Implementation Plan - AutoValuePredict ML

This document outlines the complete step-by-step implementation plan for the AutoValuePredict ML project, a machine learning system for predicting used car prices in Brazil.

Project Overview

Goal: Build an end-to-end ML pipeline that predicts the market value of used cars in Brazil, including data collection, preprocessing, feature engineering, model training, evaluation, and API deployment.

Current Status: ✅ Data collection and enrichment completed | ✅ EDA completed | ✅ Data preprocessing completed | ✅ Feature engineering completed | ✅ Baseline models completed | ✅ Advanced models completed | 🚧 Model optimization next

Development Approach: MVP-first strategy - build a functional end-to-end pipeline with essential features, then iterate and optimize.

> [!IMPORTANT] > **Data Limitations**: This project uses enriched FIPE data where features like `km` (mileage), `location`, `color`, `doors`, and `condition` are synthetically generated using statistical patterns. While realistic, these are not real-world observations. Model predictions should be validated against actual market data before production use.
>
> [!NOTE] > **Timeline**: Estimated 12-16 weeks for complete implementation. Tasks are marked as **[ESSENTIAL]** (required for MVP) or **[OPTIONAL]** (enhancements). Focus on essential tasks first to achieve a working pipeline quickly.

Phase 1: Exploratory Data Analysis (EDA) ✅ Completed

1.1 Initial Data Exploration

- [x] Load enriched datasets (`fipe_cars_enriched.csv` and `fipe_2022_enriched.csv`)
- [x] Basic data overview:
- [x] Dataset shape and memory usage
- [x] Column types and basic statistics
- [x] Missing values analysis
- [x] Duplicate records check
- [x] Create notebook: `notebooks/01_data_overview.ipynb`

1.2 Target Variable Analysis

- [x] Analyze price distribution:
- [x] Histogram and box plots
- [x] Skewness and kurtosis
- [x] Outlier detection (IQR method, Z-score)
- [x] Price ranges by vehicle category
- [x] Price trends over time (year_of_reference)
- [x] Price by brand, model, state
- [x] Create notebook: `notebooks/02_target_analysis.ipynb`

1.3 Feature Analysis

- [x] Categorical features:
- [x] Brand distribution and frequency
- [x] Model distribution
- [x] State/city distribution
- [x] Fuel type, transmission, color distributions
- [x] Condition distribution
- [x] Numerical features:
- [x] Year distribution and trends
- [x] Mileage (km) distribution and relationship with age
- [x] Engine size distribution

- [x] Doors distribution
- [x] Create notebook: `notebooks/03_feature_analysis.ipynb`

1.4 Relationships and Correlations

- [x] Correlation matrix (numerical features)
- [x] Price vs. age relationship
- [x] Price vs. mileage relationship
- [x] Price vs. brand/model analysis
- [x] Price vs. location (state) analysis
- [x] Feature interactions:
- [x] Price by brand and year
- [x] Price by fuel type and transmission
- [x] Price by condition and age
- [x] Create notebook: `notebooks/04_correlations.ipynb`

1.5 Data Quality Assessment

- [x] Identify data quality issues:
- [x] Inconsistent values
- [x] Outliers that need treatment
- [x] Missing values (if any)
- [x] Data type issues
- [x] Document findings and recommendations
- [x] Create notebook: `notebooks/05_data_quality.ipynb`

Deliverables:

- 5 Jupyter notebooks with complete EDA
- Summary document with key findings
- Data quality report

Estimated Time: 1-2 weeks

Phase 2: Data Preprocessing & Cleaning ✅ Completed

2.1 Data Cleaning Module

- [x] Create `src/data/cleaner.py`:
- [x] Remove duplicates
- [x] Handle outliers (price, km):
    - [x] IQR method for outliers
    - [x] Z-score method
    - [x] Domain knowledge-based limits
- [x] Handle missing values (if any)
- [x] Data type corrections
- [x] Standardize text fields (brand, model, city names)
- [x] Unit tests: `tests/test_cleaner.py`

2.2 Data Validation

- [x] Create `src/data/validator.py`:
- [x] Schema validation
- [x] Range checks (year, price, km)
- [x] Categorical value validation
- [x] Business rule validation
- [x] Unit tests: `tests/test_validator.py`

2.3 Data Splitting

- [x] Create `src/data/splitter.py`:
- [x] Train/validation/test split (70/15/15 or 80/10/10)
- [x] Stratified split by price ranges (optional)
- [x] Time-based split (if using year_of_reference)
- [x] Save splits to `data/processed/`
- [x] Unit tests: `tests/test_splitter.py`

Deliverables:

- ✅ Data cleaning pipeline (`src/data/cleaner.py`)

- ✅ Data validation module (`src/data/validator.py`)
- ✅ Data splitting module (`src/data/splitter.py`)
- ✅ Modular ML pipeline system (`src/pipeline/`)
- ✅ Unit tests for all modules (`tests/test_*.py`)
- ✅ Validated and cleaned datasets
- ✅ Train/validation/test splits (70/15/15)
- ✅ Pipeline execution scripts (`scripts/run_pipeline.py` , `scripts/preprocess_data.py`)
- ✅ Makefile commands for pipeline execution

Estimated Time: 1 week
Actual Time: Completed ✅

Results:

- Processed 747,948 rows (from 889,282 original)
- Train: 523,563 rows (70%)
- Validation: 112,192 rows (15%)
- Test: 112,193 rows (15%)
- All data quality checks passed ✅

Phase 3: Feature Engineering ✅ Completed

3.1 Feature Creation - Phase 1 (Essential for MVP)

- [x] Create `src/features/engineering.py` :
- [x] **[ESSENTIAL] Basic temporal features**:
  - [x] Vehicle age (already exists, verify calculation)
  - [x] Age squared (non-linear relationship)
- [x] **[ESSENTIAL] Categorical encoding**:
  - [x] One-hot encoding for low cardinality features (fuel_type, transmission, condition)
  - [x] Target encoding for high cardinality (brand, model, state)
- [x] **[ESSENTIAL] Numerical transformations**:
  - [x] Log transformation for price (if skewed)
  - [x] Log transformation for km (if skewed)

  - [x] Standardization/normalization
- [x] **[ESSENTIAL] Location features**:
  - [x] State encoding (target encoding)
  - [x] Region encoding (Norte, Nordeste, Sul, Sudeste, Centro-Oeste)

3.1.1 Feature Creation - Phase 2 (Optional Enhancements) ✅ Implemented

- [x] **[OPTIONAL] Advanced features**:

- [x] Depreciation rate calculation

- [x] Frequency encoding (brand/model frequency)
- [x] **Interaction features**:
  - [x] Brand × Year
  - [x] Fuel × Transmission
  - [x] Age × Condition
  - [x] Km per year (km / age)
- [x] **Binning**:
  - [x] Price bins (for stratified splits)
  - [x] Age bins
  - [x] Mileage bins
- [x] **Advanced location features**:
  - [x] Region encoding (Norte, Nordeste, Sul, Sudeste, Centro-Oeste) - Already in Phase 1
  - [x] City size category (if applicable)

Note: Advanced features are implemented in `AdvancedFeatureCreator` class and can be enabled via `use_advanced_features=True` parameter. They are disabled by default to maintain MVP focus.

3.2 Feature Selection

- [x] Create `src/features/selectors.py`:
- [x] Correlation-based feature selection
- [x] Mutual information
- [x] Feature importance from baseline models
- [x] Remove highly correlated features
- [x] Document selected features

3.3 Feature Pipeline

- [x] Create `src/features/pipeline.py`:
- [x] Scikit-learn Pipeline or custom pipeline
- [x] Combine all transformations
- [x] Fit on training, transform on validation/test
- [x] Save fitted pipeline for inference
- [x] Integrate FeatureEngineeringStep into main pipeline

Deliverables:

- ✅ Feature engineering pipeline (`src/features/pipeline.py`)
- ✅ Feature engineering modules (`src/features/engineering.py`, `src/features/selectors.py`)
- ✅ FeatureEngineeringStep integrated into main pipeline
- ✅ Pipeline persistence (save/load functionality)
- ⏳ Engineered feature dataset (will be created when pipeline runs)
- ⏳ Feature importance analysis (available when feature selection is enabled)

Estimated Time: 1-2 weeks
Actual Time: Completed ✅

Phase 4: Baseline Models ✅ Completed

4.1 Baseline Implementations

- [x] Create `src/models/baseline.py`:
- [x] **Mean/Median baseline**: Simple average/median price
- [x] **Linear Regression**: Basic linear model
- [x] **Ridge Regression**: L2 regularization
- [x] **Lasso Regression**: L1 regularization
- [x] **Decision Tree**: Simple tree model
- [x] Evaluate all baselines
- [x] Document baseline performance

4.2 Evaluation Metrics

- [x] Create `src/models/evaluator.py`:
- [x] **RMSE** (Root Mean Squared Error)
- [x] **MAE** (Mean Absolute Error)
- [x] **MAPE** (Mean Absolute Percentage Error)
- [x] **R^2 Score** (Coefficient of Determination)
- [x] **Residual analysis**:
    - [x] Residual plots
    - [x] Q-Q plots
    - [x] Residual distribution
- [x] Create visualization functions for metrics

Deliverables:

- ✅ Baseline model implementations (`src/models/baseline.py`)
- ✅ Baseline performance report (saved to `models/baseline_results/`)
- ✅ Evaluation metrics module (`src/models/evaluator.py`)
- ✅ TrainBaselineModelsStep integrated into main pipeline
- ✅ Training script (`scripts/train_baseline_models.py`)
- ✅ Results saved to `models/baseline_results/` (separate from `data/processed/`)

Estimated Time: 3-5 days

Actual Time: Completed ✅

Phase 5: Advanced Model Development ✅ Completed

5.1 Model Implementations - Essential

- [x] Create `src/models/trainer.py`:
- [x] **[ESSENTIAL] Random Forest**:
    - [x] Hyperparameter tuning (RandomizedSearchCV with 5-fold CV)
    - [x] n_estimators, max_depth, min_samples_split, min_samples_leaf, max_features

- [x] **[ESSENTIAL] Gradient Boosting (XGBoost)**:
    - [x] Hyperparameter tuning (validation-based search)
    - [x] learning_rate, n_estimators, max_depth, subsample, colsample_bytree
    - [x] Validation set monitoring (early stopping removed for compatibility)

5.1.1 Model Implementations - Optional

- [x] **[OPTIONAL] LightGBM**:
- [x] Hyperparameter tuning (validation-based search)
- [x] num_leaves, learning_rate, max_depth, feature_fraction
- [x] Validation set monitoring (early stopping removed for compatibility)
- [ ] **[OPTIONAL] CatBoost**:
- [ ] Good for categorical features
- [ ] Hyperparameter tuning
- **Note**: CatBoost not implemented yet (can be added in Phase 6 if needed)

5.2 Model Training

- [x] Implement cross-validation:
- [x] K-fold cross-validation (k=2) for Random Forest (reduced for memory efficiency)
- [x] Validation-based search for XGBoost and LightGBM
- [x] Train all models on training set
- [x] Validate on validation set
- [x] Track training time and model size

5.3 Model Comparison

- [x] Compare all models:
- [x] Performance metrics (RMSE, MAE, MAPE, R^2)
- [x] Training time tracking
- [x] Validation scores tracking
- [x] Feature importance analysis:
- [x] Tree-based model feature importance (available via model attributes)
- [x] Create comparison report (CSV and visualization)

5.4 Model Selection

- [x] Select best model based on:
- [x] Validation performance (metrics comparison)
- [x] Training time tracking
- [ ] Final evaluation on test set (to be done in Phase 6)
- [ ] Document model selection rationale (to be done in Phase 6)

Deliverables:

- ✅ Advanced model trainer (`src/models/trainer.py`)
- ✅ TrainAdvancedModelsStep integrated into main pipeline
- ✅ Training script (`scripts/train_advanced_models.py`)
- ✅ Trained advanced models (saved to `models/advanced_results/`)
- ✅ Model comparison report (CSV and plots)
- ✅ Model artifacts saved (joblib format)
- ⏳ Feature importance analysis (available via model attributes, detailed analysis in Phase 6)

Estimated Time: 2-3 weeks
Actual Time: Completed ✅

Phase 6: Model Optimization & Fine-tuning

6.1 Hyperparameter Optimization

- [ ] Fine-tune selected model:
- [ ] GridSearchCV or RandomSearchCV
- [ ] Bayesian optimization (Optuna) - optional
- [ ] Learning curves analysis
- [ ] Optimize for business metric (MAPE or RMSE)

6.2 Model Ensemble (Optional Enhancement)

- [ ] **[OPTIONAL]** Create `src/models/ensemble.py`:
- [ ] Voting regressor
- [ ] Stacking regressor

- [ ] Weighted average ensemble
- [ ] **[OPTIONAL]** Evaluate ensemble performance

> [!TIP] Ensemble methods can improve performance by 2-5% but add complexity. Consider only after achieving good results with single models.

6.3 Model Validation

- [ ] Final test set evaluation
- [ ] Performance by segments:
- [ ] By price range
- [ ] By brand
- [ ] By age
- [ ] By location
- [ ] Error analysis:
- [ ] Identify worst predictions
- [ ] Analyze error patterns
- [ ] Document model limitations

Deliverables:

- Optimized model
- Final performance metrics
- Model validation report
- Error analysis

Estimated Time: 1 week

Phase 7: Model Persistence & Versioning

7.1 Model Saving

- [ ] Create `src/models/persistence.py`:
- [ ] Save trained model (joblib/pickle)
- [ ] Save feature pipeline
- [ ] Save preprocessing steps

- [ ] Save model metadata:
    - [ ] Training date
    - [ ] Performance metrics
    - [ ] Feature list
    - [ ] Hyperparameters
- [ ] Save to `models/` directory

7.2 Model Versioning

- [ ] Implement model versioning:
- [ ] Version naming convention (e.g., v1.0.0)
- [ ] Model registry (simple JSON file or MLflow)
- [ ] Model comparison tracking

7.3 Model Loading

- [ ] Create model loading function
- [ ] Validate loaded model
- [ ] Test inference with loaded model

Deliverables:

- Model persistence module
- Saved model artifacts
- Model versioning system

Estimated Time: 2-3 days

Phase 8: API Development

8.1 FastAPI Application Structure

- [ ] Create `src/api/main.py`:
- [ ] FastAPI app initialization
- [ ] Model loading on startup
- [ ] Health check endpoint: `GET /health`
- [ ] Model info endpoint: `GET /model/info`

8.2 Prediction Endpoints

- [ ] Create `src/api/schemas.py`:
- [ ] Pydantic models for request/response
- [ ] Input validation schemas
- [ ] Response schemas
- [ ] Create `src/api/predictor.py`:
- [ ] Single prediction endpoint: `POST /predict`
- [ ] Batch prediction endpoint: `POST /predict/batch`
- [ ] Input validation
- [ ] Feature transformation
- [ ] Model inference
- [ ] Response formatting

8.3 Error Handling

- [ ] Create `src/api/errors.py`:
- [ ] Custom exception classes
- [ ] Error handlers
- [ ] Validation error responses
- [ ] Model inference errors

8.4 API Documentation

- [ ] Configure OpenAPI/Swagger documentation
- [ ] Add endpoint descriptions
- [ ] Add example requests/responses
- [ ] Document error codes

8.5 API Testing

- [ ] Create `tests/test_api.py`:
- [ ] Test health endpoint
- [ ] Test prediction endpoints
- [ ] Test input validation
- [ ] Test error handling
- [ ] Integration tests

Deliverables:

- FastAPI application
- Prediction endpoints
- API documentation
- API tests

Estimated Time: 1-2 weeks

Phase 9: Docker & Deployment

9.1 Docker Configuration Review

- [ ] Review and optimize Dockerfile:
- [ ] Multi-stage build (if needed)
- [ ] Minimize image size
- [ ] Optimize layer caching
- [ ] Review docker-compose.yml
- [ ] Test Docker build locally

9.2 Environment Configuration

- [ ] Create `.env.example`:
- [ ] API configuration
- [ ] Model path
- [ ] Logging level
- [ ] Update docker-compose.yml for environment variables

9.3 Deployment Preparation

- [ ] Create deployment documentation
- [ ] Test containerized application
- [ ] Performance testing:
- [ ] Load testing (optional)
- [ ] Response time testing
- [ ] Resource requirements documentation

9.4 CI/CD (Optional Enhancement)

- [ ] **[OPTIONAL]** Set up GitHub Actions:
- [ ] Run tests on push
- [ ] Code quality checks (black, flake8)
- [ ] Build Docker image
- [ ] Deploy to staging (optional)

> [!NOTE] CI/CD is valuable for production systems but not essential for MVP. Consider implementing after core functionality is complete.

Deliverables:

- Optimized Docker configuration
- Deployment documentation
- CI/CD pipeline (optional)

Estimated Time: 1 week

Phase 10: Documentation & Testing

10.1 Code Documentation

- [ ] Add docstrings to all functions and classes
- [ ] Document module purposes
- [ ] Add type hints throughout codebase
- [ ] Create API documentation

10.2 Project Documentation

- [ ] Update README.md with:
- [ ] Complete setup instructions
- [ ] Usage examples
- [ ] API usage guide
- [ ] Model information
- [ ] Create CONTRIBUTING.md (if applicable)
- [ ] Create ARCHITECTURE.md:
- [ ] Project structure

- [ ] Data flow
- [ ] Model architecture

10.3 Testing

- [ ] Unit tests for all modules:
- [ ] Data processing tests
- [ ] Feature engineering tests
- [ ] Model training tests
- [ ] API tests
- [ ] Integration tests
- [ ] Achieve >80% test coverage
- [ ] Create `tests/README.md` with testing instructions

10.4 Usage Examples

- [ ] Create example notebooks:
- [ ] Model training example
- [ ] API usage example
- [ ] Prediction example
- [ ] Create example scripts

Deliverables:

- Complete code documentation
- Updated project documentation
- Comprehensive test suite
- Usage examples

Estimated Time: 1-2 weeks

Phase 11: Optimization & Refinement

11.1 Performance Optimization

- [ ] Profile code for bottlenecks
- [ ] Optimize data loading

- [ ] Optimize feature engineering pipeline
- [ ] Optimize model inference:
- [ ] Batch processing
- [ ] Model quantization (optional)
- [ ] Caching strategies

11.2 Code Quality

- [ ] Code review and refactoring
- [ ] Follow PEP 8 style guide
- [ ] Remove unused code
- [ ] Improve error messages
- [ ] Add logging throughout application

11.3 Model Monitoring (Optional Enhancement)

- [ ] **[OPTIONAL]** Create monitoring dashboard
- [ ] **[OPTIONAL]** Track prediction distribution
- [ ] **[OPTIONAL]** Monitor model drift
- [ ] **[OPTIONAL]** Set up alerts

> [!WARNING] Model monitoring is critical for production systems but requires additional infrastructure. For portfolio/demo purposes, focus on core ML pipeline first.

Deliverables:

- Optimized codebase
- Performance improvements
- Code quality improvements

Estimated Time: 1 week

Implementation Timeline

> [!NOTE] > **MVP Strategy**: Focus on essential tasks first to build a working end-to-end pipeline (Phases 1-8 core features). Optional enhancements can be added iteratively.

Phase	Task	Estimated Time	Priority	Status
1	Exploratory Data Analysis	1-2 weeks	Essential	✅ Completed
2	Data Preprocessing & Cleaning	1 week	Essential	✅ Completed
3	Feature Engineering (Essential)	1 week	Essential	✅ Completed
3.1	Feature Engineering (Optional)	1 week	Optional	✅ Completed
4	Baseline Models	3-5 days	Essential	✅ Completed
5	Advanced Models (RF + XGBoost)	1-2 weeks	Essential	✅ Completed
5.1	Additional Models (LightGBM, CatBoost)	1 week	Optional	✅ Completed (LightGBM)
6	Model Optimization	1 week	Essential	⏳ Pending
6.1	Model Ensemble	3-5 days	Optional	⏳ Pending
7	Model Persistence	2-3 days	Essential	⏳ Pending
8	API Development	1-2 weeks	Essential	⏳ Pending
9	Docker & Deployment	1 week	Essential	⏳ Pending
9.1	CI/CD Pipeline	3-5 days	Optional	⏳ Pending
10	Documentation & Testing	1-2 weeks	Essential	⏳ Pending
11	Optimization & Refinement	1 week	Essential	⏳ Pending
11.1	Model Monitoring	3-5 days	Optional	⏳ Pending

MVP Timeline (Essential Only): 10-12 weeks
Full Implementation (Essential + Optional): 12-16 weeks

Key Deliverables Checklist

Data & Analysis

- [x] Raw datasets collected
- [x] Data enrichment completed
- [x] EDA notebooks completed
- [x] Data cleaning pipeline
- [x] Feature engineering pipeline

Models

- [x] Baseline models implemented
- [x] Advanced models trained (Random Forest, XGBoost, LightGBM)
- [ ] Best model selected and optimized (to be done in Phase 6)
- [x] Model artifacts saved (baseline results in `models/baseline_results/`, advanced results in `models/advanced_results/`)
- [x] Model performance report (baseline and advanced models)

API & Deployment

- [ ] FastAPI application
- [ ] Prediction endpoints
- [ ] Docker configuration
- [ ] Deployment documentation

Documentation & Quality

- [ ] Complete code documentation
- [ ] Project documentation
- [ ] Test suite (>80% coverage)
- [ ] Usage examples

Success Criteria

- [ ] Model achieves acceptable performance:
- [ ] MAPE < 15-20% (or RMSE < acceptable threshold)
- [ ] $R^2 > 0.85$
- [ ] API responds to predictions in < 1 second
- [ ] Code is well-documented and tested (>80% coverage)
- [ ] Project can be easily reproduced and deployed
- [ ] All phases are completed and documented

Notes & Considerations

Development Best Practices

1. **Data Quality**: Continuously monitor data quality throughout the pipeline
2. **Reproducibility**: Use random seeds for all random operations
3. **Version Control**: Track model versions and data versions
4. **Performance**: Balance model accuracy with inference speed
5. **Interpretability**: Consider model interpretability for business stakeholders
6. **Scalability**: Design pipeline to handle larger datasets in the future

Production Readiness (Future Considerations)

> [!WARNING] > **Synthetic Data Validation**: Before deploying to production, validate model predictions against real-world market data. The current dataset uses synthetic features that may not capture all market dynamics.

1. **Model Retraining**: Plan for periodic model retraining (monthly/quarterly) as car market conditions change
2. **Drift Monitoring**: Implement data drift detection to identify when model performance degrades
3. **A/B Testing**: Consider A/B testing framework for comparing model versions in production
4. **Business Validation**: Validate predictions with automotive market experts before production deployment

5. **Edge Cases**: Document and handle edge cases (luxury cars, rare models, extreme mileage)

Next Steps

1. **Start with Phase 1**: Begin Exploratory Data Analysis
2. **Create first notebook**: `notebooks/01_data_overview.ipynb`
3. **Set up development environment**: Ensure all dependencies are installed
4. **Review data**: Load and inspect enriched datasets

Last Updated: 2024-12-08
Current Phase: Phase 5 - Advanced Models ✅ Completed | Phase 6 - Model Optimization 🚧 Next
Strategy: MVP-first approach with essential features, then iterate with optional enhancements