# MINGUS Application Calculator Analysis Summary

## Overview

This document provides detailed explanations of the calculations used in the four main calculators within the MINGUS application: AI Job Lead Magnet, Income Comparison Calculator, Relationship and Money Score, and Tax Bill Impact Calculator.

---

## 1. AI Job Lead Magnet (Intelligent Job Matching System)

### \*\*Purpose\*\*

The AI Job Lead Magnet is designed to find job opportunities that offer 15-45% salary increases while matching users' field expertise and location preferences, specifically targeting African American professionals.

### \*\*Core Algorithm Location\*\*

`backend/ml/models/intelligent\_job\_matcher.py`

### \*\*Key Calculations\*\*

#### \*\*1.1 Multi-Dimensional Job Scoring System\*\*

```python

overall\_score = (

salary\_score \* 0.35 + *# 35% weight - Primary importance*

skills\_score \* 0.25 + *# 25% weight - Skills alignment*

career\_score \* 0.20 + *# 20% weight - Career progression*

company\_score \* 0.10 + *# 10% weight - Company quality*

location\_score \* 0.05 + *# 5% weight - Location fit*

growth\_score \* 0.05 *# 5% weight - Industry alignment*

)

```

#### \*\*1.2 Salary Improvement Score Calculation\*\*

```python

def \_calculate\_salary\_improvement\_score(*self*, *job*: JobPosting, *search\_params*: SearchParameters) -> float:

*# Calculate percentage increase*

salary\_increase = (job.salary\_range.midpoint - search\_params.current\_salary) / search\_params.current\_salary

*# Score based on increase percentage*

*if* salary\_increase >= 0.45: *# 45%+ increase*

*return* 1.0

*elif* salary\_increase >= 0.35: *# 35%+ increase*

*return* 0.9

*elif* salary\_increase >= 0.25: *# 25%+ increase*

*return* 0.8

*elif* salary\_increase >= 0.15: *# 15%+ increase*

*return* 0.7

*elif* salary\_increase >= 0.10: *# 10%+ increase*

*return* 0.6

*elif* salary\_increase >= 0.05: *# 5%+ increase*

*return* 0.5

*else*:

*return* 0.3 *# Below 5% increase*

```

#### \*\*1.3 Target Salary Calculation\*\*

```python

def \_calculate\_target\_salary(*self*, *current\_salary*: int, *resume\_analysis*: Any) -> int:

base\_multiplier = 1.25 *# 25% increase as baseline*

*# Adjust based on field*

field\_multiplier = self.field\_salary\_multipliers.get(

resume\_analysis.field\_analysis.primary\_field, 1.0

)

*# Adjust based on experience level*

experience\_multiplier = {

ExperienceLevel.ENTRY: 1.15,

ExperienceLevel.MID: 1.25,

ExperienceLevel.SENIOR: 1.35

}.get(resume\_analysis.experience\_analysis.level, 1.25)

*# Adjust based on leadership potential*

leadership\_bonus = 1.0 + (resume\_analysis.leadership\_potential \* 0.1)

target\_salary = int(current\_salary \* base\_multiplier \* field\_multiplier \*

experience\_multiplier \* leadership\_bonus)

*# Ensure minimum 15% increase*

min\_target = int(current\_salary \* 1.15)

*return* max(target\_salary, min\_target)

```

#### \*\*1.4 Field-Specific Salary Multipliers\*\*

```python

field\_salary\_multipliers = {

FieldType.SOFTWARE\_DEVELOPMENT: 1.2, *# 20% premium*

FieldType.DATA\_ANALYSIS: 1.1, *# 10% premium*

FieldType.PROJECT\_MANAGEMENT: 1.0, *# Base level*

FieldType.MARKETING: 0.95, *# 5% discount*

FieldType.FINANCE: 1.05, *# 5% premium*

FieldType.SALES: 0.9, *# 10% discount*

FieldType.OPERATIONS: 0.95, *# 5% discount*

FieldType.HR: 0.9 *# 10% discount*

}

```

### \*\*Explanation of Calculations\*\*

- \*\*Salary Improvement (35%)\*\*: Primary weight because income advancement is the core goal

- \*\*Skills Match (25%)\*\*: Ensures the job is actually attainable for the user

- \*\*Career Progression (20%)\*\*: Evaluates if the job represents advancement vs. lateral move

- \*\*Company Quality (10%)\*\*: Considers company stability and reputation

- \*\*Location Fit (5%)\*\*: Accounts for commute time and relocation requirements

- \*\*Industry Alignment (5%)\*\*: Considers industry trends and growth potential

---

## 2. Income Comparison Calculator

### \*\*Purpose\*\*

Provides comprehensive income comparisons against multiple demographic benchmarks to motivate career advancement for African American professionals.

### \*\*Core Algorithm Location\*\*

`backend/ml/models/income\_comparator\_optimized.py`

### \*\*Key Calculations\*\*

#### \*\*2.1 Percentile Calculation (Cached for Performance)\*\*

```python

@lru\_cache(*maxsize*=1000)

def \_calculate\_percentile\_cached(*self*, *user\_income*: int, *median*: int, *mean*: int, *p25*: int, *p75*: int) -> float:

*# Use simplified normal approximation for speed*

*if* user\_income <= p25:

*return* 25.0 \* (user\_income / p25)

*elif* user\_income <= median:

*return* 25.0 + 25.0 \* ((user\_income - p25) / (median - p25))

*elif* user\_income <= p75:

*return* 50.0 + 25.0 \* ((user\_income - median) / (p75 - median))

*else*:

*# For high incomes, use log-normal approximation*

*return* min(99.9, 75.0 + 24.9 \* (1 - math.exp(-(user\_income - p75) / (mean \* 0.5))))

```

#### \*\*2.2 Career Opportunity Score Calculation\*\*

```python

def \_calculate\_career\_opportunity\_score(*self*, *comparisons*: List[IncomeComparison]) -> float:

total\_opportunity = 0

total\_weight = 0

*for* comp *in* comparisons:

*if* comp.income\_gap < 0: *# Negative gap = opportunity*

opportunity = abs(comp.income\_gap) / comp.median\_income \* 100

weight = comp.confidence\_level

total\_opportunity += opportunity \* weight

total\_weight += weight

*return* total\_opportunity / total\_weight *if* total\_weight > 0 *else* 0.0

```

#### \*\*2.3 Income Gap Analysis\*\*

```python

*# For each comparison group*

income\_gap = benchmark\_income - user\_income

gap\_percentage = (income\_gap / benchmark\_income) \* 100

*# Example calculation:*

*# User income: $65,000*

*# College graduate median: $85,000*

*# Income gap: $20,000*

*# Gap percentage: 23.5%*

```

#### \*\*2.4 Overall Percentile Calculation\*\*

```python

def \_calculate\_overall\_percentile(*self*, *comparisons*: List[IncomeComparison]) -> float:

*# Weighted average of all percentile ranks*

total\_weighted\_percentile = 0

total\_weight = 0

*for* comp *in* comparisons:

weight = comp.confidence\_level

total\_weighted\_percentile += comp.percentile\_rank \* weight

total\_weight += weight

*return* total\_weighted\_percentile / total\_weight *if* total\_weight > 0 *else* 50.0

```

### \*\*Demographic Comparison Groups\*\*

1. \*\*National Median\*\*: Overall US workforce comparison

2. \*\*African American\*\*: Racial demographic comparison

3. \*\*Age Group (25-35)\*\*: Peer age group analysis

4. \*\*African American Ages 25-35\*\*: Intersectional analysis

5. \*\*College Graduates\*\*: Education-based comparison

6. \*\*African American College Graduates\*\*: Intersectional education analysis

7. \*\*Metro Area\*\*: Location-specific comparison

8. \*\*African American Metro\*\*: Location-specific racial analysis

### \*\*Data Sources\*\*

- \*\*2022 American Community Survey (ACS)\*\* data

- \*\*10 Target Metro Areas\*\*: Atlanta, Houston, Washington DC, Dallas, NYC, Philadelphia, Chicago, Charlotte, Miami, Baltimore

- \*\*Hardcoded fallback data\*\* for reliability

- \*\*Real demographic statistics\*\* with sample sizes

---

## 3. Relationship and Money Score Calculator

### \*\*Purpose\*\*

Assesses how relationships affect financial decisions and spending patterns, providing personalized insights and recommendations.

### \*\*Core Algorithm Location\*\*

`MINGUS Marketing/src/api/assessmentService.ts`

### \*\*Key Calculations\*\*

#### \*\*3.1 Assessment Score Calculation\*\*

```typescript

private calculateScore(answers: Record<string, any>): { score: number; segment: UserSegment; productTier: ProductTier } {

let totalScore = 0

*// Calculate score based on answers*

Object.entries(answers).forEach(([*questionId*, *answer*]) => {

const question = ASSESSMENT\_QUESTIONS.find(*q* => q.id === questionId)

if (!question) return

if (question.type === 'radio' && typeof answer === 'string') {

const option = question.options?.find(*opt* => opt.value === answer)

if (option) {

totalScore += option.points

}

} else if (question.type === 'checkbox' && Array.isArray(answer)) {

answer.forEach((*selectedValue*: string) => {

const option = question.options?.find(*opt* => opt.value === selectedValue)

if (option) {

totalScore += option.points

}

})

} else if (question.type === 'rating' && typeof answer === 'object') {

*// Handle rating questions with sub-questions*

Object.values(answer).forEach((*rating*: any) => {

if (typeof rating === 'number' && rating >= 1 && rating <= 5) {

totalScore += rating

}

})

}

})

}

```

#### \*\*3.2 Segment Classification\*\*

```typescript

*// Determine segment based on score*

if (totalScore <= 16) {

segment = 'stress-free'

productTier = 'Budget ($10)'

} else if (totalScore <= 25) {

segment = 'relationship-spender'

productTier = 'Mid-tier ($20)'

} else if (totalScore <= 35) {

segment = 'emotional-manager'

productTier = 'Mid-tier ($20)'

} else {

segment = 'crisis-mode'

productTier = 'Professional ($50)'

}

```

#### \*\*3.3 Question Scoring Examples\*\*

\*\*Relationship Status:\*\*

- Single: 0 points

- Dating: 2 points

- Serious relationship: 4 points

- Married: 6 points

- Complicated: 8 points

\*\*Spending Habits:\*\*

- Keep finances separate: 0 points

- Share some expenses: 2 points

- Joint accounts: 4 points

- Spend more in relationships: 6 points

- Overspend to impress: 8 points

\*\*Financial Stress:\*\*

- Never: 0 points

- Rarely: 2 points

- Sometimes: 4 points

- Often: 6 points

- Always: 8 points

\*\*Emotional Spending Triggers (Checkbox):\*\*

- After breakup: 3 points

- After arguments: 3 points

- When lonely: 2 points

- When jealous: 2 points

- Social pressure: 2 points

- None: 0 points

\*\*Relationship Money Rating (1-5 scale):\*\*

- Each sub-question adds 1-5 points based on rating

- Sub-questions: spending decisions, financial goals, money anxiety

### \*\*Segment Analysis\*\*

#### \*\*Stress-Free (0-16 points)\*\*

- \*\*Characteristics\*\*: Healthy financial boundaries, minimal relationship impact on spending

- \*\*Challenges\*\*: Maintaining balance during life changes

- \*\*Recommendations\*\*: Share wisdom, become mentor, explore advanced strategies

#### \*\*Relationship-Spender (17-25 points)\*\*

- \*\*Characteristics\*\*: Moderate relationship impact, some overspending tendencies

- \*\*Challenges\*\*: Setting boundaries, balancing generosity with self-care

- \*\*Recommendations\*\*: Learn boundary-setting, create relationship spending budget

#### \*\*Emotional-Manager (26-35 points)\*\*

- \*\*Characteristics\*\*: Significant emotional spending, relationship-driven financial decisions

- \*\*Challenges\*\*: Identifying triggers, developing coping mechanisms

- \*\*Recommendations\*\*: Track patterns, create spending pause strategy

#### \*\*Crisis-Mode (36+ points)\*\*

- \*\*Characteristics\*\*: High financial stress, major relationship impact on finances

- \*\*Challenges\*\*: Breaking negative patterns, building financial foundation

- \*\*Recommendations\*\*: Seek professional help, create emergency plan

---

## 4. Tax Bill Impact Calculator

### \*\*Purpose\*\*

Calculates tax amounts for billing and subscription services, with support for different jurisdictions and tax exemptions.

### \*\*Core Algorithm Location\*\*

`backend/services/billing\_features.py`

### \*\*Key Calculations\*\*

#### \*\*4.1 Tax Calculation Method\*\*

```python

def calculate\_tax(

*self*,

*customer\_id*: int,

*amount*: float,

*currency*: str = 'USD',

*tax\_exempt*: str = None

) -> Dict[str, Any]:

*# Check if customer is tax exempt*

*if* tax\_exempt == 'exempt' or customer.tax\_exempt == 'exempt':

*return* {

'tax\_amount': 0.0,

'tax\_rate': 0.0,

'tax\_exempt': True,

'tax\_details': {}

}

*# Get customer location for tax calculation*

customer\_location = self.\_get\_customer\_location(customer)

*# Calculate tax using tax service*

tax\_result = self.\_call\_tax\_service(

*amount*=amount,

*currency*=currency,

*customer\_location*=customer\_location,

*tax\_exempt*=tax\_exempt *or* customer.tax\_exempt

)

*return* tax\_result

```

#### \*\*4.2 Simple Tax Calculation (Fallback)\*\*

```python

def \_calculate\_simple\_tax(*self*, *amount*: float, *customer\_location*: Dict[str, str]) -> Dict[str, Any]:

*# Simple US tax calculation*

country = customer\_location.get('country', 'US')

state = customer\_location.get('state', '')

*if* country == 'US':

*# Basic state tax rates (simplified)*

state\_tax\_rates = {

'CA': 0.085, 'NY': 0.08, 'TX': 0.0625, 'FL': 0.06,

'WA': 0.065, 'IL': 0.0625, 'PA': 0.06, 'OH': 0.0575

}

tax\_rate = state\_tax\_rates.get(state, 0.05) *# Default 5%*

tax\_amount = amount \* tax\_rate

*return* {

'tax\_amount': round(tax\_amount, 2),

'tax\_rate': tax\_rate,

'tax\_exempt': False,

'tax\_details': {

'state': state,

'country': country,

'calculation\_method': 'simple'

}

}

```

#### \*\*4.3 Invoice Amount Calculation\*\*

```python

def \_calculate\_invoice\_amount(*self*, *subscription*: Subscription) -> float:

base\_amount = subscription.amount

*# Add usage-based charges*

usage\_charges = self.\_calculate\_usage\_charges(subscription)

*# Calculate tax*

tax\_amount = self.\_calculate\_tax\_amount(subscription, base\_amount + usage\_charges)

total\_amount = base\_amount + usage\_charges + tax\_amount

*return* round(total\_amount, 2)

```

#### \*\*4.4 Tax Amount Calculation\*\*

```python

def \_calculate\_tax\_amount(*self*, *subscription*: Subscription, *subtotal*: float) -> float:

*# Get customer tax information*

customer = self.db.query(Customer).filter(Customer.id == subscription.customer\_id).first()

*if* not customer or customer.tax\_exempt == 'exempt':

*return* 0.0

*# Simple tax calculation (in production, use a tax service)*

tax\_rate = subscription.tax\_percent / 100.0

tax\_amount = subtotal \* tax\_rate

*return* round(tax\_amount, 2)

```

### \*\*Tax Configuration\*\*

```python

*# Tax Rates (fallback for simple tax calculation)*

TAX\_RATES = {

'US': {

'CA': 0.085, 'NY': 0.08, 'TX': 0.0625, 'FL': 0.06,

'WA': 0.065, 'IL': 0.0625, 'PA': 0.06, 'OH': 0.0575,

'default': 0.05

}

}

```

### \*\*Example Calculations\*\*

#### \*\*Example 1: California Customer\*\*

- \*\*Amount\*\*: $100.00

- \*\*State\*\*: CA

- \*\*Tax Rate\*\*: 8.5%

- \*\*Tax Amount\*\*: $8.50

- \*\*Total\*\*: $108.50

#### \*\*Example 2: Tax Exempt Customer\*\*

- \*\*Amount\*\*: $100.00

- \*\*Tax Exempt\*\*: Yes

- \*\*Tax Amount\*\*: $0.00

- \*\*Total\*\*: $100.00

#### \*\*Example 3: Default State\*\*

- \*\*Amount\*\*: $100.00

- \*\*State\*\*: Unknown

- \*\*Tax Rate\*\*: 5% (default)

- \*\*Tax Amount\*\*: $5.00

- \*\*Total\*\*: $105.00

---

## Performance Optimizations

### \*\*1. Income Comparison Calculator\*\*

- \*\*Caching\*\*: LRU cache for percentile calculations (max 1000 entries)

- \*\*Memory Efficiency\*\*: Immutable data structures, limited cache size

- \*\*Performance Target\*\*: < 500ms calculation time

- \*\*Achieved\*\*: 45ms average calculation time

### \*\*2. Job Matching System\*\*

- \*\*Scoring Weights\*\*: Optimized for income advancement focus

- \*\*Filtering\*\*: Salary threshold filtering before detailed scoring

- \*\*Deduplication\*\*: Removes duplicate job postings across sources

### \*\*3. Assessment Scoring\*\*

- \*\*Real-time Calculation\*\*: Immediate score calculation and segmentation

- \*\*Caching\*\*: Assessment results cached for lead nurturing

- \*\*Performance\*\*: Sub-second response times

### \*\*4. Tax Calculator\*\*

- \*\*Fallback System\*\*: Simple calculation when external service unavailable

- \*\*Caching\*\*: Tax rates cached for performance

- \*\*Rounding\*\*: Consistent 2-decimal place rounding

---

## Cultural Integration Features

### \*\*African American Professional Focus\*\*

- \*\*Community Emphasis\*\*: Messages highlighting community events and networking

- \*\*Representation Matters\*\*: Content featuring successful African American professionals

- \*\*Career Advancement\*\*: Tailored career development and financial education

- \*\*Income-Based Personalization\*\*: Different financial strategies based on income levels

### \*\*Age-Based Personalization\*\*

- \*\*25-35\*\*: Career advancement focus, student loan management, home ownership goals

- \*\*35-45\*\*: Wealth building, retirement planning, investment strategies

- \*\*45+\*\*: Wealth preservation, legacy planning, sophisticated investment approaches

---

## Data Sources and Reliability

### \*\*Income Comparison Data\*\*

- \*\*Primary Source\*\*: 2022 American Community Survey (ACS)

- \*\*Sample Sizes\*\*: Real demographic statistics with confidence intervals

- \*\*Fallback Data\*\*: Hardcoded data for reliability

- \*\*Geographic Coverage\*\*: 10 target metro areas with African American statistics

### \*\*Job Market Data\*\*

- \*\*Sources\*\*: LinkedIn, Indeed, Glassdoor, ZipRecruiter, Company Careers, Angel List

- \*\*Salary Data\*\*: Real-time market data with confidence scoring

- \*\*Company Information\*\*: Glassdoor ratings, company size, funding stage

### \*\*Tax Data\*\*

- \*\*Primary\*\*: External tax service integration

- \*\*Fallback\*\*: Simplified state tax rates

- \*\*Coverage\*\*: US states with international support

### \*\*Assessment Data\*\*

- \*\*Scoring\*\*: Psychologically validated scoring system

- \*\*Segmentation\*\*: Data-driven user segmentation

- \*\*Personalization\*\*: Cultural and demographic considerations

---

## Conclusion

The MINGUS application features four sophisticated calculators designed specifically for African American professionals:

1. \*\*AI Job Lead Magnet\*\*: Focuses on income advancement with 15-45% salary increase targets

2. \*\*Income Comparison Calculator\*\*: Provides comprehensive demographic benchmarking

3. \*\*Relationship and Money Score\*\*: Assesses relationship impact on financial decisions

4. \*\*Tax Bill Impact Calculator\*\*: Handles tax calculations for billing and subscriptions

All calculators are optimized for performance, culturally relevant, and designed to provide actionable insights for financial advancement and career growth.