outliers

# Sort by absolute z-score (strongest outliers first)
top_outliers = results.loc[results['outlier']].sort_values(by='z_score', key=np.abs, ascending=False)

print("Top potential outliers based on improved GPR:")
print(top_outliers.head(10))



# ATTEMPT 2


# Define input features and target
features = ['country', 'shipment mode', 'vendor inco term', 'weight (kilograms)', 'freight cost (usd)', 'product group']
target = 'lead_time'
X = df_filtered[features]
y = df_filtered[target]


# Categorical and numerical separation
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numerical_cols = [col for col in X.columns if col not in categorical_cols]


preprocessor = ColumnTransformer(transformers=[('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_cols), ('num
```

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


# Define Gaussian Process with RBF kernel
# Smaller lower bound
kernel = C(1.0, (1e-3, 1e3)) * RBF(length_scale=1.0, length_scale_bounds=(1e-5, 1e5))
regressor = GaussianProcessRegressor(kernel=kernel, normalize_y=True, random_state=42)
gpr = GaussianProcessRegressor(kernel=kernel, alpha=1e-2, normalize_y=True)
# Pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', gpr)])


# Fit model
pipeline.fit(X_train, y_train)
# Predict on test data with standard deviation
y_pred, y_std = pipeline.named_steps['regressor'].predict(
    pipeline.named_steps['preprocessor'].transform(X_test), return_std=True)


# Compute residuals
residuals = y_test - y_pred
z_scores = residuals / y_std


# Identify outliers: where |z| > threshold
threshold = 2.5
outliers = np.abs(z_scores) > threshold


# Add output for analysis
results = X_test.copy()
results['actual'] = y_test
results['predicted'] = y_pred
results['std'] = y_std
results['z_score'] = z_scores
results['outlier'] = outliers


# Visualize residuals
plt.figure(figsize=(10, 6))
sns.histplot(z_scores, bins=30, kde=True)
plt.axvline(threshold, color='red', linestyle='--', label='Outlier Threshold')
plt.axvline(-threshold, color='red', linestyle='--')
plt.title('Distribution of Z-scores from GPR Residuals')
plt.xlabel('Z-score')
plt.legend()
plt.tight_layout()
plt.show()
# Show top outliers
print("Top potential outliers based on GPR:")
print(results[results['outlier']].sort_values(by='z_score', key=np.abs, ascending=False).head(10))
```