Fuel Consumption Analysis for 5m Express

Project Overview

This project analyzes the relationship between fuel consumption and various factors for 5m Express, a ride-sharing company operating in Accra and Kumasi. The analysis focuses on understanding non-linear relationships between distance, passenger load, and fuel consumption to help optimize route planning and reduce operational costs.

Project Structure

```
project-1/
├── fuel_consumption_analysis.py    # Main analysis script
├── fuel_analysis_notebook.ipynb    # Jupyter notebook for interactive analysis
├── requirements.txt                 # Python dependencies
├── README.md                       # This file
└── Generated Files/                # Output files (created when running analysis)
    ├── correlation_matrix.png
    ├── variable_distributions.png
    ├── relationship_analysis.png
    ├── 3d_relationship.html
    ├── model_comparison.png
    ├── residual_plots.png
    └── feature_importance.png
```

Requirements

- Python 3.8+
- All dependencies listed in `requirements.txt`

Installation

1. Clone or download this project
2. Install dependencies:

```
pip install -r requirements.txt
```

Usage

Option 1: Run the main script

```
python fuel_consumption_analysis.py
```

Option 2: Use Jupyter Notebook

```
jupyter notebook fuel_analysis_notebook.ipynb
```

Test Simulation Feature

What is the Test Simulation?

The test simulation is like a "what-if" calculator that tests different real-world scenarios to see how much fuel a trip will use and whether it will be profitable. It's like playing with different scenarios to find the best way to run the business.

How the Simulation Works

The simulation tests 6 different real-world scenarios:

1. **Peak Hour Commute**: Busy morning rush hour with full car
2. **Off-Peak Light Load**: Quiet time with just one passenger
3. **Long Distance Trip**: Long journey with moderate passengers
4. **Adverse Weather**: Rainy weather conditions
5. **Heavy Traffic**: Bad traffic congestion
6. **Optimal Conditions**: Perfect driving conditions

Code Example: Test Simulation

```
def run_test_simulation(self):
    """Run comprehensive test simulation with various scenarios."""

    # Define different test scenarios
    scenarios = {
        "Peak Hour Commute": {
            "distance": 25,           # 25 km trip
            "passenger_load": 4,      # 4 passengers
            "traffic_conditions": 0.9, # Heavy traffic
            "weather_conditions": 1.0, # Normal weather
            "description": "Busy morning commute with full load"
        },
        "Off-Peak Light Load": {
            "distance": 15,           # 15 km trip
            "passenger_load": 1,      # 1 passenger
            "traffic_conditions": 0.5, # Light traffic
            "weather_conditions": 1.0, # Normal weather
            "description": "Quiet period with single passenger"
        }
        # ... more scenarios
    }

    # Business parameters
    fuel_price_per_liter = 12.50  # Ghanaian Cedi
```

```
    base_fare_per_km = 2.50       # Ghanaian Cedi per km
    operational_cost_per_km = 1.20 # Ghanaian Cedi per km

    # Test each scenario
    for scenario_name, params in scenarios.items():
        # Predict fuel consumption using our models
        predicted_fuel = self.models['Polynomial Regression'].predict(...)

        # Calculate costs and revenue
        fuel_cost = predicted_fuel * fuel_price_per_liter
        operational_cost = distance * operational_cost_per_km
        total_cost = fuel_cost + operational_cost

        # Calculate revenue (more passengers = more money)
        base_revenue = distance * base_fare_per_km
        load_multiplier = 1 + (passengers - 1) * 0.3  # 30% more per extra
passenger
        total_revenue = base_revenue * load_multiplier

        # Calculate profit
        profit = total_revenue - total_cost

        # Print results
        print(f"Scenario: {scenario_name}")
        print(f"  Fuel used: {predicted_fuel:.2f} liters")
        print(f"  Cost: ₵{total_cost:.2f}")
        print(f"  Revenue: ₵{total_revenue:.2f}")
        print(f"  Profit: ₵{profit:.2f}")
```

What the Simulation Tells Us

The simulation helps us understand:

1. **Which trips are profitable**: Shows which scenarios make money vs lose money
2. **Fuel consumption patterns**: How much fuel different trips use
3. **Cost breakdown**: Separates fuel costs from other operational costs
4. **Revenue optimization**: How passenger load affects revenue
5. **Business insights**: What conditions are best for profitability

Sample Simulation Results

```
TEST SIMULATION - FUEL CONSUMPTION SCENARIOS
============================================================

Scenario             Distance   Load   Traffic  Weather  Fuel (L) Cost (₵)
Revenue (₵)  Profit (₵)
--------------------------------------------------------------------------------
Peak Hour Commute    25.0       4      0.90     1.00     11.96    149.48
118.75       -60.73
Off-Peak Light Load  15.0       1      0.50     1.00     6.30     78.78      37.50
-59.28
```

```
Long Distance Trip   45.0       3      0.70     1.10     17.21    215.09
180.00       -89.09
Adverse Weather      20.0       2      0.80     1.30     8.34     104.28     65.00
-63.28
Heavy Traffic        12.0       3      0.95     1.00     7.52     94.01      48.00
-60.41
Optimal Conditions   18.0       2      0.40     0.90     7.70     96.28      58.50
-59.38

SIMULATION SUMMARY
================================================================================
Total Fuel Consumption: 59.03 liters
Total Operational Cost: ₵899.93
Total Revenue: ₵507.75
Total Profit: ₵-392.18
Average Profit Margin: -97.2%
```

Key Insights from Simulation

1. **All scenarios are currently unprofitable** - This suggests the need for:
    - Higher fare rates
    - Better route optimization
    - Improved operational efficiency
2. **Fuel costs are high** - Fuel price of ₵12.50/liter makes trips expensive
3. **Revenue doesn't cover costs** - Current pricing model needs adjustment
4. **Traffic and weather matter** - Bad conditions increase fuel consumption significantly

How to Use the Simulation

```
# Run the complete analysis including simulation
analyzer = FuelConsumptionAnalyzer()
analyzer.generate_synthetic_data()
analyzer.build_models()
simulation_results = analyzer.run_test_simulation()

# The simulation will automatically:
# 1. Test 6 different scenarios
# 2. Calculate costs and revenue for each
# 3. Show which scenarios are profitable
# 4. Provide business recommendations
```

Project Components

1. Data Analysis (10 marks)

- **Synthetic Data Generation**: Creates realistic fuel consumption data with non-linear relationships

- **Exploratory Data Analysis**: Comprehensive analysis of variable distributions and correlations
- **Relationship Analysis**: Identifies non-linear patterns between distance, load, and fuel consumption

2. Model Development (20 marks)

- **Polynomial Regression Model**: Captures non-linear relationships with high accuracy
- **Custom Non-linear Model**: Power function model for specific fuel consumption patterns
- **Model Evaluation**: Comprehensive metrics including R^2, MSE, and RMSE
- **Prediction Tool**: Operational function for real-time fuel consumption estimation

3. Visualizations (5 marks)

- **Correlation Heatmaps**: Shows relationships between all variables
- **Distribution Plots**: Visualizes data distributions
- **Relationship Analysis**: Scatter plots with polynomial fits
- **3D Interactive Plots**: Dynamic visualization of distance vs load vs fuel consumption
- **Model Performance**: Actual vs predicted plots and residual analysis
- **Feature Importance**: Shows which factors most affect fuel consumption

4. Recommendations (10 marks)

- **Route Planning**: Optimization strategies for distance and traffic
- **Load Optimization**: Passenger distribution and vehicle utilization
- **Operational Efficiency**: Maintenance and driver training recommendations
- **Cost Management**: Dynamic pricing and fuel monitoring systems

5. Test Simulation (New Feature)

- **Scenario Testing**: Tests 6 different real-world scenarios
- **Profitability Analysis**: Shows which trips make money vs lose money
- **Cost Breakdown**: Separates fuel costs from operational costs
- **Business Insights**: Provides actionable recommendations for pricing and operations

Key Findings

1. **Non-linear Relationships**: Fuel consumption increases non-linearly with distance and passenger load
2. **Model Performance**: Polynomial regression achieves $R^2 > 0.85$
3. **Key Factors**: Distance and passenger load are the strongest predictors
4. **Environmental Impact**: Traffic and weather conditions significantly affect fuel efficiency
5. **Business Reality**: Current pricing model needs adjustment for profitability

Model Performance

Model	R^2 Score	RMSE	Description
Polynomial Regression	0.87+	< 0.5	Captures complex interactions
Custom Non-linear	0.85+	< 0.6	Power function approach

Usage Examples

```
# Example prediction
from fuel_consumption_analysis import FuelConsumptionAnalyzer

analyzer = FuelConsumptionAnalyzer()
analyzer.generate_synthetic_data()
analyzer.build_models()

# Predict fuel consumption
prediction = analyzer.predict_fuel_consumption(
    distance=25,           # km
    passenger_load=3,      # passengers
    traffic_conditions=0.7, # congestion factor
    weather_conditions=1.0  # weather factor
)
print(f"Predicted fuel consumption: {prediction:.2f} liters")

# Run test simulation
simulation_results = analyzer.run_test_simulation()
```

Recommendations Summary

Route Planning

1. Optimize routes considering traffic patterns
2. Use real-time traffic data for dynamic routing
3. Implement hub-and-spoke model for efficiency

Load Optimization

1. Balance passenger load across vehicles
2. Implement dynamic pricing based on load and distance
3. Use predictive models for optimal passenger-to-vehicle ratios

Operational Efficiency

1. Regular vehicle maintenance for fuel efficiency
2. Driver training on fuel-efficient techniques
3. Real-time monitoring of fuel consumption vs predictions

Cost Management

1. Dynamic pricing based on fuel consumption predictions
2. Fuel consumption targets and monitoring systems
3. Incentive programs for fuel-efficient operations

Business Strategy (Based on Simulation)

1. **Increase fare rates** - Current pricing doesn't cover costs
2. **Optimize routes** - Reduce distance and avoid heavy traffic

3. **Improve efficiency** - Better vehicle maintenance and driver training
4. **Dynamic pricing** - Charge more during peak hours and bad weather
5. **Load optimization** - Maximize passenger load for better profitability

Technical Details

Data Generation

- **Distance**: 5-50 km (uniform distribution)
- **Passenger Load**: 1-4 passengers (integer)
- **Traffic Conditions**: 0.3-1.0 (congestion factor)
- **Weather Conditions**: 0.8-1.2 (weather factor)
- **Sample Size**: 1000 synthetic data points

Model Features

- **Polynomial Regression**: Degree 2 polynomial features
- **Custom Model**: Power function with interaction terms
- **Cross-validation**: 80/20 train-test split
- **Metrics**: R^2, MSE, RMSE for model evaluation

Simulation Features

- **6 Real-world Scenarios**: Different trip conditions and passenger loads
- **Cost Calculation**: Fuel costs + operational costs
- **Revenue Calculation**: Base fare + passenger load multiplier
- **Profit Analysis**: Revenue minus total costs
- **Business Metrics**: Profit margins and efficiency analysis

Visualization Features

- **Static Plots**: Matplotlib and Seaborn for publication-quality figures
- **Interactive Plots**: Plotly for 3D and interactive visualizations
- **Export Formats**: PNG for static plots, HTML for interactive plots

Future Enhancements

1. **Real Data Integration**: Connect to actual 5m Express data sources
2. **Real-time Updates**: Live model updates with new data
3. **Advanced Models**: Deep learning approaches for complex patterns
4. **Mobile Integration**: API for mobile app integration
5. **Cost Optimization**: Multi-objective optimization for routes and pricing
6. **Advanced Simulation**: More scenarios and sensitivity analysis
7. **Real-time Monitoring**: Live dashboard for operational insights

Contact

For questions or contributions, please contact the data analysis team.

Code Explanation

This section explains how each part of the code works in simple terms.

1. Data Generation Code

```
def generate_synthetic_data(self, n_samples=1000):
    """Generate synthetic data that mimics real-world fuel consumption patterns"""
    np.random.seed(42)  # Makes results repeatable

    # Generate realistic parameters
    distance = np.random.uniform(5, 50, n_samples)  # Random distances between 5-
50 km
    passenger_load = np.random.randint(1, 5, n_samples)  # Random passengers 1-4
    traffic_conditions = np.random.uniform(0.3, 1.0, n_samples)  # Traffic factor
0.3-1.0
    weather_conditions = np.random.uniform(0.8, 1.2, n_samples)  # Weather factor
0.8-1.2

    # Create fuel consumption formula
    base_consumption = 2.5  # Base fuel per 100km
    distance_effect = 0.15 * distance + 0.002 * distance**2  # Fuel increases with
distance
    load_effect = 0.3 * passenger_load + 0.1 * passenger_load**2  # More
passengers = more fuel
    distance_load_interaction = 0.01 * distance * passenger_load  # Distance +
load interaction
    traffic_effect = traffic_conditions * 0.5  # Bad traffic = more fuel
    weather_effect = weather_conditions * 0.3  # Bad weather = more fuel

    # Total fuel consumption
    fuel_consumption = (base_consumption + distance_effect + load_effect +
                        distance_load_interaction + traffic_effect +
weather_effect)

    # Add realistic noise
    noise = np.random.normal(0, 0.5, n_samples)  # Random variation
    fuel_consumption += noise
    fuel_consumption = np.maximum(fuel_consumption, 0.1)  # Ensure positive values
```

What this does:

- Creates fake but realistic data for 1000 trips
- Each trip has: distance, passengers, traffic, weather
- Calculates fuel consumption using a realistic formula
- Adds random noise to make it more realistic

2. Model Building Code

```
def build_models(self):
    """Build various non-linear models for fuel consumption prediction."""
```

```
    # Prepare features and target
    X = self.data[['distance_km', 'passenger_load', 'traffic_conditions',
'weather_conditions']]
    y = self.data['fuel_consumption_liters']

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

    # Model 1: Polynomial Regression
    poly_model = Pipeline([
        ('poly', PolynomialFeatures(degree=2, include_bias=False)),  # Creates
polynomial features
        ('linear', LinearRegression())  # Fits linear regression
    ])
    poly_model.fit(X_train, y_train)  # Train the model

    # Model 2: Custom Non-linear Model
    def custom_model(X, a, b, c, d, e):
        distance, load, traffic, weather = X.T
        return a * distance**b + c * load**d + e * traffic * weather

    # Fit custom model using curve fitting
    popt, _ = curve_fit(custom_model, X_train.values.T, y_train.values)
    self.custom_params = popt
```

What this does:

- Takes the data and splits it into training (80%) and testing (20%) sets
- Creates two different models to predict fuel consumption
- **Polynomial Model**: Uses squared terms to capture curves
- **Custom Model**: Uses power functions (distance^1.2, load^1.5) for complex patterns
- Both models learn patterns from the training data

3. Test Simulation Code

```
def run_test_simulation(self):
    """Run comprehensive test simulation with various scenarios."""

    # Define test scenarios
    scenarios = {
        "Peak Hour Commute": {
            "distance": 25,
            "passenger_load": 4,
            "traffic_conditions": 0.9,
            "weather_conditions": 1.0,
            "description": "Busy morning commute with full load"
        },
        # ... more scenarios
    }
```

```
    # Business parameters
    fuel_price_per_liter = 12.50  # Ghanaian Cedi
    base_fare_per_km = 2.50       # Ghanaian Cedi per km
    operational_cost_per_km = 1.20 # Ghanaian Cedi per km

    # Test each scenario
    for scenario_name, params in scenarios.items():
        # Get predictions from both models
        poly_pred = self.models['Polynomial Regression'].predict(pd.DataFrame([{
            'distance_km': params['distance'],
            'passenger_load': params['passenger_load'],
            'traffic_conditions': params['traffic_conditions'],
            'weather_conditions': params['weather_conditions']
        }]))[0]

        # Calculate costs and revenue
        fuel_cost = poly_pred * fuel_price_per_liter
        operational_cost = params['distance'] * operational_cost_per_km
        total_cost = fuel_cost + operational_cost

        # Revenue calculation with load factor
        base_revenue = params['distance'] * base_fare_per_km
        load_multiplier = 1 + (params['passenger_load'] - 1) * 0.3  # 30% more per
passenger
        total_revenue = base_revenue * load_multiplier

        # Calculate profit
        profit = total_revenue - total_cost
        profit_margin = (profit / total_revenue) * 100 if total_revenue > 0 else 0
```

What this does:

- Tests 6 different real-world scenarios
- For each scenario:
  - Predicts fuel consumption using the trained models
  - Calculates fuel cost (fuel × price per liter)
  - Calculates operational cost (distance × cost per km)
  - Calculates revenue (distance × fare × passenger multiplier)
  - Calculates profit (revenue - total cost)

4. Visualization Code

```
def visualize_models(self):
    """Create comprehensive visualizations of the models."""

    # Model comparison plots
    fig, axes = plt.subplots(1, 2, figsize=(15, 6))

    for i, (name, result) in enumerate(self.results.items()):
        # Actual vs Predicted plots
        axes[i].scatter(self.y_test, result['predictions'], alpha=0.6, s=50)
```

```
        axes[i].plot([self.y_test.min(), self.y_test.max()],
                     [self.y_test.min(), self.y_test.max()], 'r--', linewidth=2)
        axes[i].set_title(f'{name}\nR² = {result["R²"]:.3f}')

    plt.savefig('model_comparison.png', dpi=300, bbox_inches='tight')
    plt.show()
```

What this does:

- Creates scatter plots showing actual vs predicted fuel consumption
- Perfect predictions would fall on the red diagonal line
- Shows how well each model performs
- Saves the plots as image files

5. Prediction Tool Code

```
def create_prediction_tool(self):
    """Create a prediction function for operational use."""

    def predict_fuel_consumption(distance, passenger_load, traffic_conditions=0.7,
                                 weather_conditions=1.0, model_name='Polynomial
Regression'):
        """Predict fuel consumption for given parameters."""

        # Create input array
        X_input = pd.DataFrame({
            'distance_km': [distance],
            'passenger_load': [passenger_load],
            'traffic_conditions': [traffic_conditions],
            'weather_conditions': [weather_conditions]
        })

        # Make prediction
        if model_name == 'Custom Non-linear':
            prediction = self.models[model_name](X_input)
        else:
            prediction = self.models[model_name].predict(X_input)

        return prediction[0]

    return predict_fuel_consumption
```

What this does:

- Creates a simple function that anyone can use
- Takes trip parameters (distance, passengers, traffic, weather)
- Returns predicted fuel consumption
- Can choose which model to use for prediction

6. Main Function Code

```
def main():
    """Main function to run the complete analysis."""

    # Initialize analyzer
    analyzer = FuelConsumptionAnalyzer()

    # Generate data
    analyzer.generate_synthetic_data(n_samples=1000)

    # Explore data
    analyzer.explore_data()

    # Analyze relationships
    analyzer.analyze_relationships()

    # Build models
    analyzer.build_models()

    # Visualize models
    analyzer.visualize_models()

    # Create prediction tool
    predict_func = analyzer.create_prediction_tool()

    # Generate recommendations
    recommendations = analyzer.generate_recommendations()

    # Run test simulation
    simulation_results = analyzer.run_test_simulation()

    return analyzer, predict_func, recommendations, simulation_results
```

What this does:

- Runs the complete analysis step by step
- Generates data → explores it → builds models → visualizes results → creates tools → runs simulation
- Returns all the results for further use

7. How Everything Works Together

The code follows this flow:

```
Generate Data → Explore Data → Build Models → Test Models → Visualize Results →
Run Simulation → Provide Recommendations
```

Step-by-step process:

1. **Data Generation**: Creates realistic trip data with fuel consumption patterns
2. **Data Exploration**: Analyzes relationships between variables
3. **Model Training**: Learns patterns from the data to predict fuel consumption
4. **Model Testing**: Evaluates how well the models perform
5. **Visualization**: Shows results in charts and graphs
6. **Simulation**: Tests different scenarios to see profitability
7. **Prediction Tool**: Provides easy-to-use function for real-world predictions
8. **Recommendations**: Suggests improvements based on analysis

8. Key Code Concepts Explained

Variables in the code:

- `distance_km`: How far the trip is (5-50 km)
- `passenger_load`: How many passengers (1-4 people)
- `traffic_conditions`: How bad traffic is (0.3 = good, 1.0 = bad)
- `weather_conditions`: How weather affects driving (0.8 = good, 1.2 = bad)
- `fuel_consumption_liters`: How much fuel is used

Models used:

- **Polynomial Regression**: Uses squared terms (distance^2, load^2) to capture curves
- **Custom Non-linear**: Uses power functions (distance^1.2, load^1.5) for complex patterns

Business calculations:

- **Fuel Cost**: predicted_fuel × ₵12.50 per liter
- **Operational Cost**: distance × ₵1.20 per km
- **Revenue**: distance × ₵2.50 per km × passenger_multiplier
- **Profit**: Revenue - (Fuel Cost + Operational Cost)

Why this matters:

- Helps 5m Express understand which trips are profitable
- Shows how different factors affect fuel consumption
- Provides tools for better route planning and pricing
- Identifies areas for operational improvement

Note: This project uses synthetic data to demonstrate the analysis methodology. For production use, real operational data should be used to train and validate the models. The test simulation provides valuable insights for business decision-making and helps identify areas for operational improvement in the 5m Express ride-sharing service.