```
import google.generativeai as genai

GOOGLE_API_KEY = "your-key-here"  # Exposed in notebook
genai.configure(api_key=GOOGLE_API_KEY)
model = genai.GenerativeModel('gemini-2.5-flash')
```

AWS Bedrock:

python

```
import boto3

# No API keys in code - uses AWS credentials
bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')
```

2. Model Invocation

Google Colab:

python

```
response = model.generate_content(prompt)
result = response.text
```

AWS Bedrock:

python

```
body = json.dumps({
    "anthropic_version": "bedrock-2023-05-31",
    "max_tokens": 4000,
    "messages": [{"role": "user", "content": prompt}]
})

response = bedrock.invoke_model(
    modelId="anthropic.claude-3-sonnet-20240229-v1:0",
    body=body
)

result = json.loads(response.get('body').read())['content'][0]['text']
```

3. User Interface

Google Colab:

```python
import ipywidgets as widgets
from IPython.display import display

query_box = widgets.Textarea(description='Query:')
button = widgets.Button(description='Analyze')
output = widgets.Output()

def on_button_click(b):
    with output:
        # Process query
        pass

button.on_click(on_button_click)
display(query_box, button, output)
```

AWS Bedrock (Streamlit):

```python
import streamlit as st

query = st.text_area("Enter your query:")
if st.button("Analyze"):
    result = process_query(query)
    st.write(result)
```

Cost Analysis

Google Colab Costs

```
Free Tier:
- 12 hours continuous usage
- T4 GPU access
- Limited compute units

Colab Pro ($10/month):
- Priority access to better GPUs
- Longer runtimes
- More memory

Gemini API:
- Free tier: 15 requests/minute
- Paid: $0.000125 per 1K input tokens
```

AWS Bedrock Costs

```
Claude 3 Haiku:
- Input: $0.00025 per 1K tokens
- Output: $0.00125 per 1K tokens

Claude 3 Sonnet:
- Input: $0.003 per 1K tokens
- Output: $0.015 per 1K tokens

Example: 100 queries/day = ~$2-5/day
```

Step-by-Step Migration Strategy

Phase 1: Understanding & Setup (Week 1)

Day 1-2: Set up AWS Account

```bash
# 1. Create AWS account
# 2. Set up billing alerts
# 3. Enable Bedrock service
# 4. Request model access
```

Day 3-4: Basic API Testing

```python
```

```
# Start with simplest possible test
def test_bedrock_basic():
    import boto3
    import json

    client = boto3.client('bedrock-runtime', region_name='us-east-1')

    try:
        response = client.invoke_model(
            modelId="anthropic.claude-3-haiku-20240307-v1:0",
            body=json.dumps({
                "anthropic_version": "bedrock-2023-05-31",
                "max_tokens": 100,
                "messages": [{"role": "user", "content": "Hello"}]
            })
        )
        print("✅ Bedrock working!")
        return True
    except Exception as e:
        print(f"❌ Error: {e}")
        return False

test_bedrock_basic()
```

Day 5-7: Data Processing Migration

```python
# Convert your existing P&L calculation logic
# No changes needed - this is pure Python/Pandas

def calculate_pnl_metrics():
    """Same function as in Colab - no changes needed"""
    instruments_df = pd.DataFrame(instrument_data)
    positions_df = pd.DataFrame(position_data)
    # ... rest of your existing logic
    return pnl_df

# Test this first before adding AI
pnl_data = calculate_pnl_metrics()
print(f"Processed {len(pnl_data)} positions")
```

Phase 2: AI Integration (Week 2)

Day 8-10: Convert AI Calls

```python
```

```
# Create wrapper function to replace Gemini calls
class AIAnalyzer:
    def __init__(self):
        self.bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

    def analyze_mbs_data(self, pnl_data, user_query):
        """Replace your Gemini call with this"""

        prompt = f"""
        You are an expert MBS trader. Analyze this P&L data:

        {json.dumps(pnl_data, indent=2)}

        User Question: {user_query}

        Provide specific trading recommendations.
        """

        body = json.dumps({
            "anthropic_version": "bedrock-2023-05-31",
            "max_tokens": 2000,
            "messages": [{"role": "user", "content": prompt}]
        })

        response = self.bedrock.invoke_model(
            modelId="anthropic.claude-3-sonnet-20240229-v1:0",
            body=body
        )

        result = json.loads(response.get('body').read())
        return result['content'][0]['text']

# Test with your existing data
analyzer = AIAnalyzer()
recommendation = analyzer.analyze_mbs_data(
    pnl_data.to_dict('records'),
    "Which position should I hold for 6 months?"
)
print(recommendation)
```

Day 11-14: User Interface Migration

```
python
```

```
# Option 1: Keep Jupyter-like experience with JupyterLab
# Install JupyterLab locally
pip install jupyterlab ipywidgets

# Your existing widget code works with minimal changes:
import ipywidgets as widgets
from IPython.display import display, HTML, Markdown

def create_mbs_interface():
    """Recreate your Colab interface locally"""

    query_box = widgets.Textarea(
        placeholder='Enter your MBS analysis query...',
        description='Query:',
        layout=widgets.Layout(width='90%', height='120px')
    )

    submit_button = widgets.Button(
        description="Analyze with Bedrock",
        button_style='success'
    )

    output = widgets.Output()

    def on_submit_clicked(b):
        user_query = query_box.value.strip()
        output.clear_output()

        with output:
            if not user_query:
                display(HTML("<p>Please enter a query.</p>"))
                return

            # Use Bedrock instead of Gemini
            analyzer = AIAnalyzer()
            response = analyzer.analyze_mbs_data(pnl_data, user_query)
            display(Markdown(response))

    submit_button.on_click(on_submit_clicked)

    display(HTML("""
    <div style="background: #2c3e50; padding: 15px; border-radius: 8px; color: white;">
        <h3>💰 MBS P&L Analytics - AWS Bedrock Version</h3>
        <p>Same interface, now powered by Claude on AWS</p>
    </div>
    """))

    display(query_box, submit_button, output)

# Run in Jupyter
create_mbs_interface()
```

python

```
# Option 2: Convert to Streamlit (Better for sharing)
import streamlit as st

def streamlit_mbs_interface():
    """Streamlit version of your Colab interface"""

    st.set_page_config(
        page_title="MBS P&L Analytics",
        page_icon="💰",
        layout="wide"
    )

    st.markdown("""
    <div style="background: linear-gradient(90deg, #2c3e50, #3498db);
                padding: 20px; border-radius: 10px; color: white; margin-bottom: 20px;">
        <h1>💰 MBS P&L Analytics - AWS Bedrock Edition</h1>
        <p>Advanced trading insights powered by Claude on AWS</p>
    </div>
    """, unsafe_allow_html=True)

    # Recreate your dashboard metrics
    if 'pnl_df' in globals() and not pnl_df.empty:
        col1, col2, col3 = st.columns(3)

        winners = pnl_df[pnl_df['TotalUnrealizedPnL'] > 0]
        losers = pnl_df[pnl_df['TotalUnrealizedPnL'] < 0]

        with col1:
            st.metric(
                "📈 Winning Positions",
                f"{len(winners)}",
                f"${winners['TotalUnrealizedPnL'].sum():,.2f}"
            )

        with col2:
            st.metric(
                "📉 Losing Positions",
                f"{len(losers)}",
                f"${losers['TotalUnrealizedPnL'].sum():,.2f}"
            )

        with col3:
            st.metric(
                "💼 Net P&L",
                f"${pnl_df['TotalUnrealizedPnL'].sum():,.2f}",
                f"{pnl_df['PnL_Percent'].mean():.2f}%"
            )

    # Query interface (same as your Colab version)
    st.subheader("🤖 AI Trading Analysis")

    sample_queries = [
        "Should I hold position POS0001 for next 3-6 months?",
        "Which positions have best risk-adjusted returns?",
        "Analyze carry vs duration risk for next 6 months",
        "Project P&L if rates rise 50bps over 3 months"
    ]

    selected_query = st.selectbox("Sample queries:", [""] + sample_queries)

    user_query = st.text_area(
        "Enter your analysis question:",
        value=selected_query,
        height=100
    )

    if st.button("🔍 Analyze with Bedrock Claude", type="primary"):
        if user_query.strip():
            with st.spinner("Analyzing with AWS Bedrock..."):
                analyzer = AIAnalyzer()
                response = analyzer.analyze_mbs_data(pnl_data, user_query)

                st.subheader("📊 Analysis Results")
                st.markdown(response)
        else:
            st.warning("Please enter a query to analyze.")

# To run: streamlit run your_script.py
if __name__ == "__main__":
    streamlit_mbs_interface()
```

Phase 3: Production Enhancement (Week 3-4)

Day 15-18: Error Handling & Robustness

python

```
class RobustBedrockService:
    """Production-ready Bedrock service with error handling"""

    def __init__(self, region='us-east-1', retry_attempts=3):
        self.region = region
        self.retry_attempts = retry_attempts
        self.client = None
        self._initialize_client()

    def _initialize_client(self):
        """Initialize Bedrock client with error handling"""
        try:
            self.client = boto3.client('bedrock-runtime', region_name=self.region)
            # Test the connection
            self.client.list_foundation_models()
            print(f"✅ Connected to Bedrock in {self.region}")
        except Exception as e:
            print(f"❌ Failed to connect to Bedrock: {e}")
            raise

    def call_claude_with_retry(self, prompt, model='sonnet', max_tokens=2000):
        """Call Claude with retry logic and error handling"""

        model_ids = {
            'haiku': "anthropic.claude-3-haiku-20240307-v1:0",
            'sonnet': "anthropic.claude-3-sonnet-20240229-v1:0",
            'sonnet-3.5': "anthropic.claude-3-5-sonnet-20240620-v1:0"
        }

        for attempt in range(self.retry_attempts):
            try:
                body = json.dumps({
                    "anthropic_version": "bedrock-2023-05-31",
                    "max_tokens": max_tokens,
                    "messages": [{"role": "user", "content": prompt}],
                    "temperature": 0.1
                })

                response = self.client.invoke_model(
                    modelId=model_ids.get(model, model_ids['sonnet']),
                    body=body
                )
                result = json.loads(response.get('body').read())
                return result['content'][0]['text']

            except ClientError as e:
                error_code = e.response['Error']['Code']

                if error_code == 'ThrottlingException':
                    wait_time = 2 ** attempt  # Exponential backoff
                    print(f"Rate limited, waiting {wait_time}s before retry {attempt+1}")
                    time.sleep(wait_time)
                    continue
                elif error_code == 'AccessDeniedException':
                    return f"❌ Access denied. Check model permissions in Bedrock console."
                else:
                    return f"❌ AWS Error: {e.response['Error']['Message']}"

            except Exception as e:
                if attempt == self.retry_attempts - 1:
                    return f"❌ Unexpected error after {self.retry_attempts} attempts: {str(e)}"
                time.sleep(1)

        return "❌ Failed after all retry attempts"

    def analyze_mbs_with_context(self, pnl_data, user_query, context=None):
        """Enhanced analysis with better context management"""

        # Prepare structured context
        portfolio_summary = {
            "total_positions": len(pnl_data),
            "total_unrealized_pnl": sum(pos.get('TotalUnrealizedPnL', 0) for pos in pnl_data),
            "avg_duration": np.mean([pos.get('WAM_Months', 0) for pos in pnl_data]) / 12,
            "major_positions": sorted(pnl_data,
                                    key=lambda x: abs(x.get('TotalUnrealizedPnL', 0)),
                                    reverse=True)[:5]
        }

        enhanced_prompt = f"""
        You are a senior MBS trader with 15+ years experience. Analyze this portfolio:

        PORTFOLIO SUMMARY:
        - Total Positions: {portfolio_summary['total_positions']}
        - Net Unrealized P&L: ${portfolio_summary['total_unrealized_pnl']:,.2f}
        - Average Duration: {portfolio_summary['avg_duration']:.1f} years
        TOP 5 POSITIONS BY P&L IMPACT:
        {json.dumps(portfolio_summary['major_positions'], indent=2)}

        MARKET CONTEXT:
        {context or "Current market conditions should be considered."}

        TRADER QUESTION: {user_query}

        PROVIDE:
        1. Direct answer to the question
        2. Risk assessment
        3. Specific action recommendations
        4. Timeline for decisions
        5. Key metrics to monitor

        Format with clear sections and bullet points for easy reading.
        """

        return self.call_claude_with_retry(enhanced_prompt, model='sonnet')

# Usage example
bedrock_service = RobustBedrockService()
analysis = bedrock_service.analyze_mbs_with_context(
    pnl_data.to_dict('records'),
    "Should I reduce my duration risk before the next Fed meeting?",
    context="Fed meeting in 2 weeks, inflation data trending down"
)
```

Day 19-21: Performance Optimization

```
python
```

```python
import functools
import time
from typing import Dict, Any

class OptimizedMBSAnalytics:
    """Performance-optimized version for production use"""

    def __init__(self):
        self.bedrock_service = RobustBedrockService()
        self.cache = {}
        self.cache_ttl = 300  # 5 minutes

    @functools.lru_cache(maxsize=100)
    def calculate_pnl_cached(self, data_hash: str):
        """Cache P&L calculations to avoid recomputing"""
        return self._calculate_pnl_internal()

    def _calculate_pnl_internal(self):
        """Internal P&L calculation - same as your original"""
        # Your existing calculate_pnl_metrics() function
        return calculate_pnl_metrics()

    def get_cached_analysis(self, query: str, data_signature: str) -> str:
        """Check cache before calling AI"""
        cache_key = f"{query}:{data_signature}"

        if cache_key in self.cache:
            cached_result, timestamp = self.cache[cache_key]
            if time.time() - timestamp < self.cache_ttl:
                return f"🔄 [Cached] {cached_result}"

        # Not cached or expired, make new request
        result = self.bedrock_service.analyze_mbs_with_context(
            self.pnl_data.to_dict('records'),
            query
        )

        # Cache the result
        self.cache[cache_key] = (result, time.time())
        return result

    def batch_analyze_positions(self, position_ids: list) -> Dict[str, Any]:
        """Analyze multiple positions efficiently"""
        batch_prompt = f"""
        Analyze these MBS positions and rank them by:
        1. Risk-adjusted return potential
        2. Duration risk exposure
        3. Liquidity considerations
        4. Hold vs sell recommendation

        Positions: {position_ids}

        Portfolio data: {json.dumps(self.pnl_data.to_dict('records'), indent=2)}

        Provide a ranked table with specific recommendations for each.
        """

        return self.bedrock_service.call_claude_with_retry(
            batch_prompt,
            model='sonnet-3.5',  # Use most capable model for complex analysis
            max_tokens=4000
        )

# Performance testing
analytics = OptimizedMBSAnalytics()

# Test caching
start_time = time.time()
result1 = analytics.get_cached_analysis("What's my biggest risk?", "data_v1")
print(f"First call: {time.time() - start_time:.2f}s")

start_time = time.time()
result2 = analytics.get_cached_analysis("What's my biggest risk?", "data_v1")
print(f"Cached call: {time.time() - start_time:.2f}s")
```

Phase 4: Deployment Options (Week 5)

Day 22-24: Local Development Setup

python

```
# Create: requirements.txt
"""
boto3>=1.34.0
pandas>=2.0.0
numpy>=1.24.0
streamlit>=1.28.0
python-dotenv>=1.0.0
jupyter>=1.0.0
ipywidgets>=8.0.0
"""

# Create: .env file for local development
"""
AWS_DEFAULT_REGION=us-east-1
AWS_PROFILE=default
STREAMLIT_SERVER_PORT=8501
CACHE_TTL_SECONDS=300
"""

# Create: config.py
import os
from dotenv import load_dotenv

load_dotenv()

class Config:
    AWS_REGION = os.getenv('AWS_DEFAULT_REGION', 'us-east-1')
    STREAMLIT_PORT = int(os.getenv('STREAMLIT_SERVER_PORT', 8501))
    CACHE_TTL = int(os.getenv('CACHE_TTL_SECONDS', 300))

    # Model configurations
    BEDROCK_MODELS = {
        'fast': "anthropic.claude-3-haiku-20240307-v1:0",
        'balanced': "anthropic.claude-3-sonnet-20240229-v1:0",
        'advanced': "anthropic.claude-3-5-sonnet-20240620-v1:0"
    }

    DEFAULT_MODEL = 'balanced'

# Create: run_local.py
#!/usr/bin/env python3

import subprocess
import sys
from config import Config

def setup_environment():
    """Setup local development environment"""
    print("🚀 Setting up MBS Analytics environment...")

    # Check AWS credentials
    try:
        import boto3
        sts = boto3.client('sts')
        identity = sts.get_caller_identity()
        print(f"✅ AWS credentials configured for account: {identity['Account']}")
    except Exception as e:
        print(f"❌ AWS credentials not configured: {e}")
        sys.exit(1)

    # Check Bedrock access
    try:
        bedrock = boto3.client('bedrock', region_name=Config.AWS_REGION)
        models = bedrock.list_foundation_models()
        claude_models = [m for m in models['modelSummaries'] if 'claude' in m['modelId']]
        print(f"✅ Found {len(claude_models)} Claude models available")
    except Exception as e:
        print(f"❌ Bedrock access issue: {e}")
        sys.exit(1)

    print("🎉 Environment setup complete!")
    return True

def run_streamlit():
    """Run Streamlit application"""
    if setup_environment():
        print(f"🌐 Starting Streamlit on port {Config.STREAMLIT_PORT}")
        subprocess.run([
            'streamlit', 'run', 'mbs_analytics_app.py',
            '--server.port', str(Config.STREAMLIT_PORT),
            '--server.address', 'localhost'
        ])

if __name__ == "__main__":
    run_streamlit()
```

Day 25-28: Cloud Deployment

```bash
# Option 1: Simple EC2 Deployment

# 1. Launch EC2 instance (t3.medium recommended)
# 2. Install dependencies
sudo yum update -y
sudo yum install -y python3 python3-pip git

# 3. Clone your code
git clone https://github.com/your-repo/mbs-analytics.git
cd mbs-analytics

# 4. Install requirements
pip3 install -r requirements.txt

# 5. Configure AWS credentials (using IAM role recommended)
# Attach policy: AmazonBedrockFullAccess to EC2 instance role

# 6. Run application
python3 run_local.py

# 7. Access via public IP on port 8501
# Configure security group to allow port 8501 from your IP
```

```dockerfile
```

```
# Option 2: Docker Deployment

# Dockerfile
FROM python:3.9-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements and install Python deps
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 8501

# Health check
HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health || exit 1

# Run application
CMD ["streamlit", "run", "mbs_analytics_app.py", "--server.address=0.0.0.0", "--server.port=8501"]

# Build and run
# docker build -t mbs-analytics .
# docker run -p 8501:8501 -e AWS_DEFAULT_REGION=us-east-1 mbs-analytics
```

yaml

```
# Option 3: AWS ECS Deployment

# docker-compose.yml
version: '3.8'
services:
  mbs-analytics:
    build: .
    ports:
      - "8501:8501"
    environment:
      - AWS_DEFAULT_REGION=us-east-1
    deploy:
      resources:
        limits:
          memory: 2G
        reservations:
          memory: 1G

# ecs-task-definition.json
{
  "family": "mbs-analytics",
  "taskRoleArn": "arn:aws:iam::ACCOUNT:role/ECSTaskRole",
  "executionRoleArn": "arn:aws:iam::ACCOUNT:role/ECSExecutionRole",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "512",
  "memory": "2048",
  "containerDefinitions": [
    {
      "name": "mbs-analytics",
      "image": "ACCOUNT.dkr.ecr.us-east-1.amazonaws.com/mbs-analytics:latest",
      "portMappings": [
        {
          "containerPort": 8501,
          "protocol": "tcp"
        }
      ],
      "environment": [
        {
          "name": "AWS_DEFAULT_REGION",
          "value": "us-east-1"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/mbs-analytics",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ]
}
```

Migration Checklist

Pre-Migration (Complete Before Starting)

□ AWS account created and billing configured
□ AWS CLI installed and configured locally
□ Bedrock service enabled and models requested
□ Basic understanding of AWS IAM roles and policies
□ Local development environment setup (Python 3.8+)

Week 1: Foundation

□ Successfully call Bedrock API from local machine
□ Migrate P&L calculation functions (no AI yet)
□ Test data loading and processing
□ Understand token usage and costs

Week 2: AI Integration

□ Replace Gemini calls with Bedrock calls
□ Test different Claude models for performance
□ Implement error handling and retry logic
□ Create caching mechanism for repeated queries

Week 3: Interface Migration

□ Choose UI framework (Streamlit recommended)
□ Recreate your dashboard and metrics
□ Implement user query interface
□ Test end-to-end user workflows

Week 4: Production Readiness

□ Add comprehensive error handling
□ Implement logging and monitoring
□ Optimize performance and caching
□ Create deployment scripts

Week 5: Deployment

□ Deploy to chosen platform (EC2, ECS, etc.)
□ Configure production security (IAM roles, VPC)
□ Set up monitoring and alerts
□ Document the system for users

Key Success Metrics

Technical Metrics:

- Response time < 5 seconds for typical queries
- 99%+ uptime for deployed application
- Cost < $50/month for moderate usage (100 queries/day)
- Zero security incidents

Business Metrics:

- Users can get same insights as Google Colab version
- Ability to handle confidential trading data securely
- Easy to modify based on trader feedback
- Scalable to team usage

This comprehensive approach ensures you'll have deep understanding of both the technical implementation and the business context, making you capable of quick modifications based on user requirements!