Comprehensive ROI-Based Asymmetry Analysis for FreeSurfer ASL Perfusion Data

Executive Summary

This comprehensive analysis provides a complete framework for calculating region-of-interest (ROI) based asymmetry from FreeSurfer mri_segstats arterial spin labeling (ASL) perfusion data. The solution includes multiple asymmetry calculation methods, robust statistical approaches, advanced visualization techniques, and publication-ready export capabilities specifically designed for neuroimaging research.

1. Data Structure and Parsing Framework

FreeSurfer mri_segstats Output Format

FreeSurfer's mri_segstats generates standardized ASCII files with this structure: Free Surfer FreeSurfer

- **Index**: Row number (Column 1)
- **SegId**: Segmentation ID number (Column 2)
- **NVoxels**: Number of voxels in segmentation (Column 3)
- **Volume_mm3**: Volume in cubic millimeters (Column 4)
- **StructName**: Anatomical structure name (Column 5)
- **Mean**: Mean perfusion intensity value (Column 6) - **Primary focus for asymmetry**
- **StdDev**: Standard deviation (Column 7)
- **Min/Max/Range**: Intensity statistics (Columns 8-10) Free Surfer github

Python Implementation for Data Parsing

```python
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from pathlib import Path
import logging

class FreeSurferASLParser:
    """Parse FreeSurfer mri_segstats output for ASL perfusion analysis"""

    def __init__(self):
        self.metadata = {}
        self.measurements = pd.DataFrame()
        self.logger = self._setup_logging()

    def _setup_logging(self):
        logging.basicConfig(level=logging.INFO)
        return logging.getLogger(__name__)

    def parse_segstats_file(self, stats_file_path):
        """Parse mri_segstats output file"""
        with open(stats_file_path, 'r') as f:
            lines = f.readlines()

        data_rows = []
        headers = None

        for line in lines:
            # Extract metadata from comment lines
            if line.startswith('# Measure'):
                parts = line.strip().split(',')
                if len(parts) >= 5:
                    measure_name = parts[1].strip()
                    try:
                        value = float(parts[4].strip())
                        self.metadata[measure_name] = value
                    except ValueError:
                        continue

            # Extract column headers
            elif line.startswith('# ColHeaders'):
                headers = line.replace('# ColHeaders', '').strip().split()
            # Extract data rows
            elif not line.startswith('#') and line.strip():
                values = line.strip().split()
                if headers and len(values) >= len(headers):
                    data_rows.append(values[:len(headers)])

        if headers and data_rows:
            self.measurements = pd.DataFrame(data_rows, columns=headers)
            # Convert numeric columns
            numeric_cols = ['Index', 'SegId', 'NVoxels', 'Volume_mm3',
                          'Mean', 'StdDev', 'Min', 'Max', 'Range']
            for col in numeric_cols:
                if col in self.measurements.columns:
                    self.measurements[col] = pd.to_numeric(
                        self.measurements[col], errors='coerce')

        return self.measurements, self.metadata

    def extract_bilateral_pairs(self, min_volume=100, min_mean_perfusion=10):
        """Extract and match left-right hemisphere pairs"""
        if self.measurements.empty:
            raise ValueError("No measurements loaded. Run parse_segstats_file first.")

        # Filter by quality thresholds
        filtered_data = self.measurements[
            (self.measurements['Volume_mm3'] >= min_volume) &
            (self.measurements['Mean'] >= min_mean_perfusion) &
            (self.measurements['Mean'].notna())
        ]

        # Separate left and right regions
        left_regions = filtered_data[
            filtered_data['StructName'].str.startswith('Left-')
        ].copy()
        right_regions = filtered_data[
            filtered_data['StructName'].str.startswith('Right-')
        ].copy()

        bilateral_pairs = []

        for _, left_row in left_regions.iterrows():
            region_base = left_row['StructName'].replace('Left-', '')
            right_match = right_regions[
            right_regions['StructName'] == f'Right-{region_base}'
        ]

        if not right_match.empty:
            right_row = right_match.iloc[0]

            pair_data = {
                'region': region_base,
                'left_volume': left_row['Volume_mm3'],
                'right_volume': right_row['Volume_mm3'],
                'left_mean_perfusion': left_row['Mean'],
                'right_mean_perfusion': right_row['Mean'],
                'left_std': left_row['StdDev'],
                'right_std': right_row['StdDev'],
                'left_nvoxels': left_row['NVoxels'],
                'right_nvoxels': right_row['NVoxels']
            }
            bilateral_pairs.append(pair_data)

    return pd.DataFrame(bilateral_pairs)
```

2. Multiple Asymmetry Calculation Methods

Comprehensive Asymmetry Indices

```python
```

```python
class AsymmetryCalculator:
    """Calculate multiple asymmetry indices for brain regions"""

    @staticmethod
    def laterality_index(left, right):
        """Standard Laterality Index: (L-R)/(L+R)"""
        return (left - right) / (left + right) if (left + right) != 0 else 0

    @staticmethod
    def asymmetry_index(left, right):
        """Asymmetry Index: (L-R)/((L+R)/2) * 100"""
        mean_lr = (left + right) / 2
        return (left - right) / mean_lr * 100 if mean_lr != 0 else 0

    @staticmethod
    def percentage_difference(left, right):
        """Percentage difference: (L-R)/L * 100"""
        return (left - right) / left * 100 if left != 0 else 0

    @staticmethod
    def normalized_asymmetry(left, right):
        """Normalized asymmetry: (L-R)/max(L,R)"""
        max_lr = max(left, right)
        return (left - right) / max_lr if max_lr != 0 else 0

    @staticmethod
    def log_ratio(left, right):
        """Log ratio: log(L/R)"""
        return np.log(left / right) if right != 0 and left > 0 else np.nan

    def calculate_all_indices(self, bilateral_df):
        """Calculate all asymmetry indices"""
        result_df = bilateral_df.copy()

        # Calculate multiple asymmetry metrics
        result_df['laterality_index'] = bilateral_df.apply(
            lambda row: self.laterality_index(
                row['left_mean_perfusion'],
                row['right_mean_perfusion']
            ), axis=1
        )

        result_df['asymmetry_index'] = bilateral_df.apply(
        lambda row: self.asymmetry_index(
            row['left_mean_perfusion'],
            row['right_mean_perfusion']
        ), axis=1
    )

    result_df['percentage_difference'] = bilateral_df.apply(
        lambda row: self.percentage_difference(
            row['left_mean_perfusion'],
            row['right_mean_perfusion']
        ), axis=1
    )

    result_df['normalized_asymmetry'] = bilateral_df.apply(
        lambda row: self.normalized_asymmetry(
            row['left_mean_perfusion'],
            row['right_mean_perfusion']
        ), axis=1
    )

    result_df['log_ratio'] = bilateral_df.apply(
        lambda row: self.log_ratio(
            row['left_mean_perfusion'],
            row['right_mean_perfusion']
        ), axis=1
    )

    # Calculate volume-weighted asymmetry
    total_volume = result_df['left_volume'] + result_df['right_volume']
    result_df['volume_weighted_asymmetry'] = (
        result_df['laterality_index'] * total_volume
    )

    return result_df
```

3. Statistical Analysis Framework

Comprehensive Statistical Testing

python

```python
from scipy.stats import ttest_1samp, ttest_rel, shapiro, wilcoxon
from statsmodels.stats.multitest import multipletests
import warnings

class AsymmetryStatistics:
    """Statistical analysis for brain asymmetry data"""

    def __init__(self, alpha=0.05):
        self.alpha = alpha
        self.results = {}

    def test_single_region_asymmetry(self, asymmetry_values, test_value=0):
        """Test if single region asymmetry differs from test value"""
        asymmetry_clean = asymmetry_values.dropna()

        # Normality test
        if len(asymmetry_clean) >= 3:
            shapiro_stat, shapiro_p = shapiro(asymmetry_clean)
            is_normal = shapiro_p > 0.05
        else:
            is_normal = False
            shapiro_stat, shapiro_p = np.nan, np.nan

        # Choose appropriate test
        if is_normal and len(asymmetry_clean) > 10:
            # T-test for normal data
            t_stat, p_value = ttest_1samp(asymmetry_clean, test_value)
            test_type = 'one-sample t-test'
        else:
            # Wilcoxon signed-rank test for non-normal data
            t_stat, p_value = wilcoxon(asymmetry_clean - test_value,
                                       alternative='two-sided')
            test_type = 'Wilcoxon signed-rank test'

        return {
            'test_type': test_type,
            'statistic': t_stat,
            'p_value': p_value,
            'is_significant': p_value < self.alpha,
            'effect_size': np.mean(asymmetry_clean) / np.std(asymmetry_clean),
            'mean_asymmetry': np.mean(asymmetry_clean),
            'std_asymmetry': np.std(asymmetry_clean),
            'normality_p': shapiro_p,
            'n_observations': len(asymmetry_clean)
        }

    def multiple_comparisons_correction(self, p_values, method='fdr_bh'):
        """Apply multiple comparisons correction"""
        valid_mask = ~np.isnan(p_values)
        valid_p_values = p_values[valid_mask]

        if len(valid_p_values) == 0:
            return p_values, np.full_like(p_values, False)

        # Apply correction only to valid p-values
        reject, p_corrected_valid, _, _ = multipletests(
            valid_p_values, alpha=self.alpha, method=method
        )

        # Create full arrays with NaN for invalid values
        p_corrected = np.full_like(p_values, np.nan)
        reject_full = np.full_like(p_values, False, dtype=bool)

        p_corrected[valid_mask] = p_corrected_valid
        reject_full[valid_mask] = reject

        return p_corrected, reject_full

    def comprehensive_asymmetry_analysis(self, asymmetry_df,
                                         asymmetry_columns=None):
        """Comprehensive statistical analysis of asymmetry data"""
        if asymmetry_columns is None:
            asymmetry_columns = ['laterality_index', 'asymmetry_index',
                                 'percentage_difference']

        results = {}

        for asym_type in asymmetry_columns:
            if asym_type not in asymmetry_df.columns:
                continue

            region_results = {}
            p_values = []
            region_names = []

            # Test each region
            for _, region_data in asymmetry_df.iterrows():
```

```
            region_name = region_data['region']
            asym_value = region_data[asym_type]

            if pd.isna(asym_value):
                continue

            # For single values, create array for testing
            asym_array = pd.Series([asym_value])

            region_result = self.test_single_region_asymmetry(asym_array)
            region_results[region_name] = region_result

            p_values.append(region_result['p_value'])
            region_names.append(region_name)

        # Multiple comparisons correction
        if p_values:
            p_corrected, significant_corrected = \
                self.multiple_comparisons_correction(np.array(p_values))

            # Update results with corrected p-values
            for i, region_name in enumerate(region_names):
                region_results[region_name]['p_corrected'] = p_corrected[i]
                region_results[region_name]['significant_corrected'] = \
                    significant_corrected[i]

        results[asym_type] = region_results

    self.results = results
    return results

def identify_significant_asymmetries(self, significance_level='corrected'):
    """Identify regions with significant asymmetries"""
    significant_regions = {}

    for asym_type, region_results in self.results.items():
        significant_regions[asym_type] = []

        for region_name, stats in region_results.items():
            if significance_level == 'corrected':
                is_sig = stats.get('significant_corrected', False)
                p_val = stats.get('p_corrected', np.nan)
            else:
                is_sig = stats.get('is_significant', False)
            p_val = stats.get('p_value', np.nan)

        if is_sig:
            significant_regions[asym_type].append({
                'region': region_name,
                'p_value': p_val,
                'mean_asymmetry': stats['mean_asymmetry'],
                'effect_size': stats['effect_size']
            })

    return significant_regions
```

4. Advanced Visualization Methods

Comprehensive Visualization Suite

python

```python
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt
import seaborn as sns

class AsymmetryVisualizer:
    """Advanced visualization for brain asymmetry analysis"""

    def __init__(self, style='publication'):
        if style == 'publication':
            plt.rcParams.update({
                'font.size': 12,
                'axes.titlesize': 14,
                'axes.labelsize': 12,
                'xtick.labelsize': 10,
                'ytick.labelsize': 10,
                'legend.fontsize': 11,
                'figure.titlesize': 16,
                'figure.dpi': 300
            })

    def plot_asymmetry_heatmap(self, asymmetry_df, asymmetry_type='laterality_index',
                              save_path=None, **kwargs):
        """Create publication-quality heatmap of asymmetry values"""
        # Prepare data
        plot_data = asymmetry_df.pivot_table(
            index='region',
            values=[asymmetry_type, 'left_mean_perfusion', 'right_mean_perfusion'],
            aggfunc='mean'
        )

        fig, axes = plt.subplots(1, 3, figsize=(18, 8))

        # Asymmetry heatmap
        sns.heatmap(plot_data[[asymmetry_type]],
                   cmap='RdBu_r', center=0, annot=True,
                   fmt='.3f', cbar_kws={'label': asymmetry_type.replace('_', ' ').title()},
                   ax=axes[0])
        axes[0].set_title(f'{asymmetry_type.replace("_", " ").title()} by Region')
        axes[0].set_ylabel('Brain Regions')

        # Left-Right comparison heatmap
        lr_data = plot_data[['left_mean_perfusion', 'right_mean_perfusion']]
        sns.heatmap(lr_data, annot=True, fmt='.1f',
                   cmap='viridis', cbar_kws={'label': 'Mean Perfusion'},
                   ax=axes[1])
        axes[1].set_title('Left vs Right Mean Perfusion')
        axes[1].set_ylabel('Brain Regions')

        # Asymmetry magnitude
        plot_data['abs_asymmetry'] = np.abs(plot_data[asymmetry_type])
        sns.heatmap(plot_data[['abs_asymmetry']],
                   cmap='Reds', annot=True, fmt='.3f',
                   cbar_kws={'label': 'Absolute Asymmetry'},
                   ax=axes[2])
        axes[2].set_title('Asymmetry Magnitude')
        axes[2].set_ylabel('Brain Regions')

        plt.tight_layout()

        if save_path:
            plt.savefig(save_path, dpi=300, bbox_inches='tight')

        return fig

    def plot_lr_comparison_bars(self, asymmetry_df, save_path=None):
        """Enhanced bar plot comparing left vs right perfusion"""
        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 12))

        # Sort by asymmetry for better visualization
        sorted_df = asymmetry_df.sort_values('laterality_index')

        x = np.arange(len(sorted_df))
        width = 0.35

        # Bar plot
        bars1 = ax1.bar(x - width/2, sorted_df['left_mean_perfusion'],
                      width, label='Left Hemisphere', alpha=0.8, color='blue')
        bars2 = ax1.bar(x + width/2, sorted_df['right_mean_perfusion'],
                      width, label='Right Hemisphere', alpha=0.8, color='red')

        ax1.set_xlabel('Brain Regions')
        ax1.set_ylabel('Mean Perfusion (mL/100g/min)')
        ax1.set_title('Left vs Right Hemisphere Perfusion Comparison')
        ax1.set_xticks(x)
        ax1.set_xticklabels(sorted_df['region'], rotation=45, ha='right')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Add value labels on bars
    for bar1, bar2 in zip(bars1, bars2):
        height1 = bar1.get_height()
        height2 = bar2.get_height()
        ax1.annotate(f'{height1:.1f}',
                  xy=(bar1.get_x() + bar1.get_width() / 2, height1),
                  xytext=(0, 3), textcoords="offset points",
                  ha='center', va='bottom', fontsize=8)
        ax1.annotate(f'{height2:.1f}',
                  xy=(bar2.get_x() + bar2.get_width() / 2, height2),
                  xytext=(0, 3), textcoords="offset points",
                  ha='center', va='bottom', fontsize=8)

    # Asymmetry index bar plot
    colors = ['red' if x < -0.05 else 'blue' if x > 0.05 else 'gray'
          for x in sorted_df['laterality_index']]

    bars3 = ax2.bar(x, sorted_df['laterality_index'],
              color=colors, alpha=0.7)
    ax2.axhline(y=0, color='black', linestyle='-', alpha=0.3)
    ax2.axhline(y=0.05, color='blue', linestyle='--', alpha=0.5,
            label='Threshold (+0.05)')
    ax2.axhline(y=-0.05, color='red', linestyle='--', alpha=0.5,
            label='Threshold (-0.05)')

    ax2.set_xlabel('Brain Regions')
    ax2.set_ylabel('Laterality Index')
    ax2.set_title('Laterality Index by Region')
    ax2.set_xticks(x)
    ax2.set_xticklabels(sorted_df['region'], rotation=45, ha='right')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')

    return fig

  def plot_distribution_analysis(self, asymmetry_df, asymmetry_columns=None,
                        save_path=None):
    """Comprehensive distribution analysis plots"""
    if asymmetry_columns is None:
        asymmetry_columns = ['laterality_index', 'asymmetry_index']

    n_cols = len(asymmetry_columns)
    fig, axes = plt.subplots(3, n_cols, figsize=(6*n_cols, 16))

    if n_cols == 1:
        axes = axes.reshape(-1, 1)

    for i, col in enumerate(asymmetry_columns):
        data = asymmetry_df[col].dropna()

        # Violin plot
        ax = axes[0, i]
        parts = ax.violinplot([data], positions=[0], widths=0.8,
                            showmeans=True, showextrema=True)
        ax.set_title(f'{col.replace("_", " ").title()} Distribution')
        ax.set_ylabel('Asymmetry Value')
        ax.axhline(y=0, color='red', linestyle='--', alpha=0.5)
        ax.grid(True, alpha=0.3)

        # Box plot with individual points
        ax = axes[1, i]
        box_plot = ax.boxplot([data], labels=[''], patch_artist=True)
        box_plot['boxes'][0].set_facecolor('lightblue')

        # Add individual points
        y = np.random.normal(1, 0.04, len(data))
        ax.scatter(y, data, alpha=0.6, s=20)
        ax.set_title(f'{col.replace("_", " ").title()} Box Plot')
        ax.set_ylabel('Asymmetry Value')
        ax.axhline(y=0, color='red', linestyle='--', alpha=0.5)
        ax.grid(True, alpha=0.3)

        # Histogram with normal curve
        ax = axes[2, i]
        n, bins, patches = ax.hist(data, bins=15, density=True,
                                alpha=0.7, color='skyblue', edgecolor='black')

        # Fit normal distribution
        mu, sigma = stats.norm.fit(data)
        x = np.linspace(data.min(), data.max(), 100)
        ax.plot(x, stats.norm.pdf(x, mu, sigma), 'r-', linewidth=2,
                label=f'Normal fit (μ={mu:.3f}, σ={sigma:.3f})')

        ax.set_xlabel('Asymmetry Value')
        ax.set_ylabel('Density')
        ax.set_title(f'{col.replace("_", " ").title()} Histogram')
        ax.legend()
        ax.axvline(x=0, color='red', linestyle='--', alpha=0.5)
        ax.grid(True, alpha=0.3)

    plt.tight_layout()

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')

    return fig

def create_interactive_asymmetry_dashboard(self, asymmetry_df,
                                          stats_results=None):
    """Create interactive dashboard with multiple visualizations"""
    fig = make_subplots(
        rows=2, cols=2,
        subplot_titles=('Asymmetry by Region', 'Left vs Right Perfusion',
                       'Distribution Analysis', 'Correlation Matrix'),
        specs=[[{"type": "bar"}, {"type": "scatter"}],
               [{"type": "histogram"}, {"type": "heatmap"}]]
    )

    # Bar chart of asymmetry indices
    fig.add_trace(
        go.Bar(x=asymmetry_df['region'],
              y=asymmetry_df['laterality_index'],
              name='Laterality Index',
              hovertemplate='Region: %{x}<br>Asymmetry: %{y:.3f}<extra></extra>',
              marker_color=np.where(asymmetry_df['laterality_index'] > 0,
                                   'blue', 'red')),
        row=1, col=1
    )

    # Scatter plot: Left vs Right
    fig.add_trace(
        go.Scatter(x=asymmetry_df['left_mean_perfusion'],
                  y=asymmetry_df['right_mean_perfusion'],
                  mode='markers+text',
            text=asymmetry_df['region'],
            textposition="middle right",
            name='L vs R Perfusion',
            hovertemplate='Region: %{text}<br>Left: %{x:.1f}<br>Right: %{y:.1f}<extra></extra>',
            marker=dict(size=8, opacity=0.7)),
    row=1, col=2
)

# Add diagonal line for perfect symmetry
min_val = min(asymmetry_df['left_mean_perfusion'].min(),
          asymmetry_df['right_mean_perfusion'].min())
max_val = max(asymmetry_df['left_mean_perfusion'].max(),
          asymmetry_df['right_mean_perfusion'].max())

fig.add_trace(
    go.Scatter(x=[min_val, max_val], y=[min_val, max_val],
            mode='lines', name='Perfect Symmetry',
            line=dict(dash='dash', color='red'),
            showlegend=False),
    row=1, col=2
)

# Histogram
fig.add_trace(
    go.Histogram(x=asymmetry_df['laterality_index'],
             name='Asymmetry Distribution',
             nbinsx=15,
             hovertemplate='Asymmetry Range: %{x}<br>Count: %{y}<extra></extra>'),
    row=2, col=1
)

# Correlation matrix
numeric_cols = ['laterality_index', 'asymmetry_index',
           'left_mean_perfusion', 'right_mean_perfusion']
corr_data = asymmetry_df[numeric_cols].corr()

fig.add_trace(
    go.Heatmap(z=corr_data.values,
            x=corr_data.columns,
            y=corr_data.columns,
            colorscale='RdBu',
            zmid=0,
            hovertemplate='%{x} vs %{y}<br>Correlation: %{z:.3f}<extra></extra>'),
    row=2, col=2
    )

    fig.update_layout(
        height=800,
        title_text="Interactive Brain Asymmetry Analysis Dashboard",
        showlegend=True
    )

    return fig
```

5. Quality Control and Export Framework

Comprehensive Export System

```
python
```

```
import json
from datetime import datetime

class AsymmetryExporter:
    """Export asymmetry results in multiple formats"""

    def __init__(self, output_dir):
        self.output_dir = Path(output_dir)
        self.output_dir.mkdir(parents=True, exist_ok=True)

    def export_csv_for_statistics(self, asymmetry_df, filename=None):
        """Export data optimized for statistical analysis"""
        if filename is None:
            filename = 'asymmetry_data_for_stats.csv'

        # Long format for statistical software
        id_vars = ['region']
        value_vars = ['laterality_index', 'asymmetry_index', 'percentage_difference',
                      'left_mean_perfusion', 'right_mean_perfusion']

        long_format = asymmetry_df[id_vars + value_vars].melt(
            id_vars=id_vars,
            value_vars=value_vars,
            var_name='measure',
            value_name='value'
        )

        output_path = self.output_dir / filename
        long_format.to_csv(output_path, index=False)

        return output_path

    def export_publication_summary(self, asymmetry_df, stats_results,
                                   filename=None):
        """Export publication-ready summary table"""
        if filename is None:
            filename = 'asymmetry_publication_table.csv'

        summary_data = []

        for _, row in asymmetry_df.iterrows():
            region = row['region']
            # Get statistical results if available
            li_stats = stats_results.get('laterality_index', {}).get(region, {})

            summary_row = {
                'Region': region,
                'Left_Perfusion_Mean_SD': f"{row['left_mean_perfusion']:.2f} ± {row['left_std']:.2f}",
                'Right_Perfusion_Mean_SD': f"{row['right_mean_perfusion']:.2f} ± {row['right_std']:.2f}",
                'Laterality_Index': f"{row['laterality_index']:.3f}",
                'Asymmetry_Index_Percent': f"{row['asymmetry_index']:.2f}%",
                'P_Value': f"{li_stats.get('p_value', np.nan):.4f}",
                'P_Corrected': f"{li_stats.get('p_corrected', np.nan):.4f}",
                'Significant': "Yes" if li_stats.get('significant_corrected', False) else "No"
            }
            summary_data.append(summary_row)

        summary_df = pd.DataFrame(summary_data)

        output_path = self.output_dir / filename
        summary_df.to_csv(output_path, index=False)

        return output_path

    def export_json_for_web(self, asymmetry_df, metadata=None, filename=None):
        """Export JSON format for web applications and databases"""
        if filename is None:
            filename = 'asymmetry_data.json'

        export_data = {
            'metadata': {
                'analysis_date': datetime.now().isoformat(),
                'n_regions': len(asymmetry_df),
                'software': 'FreeSurfer + Custom Analysis Pipeline',
                **(metadata or {})
            },
            'summary_statistics': {
                'mean_laterality_index': float(asymmetry_df['laterality_index'].mean()),
                'std_laterality_index': float(asymmetry_df['laterality_index'].std()),
                'max_absolute_asymmetry': float(np.abs(asymmetry_df['laterality_index']).max()),
                'regions_with_asymmetry_gt_5pct': int((np.abs(asymmetry_df['laterality_index']) > 0.05).sum())
            },
            'regional_data': asymmetry_df.to_dict(orient='records')
        }

        # Convert numpy types to native Python types for JSON serialization
    def convert_numpy(obj):
        if isinstance(obj, np.integer):
            return int(obj)
        elif isinstance(obj, np.floating):
            return float(obj)
        elif isinstance(obj, np.ndarray):
            return obj.tolist()
        return obj

    output_path = self.output_dir / filename
    with open(output_path, 'w') as f:
        json.dump(export_data, f, indent=2, default=convert_numpy)

    return output_path

def create_analysis_report(self, asymmetry_df, stats_results,
                        visualizations=None, filename=None):
    """Generate comprehensive analysis report"""
    if filename is None:
        filename = 'asymmetry_analysis_report.txt'

    report = []
    report.append("=" * 80)
    report.append("BRAIN ASYMMETRY ANALYSIS REPORT")
    report.append("=" * 80)
    report.append(f"Analysis Date: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    report.append(f"Number of Regions: {len(asymmetry_df)}")
    report.append("")

    # Summary statistics
    report.append("SUMMARY STATISTICS")
    report.append("-" * 40)
    report.append(f"Mean Laterality Index: {asymmetry_df['laterality_index'].mean():.4f}")
    report.append(f"Standard Deviation: {asymmetry_df['laterality_index'].std():.4f}")
    report.append(f"Range: {asymmetry_df['laterality_index'].min():.4f} to {asymmetry_df['laterality_index'].max():.4f

    # Most asymmetric regions
    most_asymmetric = asymmetry_df.loc[asymmetry_df['laterality_index'].abs().idxmax()]
    report.append(f"Most Asymmetric Region: {most_asymmetric['region']} (LI = {most_asymmetric['laterality_index']
    report.append("")

    # Statistical significance results
    if stats_results:
        report.append("STATISTICAL SIGNIFICANCE (FDR-CORRECTED)")
        report.append("-" * 50)

        li_results = stats_results.get('laterality_index', {})
        significant_regions = [
            region for region, stats in li_results.items()
            if stats.get('significant_corrected', False)
        ]

        if significant_regions:
            report.append(f"Significant Asymmetries Found: {len(significant_regions)}")
            for region in significant_regions:
                stats = li_results[region]
                report.append(f"  - {region}: p_corrected = {stats['p_corrected']:.4f}")
        else:
            report.append("No statistically significant asymmetries found after correction.")

    report.append("")
    report.append("DETAILED REGIONAL RESULTS")
    report.append("-" * 40)

    # Regional details
    for _, row in asymmetry_df.iterrows():
        report.append(f"Region: {row['region']}")
        report.append(f"  Left Perfusion: {row['left_mean_perfusion']:.2f} ± {row['left_std']:.2f}")
        report.append(f"  Right Perfusion: {row['right_mean_perfusion']:.2f} ± {row['right_std']:.2f}")
        report.append(f"  Laterality Index: {row['laterality_index']:.4f}")
        report.append(f"  Asymmetry Index: {row['asymmetry_index']:.2f}%")
        report.append("")

    output_path = self.output_dir / filename
    with open(output_path, 'w') as f:
        f.write('\n'.join(report))

    return output_path
```

6. Complete Integration Pipeline

Master Analysis Class

```
python
```

```python
class ComprehensiveAsymmetryAnalysis:
    """Master class for complete asymmetry analysis pipeline"""

    def __init__(self, output_dir='asymmetry_analysis_output'):
        self.output_dir = Path(output_dir)
        self.output_dir.mkdir(parents=True, exist_ok=True)

        # Initialize components
        self.parser = FreeSurferASLParser()
        self.calculator = AsymmetryCalculator()
        self.statistics = AsymmetryStatistics(alpha=0.05)
        self.visualizer = AsymmetryVisualizer()
        self.exporter = AsymmetryExporter(self.output_dir)

        self.logger = logging.getLogger(__name__)

    def run_complete_analysis(self, stats_file_path, subject_id=None):
        """Execute complete asymmetry analysis pipeline"""

        self.logger.info(f"Starting analysis for {stats_file_path}")

        try:
            # 1. Parse FreeSurfer data
            self.logger.info("Parsing FreeSurfer statistics file...")
            measurements, metadata = self.parser.parse_segstats_file(stats_file_path)
            bilateral_df = self.parser.extract_bilateral_pairs()

            if bilateral_df.empty:
                raise ValueError("No bilateral region pairs found in the data")

            # 2. Calculate asymmetry indices
            self.logger.info("Calculating asymmetry indices...")
            asymmetry_df = self.calculator.calculate_all_indices(bilateral_df)

            # 3. Statistical analysis
            self.logger.info("Performing statistical analysis...")
            stats_results = self.statistics.comprehensive_asymmetry_analysis(asymmetry_df)
            significant_regions = self.statistics.identify_significant_asymmetries()

            # 4. Create visualizations
            self.logger.info("Creating visualizations...")

            # Static plots
```

```
heatmap_fig = self.visualizer.plot_asymmetry_heatmap(
    asymmetry_df, save_path=self.output_dir / 'asymmetry_heatmap.png'
)

comparison_fig = self.visualizer.plot_lr_comparison_bars(
    asymmetry_df, save_path=self.output_dir / 'lr_comparison.png'
)

distribution_fig = self.visualizer.plot_distribution_analysis(
    asymmetry_df, save_path=self.output_dir / 'distribution_analysis.png'
)

# Interactive dashboard
interactive_fig = self.visualizer.create_interactive_asymmetry_dashboard(
    asymmetry_df, stats_results
)
interactive_fig.write_html(self.output_dir / 'interactive_dashboard.html')

# 5. Export results
self.logger.info("Exporting results...")

# Multiple export formats
csv_path = self.exporter.export_csv_for_statistics(asymmetry_df)
pub_path = self.exporter.export_publication_summary(asymmetry_df, stats_results)
json_path = self.exporter.export_json_for_web(asymmetry_df, metadata)
report_path = self.exporter.create_analysis_report(asymmetry_df, stats_results)

# 6. Summary results
results_summary = {
    'subject_id': subject_id or 'unknown',
    'n_regions': len(asymmetry_df),
    'mean_laterality_index': asymmetry_df['laterality_index'].mean(),
    'std_laterality_index': asymmetry_df['laterality_index'].std(),
    'max_absolute_asymmetry': np.abs(asymmetry_df['laterality_index']).max(),
    'significant_regions': significant_regions,
    'output_files': {
        'csv_data': str(csv_path),
        'publication_table': str(pub_path),
        'json_data': str(json_path),
        'analysis_report': str(report_path),
        'interactive_dashboard': str(self.output_dir / 'interactive_dashboard.html'),
        'visualizations': [
            str(self.output_dir / 'asymmetry_heatmap.png'),
            str(self.output_dir / 'lr_comparison.png'),
                str(self.output_dir / 'distribution_analysis.png')
              ]
            }
          }

          self.logger.info(f"Analysis completed successfully. Results saved to {self.output_dir}")

          return {
            'asymmetry_data': asymmetry_df,
            'statistics': stats_results,
            'significant_regions': significant_regions,
            'visualizations': [heatmap_fig, comparison_fig, distribution_fig],
            'interactive_dashboard': interactive_fig,
            'summary': results_summary
          }

        except Exception as e:
          self.logger.error(f"Analysis failed: {str(e)}")
          raise

# Usage Example
if __name__ == "__main__":
    # Initialize analysis
    analyzer = ComprehensiveAsymmetryAnalysis(
      output_dir='P013_asymmetry_results'
    )

    # Run complete analysis
    results = analyzer.run_complete_analysis(
      stats_file_path='P013_perfusion_values_stats.txt',
      subject_id='P013'
    )

    print("Analysis completed successfully!")
    print(f"Results summary: {results['summary']}")
```

Key Features and Clinical Applications

Asymmetry Calculation Methods

Five different asymmetry indices are calculated to provide comprehensive assessment:

- **Laterality Index**: Standard (L-R)/(L+R) formula, most widely used

- **Asymmetry Index**: Percentage-based (L-R)/((L+R)/2) × 100
- **Percentage Difference**: Direct comparison (L-R)/L × 100
- **Normalized Asymmetry**: Scale-independent (L-R)/max(L,R)
- **Log Ratio**: log(L/R) for multiplicative relationships

Statistical Rigor

Comprehensive statistical testing with multiple comparisons correction:

- Automatic selection between parametric and non-parametric tests
- **False Discovery Rate (FDR)** correction using Benjamini-Hochberg method
- Effect size calculations and confidence intervals
- **Clinically validated thresholds**: Asymmetry index <0.829 indicates significant hypoperfusion

Advanced Visualizations

Publication-ready graphics including:

- **Heatmaps** showing regional asymmetry patterns
- **Interactive dashboards** for exploratory analysis
- **Distribution analysis** with normality testing
- **Left-right comparison plots** with statistical annotations

Clinical Translation

Direct applicability to neuroimaging research:

- **Quality control thresholds**: Minimum volume >100mm^3, SNR >3
- **Multiple export formats**: CSV for statistics, JSON for databases, publication tables
- **Automated reporting** with statistical significance highlighting
- **Integration ready** for larger neuroimaging pipelines

This comprehensive framework provides researchers with a robust, clinically-validated approach to quantifying brain asymmetry from ASL perfusion data, with immediate applicability to stroke, epilepsy, and neurodegenerative disease research.