Step 1: Backend Design

The backend consists of:

1. **AWS Lambda Function**: Processes requests and interacts with SageMaker for AI predictions.
2. **Amazon API Gateway**: Provides a RESTful endpoint for the frontend to communicate with the Lambda function.
3. **Amazon DynamoDB**: Stores input data, cost estimations, and logs for tracking and analytics.

Step 1.1: DynamoDB Table

What is DynamoDB?

Amazon DynamoDB is a fully managed NoSQL database designed for fast and predictable performance. It will store the following:

- **Input Data**: Origin, destination, package details, etc.
- **Cost Breakdown**: Base cost, weight charge, dimensional weight charge, total cost.
- **Timestamp**: Date and time of the estimation for logging.

Steps to Create the Table:

1. **Log in to AWS Console**:
    - Open AWS Management Console and navigate to **DynamoDB**.
2. **Create a New Table**:
    - Click **Create Table**.
    - Enter these configurations:
        - **Table Name**: CostEstimates
        - **Partition Key**: estimate_id (data type: String).
    - Leave other settings blank unless required.
3. **Capacity Mode**:
    - Choose **On-Demand** for automatic scaling (recommended for unpredictable workloads).
4. **Encryption**:

- Leave encryption enabled (default setting).

5. **Click Create Table**:
    - Your table is now ready.

Step 1.2: Lambda Function

What is AWS Lambda?

AWS Lambda is a serverless compute service that lets you run code without managing servers. It will process cost estimation requests and interact with Amazon SageMaker for AI predictions.

Steps to Create the Lambda Function:

Step 1.2.1: Create IAM Role

1. **Log in to AWS Console**:
    - Navigate to **IAM** under **Services**.
2. **Create a Role**:
    - Click **Roles** → **Create Role**.
    - Select **AWS Service** and choose **Lambda** as the trusted entity.
3. **Attach Policies**:
    - Attach the following policies to the role:
        a. **AmazonDynamoDBFullAccess**: Allows Lambda to interact with DynamoDB.
        b. **AmazonSageMakerFullAccess**: Allows Lambda to invoke SageMaker endpoints.
        c. **AWSLambdaBasicExecutionRole**: Enables Lambda to log events to CloudWatch.
4. **Name the Role**:
    - Provide a name, e.g., CostEstimationLambdaRole.
    - Click **Create Role**.
5. **Assign the Role to Lambda**:
    - When creating the Lambda function, assign this role under the **Permissions** section.

Step 1.2.2: Create the Lambda Function

1. **Navigate to Lambda**:
    - Go to **Services → Lambda** in the AWS Console.
2. **Create a Function**:
    - Click **Create Function** → Choose **Author from Scratch**.
    - Enter:
        - **Function Name**: CostEstimationFunction
        - **Runtime**: Python 3.9 (or higher).
3. **Permissions**:
    - Assign the IAM role (CostEstimationLambdaRole) created in Step 1.2.1.
4. **Add the Code**:

    Paste the following Python code into the Lambda editor:

```
import json
import boto3
import uuid
from datetime import datetime

# Initialize AWS clients
dynamodb = boto3.resource('dynamodb')
sagemaker_runtime = boto3.client('runtime.sagemaker')
table = dynamodb.Table('CostEstimates')

# SageMaker endpoint name (update this with your endpoint name)
ENDPOINT_NAME = 'your-sagemaker-endpoint-name'

def lambda_handler(event, context):
```

```
try:
    # Parse input from API Gateway
    body = json.loads(event['body'])
    payload = {
        "origin_country": body.get("originCountry"),
        "destination_country": body.get("destinationCountry"),
        "weight": float(body.get("weight", 0)),
        "length": float(body.get("length", 0)),
        "width": float(body.get("width", 0)),
        "height": float(body.get("height", 0)),
        "service_level": body.get("serviceLevel"),
        "package_type": body.get("packageType")
    }

    # Call SageMaker endpoint for prediction
    response = sagemaker_runtime.invoke_endpoint(
        EndpointName=ENDPOINT_NAME,
        ContentType="application/json",
        Body=json.dumps(payload)
    )
    result = json.loads(response['Body'].read().decode())
    estimated_cost = result.get('predicted_cost', 0)

    # Generate cost breakdown
    cost_breakdown = {
        "base_cost": 50.00,
        "weight_cost": 10 * payload["weight"],
```

```
        "dimensional_weight_cost": 0.01 * payload["length"] *
payload["width"] * payload["height"],
        "fuel_surcharge": 20.00,
        "insurance_fee": 5.00 if body.get("insurance") else 0.00,
        "additional_services_cost": 10.00,
        "tax_duties": 15.00
    }
    cost_breakdown["estimated_cost"] =
sum(cost_breakdown.values())

    # Store estimation in DynamoDB
    estimate_id = str(uuid.uuid4())
    table.put_item(
        Item={
            'estimate_id': estimate_id,
            'input_data': payload,
            'cost_breakdown': cost_breakdown,
            'estimated_cost': cost_breakdown["estimated_cost"],
            'timestamp': datetime.utcnow().isoformat()
        }
    )

    # Return response
    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*',
```

```
                'Access-Control-Allow-Methods': 'POST, OPTIONS',
                'Access-Control-Allow-Headers': 'Content-Type'
            },
            'body': json.dumps(cost_breakdown)
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Methods': 'POST, OPTIONS',
                'Access-Control-Allow-Headers': 'Content-Type'
            },
            'body': json.dumps({'error': str(e)})
5.      }
```

Save Changes:

- Click **Deploy** to save the Lambda function.

6. **Test the Lambda Function**:
   - Use the **Test** tab in the Lambda Console to run sample requests and verify the output.

Step 1.3: Configure API Gateway

What is API Gateway?

API Gateway provides a RESTful endpoint for your Lambda function, enabling communication between frontend and backend.

Steps to Configure API Gateway:

1. **Create an API**:
    - Navigate to **API Gateway** → **Create API** → Select **REST API** → Click **Build**.
2. **Define Resources**:
    - Create a resource /estimate-cost.
    - Under /estimate-cost, click **Create Method** → Select POST.
3. **Integrate with Lambda**:
    - Choose **Lambda Function** as the integration type.
    - Enter the name of your Lambda function (CostEstimationFunction).
4. **Enable CORS**:
    - Go to the POST method → Actions → **Enable CORS**.
    - Add:
        - **Allowed Origins**: * for development or your frontend domain (https://your-frontend-domain.com).
        - **Allowed Methods**: POST, OPTIONS.
        - **Allowed Headers**: Content-Type, Authorization.
5. **Deploy the API**:
    - Create a stage (e.g., dev or prod) → Deploy the API.
    - Note the endpoint URL

🚀 STEP-BY-STEP IMPLEMENTATION

STEP 1: Open Your SageMaker Notebook

1. Go to AWS Console → SageMaker → Notebook Instances
2. Open your existing notebook
3. Create a **NEW cell** at the beginning

STEP 2: Generate Realistic Dataset

Copy and paste this entire code into a new cell:

Python

```
import pandas as pd
import numpy as np
import random
from math import radians, sin, cos, sqrt, atan2

# Set random seed for reproducibility
np.random.seed(42)
random.seed(42)

print("="*70)
print("GENERATING REALISTIC SHIPPING COST DATASET")
print("Current User: tarunkumar452004")
print(f"Generated on: 2025-10-24 15:12:29 UTC")
print("="*70)

# Define realistic parameters with actual geographic coordinates
countries = {
    'United States': {'lat': 37.09, 'lon': -95.71, 'base_rate': 10},
```

```
    'Canada': {'lat': 56.13, 'lon': -106.35, 'base_rate': 12},
    'Mexico': {'lat': 23.63, 'lon': -102.55, 'base_rate': 11},
    'United Kingdom': {'lat': 55.38, 'lon': -3.44, 'base_rate': 15},
    'Germany': {'lat': 51.17, 'lon': 10.45, 'base_rate': 15},
    'France': {'lat': 46.23, 'lon': 2.21, 'base_rate': 15},
    'Spain': {'lat': 40.46, 'lon': -3.75, 'base_rate': 14},
    'Italy': {'lat': 41.87, 'lon': 12.57, 'base_rate': 14},
    'China': {'lat': 35.86, 'lon': 104.20, 'base_rate': 18},
    'Japan': {'lat': 36.20, 'lon': 138.25, 'base_rate': 20},
    'Australia': {'lat': -25.27, 'lon': 133.78, 'base_rate': 22},
    'India': {'lat': 20.59, 'lon': 78.96, 'base_rate': 16},
    'Brazil': {'lat': -14.24, 'lon': -51.93, 'base_rate': 17},
    'South Africa': {'lat': -30.56, 'lon': 22.94, 'base_rate': 18},
    'South Korea': {'lat': 35.91, 'lon': 127.77, 'base_rate': 19},
    'Singapore': {'lat': 1.35, 'lon': 103.82, 'base_rate': 21},
}

package_types = {
    'Box': {'multiplier': 1.0, 'handling': 5},
    'Pallet': {'multiplier': 1.5, 'handling': 25},
    'Envelope': {'multiplier': 0.5, 'handling': 2},
    'Tube': {'multiplier': 0.8, 'handling': 3},
    'Container': {'multiplier': 3.0, 'handling': 100}
}

service_levels = {
    'Economy': {'multiplier': 0.7, 'days': '7-10'},
    'Standard': {'multiplier': 1.0, 'days': '3-5'},
```

```
    'Express': {'multiplier': 1.5, 'days': '2-3'},
    'Priority': {'multiplier': 2.0, 'days': '1-2'}
}

transport_modes = {
    'Road': {'cost_per_km': 0.05, 'base': 20, 'max_distance': 5000},
    'Rail': {'cost_per_km': 0.03, 'base': 30, 'max_distance': 8000},
    'Air': {'cost_per_km': 0.15, 'base': 100, 'max_distance': 20000},
    'Sea': {'cost_per_km': 0.02, 'base': 50, 'max_distance': 25000}
}

content_types = {
    'Merchandise': {'risk': 1.0},
    'Documents': {'risk': 0.8},
    'Electronics': {'risk': 1.3},
    'Sample': {'risk': 0.9},
    'Gift': {'risk': 1.0},
    'Return': {'risk': 0.9}
}

def calculate_distance(lat1, lon1, lat2, lon2):
    """Calculate distance using Haversine formula"""
    R = 6371  # Earth's radius in km
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * atan2(sqrt(a), sqrt(1-a))
```

```
    return R * c


def select_appropriate_transport(distance_km, service_level):
    """Select realistic transport mode based on distance and service"""
    if distance_km < 500:
        if service_level == 'Priority':
            return 'Air'
        return random.choice(['Road', 'Road', 'Rail'])
    elif distance_km < 2000:
        if service_level in ['Priority', 'Express']:
            return 'Air'
        return random.choice(['Road', 'Rail', 'Air'])
    elif distance_km < 5000:
        if service_level in ['Priority', 'Express']:
            return 'Air'
        return random.choice(['Rail', 'Air', 'Sea'])
    else:  # Long distance
        if service_level == 'Economy':
            return 'Sea'
        elif service_level == 'Standard':
            return random.choice(['Air', 'Sea'])
        else:
            return 'Air'


def calculate_shipping_cost(origin, destination, weight, length, width,
height,
                            package_type, service_level, transport_mode,
                            content_type, declared_value, quantity,
```

```
                    add_insurance, signature_required, tracking):
    """Calculate realistic shipping cost"""

    # Get country data
    origin_data = countries[origin]
    dest_data = countries[destination]

    # 1. DISTANCE CALCULATION
    distance_km = calculate_distance(
        origin_data['lat'], origin_data['lon'],
        dest_data['lat'], dest_data['lon']
    )

    # 2. BASE COST
    base_cost = origin_data['base_rate'] + dest_data['base_rate']

    # 3. DISTANCE COST (varies by transport mode)
    transport_data = transport_modes[transport_mode]
    distance_cost = distance_km * transport_data['cost_per_km'] +
transport_data['base']

    # 4. WEIGHT COST ($4-6 per kg depending on service)
    weight_rate = 4.0 + (service_levels[service_level]['multiplier'] * 2)
    weight_cost = weight * weight_rate

    # 5. DIMENSIONAL WEIGHT (volumetric)
    dim_weight = (length * width * height) / 5000
    if dim_weight > weight:
```

```
        dim_weight_charge = (dim_weight - weight) * 3.0
    else:
        dim_weight_charge = 0

    # 6. PACKAGE HANDLING
    package_data = package_types[package_type]
    handling_cost = package_data['handling'] * quantity

    # 7. SERVICE LEVEL MULTIPLIER
    service_mult = service_levels[service_level]['multiplier']

    # 8. CONTENT TYPE RISK
    content_risk = content_types[content_type]['risk']

    # 9. FUEL SURCHARGE (15% of transport costs)
    fuel_surcharge = distance_cost * 0.15

    # 10. INSURANCE (0.5% of declared value if selected)
    insurance_cost = (declared_value * 0.005) if add_insurance == 1
else 0

    # 11. ADDITIONAL SERVICES
    signature_cost = 5.0 if signature_required == 1 else 0
    tracking_cost = 3.0 if tracking == 1 else 0

    # 12. INTERNATIONAL FEES
    if origin != destination:
        # Different countries = customs
```

```
    customs_rate = 0.08 if distance_km > 3000 else 0.05
    customs_cost = declared_value * customs_rate
else:
    customs_cost = 0

# 13. QUANTITY DISCOUNT
if quantity > 5:
    discount = base_cost * 0.10
elif quantity > 10:
    discount = base_cost * 0.15
elif quantity > 20:
    discount = base_cost * 0.20
else:
    discount = 0

# CALCULATE SUBTOTAL
subtotal = (
    base_cost +
    distance_cost +
    weight_cost +
    dim_weight_charge +
    handling_cost +
    fuel_surcharge +
    insurance_cost +
    signature_cost +
    tracking_cost +
    customs_cost
) * service_mult * content_risk * package_data['multiplier']
```

```
    # Apply discount
    total = subtotal - discount

    # Minimum cost
    total = max(15.0, total)

    # Add realistic variation (±3%)
    variation = np.random.uniform(-0.03, 0.03)
    total *= (1 + variation)

    return round(total, 2)

# GENERATE DATASET
print("\n📦 Generating 10,000 realistic shipping records...")
print("   Using real-world pricing formulas...")

num_records = 10000
data = []

for i in range(num_records):
    # Select origin and destination
    origin = random.choice(list(countries.keys()))
    destination = random.choice(list(countries.keys()))

    # 80% different countries, 20% domestic
    if random.random() < 0.2:
        destination = origin
```

```
    # Calculate distance for transport mode selection
    origin_data = countries[origin]
    dest_data = countries[destination]
    distance_km = calculate_distance(
        origin_data['lat'], origin_data['lon'],
        dest_data['lat'], dest_data['lon']
    )

    # Service level (weighted distribution)
    service_level = random.choices(
        list(service_levels.keys()),
        weights=[20, 50, 20, 10]  # Standard most common
    )[0]

    # Transport mode (realistic based on distance)
    transport_mode = select_appropriate_transport(distance_km,
service_level)

    # Weight (log-normal: most packages 1-20kg, few heavy ones)
    weight = round(np.random.lognormal(1.8, 0.9), 2)
    weight = max(0.1, min(weight, 150))

    # Dimensions (realistic based on weight)
    if weight < 1:
        length, width, height = 15, 12, 8
    elif weight < 5:
        length, width, height = 30, 25, 20
```

```
elif weight < 20:
    length, width, height = 50, 40, 30
else:
    length, width, height = 80, 60, 50

# Add variation to dimensions
length = round(length * np.random.uniform(0.8, 1.2), 1)
width = round(width * np.random.uniform(0.8, 1.2), 1)
height = round(height * np.random.uniform(0.8, 1.2), 1)

# Package type (weighted by common usage)
package_type = random.choices(
    list(package_types.keys()),
    weights=[60, 15, 15, 5, 5]
)[0]

# Content type
content_type = random.choice(list(content_types.keys()))

# Declared value (based on content type)
if content_type == 'Electronics':
    declared_value = round(np.random.uniform(200, 3000), 2)
elif content_type == 'Documents':
    declared_value = round(np.random.uniform(20, 300), 2)
else:
    declared_value = round(np.random.uniform(50, 800), 2)

# Quantity (most are 1)
```

```
    quantity = random.choices([1, 2, 3, 5, 10, 25], weights=[70, 15, 8, 4,
2, 1])[0]

    # Zip codes
    origin_zip = random.randint(10000, 99999)
    destination_zip = random.randint(10000, 99999)

    # Additional services (realistic probabilities)
    add_insurance = 1 if (declared_value > 500 or content_type ==
'Electronics') and random.random() < 0.6 else 0
    signature_required = 1 if service_level in ['Priority', 'Express'] and
random.random() < 0.4 else 0
    tracking = 1 if random.random() < 0.9 else 0  # 90% have tracking

    # Calculate cost
    cost = calculate_shipping_cost(
        origin, destination, weight, length, width, height,
        package_type, service_level, transport_mode,
        content_type, declared_value, quantity,
        add_insurance, signature_required, tracking
    )

    # Create record
    record = {
        'origin_country': origin,
        'origin_zip': origin_zip,
        'destination_country': destination,
        'destination_zip': destination_zip,
        'package_type': package_type,
```

```
            'quantity': quantity,
            'weight_kg': weight,
            'length_cm': length,
            'width_cm': width,
            'height_cm': height,
            'declared_value': declared_value,
            'content_type': content_type,
            'service_level': service_level,
            'transport_mode': transport_mode,
            'add_insurance': add_insurance,
            'signature_required': signature_required,
            'tracking': tracking,
            'shipping_cost': cost
        }

        data.append(record)

        if (i + 1) % 2000 == 0:
            print(f"   ✓ Generated {i + 1}/{num_records} records")

# Create DataFrame
df = pd.DataFrame(data)

print("\n" + "="*70)
print(" ✅ DATASET GENERATION COMPLETE")
print("="*70)
```

```
# Statistics
print(f"\n📊 DATASET STATISTICS")
print(f"{'='*70}")
print(f"Total Records: {len(df):,}")
print(f"\n💰 Cost Distribution:")
print(f"   Mean:    ${df['shipping_cost'].mean():.2f}")
print(f"   Median:  ${df['shipping_cost'].median():.2f}")
print(f"   Std Dev: ${df['shipping_cost'].std():.2f}")
print(f"   Min:     ${df['shipping_cost'].min():.2f}")
print(f"   Max:     ${df['shipping_cost'].max():.2f}")

print(f"\n⚖️ Weight Distribution:")
print(f"   Mean:    {df['weight_kg'].mean():.2f} kg")
print(f"   Median:  {df['weight_kg'].median():.2f} kg")
print(f"   Range:   {df['weight_kg'].min():.2f} -
{df['weight_kg'].max():.2f} kg")

print(f"\n📦 Service Level Distribution:")
for service, count in df['service_level'].value_counts().items():
    pct = (count / len(df)) * 100
    print(f"   {service:10s}: {count:5d} ({pct:5.1f}%)")

print(f"\n🚚 Transport Mode Distribution:")
for mode, count in df['transport_mode'].value_counts().items():
    pct = (count / len(df)) * 100
    print(f"   {mode:10s}: {count:5d} ({pct:5.1f}%)")

print(f"\n🌍 Top 5 Routes:")
```

```
routes = df.groupby(['origin_country',
'destination_country']).size().sort_values(ascending=False).head()

for (orig, dest), count in routes.items():
    print(f"   {orig:20s} → {dest:20s}: {count:4d} shipments")


# Correlation analysis
print(f"\n📈 COST CORRELATION ANALYSIS")
print(f"{'='*70}")


print(f"\n💵 Average Cost by Service Level:")
service_costs =
df.groupby('service_level')['shipping_cost'].mean().sort_values()
for service, cost in service_costs.items():
    print(f"   {service:10s}: ${cost:7.2f}")


print(f"\n💵 Average Cost by Transport Mode:")
transport_costs =
df.groupby('transport_mode')['shipping_cost'].mean().sort_values()
for mode, cost in transport_costs.items():
    print(f"   {mode:10s}: ${cost:7.2f}")


print(f"\n💵 Average Cost by Weight Range:")
df['weight_range'] = pd.cut(df['weight_kg'], bins=[0, 5, 10, 20, 50,
200],
                            labels=['0-5kg', '5-10kg', '10-20kg', '20-50kg',
'50kg+'])
weight_costs = df.groupby('weight_range')['shipping_cost'].mean()
for range_label, cost in weight_costs.items():
    print(f"   {range_label:10s}: ${cost:7.2f}")
```

```
# Sample records
print(f"\n📋 SAMPLE RECORDS (First 5)")
print("="*70)
sample_cols = ['origin_country', 'destination_country', 'weight_kg',
               'service_level', 'transport_mode', 'shipping_cost']
print(df[sample_cols].head().to_string(index=False))

# Save to CSV
csv_filename = 'shipping_data_realistic.csv'
df.to_csv(csv_filename, index=False)
print(f"\n✅ Dataset saved to: {csv_filename}")
print(f"   Size: {len(df):,} records")
print(f"   File size: ~{len(df) * 0.5 / 1024:.1f} MB")

print("\n" + "="*70)
print("🎉 READY FOR MODEL TRAINING!")
print("="*70)
```

Run this cell and wait for completion (~1-2 minutes).

STEP 3: Verify the Data Quality

Add a new cell to visualize the data:

```
Python
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
```

```
sns.set_style("whitegrid")
plt.figure(figsize=(16, 10))

# 1. Cost Distribution
plt.subplot(2, 3, 1)
plt.hist(df['shipping_cost'], bins=50, edgecolor='black', alpha=0.7)
plt.xlabel('Shipping Cost ($)')
plt.ylabel('Frequency')
plt.title('Cost Distribution')
plt.axvline(df['shipping_cost'].mean(), color='red', linestyle='--',
label=f'Mean: ${df["shipping_cost"].mean():.2f}')
plt.legend()

# 2. Weight Distribution
plt.subplot(2, 3, 2)
plt.hist(df['weight_kg'], bins=50, edgecolor='black', alpha=0.7,
color='green')
plt.xlabel('Weight (kg)')
plt.ylabel('Frequency')
plt.title('Weight Distribution')
plt.axvline(df['weight_kg'].mean(), color='red', linestyle='--',
label=f'Mean: {df["weight_kg"].mean():.2f} kg')
plt.legend()

# 3. Cost vs Weight
plt.subplot(2, 3, 3)
plt.scatter(df['weight_kg'], df['shipping_cost'], alpha=0.3, s=10)
plt.xlabel('Weight (kg)')
plt.ylabel('Shipping Cost ($)')
```

```
plt.title('Cost vs Weight Correlation')

# 4. Cost by Service Level
plt.subplot(2, 3, 4)
service_data = df.groupby('service_level')['shipping_cost'].mean().sort_values()
service_data.plot(kind='bar', color='skyblue', edgecolor='black')
plt.xlabel('Service Level')
plt.ylabel('Average Cost ($)')
plt.title('Average Cost by Service Level')
plt.xticks(rotation=45)

# 5. Cost by Transport Mode
plt.subplot(2, 3, 5)
transport_data = df.groupby('transport_mode')['shipping_cost'].mean().sort_values()
transport_data.plot(kind='bar', color='coral', edgecolor='black')
plt.xlabel('Transport Mode')
plt.ylabel('Average Cost ($)')
plt.title('Average Cost by Transport Mode')
plt.xticks(rotation=45)

# 6. Service Level Distribution
plt.subplot(2, 3, 6)
df['service_level'].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.ylabel('')
plt.title('Service Level Distribution')
```

plt.tight_layout()

plt.savefig('data_visualization.png', dpi=150, bbox_inches='tight')

plt.show()

print(" ✅ Visualization saved: data_visualization.png")

STEP 4: Upload to S3

Python

import boto3

Upload the new dataset

s3 = boto3.client('s3')

bucket_name = 'cost-estimation-capstone-2025'

print("Uploading realistic dataset to S3...")

s3.upload_file('shipping_data_realistic.csv', bucket_name, 'shipping_data_realistic.csv')

print(f" ✅ Uploaded to: s3://{bucket_name}/shipping_data_realistic.csv")

STEP 5: Retrain the Model with New Data

Python

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error

import pickle

import json

```
print("="*70)
print("RETRAINING MODEL WITH REALISTIC DATA")
print("="*70)

# Load the realistic data
df = pd.read_csv('shipping_data_realistic.csv')

# Separate features and target
X = df.drop('shipping_cost', axis=1)
y = df['shipping_cost']

# Apply one-hot encoding
categorical_features = ['origin_country', 'destination_country',
'package_type',
                        'content_type', 'service_level', 'transport_mode']

X_encoded = pd.get_dummies(X, columns=categorical_features)

print(f"\n📊 Feature Engineering Complete")
print(f"   Original features: {len(X.columns)}")
print(f"   Encoded features: {len(X_encoded.columns)}")

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

```
print(f"\n📦 Dataset Split:")
print(f"   Training samples: {len(X_train):,}")
print(f"   Testing samples: {len(X_test):,}")

# Train Random Forest
print(f"\n🌲 Training Random Forest Model...")
model = RandomForestRegressor(
    n_estimators=100,
    max_depth=20,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1  # Use all CPU cores
)

model.fit(X_train, y_train)
print(f"   ✓ Model trained successfully!")

# Evaluate
print(f"\n📈 MODEL EVALUATION")
print("="*70)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

train_mae = mean_absolute_error(y_train, y_pred_train)
test_mae = mean_absolute_error(y_test, y_pred_test)
```

```
train_r2 = r2_score(y_train, y_pred_train)
test_r2 = r2_score(y_test, y_pred_test)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)

print(f"\n🎯 Training Set Performance:")
print(f"   MAE: ${train_mae:.2f}")
print(f"   R² Score: {train_r2:.4f}")

print(f"\n🎯 Test Set Performance:")
print(f"   MAE: ${test_mae:.2f}")
print(f"   RMSE: ${test_rmse:.2f}")
print(f"   R² Score: {test_r2:.4f}")
print(f"   Accuracy: {test_r2*100:.2f}%")

# Feature importance
print(f"\n⭐ TOP 10 MOST IMPORTANT FEATURES:")
feature_importance = pd.DataFrame({
    'feature': X_encoded.columns,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)

for idx, row in feature_importance.head(10).iterrows():
    print(f"   {row['feature']:40s}: {row['importance']:.4f}")

# Save model
print(f"\n💾 Saving Model...")
with open('shipping_cost_model_v2.pkl', 'wb') as f:
```

```
    pickle.dump(model, f)

with open('feature_names_v2.json', 'w') as f:
    json.dump(list(X_encoded.columns), f)

print(f"   ✓ Model saved: shipping_cost_model_v2.pkl")
print(f"   ✓ Features saved: feature_names_v2.json")

# Test predictions
print(f"\n🧪 SAMPLE PREDICTIONS:")
print("="*70)
sample_indices = np.random.choice(X_test.index, 5, replace=False)
for idx in sample_indices:
    actual = y_test.loc[idx]
    predicted = model.predict(X_encoded.loc[[idx]])[0]
    error = abs(actual - predicted)
    error_pct = (error / actual) * 100

    print(f"\nSample {idx}:")
    print(f"   Actual:    ${actual:.2f}")
    print(f"   Predicted: ${predicted:.2f}")
    print(f"   Error:     ${error:.2f} ({error_pct:.1f}%)")

print("\n" + "="*70)
print("✅ MODEL RETRAINING COMPLETE!")
print("="*70)
```

STEP 6: Upload New Model to S3 and Redeploy

Python

```
import tarfile
import os
import shutil

print("="*70)
print("PREPARING MODEL FOR SAGEMAKER DEPLOYMENT")
print("="*70)

# Clean up old files
for dirname in ['model_v2', 'code']:
    if os.path.exists(dirname):
        shutil.rmtree(dirname)

# Create model directory
os.makedirs('model_v2')

# Copy files
shutil.copy('shipping_cost_model_v2.pkl',
'model_v2/shipping_cost_model.pkl')
shutil.copy('feature_names_v2.json', 'model_v2/feature_names.json')
print("✓ Model files prepared")

# Create model.tar.gz
if os.path.exists('model_v2.tar.gz'):
    os.remove('model_v2.tar.gz')
```

```
with tarfile.open('model_v2.tar.gz', 'w:gz') as tar:
    tar.add('model_v2/shipping_cost_model.pkl', arcname='shipping_cost_model.pkl')
    tar.add('model_v2/feature_names.json', arcname='feature_names.json')

print("✓ model_v2.tar.gz created")

# Upload to S3
model_s3_path_v2 = sess.upload_data(
    path='model_v2.tar.gz',
    bucket=bucket_name,
    key_prefix='sagemaker-model-v2'
)

print(f"✓ Uploaded to: {model_s3_path_v2}")
print("\n" + "="*70)
print(" 📊 READY TO UPDATE SAGEMAKER ENDPOINT")
print("="*70)
print(f"\nNew Model Path: {model_s3_path_v2}")
```

STEP 7: Update SageMaker Endpoint with New Model

```
Python
from sagemaker.sklearn import SKLearnModel
from datetime import datetime

print("="*70)
print("UPDATING SAGEMAKER ENDPOINT WITH IMPROVED MODEL")
```

```
print("="*70)

endpoint_name = "shipping-endpoint-20251024-105923"

# Create new model
sklearn_model_v2 = SKLearnModel(
    model_data=model_s3_path_v2,
    role=role,
    entry_point='inference.py',
    source_dir='code',
    framework_version='1.0-1',
    py_version='py3',
    sagemaker_session=sess
)

print(f"\n⏳ Updating endpoint: {endpoint_name}")
print(f"   This will take 5-8 minutes...")

try:
    predictor = sklearn_model_v2.deploy(
        initial_instance_count=1,
        instance_type='ml.t2.medium',
        endpoint_name=endpoint_name,
        update_endpoint=True,
        wait=True
    )

    print("\n" + "="*70)
```

```
    print("✅ ENDPOINT UPDATED WITH NEW MODEL!")
    print("="*70)
    print(f"Endpoint: {endpoint_name}")
    print(f"Status: InService")
    print(f"Model: v2 (Realistic Data)")

except Exception as e:
    print(f"\n❌ Update failed: {e}")
    print("\nTrying to create new endpoint...")

    new_endpoint_name = f"shipping-realistic-{datetime.now().strftime('%Y%m%d-%H%M%S')}"
    predictor = sklearn_model_v2.deploy(
        initial_instance_count=1,
        instance_type='ml.t2.medium',
        endpoint_name=new_endpoint_name,
        wait=True
    )

    print(f"\n✅ New endpoint created: {new_endpoint_name}")
    print(f"\n⚠️  Update your Lambda function with new endpoint name:")
    print(f"   ENDPOINT_NAME = '{new_endpoint_name}'")
```

STEP 8: Test the Updated Endpoint

```
Python

import boto3

import json
```

```
runtime_client = boto3.client('sagemaker-runtime')
endpoint_name = "shipping-endpoint-20251024-105923"

print("="*70)
print("TESTING UPDATED ENDPOINT WITH REALISTIC MODEL")
print("="*70)

# Test cases
test_cases = [
  {
    "name": "USA to Canada - Standard Ground",
    "data": {
      "origin_country": "United States",
      "destination_country": "Canada",
      "weight_kg": 5.0,
      "length_cm": 30.0,
      "width_cm": 25.0,
      "height_cm": 20.0,
      "package_type": "Box",
      "service_level": "Standard",
      "transport_mode": "Road",
      "content_type": "Merchandise",
      "declared_value": 100,
      "quantity": 1,
      "origin_zip": 10001,
      "destination_zip": 12345,
      "add_insurance": 0,
```

```
      "signature_required": 0,
      "tracking": 1
    }
  },
  {
    "name": "USA to Japan - Express Air",
    "data": {
      "origin_country": "United States",
      "destination_country": "Japan",
      "weight_kg": 10.0,
      "length_cm": 50.0,
      "width_cm": 40.0,
      "height_cm": 30.0,
      "package_type": "Box",
      "service_level": "Express",
      "transport_mode": "Air",
      "content_type": "Electronics",
      "declared_value": 1500,
      "quantity": 1,
      "origin_zip": 10001,
      "destination_zip": 12345,
      "add_insurance": 1,
      "signature_required": 1,
      "tracking": 1
    }
  },
  {
    "name": "Domestic USA - Economy",
```

```
        "data": {
            "origin_country": "United States",
            "destination_country": "United States",
            "weight_kg": 2.0,
            "length_cm": 25.0,
            "width_cm": 20.0,
            "height_cm": 15.0,
            "package_type": "Box",
            "service_level": "Economy",
            "transport_mode": "Road",
            "content_type": "Documents",
            "declared_value": 50,
            "quantity": 1,
            "origin_zip": 10001,
            "destination_zip": 90001,
            "add_insurance": 0,
            "signature_required": 0,
            "tracking": 1
        }
    }
]

print(f"\nRunning {len(test_cases)} test predictions...\n")

for i, test_case in enumerate(test_cases, 1):
    print(f"{i}. {test_case['name']}")
    print("   " + "-" * 60)
```

```
    try:
        response = runtime_client.invoke_endpoint(
            EndpointName=endpoint_name,
            ContentType='application/json',
            Body=json.dumps(test_case['data'])
        )

        result = json.loads(response['Body'].read().decode())
        cost = result['predicted_cost']

        print(f"   ✅ Predicted Cost: ${cost:.2f}")
        print(f"   Route: {test_case['data']['origin_country']} → {test_case['data']['destination_country']}")
        print(f"   Weight: {test_case['data']['weight_kg']} kg")
        print(f"   Service: {test_case['data']['service_level']}")
        print(f"   Transport: {test_case['data']['transport_mode']}")

    except Exception as e:
        print(f"   ❌ Error: {e}")

    print()

print("="*70)
print("✅ TESTING COMPLETE!")
print("="*70)
```