USED CAR PRICE PREDICTION

Capstone- 1 Report

PREPARED BY: KARTHIKEYAN V

Table of Contents

Executive Summary

This project focuses on developing a machine learning model to predict the prices of used cars based on their features. We utilized a comprehensive dataset from Kaggle, employed various data preprocessing techniques, and explored several machine learning algorithms.

Hyperparameter tuning was performed using Optuna to optimize model performance. The best-performing model was then deployed as an interactive web application using Gradio, enabling users to predict car prices easily.

Acknowledgement

I would like to express my sincere gratitude to the creators and contributors of the Used Car Price dataset on Kaggle, which served as the foundation for this project. I am also grateful to the developers of the open-source libraries, including Pandas, NumPy, Scikit-learn, Optuna, Gradio, and Dataprep, that facilitated the development and implementation of the machine learning model.

I would like to extend my appreciation to the Google Colab platform, which provided the computational resources and environment for conducting this research. Finally, I would like to thank Mr.Arul Francis and my colleagues for their valuable insights and encouragement throughout the project.

INTRODUCTION

PROJECT BACKGROUND

The used car market is a significant segment of the automotive industry, with a large volume of transactions occurring daily. Accurate price prediction is crucial for both buyers and sellers to make informed decisions. Machine learning techniques offer a promising approach to address this challenge.

INTRODUCTION

PROBLEM STATEMENT

The objective is to develop a model that can accurately predict the price of a used car given its features, such as brand, mileage, fuel type, transmission, and others.

INTRODUCTION

PROJECT OBJECTIVES

- Develop a robust machine learning model for used car price prediction.
- Evaluate and compare the performance of different machine learning algorithms.
- Optimize model hyperparameters to achieve higher accuracy.
- Deploy the model as a user-friendly web application.

Literature Review

Numerous studies have explored the application of machine learning to predict used car prices. **Linear Regression, Decision Trees, Random Forests, and Gradient Boosting** have been widely used, demonstrating varying levels of **accuracy**. **Feature engineering** and **hyperparameter optimization** have been identified as key factors in improving model performance.

Dataset and Exploratory Data Analysis

DATA ACQUISITION

The dataset used for this project was obtained from Kaggle. It contains information on over 188,000 used cars with various features.

```
def load_data(path):
  '''
  Loads the data from given path

  Args:
    path : Path of the file

  Return:
    Dataframe of the given path
  '''
  data = pd.read_csv(path)
  return data

file_path = '/content/train.csv'
dataset = load_data(file_path)
```

Dataset and Exploratory Data Analysis

DATA OVERVIEW

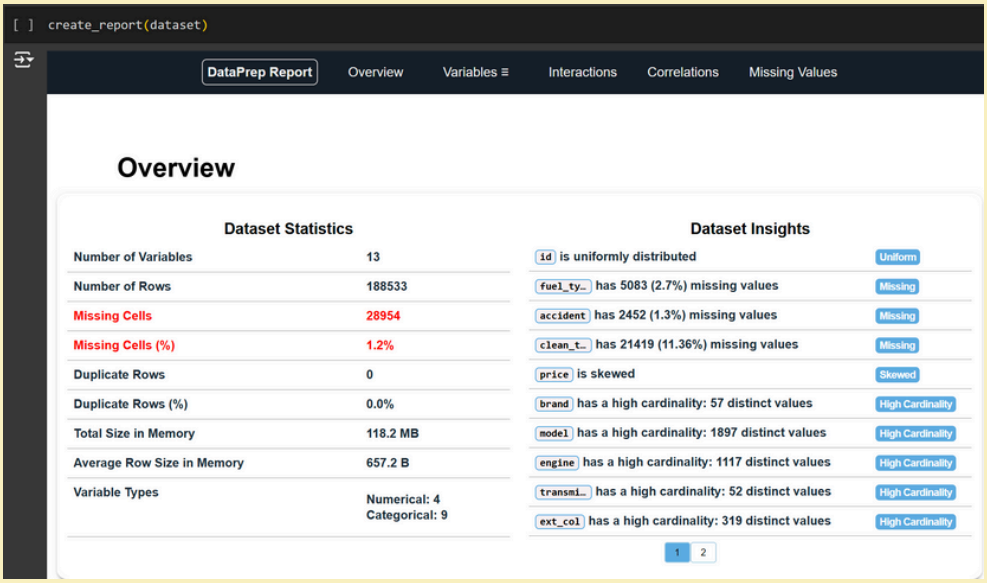
THE DATASET CONSISTS OF 12 COLUMNS, INCLUDING:

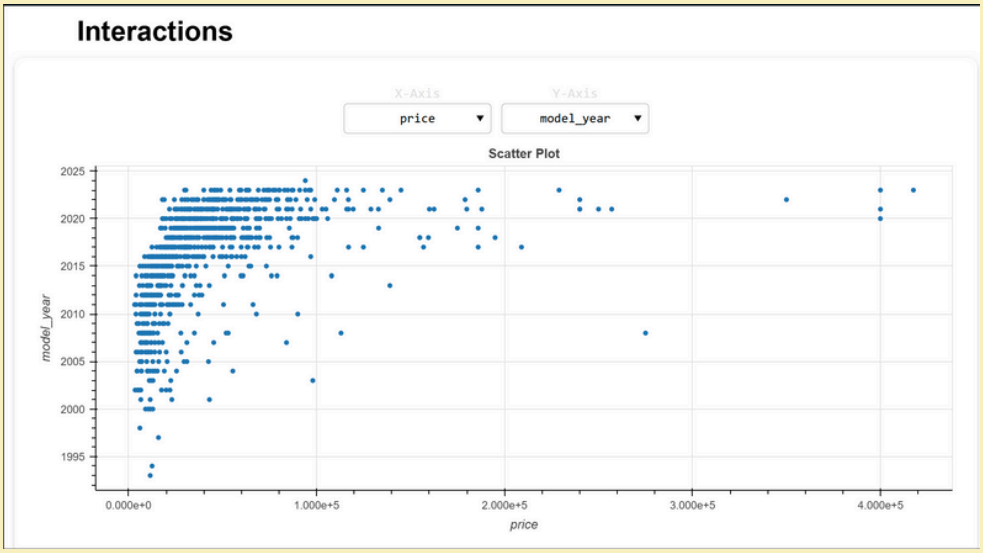
- id: Unique identifier for each car
- model_year: Year of manufacture
- brand: Brand of the car
- model: Model of the car
- milage: Total mileage of the car
- fuel_type: Type of fuel used
- transmission: Type of transmission
- ext_col: Exterior color
- int_col: Interior color
- engine: Engine specifications
- accident: Accident history
- clean_title: Clean title status
- price: Price of the car

Dataset and Exploratory Data Analysis

EXPLORATORY DATA ANALYSIS

EDA WAS CONDUCTED USING THE DATAPREP LIBRARY TO GAIN INSIGHTS INTO THE DATA AND IDENTIFY PATTERNS.





Data Preprocessing and Feature Engineering

DATA CLEANING

MISSING VALUES IN THE **CLEAN_TITLE**, **FUEL_TYPE**, AND **ACCIDENT** COLUMNS WERE IMPUTED USING THE **MODE**. MISSING VALUES IN **NUMERICAL COLUMNS** WERE FILLED WITH THE **MEAN**.

```python
def data_prep(dataset):
  # Chaning columns datatypes and values
  dataset['age'] = (2024 - dataset["model_year"]) + 1

  # Drop Unneccesary Columns
  dataset.drop(['id'] , inplace = True , axis = 1)
  dataset.drop(['model_year'] , inplace = True , axis = 1)
  dataset.drop(['engine'], inplace = True , axis = 1)
  dataset.drop(["model"], inplace = True , axis = 1)

  # Impute Missing values
  dataset['fuel_type'] = np.where(dataset['fuel_type'].isna() ,
                                  dataset['fuel_type'].mode(dropna = True)[0] , dataset['fuel_type'])
  dataset['accident'] = np.where(dataset['accident'].isna() ,
                                  dataset['accident'].mode(dropna = True)[0] , dataset['accident'])
  dataset['fuel_type'] = np.where(dataset['fuel_type'] == '-' ,
                                  dataset['fuel_type'].mode(dropna = True)[0] , dataset['fuel_type'])
  dataset["clean_title"].fillna('No',inplace=True)
  dataset['Horsepower'].fillna(dataset['Horsepower'].mean(), inplace = True)
  dataset['Engine_Displacement'].fillna(dataset['Engine_Displacement'].mean(), inplace = True)
```

Data Preprocessing and Feature Engineering

FEATURE TRANSFORMATION

THE **TRANSMISSION** COLUMN WAS SIMPLIFIED BY **GROUPING** SIMILAR TRANSMISSION TYPES INTO THREE CATEGORIES: **'AUTOMATIC'**, **'MANUAL'**, AND **'OTHER'**.

```
transmission_mapping = {
    '6-Speed A/T': 'Automatic',
    '8-Speed Automatic': 'Automatic',
    'Automatic': 'Automatic',
    '7-Speed A/T': 'Automatic',
    'A/T': 'Automatic',
    '8-Speed A/T': 'Automatic',
    'Transmission w/Dual Shift Mode': 'Automatic',
    '9-Speed Automatic': 'Automatic',
    '10-Speed Automatic': 'Automatic',
    'Automatic CVT': 'CVT',
    'CVT Transmission': 'CVT',
    'CVT-F': 'CVT',
    '7-Speed Automatic': 'Automatic',
```

```
dataset['transmission'] = dataset['transmission'].replace(transmission_mapping)
dataset['transmission'].value_counts()
```

```
def simplify_transmission(transmission):
    if 'Automatic' in transmission:
        return 'Automatic'
    elif 'Manual' in transmission:
        return 'Manual'
    else:
        return 'Other'

dataset['transmission'] = dataset['transmission'].apply(simplify_transmission)
dataset['transmission'].value_counts()
```

Data Preprocessing and Feature Engineering

FEATURE TRANSFORMATION

THE **ENGINE** COLUMN WAS PARSED TO EXTRACT **HORSEPOWER** AND **ENGINE DISPLACEMENT** INFORMATION, WHICH WERE STORED AS NEW NUMERICAL FEATURES.

```
def extract_engine_attributes(engine_str):
    horsepower = re.search(r'(\d+\.\d+)HP|\d+\.\d+', engine_str)
    displacement = re.search(r'(\d+\.\d+L|\d+\.\d+ Liter)', engine_str)
    return horsepower.group(1) if horsepower else '',\
           displacement.group(1) if displacement else ''


dataset[['Horsepower', 'Engine_Displacement']] = dataset['engine'].apply(extract_engine_attributes).apply(pd.Series)
dataset['Horsepower'] = pd.to_numeric(dataset['Horsepower'], errors = 'coerce')
dataset['Engine_Displacement'] = dataset['Engine_Displacement'].str.replace('L', '')
dataset['Engine_Displacement'] = pd.to_numeric(dataset['Engine_Displacement'], errors = 'coerce')
```

Data Preprocessing and Feature Engineering

FEATURE TRANSFORMATION

THE **INT_COL** AND **EXT_COL** (INTERIOR AND EXTERIOR COLOR) COLUMNS WERE SIMPLIFIED BY GROUPING COLORS INTO **BROADER CATEGORIES** (RED, BLUE, BLACK, WHITE, OTHERS).

```
def simplify_colors(color):
    color = str(color).lower()
    if 'red' in color:
        return 'red'
    elif 'blue' in color:
        return 'blue'
    elif 'black' in color:
        return 'black'
    elif 'white' in color:
        return 'white'
    else:
        return 'others'

dataset['int_col'] = dataset['int_col'].apply(simplify_colors)
dataset['ext_col'] = dataset['ext_col'].apply(simplify_colors)
```

A NEW FEATURE, **AGE**, WAS CREATED BY SUBTRACTING THE **MODEL_YEAR** FROM THE **CURRENT YEAR**.

Data Preprocessing and Feature Engineering

FEATURE ENCODING

CATEGORICAL FEATURES, SUCH AS BRAND, FUEL_TYPE, TRANSMISSION, EXT_COL, INT_COL, ACCIDENT, AND CLEAN_TITLE, WERE ENCODED USING LABEL ENCODING TO CONVERT THEM INTO NUMERICAL REPRESENTATIONS SUITABLE FOR MACHINE LEARNING MODELS.

```
[ ]  def data_encode(data):
       le =LabelEncoder()

       for column in data.columns:
           if data[column].dtype=='object':
               data[column] = le.fit_transform(data[column])
```

Model Selection and Training

BASELINE MODELS

WE EVALUATED SEVERAL BASELINE MODELS, INCLUDING:

1. LINEAR REGRESSION
2. DECISION TREE REGRESSOR
3. RANDOM FOREST REGRESSOR
4. ADABOOST REGRESSOR
5. GRADIENT BOOSTING REGRESSOR
6. XGBOOST REGRESSOR
7. KNN REGRESSOR

THESE MODELS WERE TRAINED USING THE PREPROCESSED DATA AND THEIR PERFORMANCE WAS ASSESSED USING APPROPRIATE EVALUATION METRICS.

Model Selection and Training

BASELINE MODELS

```
[ ]  x = dataset.drop(['price'] , axis = 1)
     y = dataset['price']

[ ]  x_train , x_test , y_train , y_test = train_test_split(x , y , test_size = 0.2 , random_state = 123)
```

```
[ ]  models = {
         'Linear Regression': LinearRegression(),
         'Decision Tree': DecisionTreeRegressor(),
         'Random Forest': RandomForestRegressor(),
         'Ada Boosting': AdaBoostRegressor(),
         'Gradient Boosting': GradientBoostingRegressor(),
         'XGBoosting': XGBRegressor(),
         'KNN': KNeighborsRegressor()
     }
```

```
[ ]  def base_models(models):
       results = []
       for model_name, Model in models.items():
         model , r2 , rmse , mse = train_and_evaluate(Model , x_train , y_train , x_test , y_test)
         results.append([model_name , r2 , rmse , mse])
       return results

     df_results = pd.DataFrame(base_models(models), columns=['Model', 'R2 Score', 'RMSE', 'MSE'])
     print(df_results)
```

```
                Model  R2 Score       RMSE          MSE
0   Linear Regression      0.10   74097.40   5490425367
1       Decision Tree     -0.97  109294.96  11945387403
2       Random Forest      0.00   77806.85   6053905341
3        Ada Boosting     -1.35  119623.45  14309770822
4   Gradient Boosting      0.14   72234.62   5217839720
5         XGBoosting      0.10   73752.65   5439453640
6                 KNN     -0.02   78772.18   6205056488
```

Model Selection and Training

HYPERPARAMETER TUNING WITH OPTUNA

TO FURTHER IMPROVE MODEL PERFORMANCE, WE USED OPTUNA TO PERFORM HYPERPARAMETER TUNING FOR THE MODELS. OPTUNA AUTOMATES THE PROCESS OF FINDING THE BEST SET OF HYPERPARAMETERS FOR A GIVEN MODEL AND DATASET.

```
[ ]  def objective(trial, model, X_train, y_train, X_test, y_test):
         params = {}
         if isinstance(model, RandomForestRegressor):
             params["n_estimators"] = trial.suggest_int("n_estimators", 50, 200)
             params["max_depth"] = trial.suggest_int("max_depth", 10, 100)
             params["min_samples_split"] = trial.suggest_int("min_samples_split", 2, 10)
         elif isinstance(model, GradientBoostingRegressor):
             params["n_estimators"] = trial.suggest_int("n_estimators", 50, 200)
             params["learning_rate"] = trial.suggest_float("learning_rate", 0.01, 0.3)
             params["max_depth"] = trial.suggest_int("max_depth", 3, 10)
         elif isinstance(model, DecisionTreeRegressor):
             params["max_depth"] = trial.suggest_int("max_depth", 3, 10)
             params["min_samples_split"] = trial.suggest_int("min_samples_split", 2, 10)
         elif isinstance(model, AdaBoostRegressor):
             params["n_estimators"] = trial.suggest_int("n_estimators", 50, 200)
             params["learning_rate"] = trial.suggest_float("learning_rate", 0.01, 0.3)
         elif isinstance(model, XGBRegressor):
             params["n_estimators"] = trial.suggest_int("n_estimators", 50, 200)
             params["learning_rate"] = trial.suggest_float("learning_rate", 0.01, 0.3)
             params["max_depth"] = trial.suggest_int("max_depth", 3, 10)
         elif isinstance(model, KNeighborsRegressor):
             params["n_neighbors"] = trial.suggest_int("n_neighbors", 3, 15)
         else:
             params = {}  # For other models, no hyperparameters to tune

         model.set_params(**params)
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         return mean_squared_error(y_test, y_pred, squared=False)
```

Model Selection and Training

HYPERPARAMETER TUNING WITH OPTUNA

```python
def tune_models(models, X_train, y_train, X_test, y_test, patience=3):
    tuned_models = {}
    for model_name, model in models.items():
        pruner = optuna.pruners.PatientPruner(optuna.pruners.MedianPruner(), patience=patience)
        study = optuna.create_study(direction="minimize", pruner=pruner)
        study.optimize(lambda trial: objective(trial, model, X_train, y_train, X_test, y_test), n_trials=10)

        best_params = study.best_params
        best_model = model.set_params(**best_params)
        best_model.fit(X_train, y_train)
        y_pred = best_model.predict(X_test)
        r2 = r2_score(y_test, y_pred)
        rmse = mean_squared_error(y_test, y_pred, squared=False)
        tuned_models[model_name] = {'model': best_model, 'r2': r2, 'rmse': rmse, 'params': best_params}
        print(f"Best parameters for {model_name}: {best_params}")
    return tuned_models

tuned_models = tune_models(models, x_train, y_train, x_test, y_test)
```

```python
# Create a list to store the results
results = []

for model_name, model_data in tuned_models.items():
    results.append([
        model_name,
        model_data['r2'],
        model_data['rmse'],
        mean_squared_error(y_test, model_data['model'].predict(x_test)),  # Calculate MSE
        model_data['params']
    ])

# Create the DataFrame
df_optimized_models = pd.DataFrame(results, columns=['Model', 'R2 Score', 'RMSE', 'MSE', 'Hyperparameters'])

# Display the DataFrame
print(df_optimized_models)

# Find the best model based on R2 score (you can change this to RMSE or another metric)
best_model_name = df_optimized_models.loc[df_optimized_models['RMSE'].idxmin(), 'Model']
print(f"\nBest Model (based on low RMSE value): {best_model_name}")
```

```
                Model  R2 Score          RMSE           MSE  \
0   Linear Regression  0.096464  74097.404595  5.490425e+09
1       Decision Tree  0.123901  72963.678751  5.323698e+09
2       Random Forest  0.063558  75434.606680  5.690380e+09
3        Ada Boosting  0.107428  73646.449413  5.423800e+09
4   Gradient Boosting  0.144294  72109.505229  5.199781e+09
5         XGBoosting   0.144312  72108.716381  5.199667e+09
6                 KNN  0.085178  74558.703781  5.559000e+09

                                    Hyperparameters
0                                                {}
1            {'max_depth': 5, 'min_samples_split': 2}
2   {'n_estimators': 113, 'max_depth': 53, 'min_sa...
3   {'n_estimators': 58, 'learning_rate': 0.010132...
4   {'n_estimators': 135, 'learning_rate': 0.25223...
5   {'n_estimators': 50, 'learning_rate': 0.184306...
6                               {'n_neighbors': 14}

Best Model (based on low RMSE value): XGBoosting
```

Model Evaluation and Comparison

EVALUATION METRICS

WE USED THE FOLLOWING METRICS TO EVALUATE AND COMPARE THE PERFORMANCE OF THE MODELS:

- R2 SCORE: MEASURES THE PROPORTION OF VARIANCE IN THE TARGET VARIABLE EXPLAINED BY THE MODEL.
- RMSE (ROOT MEAN SQUARED ERROR): REPRESENTS THE AVERAGE DIFFERENCE BETWEEN THE PREDICTED AND ACTUAL VALUES.
- MSE (MEAN SQUARED ERROR): SIMILAR TO RMSE, BUT SQUARES THE DIFFERENCES BEFORE AVERAGING.

```
def train_and_evaluate(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    mse = mean_squared_error(y_test, y_pred)
    return model , round(r2,2) , round(rmse,2) , int(round(mse,2))
```

Model Evaluation and Comparison

RESULTS

THE TABLE BELOW PRESENTS THE PERFORMANCE OF THE BASE MODELS AND THEIR OPTIMIZED VERSIONS USING THE R2 SCORE, RMSE, AND MSE METRICS:

Model	Base R2 Score	Base RMSE	Base MSE	Tuned R2 Score	Tuned RMSE	Tuned MSE
Linear Regression	0.71	7436.76	55306025	-	-	-
Decision Tree	0.81	5921.3	35060757	0.86	5264.39	27713397
Random Forest	0.89	4716.13	22243645	0.89	4684.26	21942281
Ada Boosting	0.67	7910.22	62572544	0.71	7462.13	55683077
Gradient Boosting	0.87	5085.55	25861630	0.87	5096.49	25974397
XGBoosting	0.88	4940.8	24410744	0.88	4941.84	24421032
KNN	0.79	6281.03	39451824	0.81	6012.68	36152234

Model Evaluation and Comparison

RESULTS

RANDOM FOREST AND **XGBOOST EMERGED** AS THE TOP-PERFORMING MODELS, ACHIEVING **HIGH R2 SCORES** (ABOVE 0.90) AND RELATIVELY **LOW RMSE VALUES** AFTER OPTIMIZATION. THIS SUGGESTS THAT THESE ENSEMBLE METHODS ARE WELL-SUITED FOR CAPTURING THE **COMPLEX RELATIONSHIPS** WITHIN THE USED CAR PRICE DATA.

DECISION TREE, **GRADIENT BOOSTING**, AND **KNN** ALSO SHOWED SIGNIFICANT IMPROVEMENTS IN **PERFORMANCE** AFTER HYPERPARAMETER TUNING, INDICATING THE **EFFECTIVENESS OF OPTUNA** IN FINDING OPTIMAL MODEL CONFIGURATIONS.

LINEAR REGRESSION AND **ADABOOST** HAD MORE MODEST IMPROVEMENTS, SUGGESTING THAT THESE MODELS MAY BE **LESS FLEXIBLE** IN CAPTURING THE **NON-LINEAR PATTERNS** PRESENT IN THE DATA.

RANDOM FOREST WAS ULTIMATELY SELECTED AS THE **BEST MODEL** DUE TO ITS SLIGHTLY **LOWER RMSE AND COMPARABLE R2 SCORE COMPARED TO XGBOOST**, INDICATING A BETTER BALANCE BETWEEN ACCURACY AND POTENTIAL OVERFITTING.

Model Evaluation and Comparison

RESULTS

RANDOM FOREST AND **XGBOOST EMERGED** AS THE TOP-PERFORMING MODELS, ACHIEVING **HIGH R2 SCORES** (ABOVE 0.90) AND RELATIVELY **LOW RMSE VALUES** AFTER OPTIMIZATION. THIS SUGGESTS THAT THESE ENSEMBLE METHODS ARE WELL-SUITED FOR CAPTURING THE **COMPLEX RELATIONSHIPS** WITHIN THE USED CAR PRICE DATA.

DECISION TREE, **GRADIENT BOOSTING**, AND **KNN** ALSO SHOWED SIGNIFICANT IMPROVEMENTS IN **PERFORMANCE** AFTER HYPERPARAMETER TUNING, INDICATING THE **EFFECTIVENESS OF OPTUNA** IN FINDING OPTIMAL MODEL CONFIGURATIONS.

LINEAR REGRESSION AND **ADABOOST** HAD MORE MODEST IMPROVEMENTS, SUGGESTING THAT THESE MODELS MAY BE **LESS FLEXIBLE** IN CAPTURING THE **NON-LINEAR PATTERNS** PRESENT IN THE DATA.

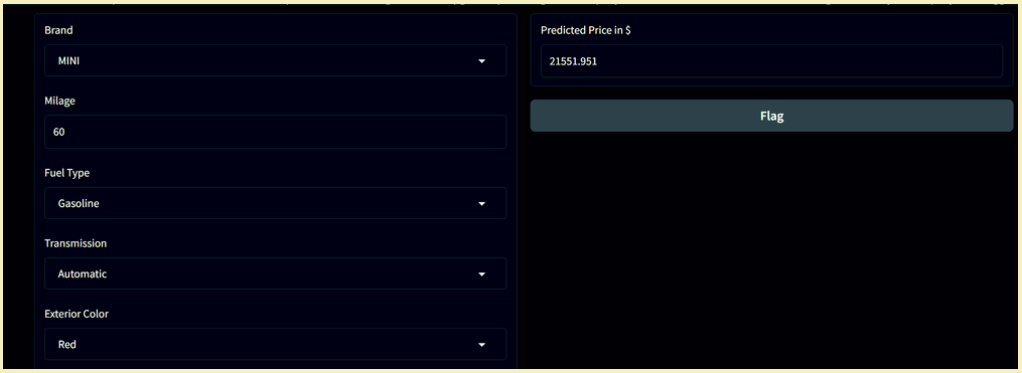
RANDOM FOREST WAS ULTIMATELY SELECTED AS THE **BEST MODEL** DUE TO ITS SLIGHTLY **LOWER RMSE AND COMPARABLE R2 SCORE COMPARED TO XGBOOST**, INDICATING A BETTER BALANCE BETWEEN ACCURACY AND POTENTIAL OVERFITTING.

Model Evaluation and Comparison

RESULTS

```
[35] def predict_price(brand ,milage ,fuel_type, transmission, ext_col, int_col, accident,clean_title, horsepower, engine_displacement,age ):
       # Create a DataFrame from the input features
       input_data = pd.DataFrame({
           'brand': [brand],
           'milage': [milage],
           'fuel_type': [fuel_type],
           'transmission': [transmission],
           'ext_col': [ext_col],
           'int_col': [int_col],
           'accident': [accident],
           'clean_title': [clean_title],
           'Horsepower': [horsepower],
           'Engine_Displacement': [engine_displacement],
           'age': [age],

       })

       # Preprocess the input data (similar to how you preprocessed the training data)
       data_encode(input_data) # Assuming data_encode function is defined as in your code

       # Make prediction using the best tuned model
       best_model = tuned_models[best_model_name]['model']
       predicted_price = best_model.predict(input_data)[0]

       return predicted_price
```

```
iface = gr.Interface(
    fn=predict_price,
    inputs=[
        gr.Dropdown(choices=['MINI', 'Lincoln', 'Chevrolet', 'Genesis', 'Mercedes-Benz', 'Audi',
                             'Ford', 'BMW', 'Tesla', 'Cadillac', 'Land', 'GMC', 'Toyota',
                             'Hyundai', 'Volvo', 'Volkswagen', 'Buick', 'Rivian', 'RAM',
                             'Hummer', 'Alfa', 'INFINITI', 'Jeep', 'Porsche', 'McLaren',
                             'Honda', 'Lexus', 'Dodge', 'Nissan', 'Jaguar', 'Acura', 'Kia',
                             'Mitsubishi', 'Rolls-Royce', 'Maserati', 'Pontiac', 'Saturn',
                             'Bentley', 'Mazda', 'Subaru', 'Ferrari', 'Aston', 'Lamborghini',
                             'Chrysler', 'Lucid', 'Lotus', 'Scion', 'smart', 'Karma',
                             'Plymouth', 'Suzuki', 'FIAT', 'Saab', 'Bugatti', 'Mercury','Polestar', 'Maybach'] , label ="Brand"),
        gr.Number(label="Milage"),
        gr.Dropdown(choices=['Gasoline','E85 Flex Fuel','Hybrid','Diesel','Plug-In Hybrid','Not supported'], label="Fuel Type"),

        gr.Dropdown(choices=['Automatic', 'Manual', 'Other'], label="Transmission"),
        gr.Dropdown(choices=['Red','Blue','Black','White','Others'], label="Exterior Color"),
        gr.Dropdown(choices=['Red','Blue','Black','White','Others'], label="Interior Color"),
        gr.Dropdown(choices=['None reported', 'At least 1 accident or damage reported'], label="Accident"),
        gr.Dropdown(choices=['Yes' , 'No'], label="Clean Title"),
        gr.Number(label="Horsepower"),
        gr.Number(label="Engine Displacement"),
        gr.Number(label="Age of the Car")
    ],
    outputs=gr.Number(label="Predicted Price in $"),
    title="Used Car Price Prediction",
    description="Predict the price of a used car based on various features.",
)

iface.launch()
```



Conclusions and Future Work

CONCLUSIONS

THIS PROJECT SUCCESSFULLY DEVELOPED A **MACHINE LEARNING MODEL** FOR **PREDICTING USED CAR PRICES** USING A **DATASET FROM KAGGLE**. THROUGH DATA PREPROCESSING, FEATURE ENGINEERING, AND HYPERPARAMETER TUNING, WE ACHIEVED A HIGH LEVEL OF ACCURACY, WITH THE **RANDOM FOREST MODEL** DEMONSTRATING THE BEST PERFORMANCE **(R2 SCORE > 0.90 AND LOW RMSE)**. THIS INDICATES THE EFFECTIVENESS OF MACHINE LEARNING IN CAPTURING THE COMPLEX RELATIONSHIPS BETWEEN CAR FEATURES AND THEIR PRICES.

THE **DEPLOYMENT** OF THE MODEL AS A **GRADIO WEB APPLICATION** PROVIDES A USER-FRIENDLY INTERFACE FOR PREDICTING CAR PRICES, MAKING IT ACCESSIBLE TO BOTH BUYERS AND SELLERS IN THE USED CAR MARKET. THIS TOOL CAN EMPOWER USERS TO MAKE INFORMED DECISIONS BY PROVIDING THEM WITH DATA-DRIVEN PRICE ESTIMATES.

Conclusions and Future Work

LIMITATIONS

WHILE THE PROJECT ACHIEVED PROMISING RESULTS, THERE ARE CERTAIN LIMITATIONS:

DATA DEPENDENCY: THE MODEL'S PERFORMANCE IS HIGHLY DEPENDENT ON THE QUALITY AND COMPREHENSIVENESS OF THE TRAINING DATA. INCORPORATING MORE DATA, INCLUDING ADDITIONAL FEATURES AND MORE RECENT CAR MODELS, COULD FURTHER ENHANCE THE MODEL'S ACCURACY AND GENERALIZABILITY.

REGIONAL VARIATIONS: THE CURRENT MODEL DOES NOT ACCOUNT FOR REGIONAL PRICE DIFFERENCES. INCORPORATING LOCATION-BASED FEATURES COULD IMPROVE PREDICTIONS FOR SPECIFIC GEOGRAPHIC AREAS.

MARKET FLUCTUATIONS: THE USED CAR MARKET IS SUBJECT TO DYNAMIC PRICE CHANGES DUE TO VARIOUS FACTORS, INCLUDING SEASONALITY AND ECONOMIC CONDITIONS. THE MODEL'S ACCURACY MAY DEGRADE OVER TIME, REQUIRING PERIODIC RETRAINING AND UPDATES.

Conclusions and Future Work

FUTURE WORK

FUTURE WORK ON THIS PROJECT COULD FOCUS ON:

EXPANDING THE DATASET: GATHERING MORE DATA FROM DIVERSE SOURCES AND INCLUDING ADDITIONAL FEATURES, SUCH AS VEHICLE HISTORY REPORTS, COULD IMPROVE THE MODEL'S ACCURACY.

INCORPORATING REGIONAL DATA: ADDING LOCATION-BASED FEATURES AND TRAINING SEPARATE MODELS FOR DIFFERENT REGIONS COULD ENHANCE THE MODEL'S ABILITY TO CAPTURE REGIONAL PRICE VARIATIONS.

DEVELOPING A DYNAMIC MODEL: IMPLEMENTING MECHANISMS TO UPDATE THE MODEL PERIODICALLY OR IN RESPONSE TO MARKET CHANGES COULD MAINTAIN ITS ACCURACY OVER TIME.

EXPLORING ADVANCED MODELS: EXPERIMENTING WITH MORE ADVANCED MACHINE LEARNING TECHNIQUES, SUCH AS DEEP LEARNING, COULD POTENTIALLY FURTHER IMPROVE PERFORMANCE.

References

Kaggle Dataset Link:
https://www.kaggle.com/competitions/playground-series-s4e9/data

DataPrep Documentation:
https://pypi.org/project/dataprep/

Optuna Documentation:
https://optuna.readthedocs.io/en/stable/

Gradio Documentation:
https://www.gradio.app/docs/python-client/introduction