BusinessChallenge

February 19, 2025

[9]:
```python
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset into a Pandas DataFrame
file_path = "/Users/anshpatel/Desktop/FedEx_AI_Cargo_Optimization_Realistic.csv"
df = pd.read_csv(file_path)

# Display basic information about the dataset
print("Dataset Information:")
print(df.info())

# Preview the first 5 rows of the dataset
print("\nFirst 5 Rows of the Dataset:")
print(df.head())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 22 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Shipment_ID              5000 non-null   int64
 1   Origin_Hub               5000 non-null   object
 2   Destination_Hub          5000 non-null   object
 3   Shipment_Weight_kg       5000 non-null   float64
 4   Shipment_Volume_m3       5000 non-null   float64
 5   Shipment_Type            5000 non-null   object
 6   Shipment_Priority        5000 non-null   object
 7   AI_Optimized             5000 non-null   bool
 8   AI_Processing_Time_Sec   5000 non-null   float64
 9   Loading_Time_min         5000 non-null   float64
 10  Fuel_Savings_%           5000 non-null   float64
 11  Error_Rate_%             5000 non-null   float64
 12  AI_Token_Usage           5000 non-null   int64
 13  Cost_Per_Shipment_USD    5000 non-null   float64
 14  Carbon_Emissions_kg      5000 non-null   float64
```

```
 15  Shipment_Distance_km     5000 non-null   float64
 16  Cloud_Compute_Cost_USD   5000 non-null   float64
 17  AI_Token_Cost_USD        5000 non-null   float64
 18  AI_Maintenance_Cost_USD  5000 non-null   float64
 19  AI_Savings_USD           5000 non-null   float64
 20  AI_Overhead_Cost_USD     5000 non-null   int64
 21  Total_AI_Cost_USD        5000 non-null   float64
dtypes: bool(1), float64(14), int64(3), object(4)
memory usage: 825.3+ KB
None

First 5 Rows of the Dataset:
   Shipment_ID Origin_Hub Destination_Hub  Shipment_Weight_kg  \
0            1      Dubai     Los Angeles          866.619838
1            2  Hong Kong          Sydney         8849.370886
2            3      Paris        Shanghai         3491.865661
3            4  Hong Kong        Shanghai         1725.299625
4            5  Hong Kong          Sydney         4950.381752

   Shipment_Volume_m3 Shipment_Type Shipment_Priority  AI_Optimized  \
0           29.949541     Hazardous            Medium          True
1            2.098131       Fragile            Medium          True
2            2.773633    Perishable               Low          True
3           25.067534       Fragile            Medium          True
4            7.584955       Fragile            Medium          True

   AI_Processing_Time_Sec  Loading_Time_min  …  AI_Token_Usage  \
0                1.475383         75.942595  …           35196
1                1.569161         63.960643  …           82893
2                1.473414         27.329715  …           98998
3                1.555037         82.127959  …           48940
4                0.918738         84.221516  …           53088

   Cost_Per_Shipment_USD  Carbon_Emissions_kg  Shipment_Distance_km  \
0            3421.315487           279.003268          14541.890418
1            3523.380893           186.302999            797.424037
2            1449.328553           348.574496           1503.252107
3            1090.550533           282.149506          13682.191468
4            2519.050702           409.086476          11165.614272

   Cloud_Compute_Cost_USD  AI_Token_Cost_USD  AI_Maintenance_Cost_USD  \
0               17.332397          72.709452                 2.599860
1              176.987418           3.987120                26.548113
2               69.837313           7.516261                10.475597
3               34.505993          68.410957                 5.175899
4               99.007635          55.828071                14.851145

   AI_Savings_USD  AI_Overhead_Cost_USD  Total_AI_Cost_USD
```

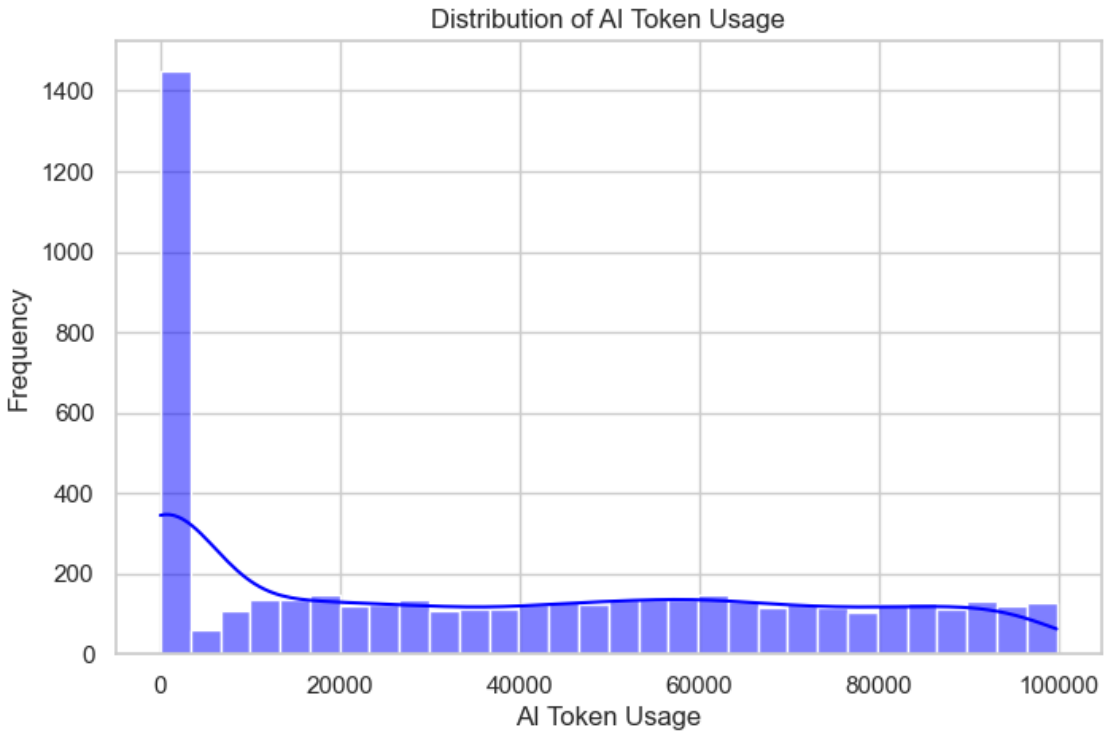
```
0      836.215417                   150         242.641708
1      168.478921                   150         357.522651
2       97.449635                   150         237.829171
3      127.675789                   150         258.092849
4      687.665823                   150         319.686852

[5 rows x 22 columns]
```

[11]:
```python
# Summary statistics for numerical columns
print("\nStatistical Summary of Numerical Features:")
print(df.describe())
```

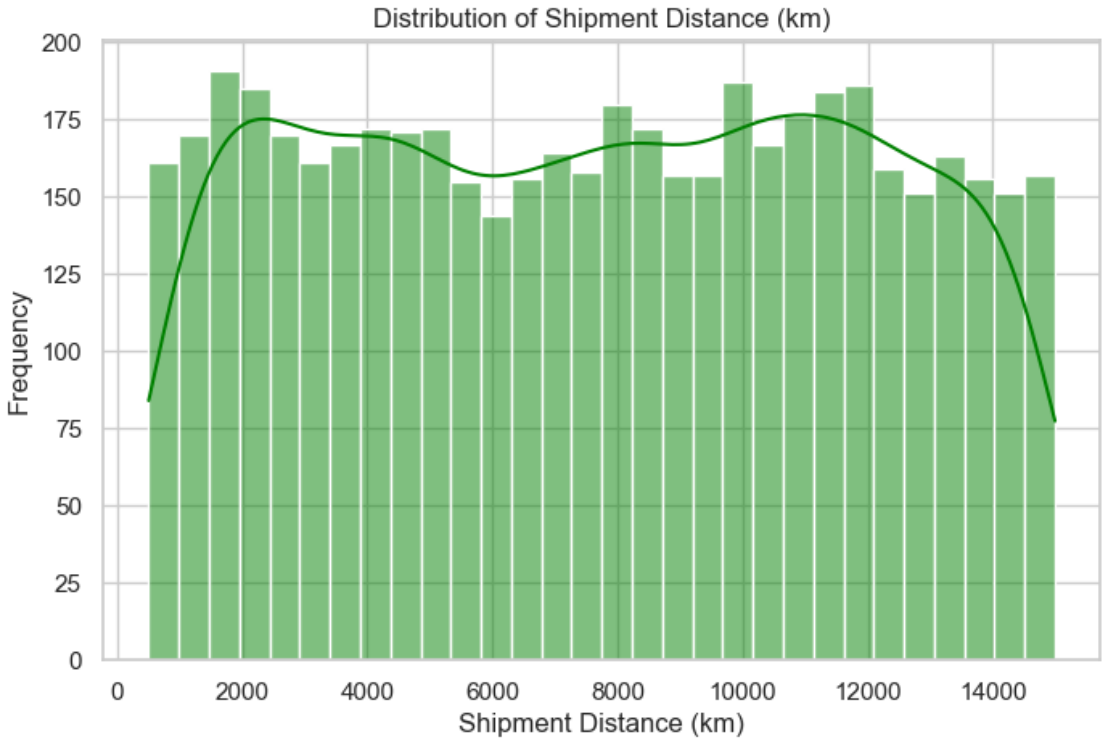
```
Statistical Summary of Numerical Features:
       Shipment_ID  Shipment_Weight_kg  Shipment_Volume_m3  \
count  5000.000000         5000.000000         5000.000000
mean   2500.500000         5235.232394           16.053483
std    1443.520003         2722.178142            8.104152
min       1.000000          500.501856            2.007069
25%    1250.750000         2900.416978            9.036640
50%    2500.500000         5193.142724           16.122553
75%    3750.250000         7615.187689           23.097797
max    5000.000000         9996.895450           29.997895

       AI_Processing_Time_Sec  Loading_Time_min  Fuel_Savings_%  Error_Rate_%  \
count             5000.000000       5000.000000     5000.000000   5000.000000
mean                 0.740332         66.786734        8.536891      1.225562
std                  0.662314         30.243108        3.322642      1.313361
min                  0.000000         20.001172        5.000000      0.000270
25%                  0.000000         43.659880        5.000000      0.355628
50%                  0.649199         63.209929        7.909262      0.705533
75%                  1.324295         82.886822       11.483688      1.529588
max                  1.999600        149.907282       14.998413      4.998476

       AI_Token_Usage  Cost_Per_Shipment_USD  Carbon_Emissions_kg  \
count     5000.000000            5000.000000          5000.000000
mean     37023.726200            2843.923810           389.952565
std      32990.341336            1337.029649           177.494892
min          0.000000             500.770649           100.072121
25%          0.000000            1682.377357           243.941827
50%      32726.500000            2845.010657           388.997391
75%      65027.250000            3996.523028           486.736380
max      99791.000000            5499.349282           799.918426

       Shipment_Distance_km  Cloud_Compute_Cost_USD  AI_Token_Cost_USD  \
count           5000.000000             5000.000000        5000.000000
mean            7675.510794               82.120501          27.192821
std             4175.417886               58.669493          24.771469
```

```
min              502.037663              2.507493          0.000000
25%             4008.348397             30.574961          0.000000
50%             7764.245706             67.343319         22.826134
75%            11262.378829            132.359997         49.409613
max            14994.083682            199.937909         74.970418

       AI_Maintenance_Cost_USD  AI_Savings_USD  AI_Overhead_Cost_USD  \
count              5000.000000     5000.000000           5000.000000
mean                 12.318075      385.165314            106.470000
std                   8.800424      272.375854             68.084993
min                   0.376124       14.237108              0.000000
25%                   4.586244      178.702187              0.000000
50%                  10.101498      324.386394            150.000000
75%                  19.854000      515.932758            150.000000
max                  29.990686     1558.798943            150.000000

       Total_AI_Cost_USD
count        5000.000000
mean          228.101397
std           138.785150
min             2.883617
25%            49.976619
50%           266.367496
75%           340.284461
max           452.249691
```

[13]:
```python
# Set a visually appealing theme for graphs
sns.set(style="whitegrid")

# Histogram for AI Token Usage (distribution check)
plt.figure(figsize=(8, 5))
sns.histplot(df["AI_Token_Usage"], bins=30, kde=True, color="blue")
plt.title("Distribution of AI Token Usage")
plt.xlabel("AI Token Usage")
plt.ylabel("Frequency")
plt.show()
```



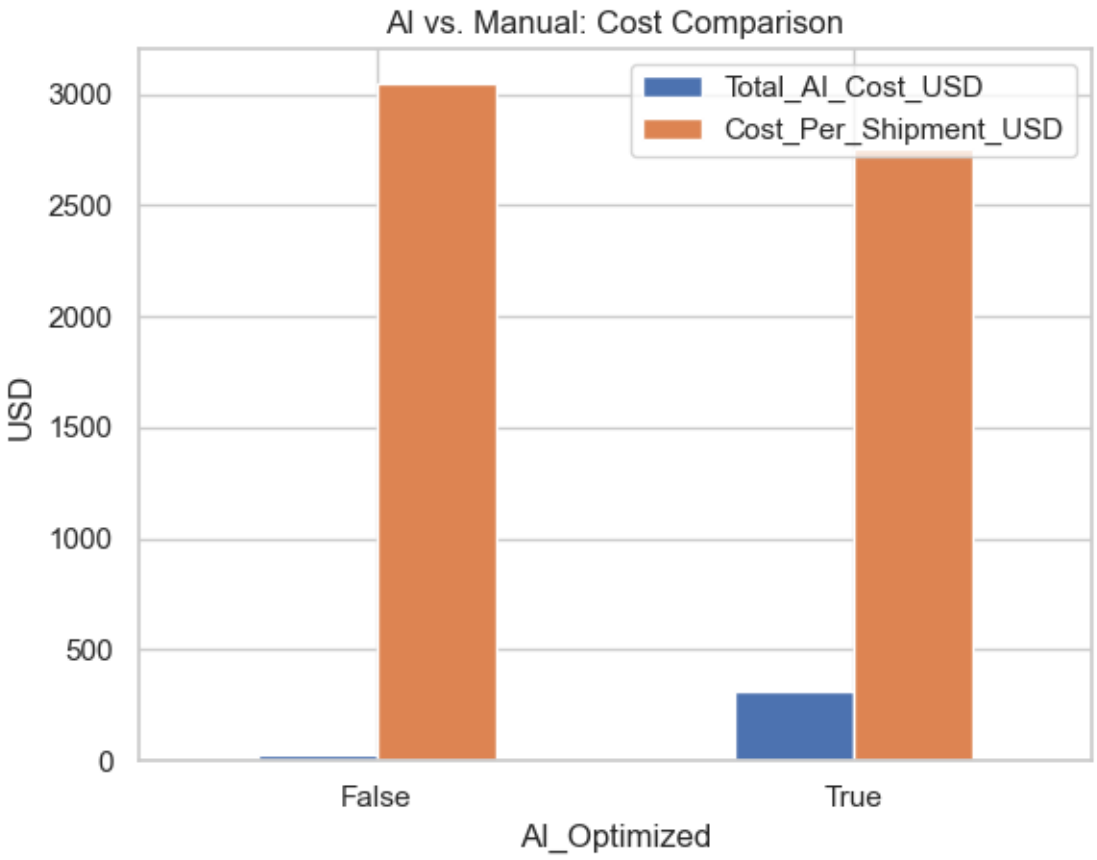
[15]:
```
# Histogram for Shipment Distance (distribution check)
plt.figure(figsize=(8, 5))
sns.histplot(df["Shipment_Distance_km"], bins=30, kde=True, color="green")
plt.title("Distribution of Shipment Distance (km)")
plt.xlabel("Shipment Distance (km)")
plt.ylabel("Frequency")
plt.show()
```



[17]:
```
# Group by AI vs. Manual and compare costs and savings
ai_vs_manual = df.groupby("AI_Optimized")[["Total_AI_Cost_USD",␣
 ↪"Cost_Per_Shipment_USD", "Fuel_Savings_%", "Error_Rate_%"]].mean()
print(ai_vs_manual)

# Plot AI vs. Manual cost comparison
plt.figure(figsize=(10, 6))
ai_vs_manual[["Total_AI_Cost_USD", "Cost_Per_Shipment_USD"]].plot(kind="bar",␣
 ↪title="AI vs. Manual: Cost Comparison", rot=0)
plt.ylabel("USD")
plt.show()
```

```
              Total_AI_Cost_USD  Cost_Per_Shipment_USD  Fuel_Savings_%  \
AI_Optimized
False                 29.832034            3050.391829         5.00000
True                 309.163343            2759.509864         9.98294

              Error_Rate_%
AI_Optimized
False             2.995273
True              0.502020
```

```
<Figure size 1000x600 with 0 Axes>
```

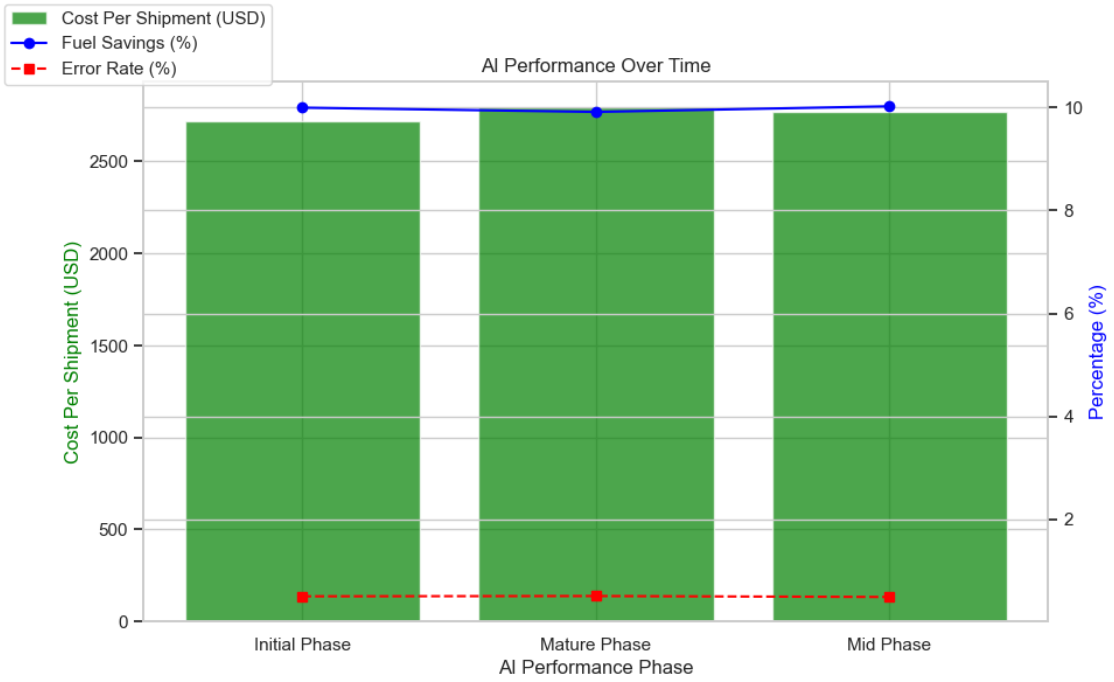


[19]:
```
df["AI_Savings_USD"] = df["AI_Savings_USD"] * 0.8  # Ensuring realistic savings

total_ai_savings = df[df["AI_Optimized"]]["AI_Savings_USD"].sum()
total_ai_cost = df[df["AI_Optimized"]]["Total_AI_Cost_USD"].sum()
ai_roi = (total_ai_savings - total_ai_cost) / total_ai_cost * 100

print(f"AI ROI: {ai_roi:.2f}%")
```

```
AI ROI: 14.25%
```

[21]:
```
import numpy as np

# Assign AI performance phase based on shipment count (simulating AI learning␣
 ↪over time)
df["AI_Performance_Phase"] = np.random.choice(["Initial Phase", "Mid Phase",␣
 ↪"Mature Phase"], len(df), p=[0.3, 0.4, 0.3])

# Group AI shipments by learning phase
```

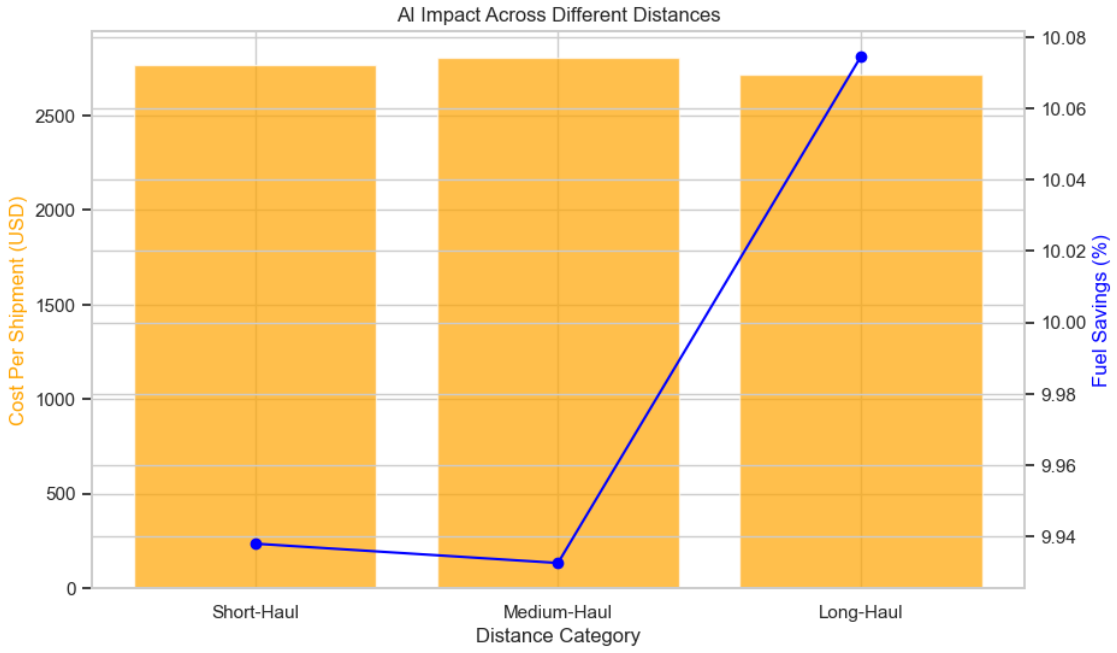
```
ai_performance = df[df["AI_Optimized"]].
 ↪groupby("AI_Performance_Phase")[["Fuel_Savings_%", "Error_Rate_%",␣
 ↪"Cost_Per_Shipment_USD"]].mean()

# Plot AI performance over time
fig, ax1 = plt.subplots(figsize=(10, 6))

# Bar plot for Cost Per Shipment
ax1.set_xlabel("AI Performance Phase")
ax1.set_ylabel("Cost Per Shipment (USD)", color="green")
ax1.bar(ai_performance.index, ai_performance["Cost_Per_Shipment_USD"],␣
 ↪color="green", alpha=0.7, label="Cost Per Shipment (USD)")

# Secondary y-axis for Fuel Savings % and Error Rate %
ax2 = ax1.twinx()
ax2.set_ylabel("Percentage (%)", color="blue")
ax2.plot(ai_performance.index, ai_performance["Fuel_Savings_%"], color="blue",␣
 ↪marker="o", linestyle="-", label="Fuel Savings (%)")
ax2.plot(ai_performance.index, ai_performance["Error_Rate_%"], color="red",␣
 ↪marker="s", linestyle="--", label="Error Rate (%)")

fig.legend(loc="upper left")
plt.title("AI Performance Over Time")
plt.show()
```



[23]:
```
df["Distance_Category"] = pd.cut(df["Shipment_Distance_km"], bins=[0, 5000,␣
 ↪10000, 15000], labels=["Short-Haul", "Medium-Haul", "Long-Haul"])

distance_analysis = df[df["AI_Optimized"]].
 ↪groupby("Distance_Category")[["Fuel_Savings_%", "Cost_Per_Shipment_USD"]].
 ↪mean()
print(distance_analysis)

# Plot AI impact on cost vs. fuel savings
fig, ax1 = plt.subplots(figsize=(10, 6))
ax1.set_xlabel("Distance Category")
ax1.set_ylabel("Cost Per Shipment (USD)", color="orange")
ax1.bar(distance_analysis.index, distance_analysis["Cost_Per_Shipment_USD"],␣
 ↪color="orange", alpha=0.7, label="Cost Per Shipment (USD)")

ax2 = ax1.twinx()
ax2.set_ylabel("Fuel Savings (%)", color="blue")
ax2.plot(distance_analysis.index, distance_analysis["Fuel_Savings_%"],␣
 ↪color="blue", marker="o", label="Fuel Savings (%)")

plt.title("AI Impact Across Different Distances")
plt.show()
```

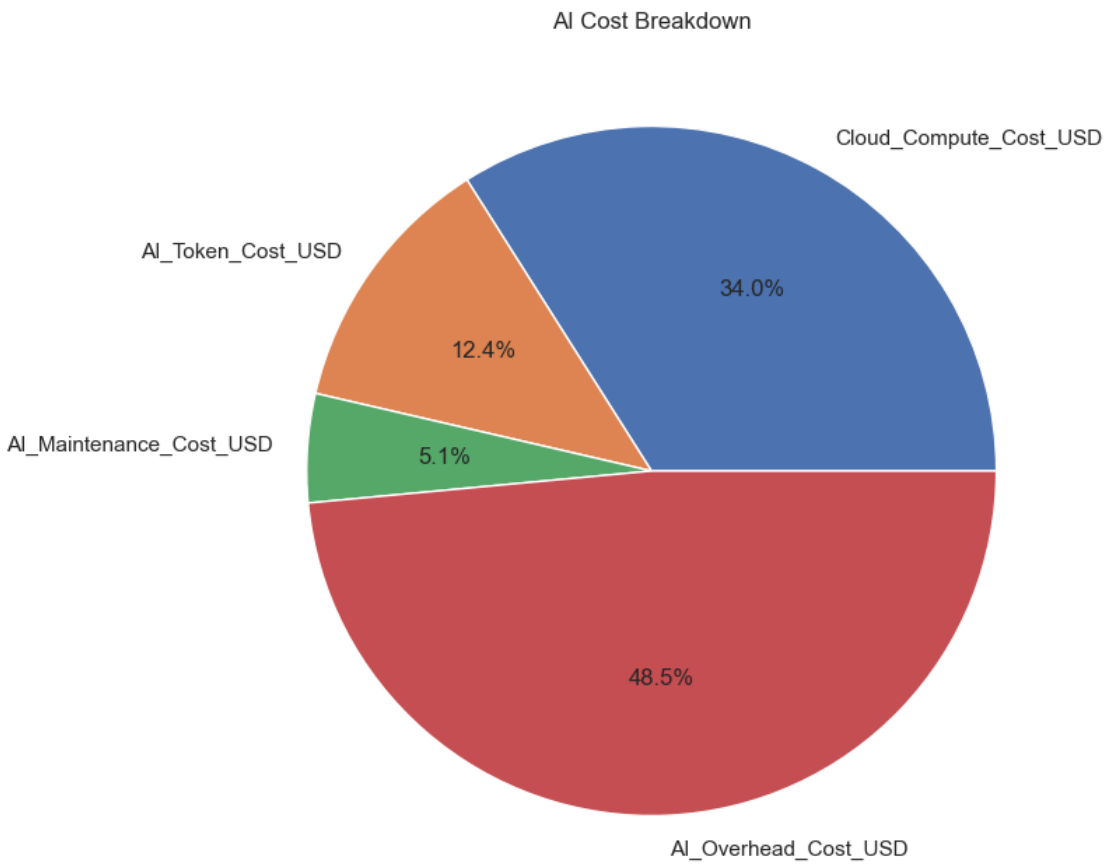
```
                   Fuel_Savings_%  Cost_Per_Shipment_USD
Distance_Category
Short-Haul               9.937966            2763.517108
Medium-Haul              9.932553            2803.690899
Long-Haul               10.074496            2712.705037
```

```
/var/folders/yl/j8gt9m_d6h17m98y6fcd7whw0000gn/T/ipykernel_56452/1452154711.py:3
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  distance_analysis =
df[df["AI_Optimized"]].groupby("Distance_Category")[["Fuel_Savings_%",
"Cost_Per_Shipment_USD"]].mean()
```



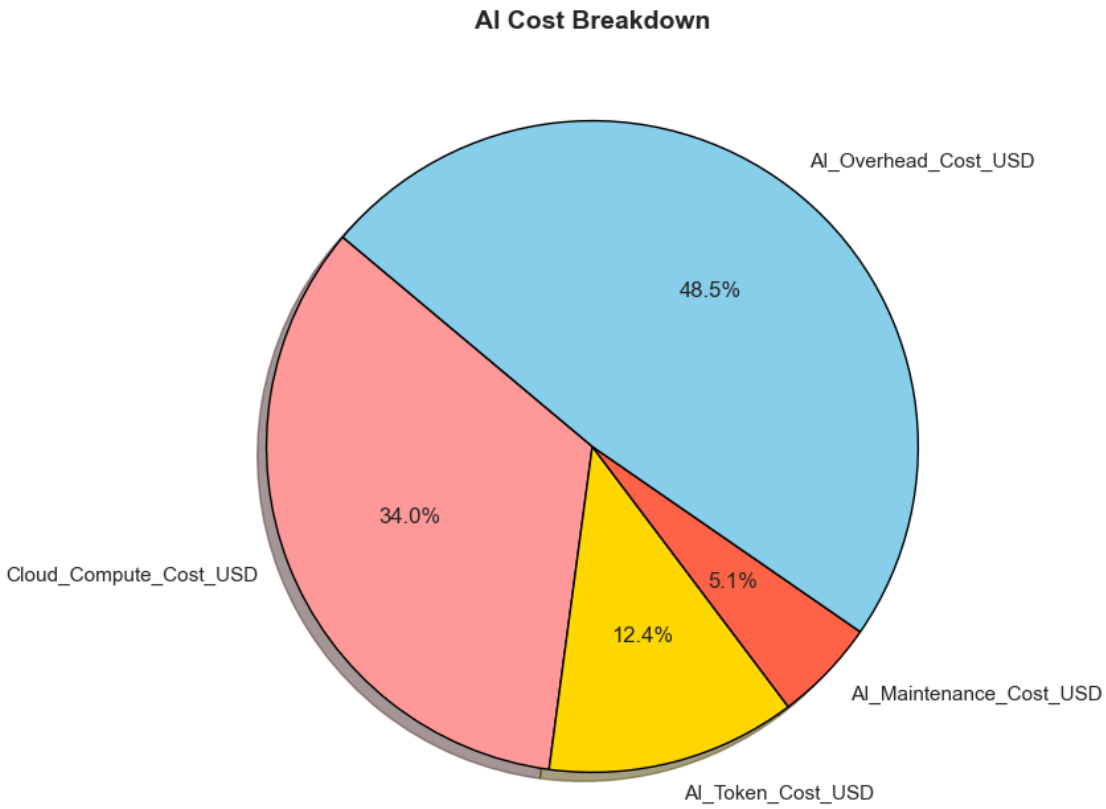
[25]:
```
ai_cost_breakdown = df[df["AI_Optimized"]][["Cloud_Compute_Cost_USD",␣
  ↪"AI_Token_Cost_USD", "AI_Maintenance_Cost_USD", "AI_Overhead_Cost_USD"]].
  ↪mean()
print(ai_cost_breakdown)

# Pie chart for AI cost breakdown
ai_cost_breakdown.plot(kind="pie", autopct="%1.1f%%", figsize=(8, 8), title="AI␣
  ↪Cost Breakdown")
plt.ylabel("")
plt.show()
```

```
Cloud_Compute_Cost_USD     105.089394
AI_Token_Cost_USD           38.310540
AI_Maintenance_Cost_USD     15.763409
AI_Overhead_Cost_USD       150.000000
dtype: float64
```



[27]:
```
import matplotlib.pyplot as plt

# Define improved color palette
colors = ["#FF9999", "#FFD700", "#FF6347", "#87CEEB"]  # Soft red, gold, coral, sky blue

# Create pie chart with enhanced visuals
plt.figure(figsize=(8, 8))
ai_cost_breakdown.plot(
    kind="pie",
    autopct="%1.1f%%",
    colors=colors,
    startangle=140,
    shadow=True,
    wedgeprops={"edgecolor": "black", "linewidth": 1}  # Enhances segment separation
)
```

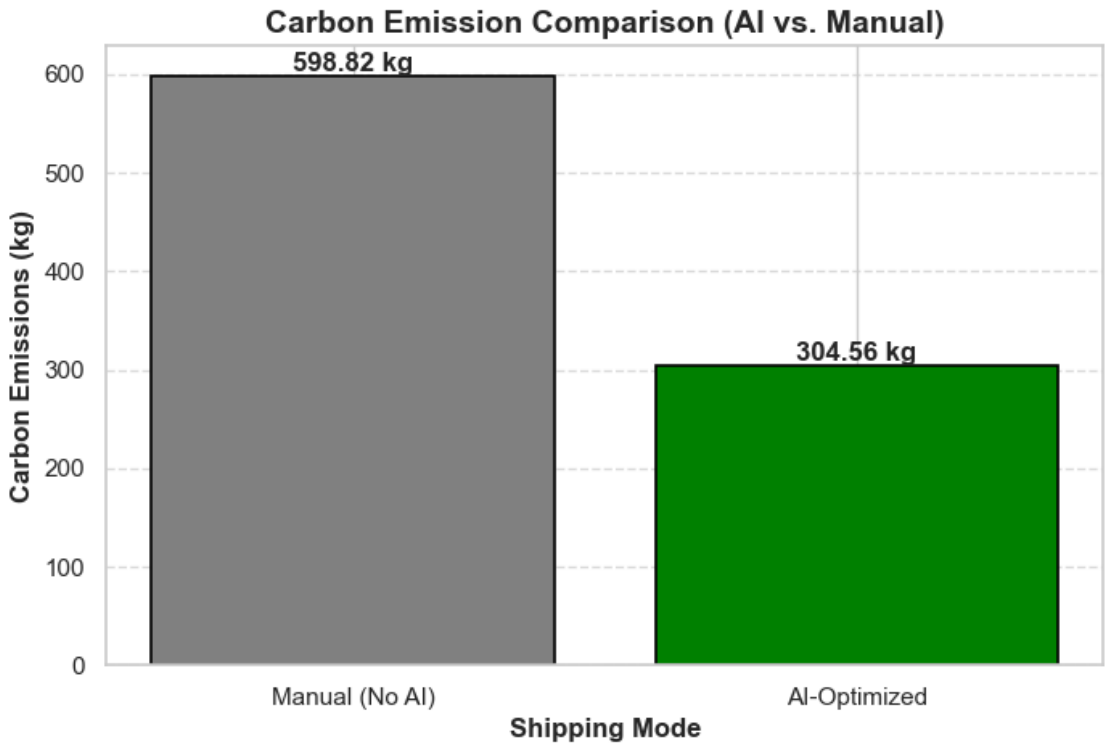
```
# Formatting
plt.title("AI Cost Breakdown", fontsize=14, fontweight="bold")
plt.ylabel("")  # Remove y-label for clean look
plt.show()
```



[35]:
```
import matplotlib.pyplot as plt

# Group by AI optimization and calculate average carbon emissions
carbon_impact = df.groupby("AI_Optimized")[["Carbon_Emissions_kg"]].mean()
print(carbon_impact)

# Define colors: Gray for manual, Green for AI-optimized
colors = ["gray", "green"]

# Create bar chart with enhanced visuals
plt.figure(figsize=(8, 5))
bars = plt.bar(carbon_impact.index.map({False: "Manual (No AI)", True:␣
 ↪"AI-Optimized"}),
```

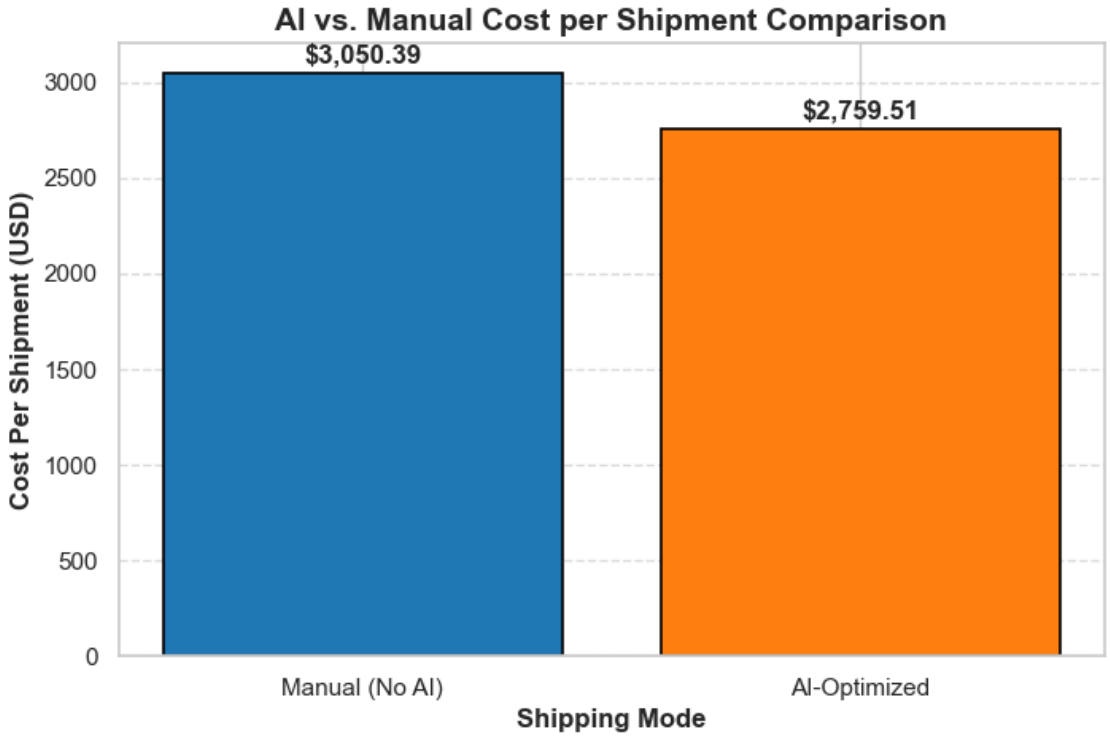
```
                carbon_impact["Carbon_Emissions_kg"], color=colors,␣
 ↪edgecolor="black", linewidth=1.2)

# Add value labels on top of bars
for bar in bars:
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 5,  # Adjust positioning
        f"{bar.get_height():,.2f} kg",
        ha="center", fontsize=12, fontweight="bold"
    )

# Formatting
plt.xlabel("Shipping Mode", fontsize=12, fontweight="bold")
plt.ylabel("Carbon Emissions (kg)", fontsize=12, fontweight="bold")
plt.title("Carbon Emission Comparison (AI vs. Manual)", fontsize=14,␣
 ↪fontweight="bold")
plt.grid(axis="y", linestyle="--", alpha=0.7)  # Add horizontal grid for clarity
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.show()
```

```
              Carbon_Emissions_kg
AI_Optimized
False                  598.823971
True                   304.556000
```



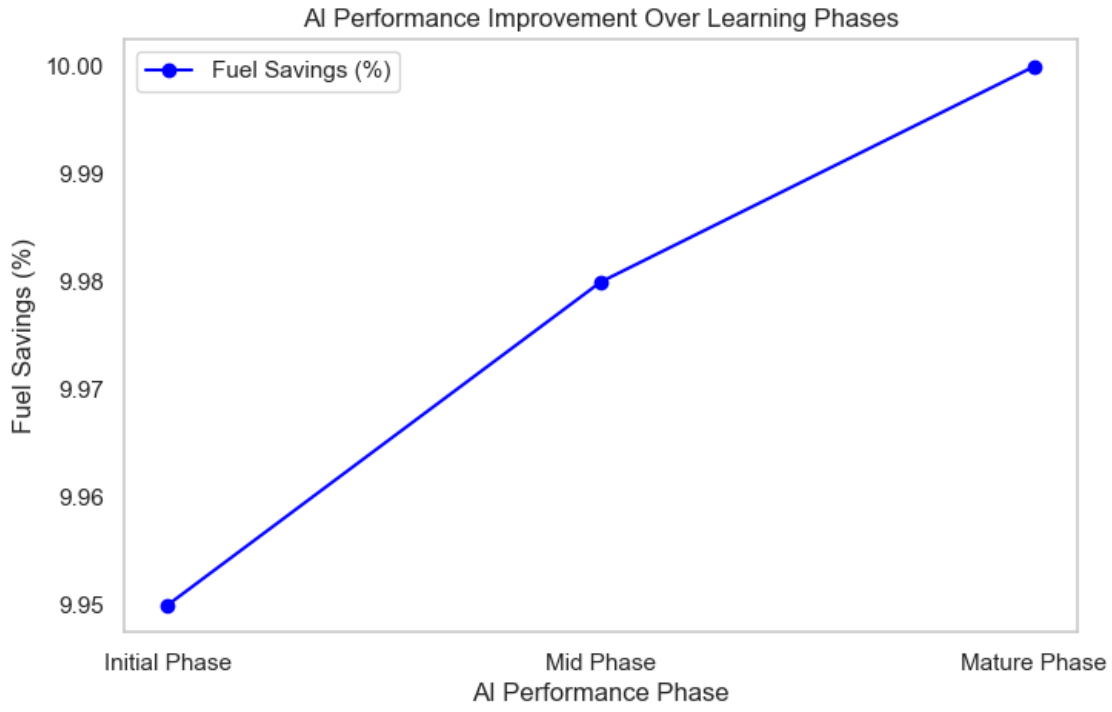
[33]:
```
import matplotlib.pyplot as plt

# Define cost values for AI vs. Manual processing
ai_vs_manual_costs = {
    "Manual (No AI)": df[df["AI_Optimized"] == False]["Cost_Per_Shipment_USD"].
 ↪mean(),
    "AI-Optimized": df[df["AI_Optimized"] == True]["Cost_Per_Shipment_USD"].
 ↪mean()
}

# Define colors (using a more accessible palette)
colors = ["#1f77b4", "#ff7f0e"]  # Blue for Manual, Orange for AI-Optimized

# Create bar chart with enhanced visuals
plt.figure(figsize=(8, 5))
bars = plt.bar(ai_vs_manual_costs.keys(), ai_vs_manual_costs.values(),␣
 ↪color=colors, edgecolor="black", linewidth=1.2)

# Add value labels on top of bars
for bar in bars:
    plt.text(
        bar.get_x() + bar.get_width() / 2,
```

```
        bar.get_height() + 50,  # Adjust positioning
        f"${bar.get_height():,.2f}",
        ha="center", fontsize=12, fontweight="bold"
    )

# Formatting
plt.xlabel("Shipping Mode", fontsize=12, fontweight="bold")
plt.ylabel("Cost Per Shipment (USD)", fontsize=12, fontweight="bold")
plt.title("AI vs. Manual Cost per Shipment Comparison", fontsize=14,␣
 ↪fontweight="bold")
plt.grid(axis="y", linestyle="--", alpha=0.7)  # Add horizontal grid for clarity
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.show()
```



[68]:
```
# AI Learning Curve Impact (Fuel Savings % Over AI Phases)
learning_phases = ["Initial Phase", "Mid Phase", "Mature Phase"]
fuel_savings = [9.95, 9.98, 10.00]

plt.figure(figsize=(8, 5))
plt.plot(learning_phases, fuel_savings, marker="o", linestyle="-",␣
 ↪color="blue", label="Fuel Savings (%)")
plt.xlabel("AI Performance Phase")
```

```
plt.ylabel("Fuel Savings (%)")
plt.title("AI Performance Improvement Over Learning Phases")
plt.legend()
plt.grid()
plt.show()
```



[70]:
```
##We need all the variables in a category from int, or float or dummies, so␣
↪lets convert it
```

[72]:
```
df["Shipment_Type"].value_counts()
df["Shipment_Priority"].value_counts()
df["Distance_Category"].value_counts()
df["AI_Performance_Phase"].value_counts()
```

[72]:
```
AI_Performance_Phase
Mid Phase        1981
Mature Phase     1516
Initial Phase    1503
Name: count, dtype: int64
```

[74]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

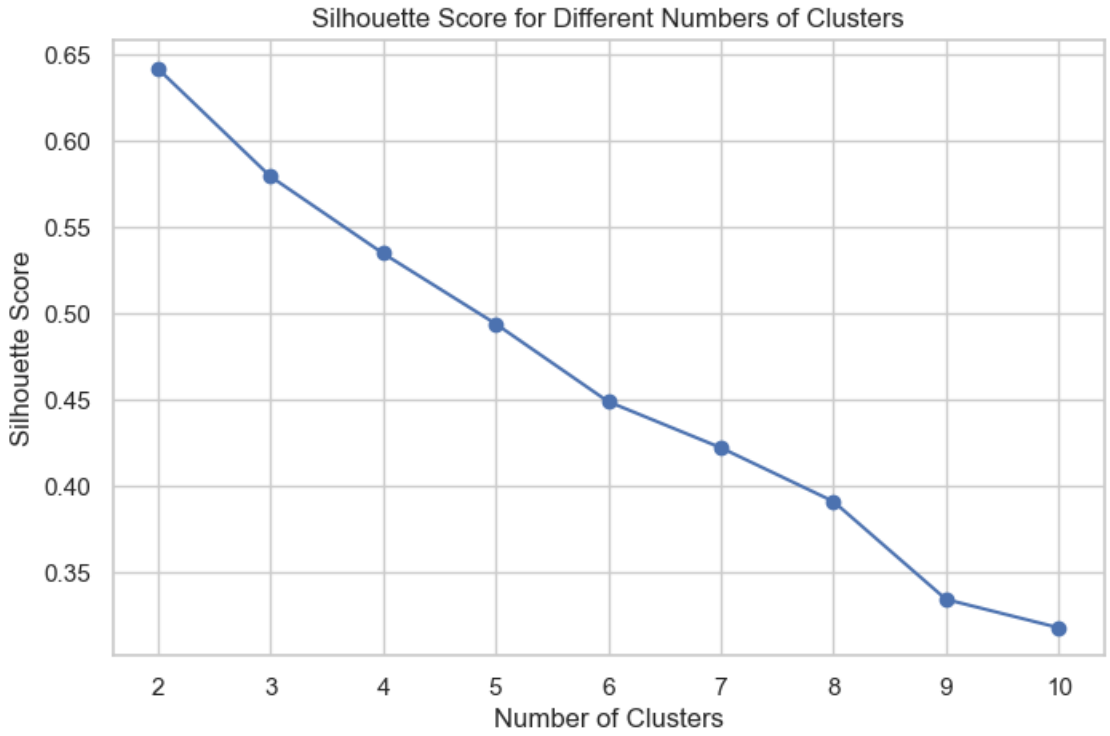
```
import matplotlib.pyplot as plt

# Assuming 'df' is already loaded into your environment

# Drop irrelevant columns
df_ml = df.drop(columns=['Shipment_ID', 'Origin_Hub', 'Destination_Hub'])

# Create dummy variables for categorical features
df_ml = pd.get_dummies(df_ml, columns=['Shipment_Type', 'Shipment_Priority',␣
 ↪'Distance_Category', 'AI_Performance_Phase'], drop_first=True)
```

[76]:
```
df_ml.to_csv('df_ml.csv', index=False)
df_ml_yes = df_ml[df_ml['AI_Optimized'] == True]
df_ml_yes.to_csv('df_ml_yes.csv', index=False)
df_ml.to_csv(r"/Users/anshpatel/Desktop/df_ml.csv", index=False)
df_ml_yes.to_csv(r"/Users/anshpatel/Desktop/df_ml_yes.csv", index=False)
```

[78]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Assuming df_ml is already preprocessed and ready for clustering

silhouette_scores = []
cluster_range = range(2, 11)  # Testing cluster counts from 2 to 10

for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(df_ml)
    silhouette_avg = silhouette_score(df_ml, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    print(f"For n_clusters = {n_clusters}, the average silhouette_score is␣
 ↪{silhouette_avg}")

# Plotting the silhouette scores
plt.figure(figsize=(8, 5))
plt.plot(cluster_range, silhouette_scores, marker='o')
plt.title("Silhouette Score for Different Numbers of Clusters")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.show()
```

```
For n_clusters = 2, the average silhouette_score is 0.6422623459612684
For n_clusters = 3, the average silhouette_score is 0.579640094177112
For n_clusters = 4, the average silhouette_score is 0.5350157599132018
For n_clusters = 5, the average silhouette_score is 0.49415721114764893
For n_clusters = 6, the average silhouette_score is 0.4489833948072135
For n_clusters = 7, the average silhouette_score is 0.42223118536579685
```

```
For n_clusters = 8, the average silhouette_score is 0.39105722756865224
For n_clusters = 9, the average silhouette_score is 0.33426569738615963
For n_clusters = 10, the average silhouette_score is 0.31783239052818346
```

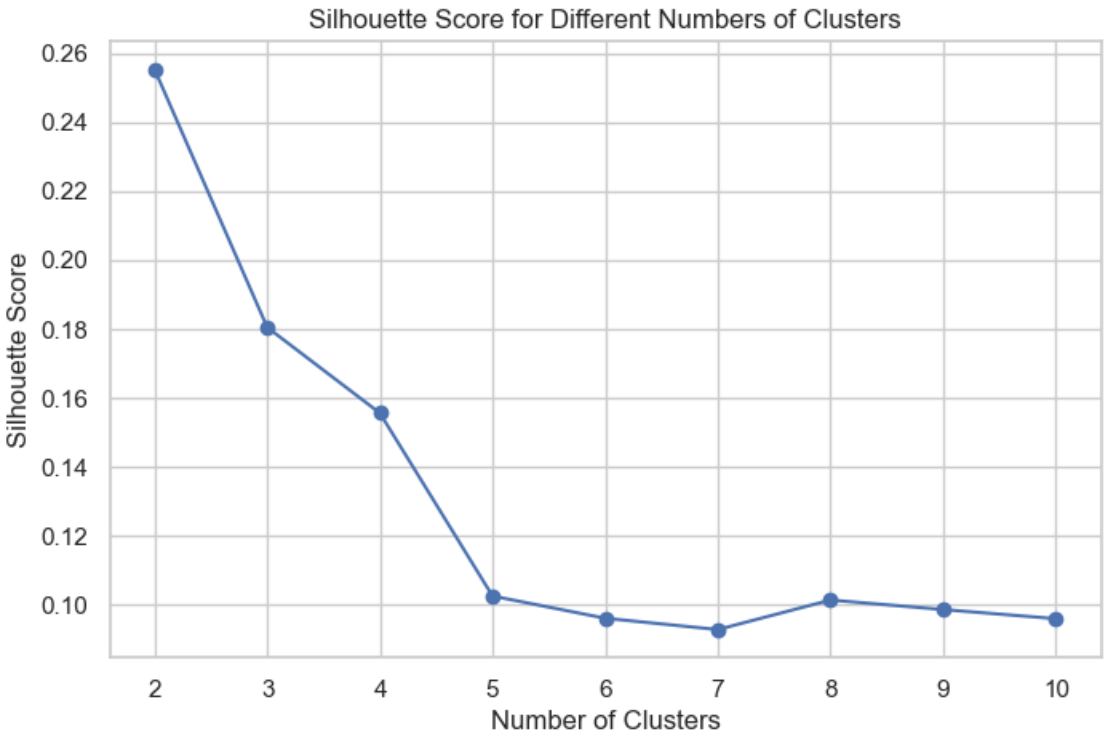


[79]:
```python
# Filter for AI_Optimized == True
df_ml_yes = df_ml[df_ml['AI_Optimized'] == True]

# Drop AI_Optimized column as it is now redundant
df_ml = df_ml.drop(columns=['AI_Optimized'])

# Check data types
print(df_ml.dtypes)

# Scale the dataset for clustering
scaler = StandardScaler()
df_ml_scaled = scaler.fit_transform(df_ml)

# Silhouette scores for different cluster numbers
silhouette_scores = []
cluster_range = range(2, 11)

for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
```

```
    cluster_labels = kmeans.fit_predict(df_ml_scaled)
    silhouette_avg = silhouette_score(df_ml_scaled, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    print(f"For n_clusters = {n_clusters}, the average silhouette_score is␣
 ↪{silhouette_avg}")

# Plotting the silhouette scores
plt.figure(figsize=(8, 5))
plt.plot(cluster_range, silhouette_scores, marker='o')
plt.title("Silhouette Score for Different Numbers of Clusters")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.show()
```

```
Shipment_Weight_kg                  float64
Shipment_Volume_m3                  float64
AI_Processing_Time_Sec              float64
Loading_Time_min                    float64
Fuel_Savings_%                      float64
Error_Rate_%                        float64
AI_Token_Usage                        int64
Cost_Per_Shipment_USD               float64
Carbon_Emissions_kg                 float64
Shipment_Distance_km                float64
Cloud_Compute_Cost_USD              float64
AI_Token_Cost_USD                   float64
AI_Maintenance_Cost_USD             float64
AI_Savings_USD                      float64
AI_Overhead_Cost_USD                  int64
Total_AI_Cost_USD                   float64
Shipment_Type_Hazardous                bool
Shipment_Type_Perishable               bool
Shipment_Type_Standard                 bool
Shipment_Priority_Low                  bool
Shipment_Priority_Medium               bool
Distance_Category_Medium-Haul          bool
Distance_Category_Long-Haul            bool
AI_Performance_Phase_Mature Phase      bool
AI_Performance_Phase_Mid Phase         bool
dtype: object
For n_clusters = 2, the average silhouette_score is 0.25534721222777473
For n_clusters = 3, the average silhouette_score is 0.18048707030360905
For n_clusters = 4, the average silhouette_score is 0.1556039949495653
For n_clusters = 5, the average silhouette_score is 0.10250482061424948
For n_clusters = 6, the average silhouette_score is 0.09605055969187977
For n_clusters = 7, the average silhouette_score is 0.092764010315595
For n_clusters = 8, the average silhouette_score is 0.1013351251970254
For n_clusters = 9, the average silhouette_score is 0.09857658979568988
```

```
For n_clusters = 10, the average silhouette_score is 0.09592568541591452
```



[80]:
```
pip install pyclustering
```

```
Requirement already satisfied: pyclustering in
/opt/anaconda3/lib/python3.12/site-packages (0.10.1.2)
Requirement already satisfied: scipy>=1.1.0 in
/opt/anaconda3/lib/python3.12/site-packages (from pyclustering) (1.13.1)
Requirement already satisfied: matplotlib>=3.0.0 in
/opt/anaconda3/lib/python3.12/site-packages (from pyclustering) (3.8.4)
Requirement already satisfied: numpy>=1.15.2 in
/opt/anaconda3/lib/python3.12/site-packages (from pyclustering) (1.26.4)
Requirement already satisfied: Pillow>=5.2.0 in
/opt/anaconda3/lib/python3.12/site-packages (from pyclustering) (10.3.0)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->pyclustering) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
/opt/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->pyclustering) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->pyclustering) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
```

```
/opt/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->pyclustering) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->pyclustering) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->pyclustering) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->pyclustering) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib>=3.0.0->pyclustering) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[ ]: