In [42]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [43]:
```python
# Loading the certificate summary data set and viewing top 10 rows of dataframe

df_cert = pd.read_excel(r'C:\Users\mdyoung\OneDrive - Bankers Financial Corporation\Desktop\Bellevue\DSC680_T3(
df_cert.head(10)
```

Out[43]:

	CertificateNumber	WarrantyEnrollmentAppID	ClosingDate	ActivatedAt	FormattedPurchasePrice	BuilderNumber	Company Name	BuilderApprova
0	W10657342	657342	2023-06-02 05:00:00	2024-02-28 15:27:10.735	1600000.00	23093	11/11 Developers LLC	2023-
1	1113400	685163	2023-03-27 00:00:00	2023-05-30 00:00:00.000	1450000.00	23093	11/11 Developers LLC	2023-
2	W10701958	701958	2024-02-27 06:00:00	2024-03-14 21:09:02.635	1555000.00	23093	11/11 Developers LLC	2023-
3	1104252	674932	2022-12-09 00:00:00	2023-01-09 00:00:00.000	345000.00	22253	11/11 Developers LLC	2022-
4	W10701820	701820	2024-02-28 06:00:00	2024-03-21 14:05:14.343	367000.00	22253	11/11 Developers LLC	2022-
5	1094094	313666	2022-05-15 00:00:00	2022-09-21 00:00:00.000	310063.00	7627	First Choice Home Builders	2007-
6	1103435	313668	2022-09-12 00:00:00	2022-12-28 00:00:00.000	224817.00	7627	First Choice Home Builders	2007-
7	1098826	426337	2022-07-29 00:00:00	2022-11-07 00:00:00.000	217785.00	7627	First Choice Home Builders	2007-
8	1094291	448835	2022-05-15 00:00:00	2022-09-21 00:00:00.000	666580.00	7627	First Choice Home Builders	2007-
9	1098808	458452	2022-07-15 00:00:00	2022-11-07 00:00:00.000	375569.75	7627	First Choice Home Builders	2007-

In [44]:
```python
# Viewing summary of certificate data

df_cert.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40191 entries, 0 to 40190
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CertificateNumber       40191 non-null  object
 1   WarrantyEnrollmentAppID 40191 non-null  int64
 2   ClosingDate             39901 non-null  datetime64[ns]
 3   ActivatedAt             40191 non-null  datetime64[ns]
 4   FormattedPurchasePrice  40115 non-null  float64
 5   BuilderNumber           40191 non-null  int64
 6   Company Name            40191 non-null  object
 7   BuilderApprovalDate     40175 non-null  datetime64[ns]
 8   StateId                 40191 non-null  object
 9   ZipCode                 40191 non-null  object
 10  County                  40185 non-null  object
 11  ProductSegment          40191 non-null  object
 12  Premium                 40191 non-null  int64
dtypes: datetime64[ns](3), float64(1), int64(3), object(6)
memory usage: 4.0+ MB
```

In [45]:
```python
# Viewing stat summary for certificate data

df_cert.describe().T
```

Out[45]:

	count	mean	min	25%	50%	75%	max	std
WarrantyEnrollmentAppID	40191.0	631823.399592	305774.0	654932.5	679809.0	690461.5	706396.0	103170.935688
ClosingDate	39901	2023-05-07 18:09:41.038069248	2001-05-25 04:00:00	2022-12-21 00:00:00	2023-05-05 00:00:00	2023-09-15 04:00:00	2026-06-28 00:00:00	NaN
ActivatedAt	40191	2023-06-08 07:23:44.749024512	2022-09-06 00:00:00	2023-01-18 00:00:00	2023-05-31 00:00:00	2023-10-23 19:35:12.892499968	2024-03-22 21:31:29.272000	NaN
FormattedPurchasePrice	40115.0	476463.426457	0.0	327022.0	405000.0	539414.5	16518000.0	278596.323896
BuilderNumber	40191.0	51565899.184096	124.0	13104.0	16439.0	22086.0	2300000018115.0	3441449294.095469
BuilderApprovalDate	40175	2011-11-30 23:53:45.797137664	1993-04-20 00:00:00	2008-02-20 00:00:00	2014-10-13 00:00:00	2018-03-05 00:00:00	2024-03-20 00:00:00	NaN
Premium	40191.0	540.616108	0.0	173.0	341.0	755.0	14350.0	619.101711

In [46]:
```python
# Loading the CRM data set and viewing dataframe.

df_crm = pd.read_excel(r'C:\Users\mdyoung\OneDrive - Bankers Financial Corporation\Desktop\Bellevue\DSC680_T30l
df_crm.head(10)
```

Out[46]:

	Company Name	Company Id	Year of Createdate	Month of Createdate	Pod
0	Longo Homes INC	9505516456	2022	September	2
1	Heritage Crafted Homes - JBB Ventures, LLC	9507836967	2022	September	1
2	Hempstead Premier Homes	9512394395	2022	September	3
3	Extra Mile Properties LLC	9512480614	2022	September	4
4	Evolve Homebuilders LLC	9512578183	2022	September	4
5	Group 3 Builders	9513840902	2022	September	1
6	Gryphyn Custom Homes	9514062734	2022	September	1
7	Chambliss Homes Inc	9533992089	2022	September	2
8	Charlie Sanders Lumber & Construction	9533992259	2022	September	2
9	Delta Traditions	9533992532	2022	September	2

In [47]:
```python
# Viewing summary of crm data

df_crm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3062 entries, 0 to 3061
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Company Name         2598 non-null   object
 1   Company Id           3062 non-null   int64
 2   Year of Createdate   3062 non-null   int64
 3   Month of Createdate  3062 non-null   object
 4   Pod                  3062 non-null   int64
dtypes: int64(3), object(2)
memory usage: 119.7+ KB
```

In [48]:
```python
# Viewing stat summary for crm data

df_crm.describe().T
```

Out[48]:

	count	mean	std	min	25%	50%	75%	max
Company Id	3062.0	1.494679e+10	3.000597e+09	9.505516e+09	1.234594e+10	1.574674e+10	1.752934e+10	1.856221e+10
Year of Createdate	3062.0	2.022816e+03	3.874388e-01	2.022000e+03	2.023000e+03	2.023000e+03	2.023000e+03	2.023000e+03
Pod	3062.0	2.661659e+00	8.956461e-01	1.000000e+00	2.000000e+00	2.000000e+00	3.000000e+00	4.000000e+00

In [49]:
```python
# Initial cleaning steps (e.g., dropping duplicates, handling missing values)
df_cert.drop_duplicates(inplace=True)
df_crm.drop_duplicates(inplace=True)
```

In [50]:
```python
df_cert.shape
```

Out[50]: (40191, 13)

In [51]:
```python
df_crm.shape
```

Out[51]: (3062, 5)

Both data frames remain the same after running the drop duplicates code.

First, we'll normalize the "Company Name" fields in both datasets to standardize their format. This involves converting to lowercase, removing extra spaces, and eliminating special characters.

In [52]:
```python
import re

# Updated normalization function
def normalize_company_name(name):
    # Convert any non-string input to string
    name = str(name)
    name = name.lower()  # Convert to lowercase
    name = re.sub(r'\s+', ' ', name)  # Replace multiple spaces with a single space
    name = re.sub(r'[^\w\s]', '', name)  # Remove special characters
    name = name.strip()  # Trim whitespace from the ends
    # Handle the case where original name was NaN (converted to 'nan' by str())
    if name == 'nan':
        return None  # Return None or an empty string '' as appropriate
    return name

# Apply the normalization function to the "Company Name" columns in both datasets
df_cert['Company Name'] = df_cert['Company Name'].apply(normalize_company_name)
df_crm['Company Name'] = df_crm['Company Name'].apply(normalize_company_name)

# Proceed with a direct merge/join
merged_df = pd.merge(df_cert, df_crm, on='Company Name', how='inner')
```

In [53]:
```python
merged_df.shape
```

Out[53]: (3041, 17)

The direct merge after normalization still leaves many unmatched records and I will proceed with fuzzy matching.

In [54]:
```python
from fuzzywuzzy import process

def fuzzy_merge(df_left, df_right, key_left, key_right, threshold=90, limit=2):
    # Create a dictionary to hold the best matches
    match_dict = {}
    # Get a list of unique company names from the right DataFrame
    right_names = df_right[key_right].unique()

    # For each unique name in the left DataFrame, find the best match in the right DataFrame
    for name in df_left[key_left].unique():
        matches = process.extract(name, right_names, limit=limit)
        best_match = [match for match in matches if match[1] >= threshold]
        if best_match:
            match_dict[name] = best_match[0][0]

    # Map the matches based on the dictionary
    df_left[f'{key_left}_matched'] = df_left[key_left].map(match_dict)

    # Perform the merge
    return pd.merge(df_left, df_right, left_on=f'{key_left}_matched', right_on=key_right, how='left')

# Apply fuzzy matching and merge
merged_df_fuzzy = fuzzy_merge(df_cert, df_crm, 'Company Name', 'Company Name')
```

In [55]:
```python
merged_df_fuzzy.shape
```

Out[55]: (15413236, 19)

Removing Duplicates Based on a Subset of Columns

Often, especially after fuzzy matching, you'll want to remove duplicates based on specific criteria. For example, you might consider a row a duplicate if the same Company Name from df_cert matches multiple entries in df_crm, and you only want to keep the match with the highest confidence score.

First, sort the DataFrame based on some criteria, such as a match score or another metric indicating match quality. If you don't have a match score, you might sort by another relevant column. Assuming match_score indicates the confidence of each match:

In [56]:
```python
# Keeping the first occurrence of each CertificateNumber
merged_df_fuzzy_unique = merged_df_fuzzy.drop_duplicates(subset=['CertificateNumber'], keep='first')
```

In [57]:
```python
merged_df_fuzzy_unique.shape
```

Out[57]: (40191, 19)

In [58]:
```python
merged_df_fuzzy_unique.head(10)
```

Out[58]:

	CertificateNumber	WarrantyEnrollmentAppID	ClosingDate	ActivatedAt	FormattedPurchasePrice	BuilderNumber	Company Name_x	BuilderApproval
0	W10657342	657342	2023-06-02 05:00:00	2024-02-28 15:27:10.735	1600000.00	23093	1111 developers llc	2023-(
1	1113400	685163	2023-03-27 00:00:00	2023-05-30 00:00:00.000	1450000.00	23093	1111 developers llc	2023-(
2	W10701958	701958	2024-02-27 06:00:00	2024-03-14 21:09:02.635	1555000.00	23093	1111 developers llc	2023-(
3	1104252	674932	2022-12-09 00:00:00	2023-01-09 00:00:00.000	345000.00	22253	1111 developers llc	2022-'
4	W10701820	701820	2024-02-28 06:00:00	2024-03-21 14:05:14.343	367000.00	22253	1111 developers llc	2022-'
5	1094094	313666	2022-05-15 00:00:00	2022-09-21 00:00:00.000	310063.00	7627	first choice home builders	2007-(
6	1103435	313668	2022-09-12 00:00:00	2022-12-28 00:00:00.000	224817.00	7627	first choice home builders	2007-(
7	1098826	426337	2022-07-29 00:00:00	2022-11-07 00:00:00.000	217785.00	7627	first choice home builders	2007-(
8	1094291	448835	2022-05-15 00:00:00	2022-09-21 00:00:00.000	666580.00	7627	first choice home builders	2007-(
9	1098808	458452	2022-07-15 00:00:00	2022-11-07 00:00:00.000	375569.75	7627	first choice home builders	2007-(

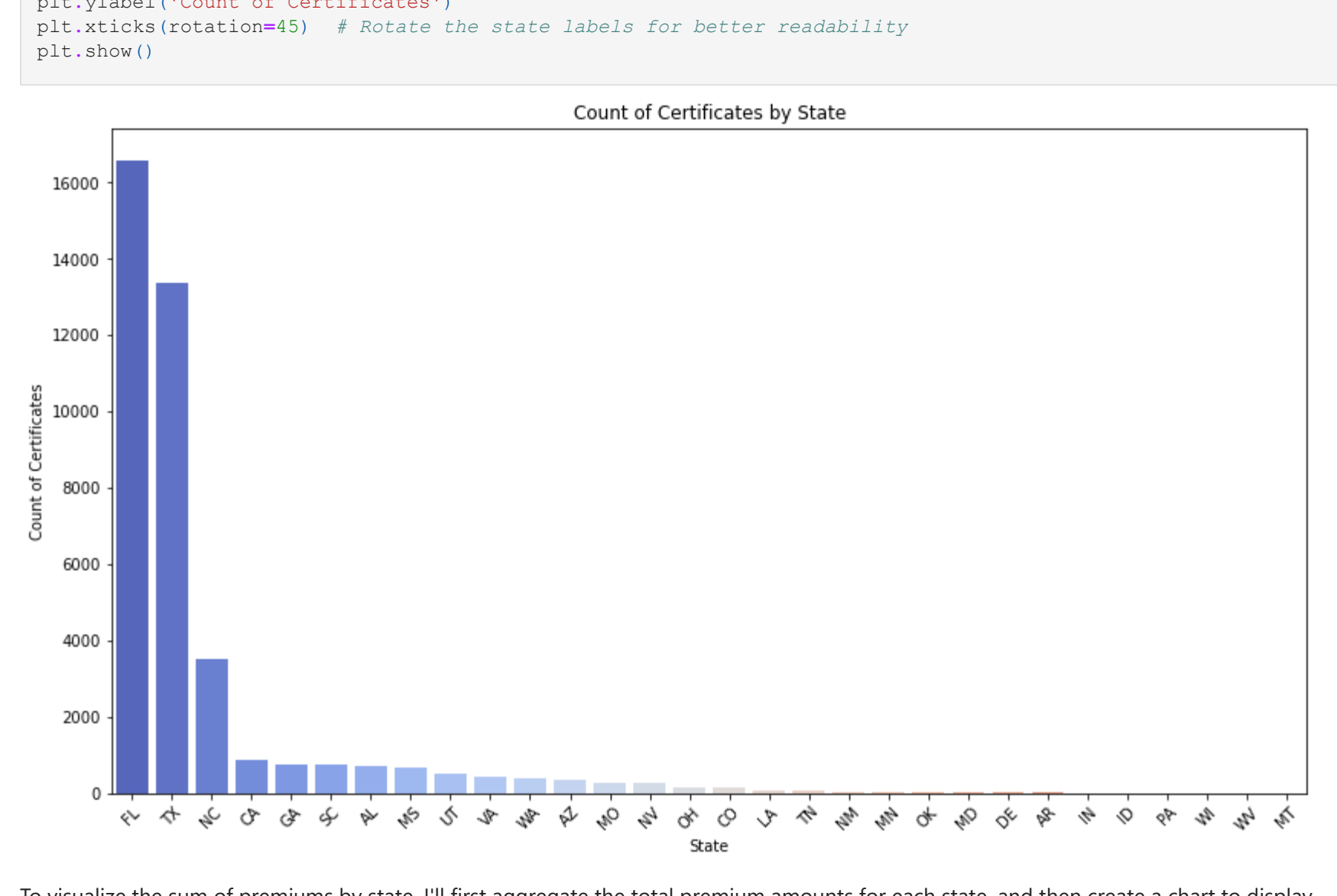
In [59]:
```python
merged_df_fuzzy_unique.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 40191 entries, 0 to 15412772
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CertificateNumber       40191 non-null  object
 1   WarrantyEnrollmentAppID 40191 non-null  int64
 2   ClosingDate             39901 non-null  datetime64[ns]
 3   ActivatedAt             40191 non-null  datetime64[ns]
 4   FormattedPurchasePrice  40115 non-null  float64
 5   BuilderNumber           40191 non-null  int64
 6   Company Name_x          40191 non-null  object
 7   BuilderApprovalDate     40175 non-null  datetime64[ns]
 8   StateId                 40191 non-null  object
 9   ZipCode                 40191 non-null  object
 10  County                  40185 non-null  object
 11  ProductSegment          40191 non-null  object
 12  Premium                 40191 non-null  int64
 13  Company Name_matched    6988 non-null   object
 14  Company Name_y          6988 non-null   object
 15  Company Id              40191 non-null  int64
 16  Year of Createdate      40191 non-null  int64
 17  Month of Createdate     40191 non-null  object
 18  Pod                     40191 non-null  int64
dtypes: datetime64[ns](3), float64(1), int64(6), object(9)
memory usage: 6.1+ MB
```

I want to aggregate the count of certificates by state. StateId is the column representing states in merged_df_fuzzy_unique:

In [60]:
```python
state_certificate_counts = merged_df_fuzzy_unique['StateId'].value_counts().reset_index()
state_certificate_counts.columns = ['State', 'Count']
state_certificate_counts = state_certificate_counts.sort_values(by='Count', ascending=False)
```

In [61]:
```python
plt.figure(figsize=(14, 8))  # Adjust the figure size as needed
sns.barplot(data=state_certificate_counts, x='State', y='Count', palette='coolwarm')
plt.title('Count of Certificates by State')
plt.xlabel('State')
plt.ylabel('Count of Certificates')
plt.xticks(rotation=45)  # Rotate the state labels for better readability
plt.show()
```

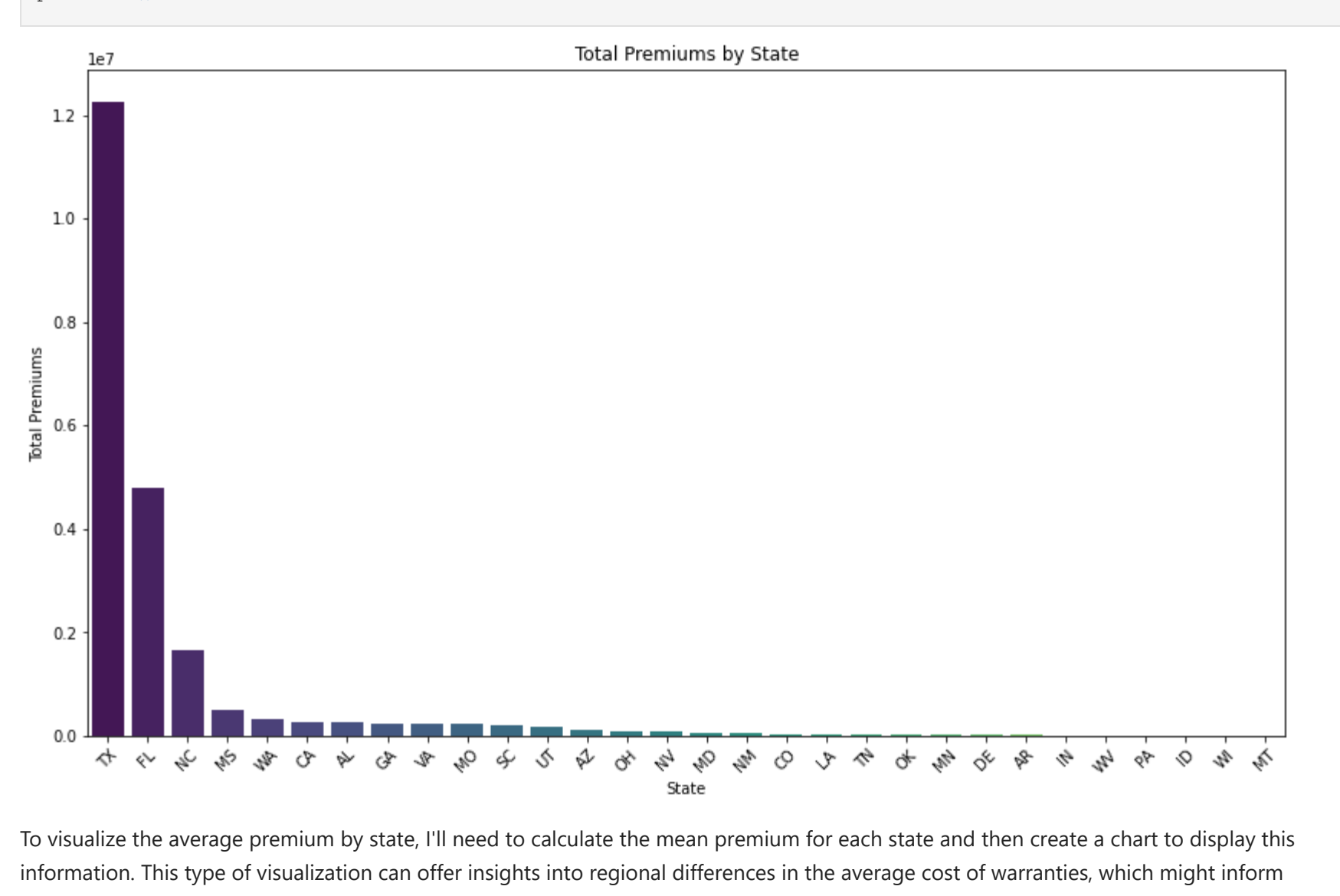


To visualize the sum of premiums by state, I'll first aggregate the total premium amounts for each state, and then create a chart to display this information. This visualization can provide insights into which states generate the most revenue in terms of premiums, potentially informing resource allocation and regional strategies.

First, calculate the sum of premiums for each state. Premium is the column representing the premium amounts in merged_df_fuzzy_unique and StateId represents the states:

In [62]:
```python
premiums_by_state = merged_df_fuzzy_unique.groupby('StateId')['Premium'].sum().reset_index()
premiums_by_state = premiums_by_state.sort_values(by='Premium', ascending=False)
```

In [63]:
```python
plt.figure(figsize=(14, 8))  # Adjust the figure size as needed
sns.barplot(data=premiums_by_state, x='StateId', y='Premium', palette='viridis')
plt.title('Total Premiums by State')
plt.xlabel('State')
plt.ylabel('Total Premiums')
plt.xticks(rotation=45)  # Rotate the state labels for better readability
plt.show()
```

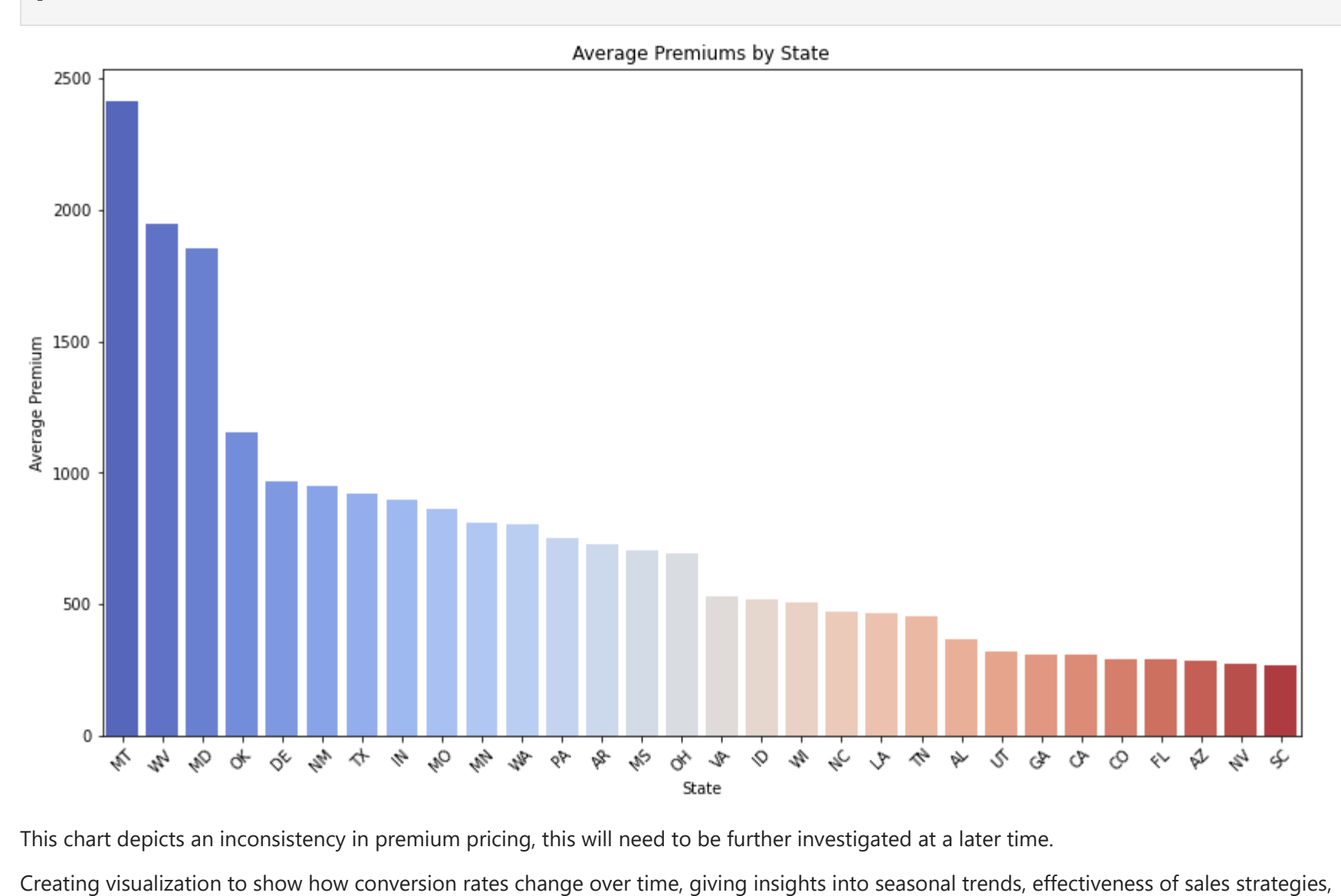


To visualize the average premium by state, I'll need to calculate the mean premium for each state and then create a chart to display this information. This type of visualization can offer insights into regional differences in the average cost of warranties, which might inform pricing strategies or highlight areas with higher or lower than average premiums.

First, calculate the average (mean) premium for each state. Premium is the column for premium amounts and StateId represents the states in merged_df_fuzzy_unique:

In [64]:
```python
average_premiums_by_state = merged_df_fuzzy_unique.groupby('StateId')['Premium'].mean().reset_index()
average_premiums_by_state = average_premiums_by_state.sort_values(by='Premium', ascending=False)
```

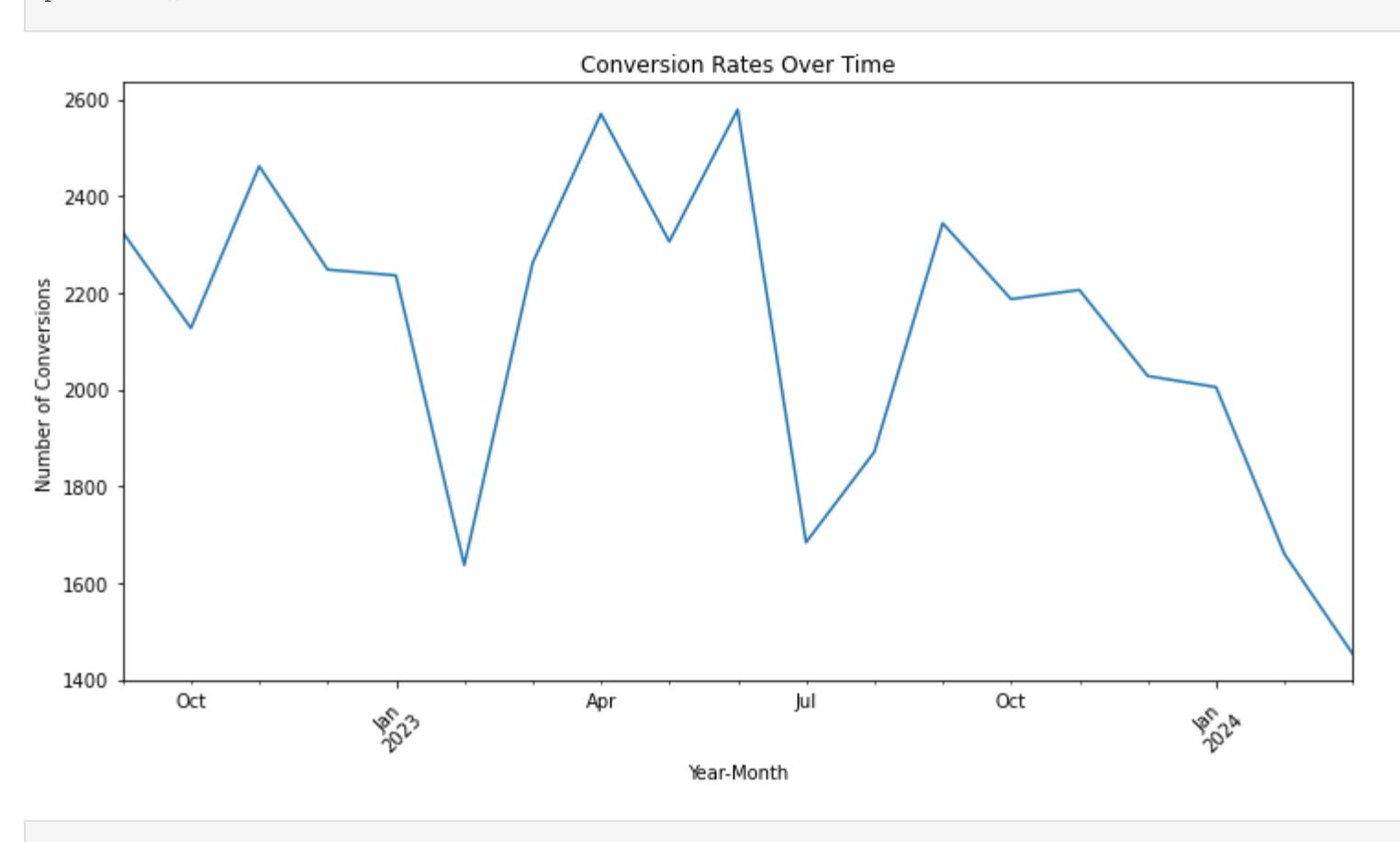
In [65]:
```python
plt.figure(figsize=(14, 8))  # You might need to adjust this size depending on the number of states
sns.barplot(data=average_premiums_by_state, x='StateId', y='Premium', palette='coolwarm')
plt.title('Average Premiums by State')
plt.xlabel('State')
plt.ylabel('Average Premium')
plt.xticks(rotation=45)  # Rotate state labels for better readability
plt.show()
```



This chart depicts an inconsistency in premium pricing, this will need to be further investigated at a later time.

Creating visualization to show how conversion rates change over time, giving insights into seasonal trends, effectiveness of sales strategies, and market dynamics.

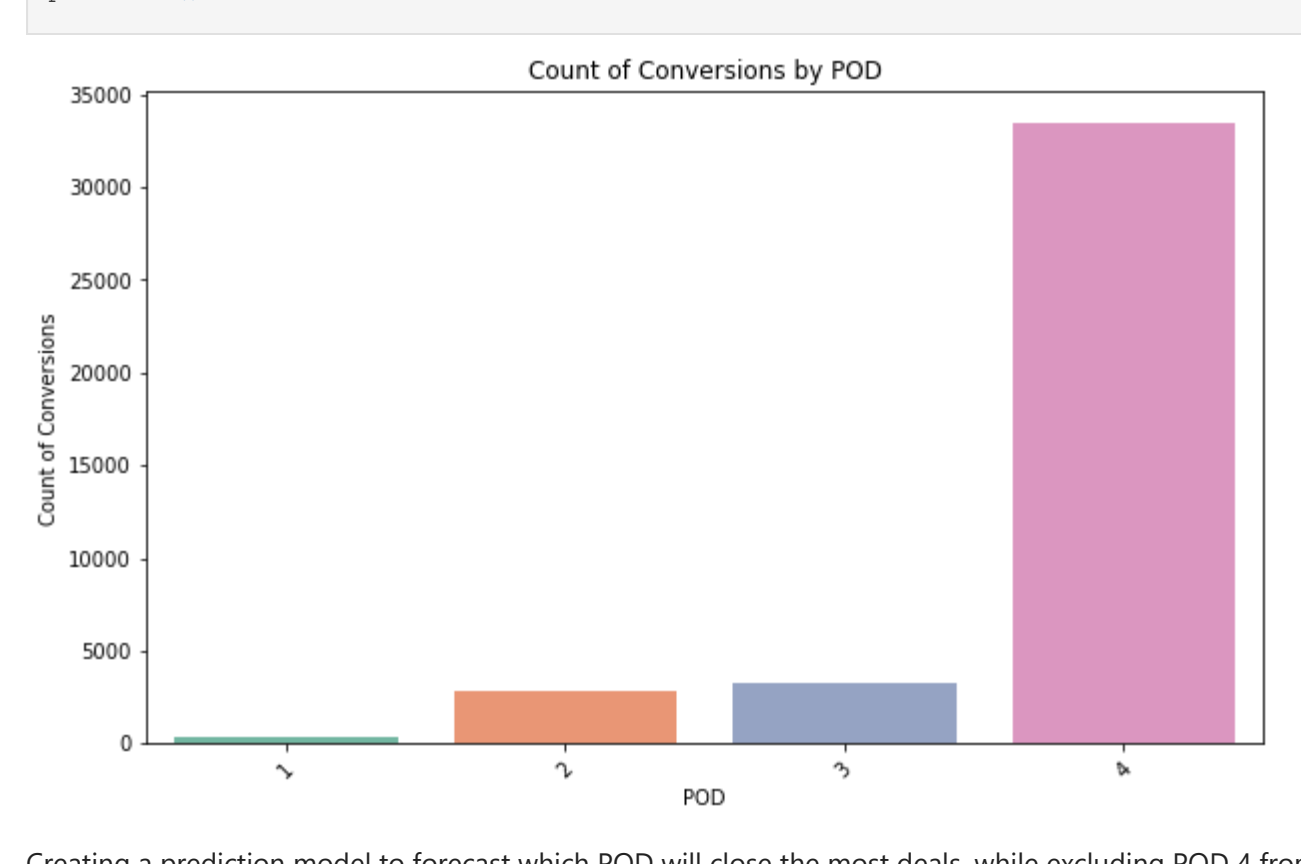
In [66]:
```python
# 'ClosingDate' indicates a successful sale
# First, created a 'SaleYearMonth' column for aggregation
merged_df_fuzzy_unique['SaleYearMonth'] = merged_df_fuzzy_unique['ActivatedAt'].dt.to_period('M')

# Calculate conversion rates by month
conversion_rates = merged_df_fuzzy_unique.groupby('SaleYearMonth').size()

plt.figure(figsize=(12, 6))
conversion_rates.plot(kind='line')
plt.title('Conversion Rates Over Time')
plt.xlabel('Year-Month')
plt.ylabel('Number of Conversions')
plt.xticks(rotation=45)
plt.show()
```



In [67]:
```python
merged_df_fuzzy_unique['IsConverted'] = ~merged_df_fuzzy_unique['ClosingDate'].isnull()
```

In [68]:
```python
# Count the number of conversions by POD
conversions_count_by_pod = merged_df_fuzzy_unique[merged_df_fuzzy_unique['IsConverted']].groupby('Pod').size().
```

In [69]:
```python
plt.figure(figsize=(10, 6))
sns.barplot(data=conversions_count_by_pod, x='Pod', y='Conversions', palette='Set2')
plt.title('Count of Conversions by POD')
plt.xlabel('POD')
plt.ylabel('Count of Conversions')
plt.xticks(rotation=45)  # Adjust the rotation as needed for better label readability
plt.show()
```



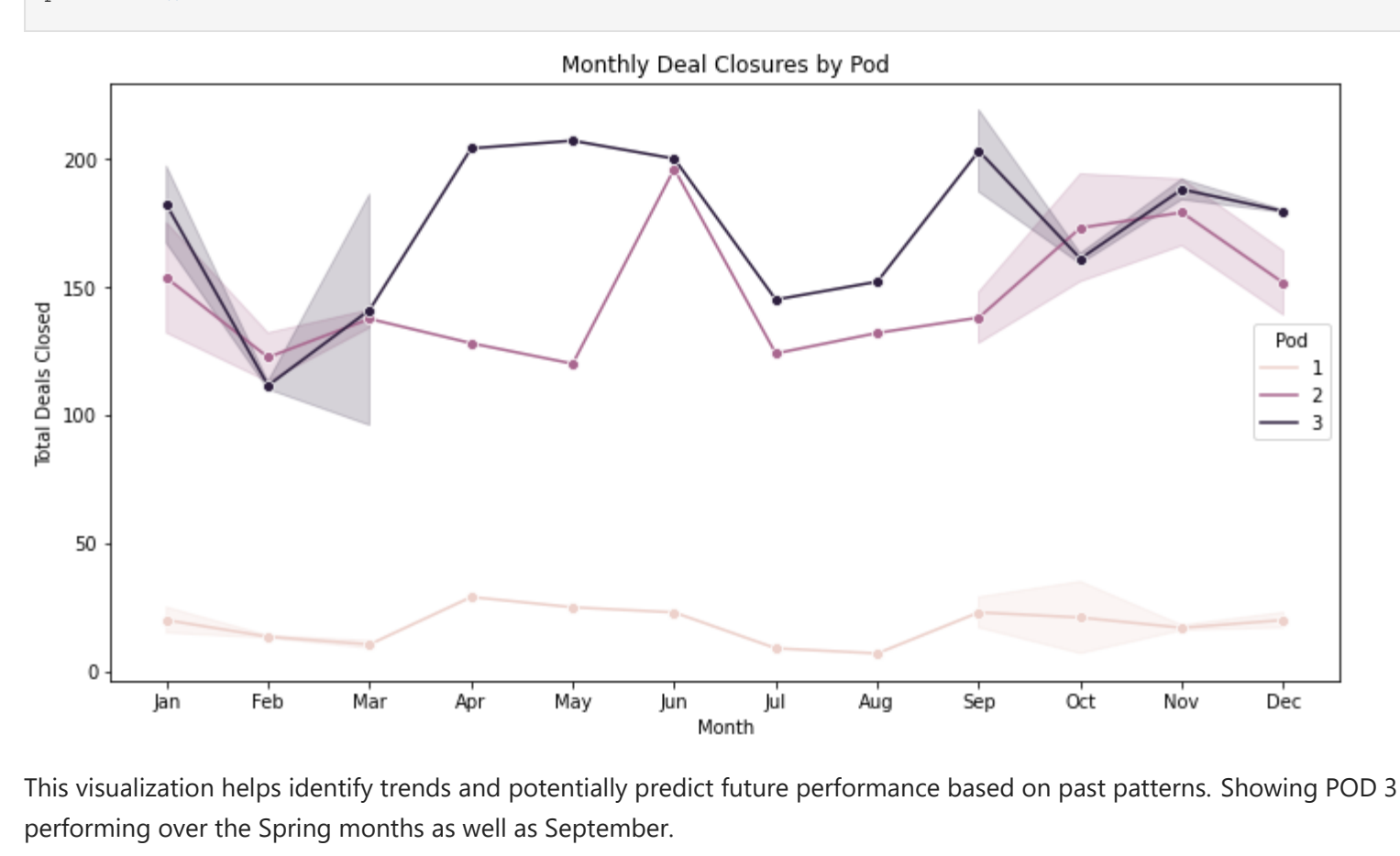
Creating a prediction model to forecast which POD will close the most deals, while excluding POD 4 from the analysis, involves several steps, including data preparation, feature selection, model training, and evaluation.

Excluding POD 4 from dataset and preparing the features (X) and target (y). The target variable (y) the count of closed deals for each POD, and features (X) include any variables that might predict sales performance - average purchase price, conversion rate, temporal features like month or quarter.

In [70]:
```python
# Filter out Pod 4
df_filtered = merged_df_fuzzy_unique[merged_df_fuzzy_unique['Pod'] != 4]

# Ensure 'ActivatedAt' is a datetime type
df_filtered['ActivatedAt'] = pd.to_datetime(df_filtered['ActivatedAt'])

# Extract year and month from 'ActivatedAt' for temporal analysis
df_filtered['Year'] = df_filtered['ActivatedAt'].dt.year
df_filtered['Month'] = df_filtered['ActivatedAt'].dt.month
```

In [71]:
```python
# Create a 'DealClosed' flag based on 'ClosingDate' not being null
df_filtered['DealClosed'] = ~df_filtered['ClosingDate'].isna()

# Aggregate data by Pod, Year, and Month
aggregated_data = df_filtered.groupby(['Pod', 'Year', 'Month']).agg(
    TotalDeals=('DealClosed', 'sum')  # Count of deals closed
).reset_index()
```

In [72]:
```python
plt.figure(figsize=(12, 6))
sns.lineplot(data=aggregated_data, x='Month', y='TotalDeals', hue='Pod', marker='o')
plt.title('Monthly Deal Closures by Pod')
plt.xlabel('Month')
plt.ylabel('Total Deals Closed')
plt.legend(title='Pod')
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.show()
```



This visualization helps identify trends and potentially predict future performance based on past patterns. Showing POD 3 as top performing over the Spring months as well as September.

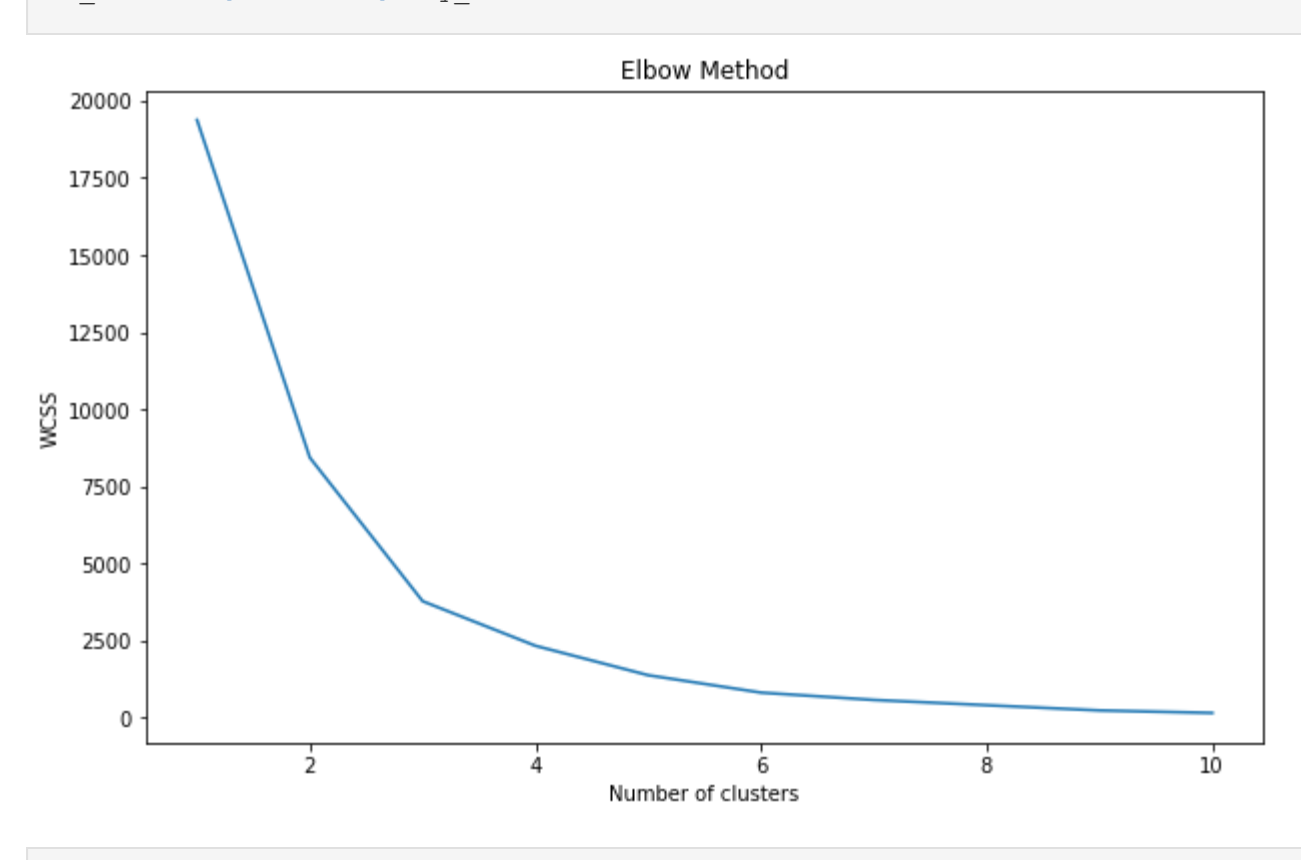
Next I will calculate the percent of converting leads.

In [73]:
```python
# Convert company names in both datasets to sets for efficient comparison
unique_crm_companies = set(df_crm['Company Name'].unique())
unique_cert_companies = set(df_cert['Company Name'].unique())

# Identify non-converting leads (companies present in CRM but not in Cert)
non_converting_leads = unique_crm_companies.difference(unique_cert_companies)

# Convert back to list if you want to filter or further manipulate the DataFrame
non_converting_list = list(non_converting_leads)

# Count of non-converting leads
count_non_converting = len(non_converting_list)

# Count of converting leads (companies present in both CRM and Cert)
converting_leads = unique_crm_companies.intersection(unique_cert_companies)
count_converting = len(converting_leads)

# Total counts from original DataFrames for context
total_crm_leads = df_crm.shape[0]
total_cert_leads = df_cert.shape[0]

# Printing counts for clarity
print(f"Total CRM Leads: {total_crm_leads}")
print(f"Total Certificate Entries: {total_cert_leads}")
print(f"Count of Non-Converting Leads: {count_non_converting}")
print(f"Count of Converting Leads: {count_converting}")
print(f"Percent of Converting Leads: {(count_converting/count_non_converting)*100:.2f}%")
```

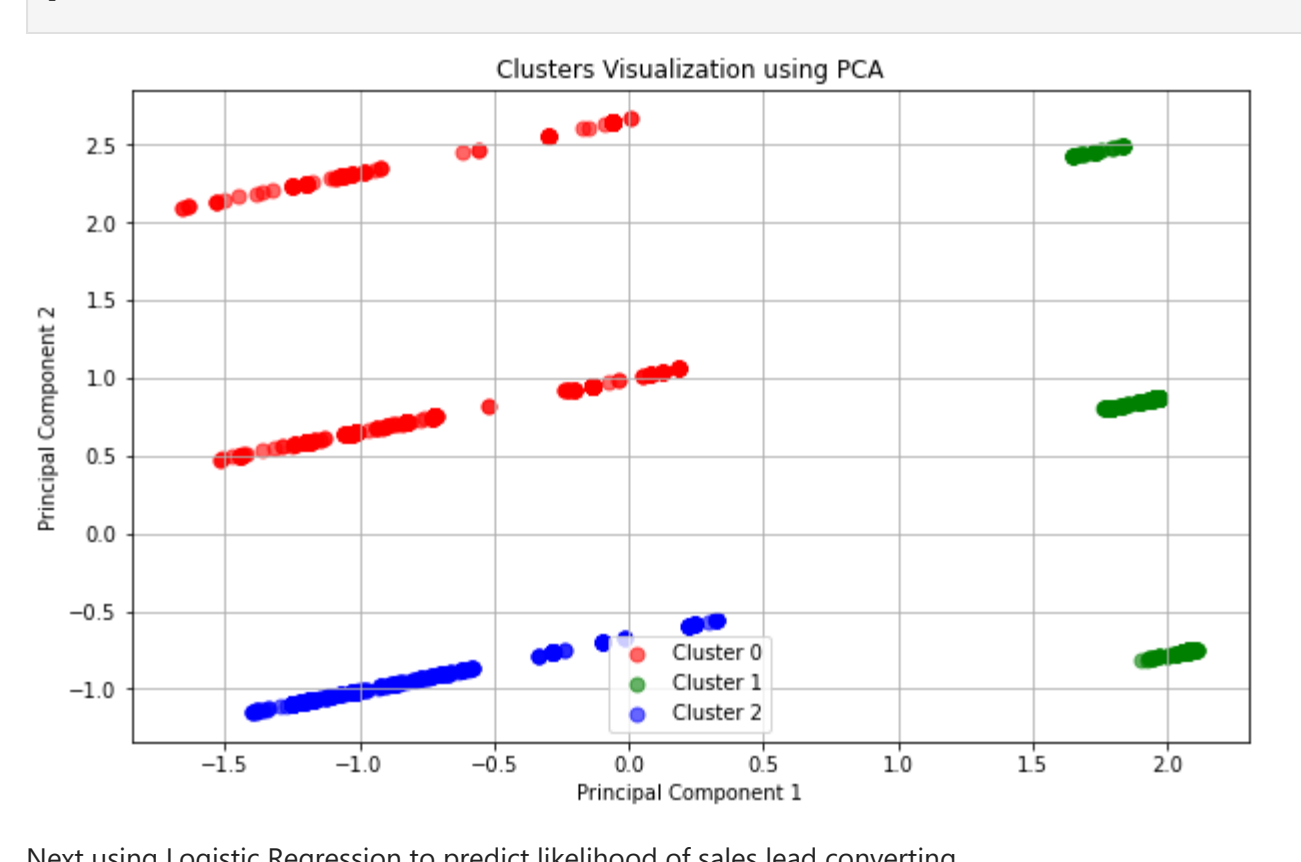
```
Total CRM Leads: 3062
Total Certificate Entries: 40191
Count of Non-Converting Leads: 2327
Count of Converting Leads: 190
Percent of Converting Leads: 8.17%
```

Running cluster analysis through K-means and determining the number of clusters using the elbow method.

In [74]:
```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

#Selecting numerical columns only
numerical_cols = ['Company Id', 'Year of Createdate', 'Pod']  # Add your numerical columns here
df_numerical = df_filtered[numerical_cols]

# Scale the features
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numerical)

# Determine the optimal number of clusters using the elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=42)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

# Plot the WCSS to observe the 'elbow'
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Assuming you choose k=3 based on the elbow plot
k = 3
kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10, random_state=42)
y_kmeans = kmeans.fit_predict(df_scaled)

# Add the cluster labels to your original DataFrame
df_filtered['Cluster'] = y_kmeans
```



In [75]:
```python
from sklearn.decomposition import PCA

# Applying PCA to reduce dimensions to 2 for visualization
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

# Creating a DataFrame for the PCA results
df_pca = pd.DataFrame(df_pca, columns=['PC1', 'PC2'])

# Adding the cluster labels
df_pca['Cluster'] = y_kmeans

# Plotting the clusters
plt.figure(figsize=(10, 6))
colors = ['red', 'green', 'blue']
for i in range(k):
    plt.scatter(df_pca[df_pca['Cluster'] == i]['PC1'], df_pca[df_pca['Cluster'] == i]['PC2'],
                label=f'Cluster {i}', s=50, c=colors[i], alpha=0.6)

plt.title('Clusters Visualization using PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```



Next using Logistic Regression to predict likelihood of sales lead converting.

In [76]:
```python
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.impute import SimpleImputer

# Excluding non-feature columns and the target column
feature_cols = ['WarrantyEnrollmentAppID', 'FormattedPurchasePrice', 'BuilderNumber',
                'Premium', 'Company Id', 'Year of Createdate', 'Pod',
                'Year', 'Month', 'Cluster']  # Add other columns you deem as features
target_col = 'IsConverted'

# Convert boolean target to int
df_filtered[target_col] = df_filtered[target_col].astype(int)

# Split the DataFrame into features (X) and target (y)
X = df_filtered[feature_cols]
y = df_filtered[target_col]

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

# Identify categorical and numerical columns in the features
categorical_cols = X_train.select_dtypes(include=['object', 'category']).columns
numerical_cols = X_train.select_dtypes(include=['int64', 'float64', 'int32']).columns

# Modify the preprocessor to include SimpleImputer for numerical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with the mean
            ('scaler', StandardScaler())]), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ])

# Adjust the model in the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42))
])

# Fit the model with the adjusted pipeline
pipeline.fit(X_train, y_train)

# Make predictions and evaluate the model again
y_pred = pipeline.predict(X_test)
print("Adjusted Classification Report:\n", classification_report(y_test, y_pred))
print("Adjusted Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

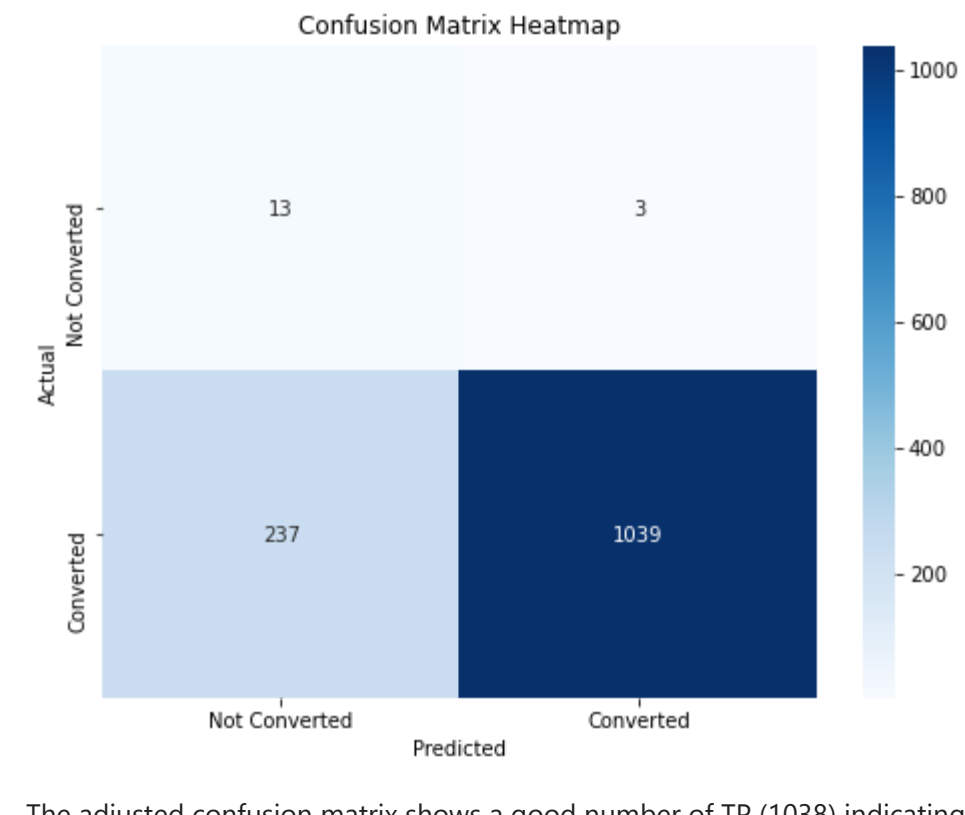
```
Adjusted Classification Report:
               precision    recall  f1-score   support

           0       0.05      0.81      0.10        16
           1       1.00      0.81      0.90      1276

    accuracy                           0.81      1292
   macro avg       0.52      0.81      0.50      1292
weighted avg       0.99      0.81      0.89      1292

Adjusted Confusion Matrix:
 [[  13    3]
 [ 237 1039]]
```

Class 0 (Not Converted): Represents the leads that did not convert into sales.

Class 1 (Converted): Represents the leads that successfully converted into sales.

A confusion matrix provides a breakdown of the model's performance:

In [78]:
```python
from sklearn.metrics import confusion_matrix

# y_test and y_pred are true labels and predictions, respectively
cm = confusion_matrix(y_test, y_pred)

# Plotting the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Not Converted", "Converted"], yticklabels=["No
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



The adjusted confusion matrix shows a good number of TP (1038) indicating successful prediction of conversions.

The TN (13) is relatively low, which suggests that there are few cases of correctly identifying non-conversions. Given the small number of actual non-conversions (16), this isn't necessarily a problem but indicates the rarity of non-conversion cases or possible class imbalance.

FPs (3) are very low, indicating the model seldom incorrectly predicts a conversion.

FNs (238) suggest there are a considerable number of conversions that the model fails to identify. This could be an area to improve, as each missed conversion represents a potential revenue opportunity that wasn't anticipated.

A high number of TP and TN values relative to FP and FN indicates a good model performance.

A high FP rate might indicate that the model is too optimistic, seeing conversions where there are none. This might waste resources by overinvesting in leads unlikely to convert.

A high FN rate might indicate the model is too pessimistic, potentially missing out on genuine opportunities for conversion. This could mean missed revenue or lost opportunities for growth.

The high recall for the "Converted" class (0.81) indicates that the model is quite good at capturing the majority of actual conversions, even though it's somewhat prone to missing some (indicated by the FN count). The precision for the "Not Converted" class is low (0.05), partly because the actual number of non-conversions is very small, making any error more impactful on precision and recall metrics.

Model Overview: The predictive model focuses on identifying which sales leads are most likely to convert into actual sales. The model uses historical data from our CRM and warranty systems to learn patterns that indicate a successful conversion.

High-Level Performance: Overall, the model is highly effective, with an accuracy rate of approximately 81%. This means that in 81% of cases, the model accurately predicts whether a lead will convert or not.

Strengths of the Model: One of the model's key strengths is its ability to correctly identify a large majority of the leads that will convert. Specifically, it correctly identified conversions 81% of the time, which suggests it is a valuable tool for focusing our sales efforts on the most promising leads.

Areas for Improvement: However, the model tends to miss some opportunities, as indicated by a false negative rate. This means that there are leads that could have converted but were not identified by the model as high-potential. We're looking into ways to reduce this rate, which could involve further refining our data inputs or adjusting the model's parameters.

Implications for Business Strategy: The model's current performance suggests that it can significantly enhance our sales strategy by prioritizing leads with the highest likelihood of conversion. This could lead to more efficient use of our sales resources and potentially higher conversion rates overall. The insights from the model also provide us with opportunities to revisit and optimize our lead engagement strategies, especially for those leads that the model predicts as non-converting but might still hold potential.

Next Steps: We plan to integrate the model's predictions into our sales processes, enabling more targeted follow-ups. Additionally, we will continue refining the model by incorporating more detailed data and feedback from the sales team to further improve its accuracy.

Call to Action: To fully leverage the model's capabilities, we recommend a collaborative approach where sales and data teams work closely to continually refine our sales targeting strategies based on model predictions and real-world outcomes.

By focusing on these key points, you convey the value of the model in enhancing business strategies, while also being transparent about its limitations and the need for ongoing improvement.