RMarkdown dokument

Vukoje, Leo

Supplement / Prilog

Publication year / Godina izdavanja: **2024**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:052438

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-19**

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Preparation and exploratory analysis of a dataset with car prices

Ivan Malin
Kate Koval

Imports

Required Libraries

```
!pip install kaggle
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install seaborn
!pip install scipy
```

Data loading

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle datasets download -d hamdallak/car-prices-data
```

```
Dataset URL: https://www.kaggle.com/datasets/hamdallak/car-prices-data
License(s): unknown
car-prices-data.zip: Skipping, found more recently written local file, use "--force" to force download
```

```
!unzip car-prices-data.zip
```

```
Archive:  car-prices-data.zip
  inflating: CarPrice_Assignment.csv
  inflating: Description.txt
```

Descriptive statistics

Range, median, mode and quantiles for each variable except the car identifier

```
df.describe(include=['object', 'bool'], percentiles=[0.25, 0.50, 0.75, 0.90, 0.95, 0.99])
```

```
       CarName  fueltype aspiration  ... enginetype  cylinder  fuelsystem
count      205       205       205  ...       205       205         205
unique     147         2         2  ...         7         7           8
top       toyota       gas       std  ...       ohc      four        mpfi
freq        32       185       168  ...       148       159         94
25%        NaN       NaN       NaN  ...       NaN       NaN         NaN
50%        NaN       NaN       NaN  ...       NaN       NaN         NaN
75%        NaN       NaN       NaN  ...       NaN       NaN         NaN
90%        NaN       NaN       NaN  ...       NaN       NaN         NaN
95%        NaN       NaN       NaN  ...       NaN       NaN         NaN
99%        NaN       NaN       NaN  ...       NaN       NaN         NaN
top       NaN       NaN       NaN  ...       NaN       NaN         NaN
freq      NaN       NaN       NaN  ...       NaN       NaN         NaN
       car_ID  symboling  wheelbase  ...  citympg  highway  ...  price
count     NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
unique    NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
top       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
freq      NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
25%       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
50%       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
75%       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
90%       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
95%       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
99%       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
top       NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
freq      NaN     NaN     NaN     NaN  ...  NaN     NaN     ...  NaN
       [11 rows x 26 columns]
```

Standard Deviation

```
numeric_cols = df.select_dtype(include=['int64', 'float64'])
numeric_cols.drop(['car_ID'], axis=1).apply(lambda x: x.mode[0])
```

```
symboling    0.83
wheelbase   94.50
carlength   173.2
carwidth    65.54
carheight   54.10
curbweight  2145
enginesize   120
boreratio     3.1
stroke        3.4
compression   9.0
peakrpm      5500
citympg        31
highway        38
price       7295
dtype: float64
```

Variance

```
numeric_cols.var(ddof=1).sort_values(ascending=False)
```

```
curbweight    2.869000e+06
enginesize    1.700000e+04
price         6.000000e+03
carlength     1.000000e+03
carwidth      1.000000e+03
wheelbase     1.000000e+02
carheight     1.000000e+02
highway       1.000000e+02
citympg       1.000000e+02
peakrpm       1.000000e+02
car_ID        1.000000e+01
symboling     1.000000e+01
stroke        1.000000e+01
compression   1.000000e+01
boreratio     1.000000e+00
dtype: float64
```

Visualization

Price Range Frequency Histogram

```
price_range = pd.cut(df['price'], bins=[0, 5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000])
df['price_range'] = price_range
plt.figure(figsize=(10, 6))
sns.countplot(x='price_range', data=df, palette='viridis')
plt.title('Frequency of Cars in Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Count')
plt.xticks(rotation=45)
for p in plt.gca().patches:
    height = p.get_height()
    if height > 0:
        plt.gca().text(p.get_x() + p.get_width() / 2, height + 0.1, f'{int(height)}', ha='center', fontsize=10)
plt.tight_layout()
plt.show
```

```
<ipykrnl_1>:2: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will be treated as labels.
```

Average Price by Car Length Line Chart

```
df['carlength'] = df['carlength'].astype(int)
average_price = df.groupby('carlength')['price'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_price.index, y=average_price.values, marker='o', color='red')
plt.title('Average Price by Car Length')
plt.xlabel('Car Length')
plt.ylabel('Average Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Average Price by Car Width Line Chart

```
df['carwidth'] = df['carwidth'].astype(int)
average_price = df.groupby('carwidth')['price'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_price.index, y=average_price.values, marker='o', color='red')
plt.title('Average Price by Car Width')
plt.xlabel('Car Width')
plt.ylabel('Average Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Average Price by Curb Weight Line Chart

```
df['curbweight'] = df['curbweight'].astype(int)
average_price = df.groupby('curbweight')['price'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_price.index, y=average_price.values, marker='o', color='red')
plt.title('Average Price by Curb Weight')
plt.xlabel('Curb Weight')
plt.ylabel('Average Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Average Price by Engine Size Line Chart

```
df['enginesize'] = df['enginesize'].astype(int)
average_price = df.groupby('enginesize')['price'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_price.index, y=average_price.values, marker='o', color='red')
plt.title('Average Price by Engine Size')
plt.xlabel('Engine Size')
plt.ylabel('Average Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Average Price by Horsepower Line Chart

```
df['horsepower'] = df['horsepower'].astype(int)
average_price = df.groupby('horsepower')['price'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_price.index, y=average_price.values, marker='o', color='red')
plt.title('Average Price by Horsepower')
plt.xlabel('Horsepower')
plt.ylabel('Average Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Average Price by City MPG Line Chart

```
df['citympg'] = df['citympg'].astype(int)
average_price = df.groupby('citympg')['price'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_price.index, y=average_price.values, marker='o', color='red')
plt.title('Average Price by City MPG')
plt.xlabel('City MPG')
plt.ylabel('Average Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Average Price by Highway MPG Line Chart

```
df['highway'] = df['highway'].astype(int)
average_price = df.groupby('highway')['price'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=average_price.index, y=average_price.values, marker='o', color='red')
plt.title('Average Price by Highway MPG')
plt.xlabel('Highway MPG')
plt.ylabel('Average Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Correlational Matrix

```
numeric_cols = df.select_dtype(include=['int64', 'float64'])
correlation_matrix = numeric_cols.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.tight_layout()
plt.show()
```

Inferential Statistics

Hypothesis Testing

T-test

```
t_test_result = ttest_ind(fuel_gas, fuel_diesel)
print(f"T-test result: {t_test_result}")
```

```
T-test result: Ttest_indResult(statistic=-0.7411344171922661, pvalue=0.4592889265129488)
```

```
fuel_gas = df[df['fueltype'] == 'gas']['price']
fuel_diesel = df[df['fueltype'] == 'diesel']['price']
t_test_result = ttest_ind(fuel_gas, fuel_diesel)
print(f"T-test result: {t_test_result}")
```

```
T-test result: Ttest_indResult(statistic=-0.7411344171922661, pvalue=0.4592889265129488)
```

ANOVA test

```
cylinder_groups = [group['price'] for name, group in df.groupby('cylinder')]
f_statistic, p_value = f_oneway(*cylinder_groups)
print("ANOVA test results:")
print("F-statistic:", f_statistic)
print("p-value:", p_value)
```

```
ANOVA test results:
F-statistic: 16.777777777777777
p-value: 1.1666666666666666e-14
```

Chi-square test

```
contingency_table = pd.crosstab(df['cylinder'], df['fueltype'])
print("Contingency table:")
print(contingency_table)
print("")
chi2, p, dof, expected = chi2_contingency(contingency_table)
print("Chi-square test results:")
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("Expected frequencies:")
print(expected)
```

```
Contingency table:
fueltype  diesel  gas
cylinder
eight          0    5
five           1    4
four           9  150
six            2    6
three          0    1
twelve         0    1
two            0    2

Chi-square test results:
Chi-square statistic: 14.000000000000000
p-value: 0.0365000000000000
Degrees of freedom: 6
Expected frequencies:
[[ 0.  5.]
 [ 0.  5.]
 [ 8. 151.]
 [ 2.  6.]
 [ 0.  1.]
 [ 0.  1.]
 [ 0.  2.]]
```

Category table

```
cylinder_counts = df['cylinder']
```

```
cylinder
four      159
six        24
five       11
eight      10
two         2
three       1
twelve      1
Name: count, dtype: int64
```

Confidence Intervals

```
fuel_diesel = df[df['fueltype'] == 'diesel']['price']
fuel_gas = df[df['fueltype'] == 'gas']['price']

mean_diesel = fuel_diesel.mean()
mean_gas = fuel_gas.mean()
```

```
mean_diesel, mean_gas = 15843.125, 13056.7
```

```
confidence_interval_diesel = t.interval(0.95, len(fuel_diesel) - 1, loc=mean_diesel, scale=sem_diesel)
print("95% confidence interval for diesel:", confidence_interval_diesel)
```

```
95% confidence interval for diesel: (13000.000000000000, 18686.250000000000)
```

```
confidence_interval_gas = t.interval(0.95, len(fuel_gas) - 1, loc=mean_gas, scale=sem_gas)
print("95% confidence interval for gas:", confidence_interval_gas)
```

```
95% confidence interval for gas: (12000.000000000000, 14113.400000000000)
```

Manual Insertion of NA Values

```
df.loc[0, 'price'] = np.nan
df.loc[1, 'price'] = np.nan
```

```
nan, nan
```

Determination of Numeric Variables

```
numeric_columns = df.select_dtype(include=['int64', 'float64'])
print("Numeric variables:", numeric_columns.columns.tolist())
```

```
Numeric variables: ['car_ID', 'symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke', 'compression', 'peakrpm', 'citympg', 'highway', 'price']
```

Normalization and Standardization of Numeric Variables

```
numeric_columns = df.select_dtype(include=['int64', 'float64'])
scaler = MinMaxScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
print("Normalized data:")
print(df[numeric_columns].head())
```

```
Normalized data:
   car_ID  symboling  wheelbase  ...  highway     price
0   0.000   0.666667   0.058309  ...  0.222222       NaN
1   0.005   0.666667   0.058309  ...  0.222222       NaN
2   0.010   0.466667   0.230321  ...  0.333333  0.413433
3   0.015   0.733333   0.384840  ...  0.444444  0.449254
4   0.020   0.733333   0.373178  ...  0.333333  0.529851
[5 rows x 15 columns]
```

```
numeric_columns = df.select_dtype(include=['int64', 'float64'])
scaler = StandardScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
print("Standardized data:")
print(df[numeric_columns].head())
```

```
Standardized data:
   car_ID  symboling  wheelbase  ...  highway     price
0  -1.664   1.475829   -0.08744  ...  -0.546   -0.033
1  -1.657   1.475829   -0.08744  ...  -0.546   -0.033
2  -1.650   0.683485    0.744592  ...  -0.691   0.620
3  -1.643   1.267243    1.395489  ...  -0.836   0.697
4  -1.636   1.267243    1.316689  ...  -0.691   0.949
[5 rows x 15 columns]
```

Mean and Standard Deviation of Numeric Variables

```
numeric_columns = df.select_dtype(include=['int64', 'float64'])
```

```
car_ID        -1.664000e-17
symboling     -1.246800e-16
wheelbase     -1.246800e-16
carlength     -1.246800e-16
carwidth      -1.246800e-16
carheight     -1.246800e-16
curbweight    -1.246800e-16
enginesize    -1.246800e-16
boreratio     -1.246800e-16
stroke        -1.246800e-16
compression   -1.246800e-16
peakrpm       -1.246800e-16
citympg       -1.246800e-16
highway       -1.246800e-16
price         -1.246800e-16
dtype: float64
```

Standardization and Normalization with dplyr

```
!pip install dplyr
```

Brand Values Consistency Fix

```
df['CarName'] = df['CarName'].replace({'maxda': 'mazda', 'toyouta': 'toyota', 'vokswagen': 'volkswagen', 'vw': 'volkswagen'})
```

```
df['CarName'].value_counts()
```

```
CarName
toyota             32
nissan             18
mazda              17
mitsubishi         13
honda              13
volkswagen         12
subaru             12
peugeot            11
volvo              11
dodge               9
buick               8
bmw                 8
audi                7
plymouth            7
saab                6
porsche             5
chevrolet           3
alfa-romeo          3
jaguar              3
isuzu               4
maxda               2
toyouta             1
vokswagen           1
vw                  1
Name: count, dtyp
```

Frequency of Cars per Brand

```
plt.figure(figsize=(10, 6))
sns.countplot(x='CarName', data=df, palette='viridis')
plt.title('Frequency of Cars per Brand')
plt.xlabel('Car Brand')
plt.ylabel('Count')
plt.xticks(rotation=90)
for p in plt.gca().patches:
    height = p.get_height()
    if height > 0:
        plt.gca().text(p.get_x() + p.get_width() / 2, height + 0.1, f'{int(height)}', ha='center', fontsize=10)
plt.tight_layout()
plt.show()
```

Categories of Categorical Variables

```
fueltype_unique = df['fueltype'].unique()
```

```
fueltype_unique
```

```
array(['gas', 'diesel'],
      dtype=object)
```

```
aspiration_unique = df['aspiration'].unique()
```

```
aspiration_unique
```

```
array(['std', 'turbo'],
      dtype=object)
```

```
doornumber_unique = df['doornumber'].unique()
```

```
doornumber_unique
```

```
array(['two', 'four'],
      dtype=object)
```

```
carbody_unique = df['carbody'].unique()
```

```
carbody_unique
```

```
array(['convertible', 'hatchback', 'sedan', 'wagon', 'hardtop'],
      dtype=object)
```

```
drivewheel_unique = df['drivewheel'].unique()
```

```
drivewheel_unique
```

```
array(['rwd', 'fwd', '4wd'],
      dtype=object)
```

```
enginelocation_unique = df['enginelocation'].unique()
```

```
enginelocation_unique
```

```
array(['front', 'rear'],
      dtype=object)
```

```
enginetype_unique = df['enginetype'].unique()
```

```
enginetype_unique
```

```
array(['dohc', 'ohcv', 'ohc', 'l', 'rotor', 'ohcf', 'dohcv'],
      dtype=object)
```

```
cylinder_unique = df['cylinder'].unique()
```

```
cylinder_unique
```

```
array(['four', 'six', 'five', 'three', 'twelve', 'two', 'eight'],
      dtype=object)
```

```
fuelsystem_unique = df['fuelsystem'].unique()
```

```
fuelsystem_unique
```

```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', idi'],
      dtype=object)
```

Ordinal Encoding

```
cylinder_mapping = {'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'eight': 8, 'twelve': 12}
df['cylinder'] = df['cylinder'].map(cylinder_mapping)
```

```
df['cylinder'].unique()
```

```
array[nan, nan, 4, 6, 5, 3, 12, 8]
```

```
fueltype_mapping = {'gas': 0, 'diesel': 1}
df['fueltype'] = df['fueltype'].map(fueltype_mapping)
```

```
df['fueltype'].unique()
```

```
array[0, 1]
```

```
aspiration_mapping = {'std': 0, 'turbo': 1}
df['aspiration'] = df['aspiration'].map(aspiration_mapping)
```

```
df['aspiration'].unique()
```

```
array[0, 1]
```

```
doornumber_mapping = {'two': 2, 'four': 4}
df['doornumber'] = df['doornumber'].map(doornumber_mapping)
```

```
df['doornumber'].unique()
```

```
array[2, 4]
```

One-Hot Encoding

```
carbody_dummies = pd.get_dummies(df['carbody'], prefix='carbody', dtype=int)
df = pd.concat([df, carbody_dummies], axis=1)
print("DataFrame after one-hot encoding:")
print(df.head())
```

```
DataFrame after one-hot encoding:
   car_ID  symboling                   CarName  fueltype  aspiration  ...  price_range  carbody_convertible  carbody_hatchback  carbody_sedan  carbody_wagon
0       1          3        alfa-romeo giulia         0           0  ...  NaN          0                    0                  1              0
1       2          3       alfa-romeo stelvio         0           0  ...  NaN          0                    0                  1              0
2       3          1  alfa-romeo quadrifoglio         0           0  ...  NaN          0                    0                  0              0
3       4          2               audi 100ls         0           0  ...  NaN          0                    0                  1              0
4       5          2                audi 100le         0           0  ...  NaN          0                    0                  1              0
[5 rows x 32 columns]
```

Pervading Preprocessed Dataset

```
df.to_csv('car_prices_preprocessed.csv', sep=',', header=True)
```