빅데이터 분석3

강사님	강희재
강의 과목	빅데이터 분석
복습	☐
자료	https://docs.microsoft.com/ko-kr/learn/modules/predict-flight-delays-with-python/
작성일시	@2021년 5월 16일 오후 4:59

수업 형식

아래의 읽기 자료 검토하기

MS Learn(SD)

1. Exploratory Data Analysis

▼ 1) Exploratory Data Analysis (EDA): Car prices

⇒ used cars 데이터(csv)를 읽어와서 EDA에 대해 공부해보자

▼ 2) Loading data with Azure Databricks

⇒ 일단 UsedCars.csv 파일을 좌측 Data - Create New Table - Upload File해버리자

그리고 테이블 설정할때 첫행이 칼럼명이 되도록(First row is header) 설정할것.

```
# sql문을 사용해서 데이터를 가져오자
%sql
SELECT * FROM usedcars_
```

▸ (1) Spark Jobs

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
1	7450.0	65.0	82000.0	Petrol	86	1	0	1300	3	1015
2	7250.0	74.0	130025.0	Petrol	110	1	0	1600	3	1050
3	8950.0	80.0	64000.0	Petrol	110	0	0	1600	3	1055
4	11450.0	54.0	62987.0	Petrol	110	0	0	1600	5	1080
5	null	42.0	38932.0	Petrol	110	1	0	1600	3	1040
6	6950.0	80.0	62581.0	Petrol	110	0	0	1600	5	1075
7	8250.0	70.0	59017.0	petrol	107	1	1	1600	3	1080

Showing the first 1000 rows.

```
# df에 sql을 써서 가져온 데이터를 넣어주자
df = spark.sql("SELECT * FROM usedcars_")
df
```

```
# 데이터는 1446개
df.count()
```

```
df = spark.sql("SELECT * FROM usedcars_")
df
```

▸ df: pyspark.sql.dataframe.DataFrame = [Price: string, Age: string ... 8 more fields]

```
Out[1]: DataFrame[Price: string, Age: string, KM: string, FuelType: string, HP: string, MetColor: str
ing, Automatic: string, CC: string, Doors: string, Weight: string]
```

Command took 0.19 seconds -- by kmhkmh7749@hyuscec.onmicrosoft.com at 2021. 5. 17. 오후 10:54:10 on DBRCLS5

Cmd 12

Run the following cell to understand how many rows of data we have in this dataset.

Cmd 13

```
df.count()
```

▸ (1) Spark Jobs

```
Out[2]: 1446
```

Command took 0.48 seconds -- by kmhkmh7749@hyuscec.onmicrosoft.com at 2021. 5. 17. 오후 10:54:13 on DBRCLS5

▼ 3) Basic EDA with Azure Databricks

- Spark dataframe과 Pandas dataframe은 몇몇 특징은 비슷한데 꽤 다르다. 다행히도 Spark df를 Pandas df로 바꾸는 건 겁나 쉽다. 한번 보자.

```
# Spark df를 Pandas df로 바꿔주기
pdf = df.toPandas()
pdf
```

▸ (1) Spark Jobs

Out[21]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	7450.0	65.0	82000.0	Petrol	86	1	0	1300	3	1015
1	7250.0	74.0	130025.0	Petrol	110	1	0	1600	3	1050
2	8950.0	80.0	64000.0	Petrol	110	0	0	1600	3	1055
3	11450.0	54.0	62987.0	Petrol	110	0	0	1600	5	1080
4	None	42.0	38932.0	Petrol	110	1	0	1600	3	1040
5	6950.0	80.0	62581.0	Petrol	110	0	0	1600	5	1075
6	8250.0	70.0	59017.0	petrol	107	1	1	1600	3	1080
7	12950.0	44.0	41499.0	CNG	110	1	0	1600	5	1103
8	9950.0	65.0	65513.0	Petrol	110	1	1	1600	4	1070
9	7900.0	75.0	125400.0	Petrol	110	0	0	1600	3	1050

- 웃긴건 df.show와 df.head의 결과 값이 다르다는 것이다.

```
df.head(10)
```

▸ (1) Spark Jobs

```
Out[3]: [Row(Price='7450.0', Age='65.0', KM='82000.0', FuelType='Petrol', HP='86', MetColor='1', Automatic='0', CC='1300', Doors='3', Weight='1015'),
 Row(Price='7250.0', Age='74.0', KM='130025.0', FuelType='Petrol', HP='110', MetColor='1', Automatic='0', CC='1600', Doors='3', Weight='1050'),
 Row(Price='8950.0', Age='80.0', KM='64000.0', FuelType='Petrol', HP='110', MetColor='0', Automatic='0', CC='1600', Doors='3', Weight='1055'),
 Row(Price='11450.0', Age='54.0', KM='62987.0', FuelType='Petrol', HP='110', MetColor='0', Automatic='0', CC='1600', Doors='5', Weight='1080'),
 Row(Price=None, Age='42.0', KM='38932.0', FuelType='Petrol', HP='110', MetColor='1', Automatic='0', CC='1600', Doors='3', Weight='1040'),
 Row(Price='6950.0', Age='80.0', KM='62581.0', FuelType='Petrol', HP='110', MetColor='0', Automatic='0', CC='1600', Doors='5', Weight='1075'),
 Row(Price='8250.0', Age='70.0', KM='59017.0', FuelType='petrol', HP='107', MetColor='1', Automatic='1', CC='1600', Doors='3', Weight='1080'),
 Row(Price='12950.0', Age='44.0', KM='41499.0', FuelType='CNG', HP='110', MetColor='1', Automatic='0', CC='1600', Doors='5', Weight='1103'),
 Row(Price='9950.0', Age='65.0', KM='65513.0', FuelType='Petrol', HP='110', MetColor='1', Automatic='1', CC='1600', Doors='4', Weight='1070'),
 Row(Price='7900.0', Age='75.0', KM='125400.0', FuelType='Petrol', HP='110', MetColor='0', Automatic='0', CC='1600', Doors='3', Weight='1050')]
```

```
df.show(10)
```

(1) Spark Jobs

```
+-------+----+--------+--------+---+---------+---------+----+-----+------+
|  Price| Age|      KM|FuelType| HP|MetColor|Automatic|  CC|Doors|Weight|
+-------+----+--------+--------+---+---------+---------+----+-----+------+
| 7450.0|65.0| 82000.0|  Petrol| 86|        1|        0|1300|    3|  1015|
| 7250.0|74.0|130025.0|  Petrol|110|        1|        0|1600|    3|  1050|
| 8950.0|80.0| 64000.0|  Petrol|110|        0|        0|1600|    3|  1055|
|11450.0|54.0| 62987.0|  Petrol|110|        0|        0|1600|    5|  1080|
|   null|42.0| 38932.0|  Petrol|110|        1|        0|1600|    3|  1040|
| 6950.0|80.0| 62581.0|  Petrol|110|        0|        0|1600|    5|  1075|
| 8250.0|70.0| 59017.0|  petrol|107|        1|        1|1600|    3|  1080|
|12950.0|44.0| 41499.0|     CNG|110|        1|        0|1600|    5|  1103|
| 9950.0|65.0| 65513.0|  Petrol|110|        1|        1|1600|    4|  1070|
| 7900.0|75.0|125400.0|  Petrol|110|        0|        0|1600|    3|  1050|
+-------+----+--------+--------+---+---------+---------+----+-----+------+
only showing top 10 rows
```

- df의 기술통계량을 살펴보자

```
summary = df.describe()
display(summary)
```

(1) Spark Jobs

summary: pyspark.sql.dataframe.DataFrame = [summary: string, Price: string ... 9 more fields]

	summary	Price	Age	KM	FuelType	HP
1	count	1439	1441	1440	1446	1446
2	mean	10728.397498262682	55.95766828591256	68534.575	null	101.4661134
3	stddev	3623.832644481436	18.57766015357036	37476.02400668296	null	14.98931198
4	min	10000.0	1.0	1.0	CNG	107
5	max	9995.0	9.0	99971.0	petrol	98

Showing all 5 rows.

```
# df_typed에 칼럼별로 int로 형변환해서 값 넣어주기
df_typed = spark.sql("SELECT cast(Price as int), cast(Age as int), cast(KM as int), FuelType, cast(HP as int), cast(MetColor as ir
df_typed
```

```
df_typed = spark.sql("SELECT cast(Price as int), cast(Age as int), cast(KM as int), FuelType,
cast(HP as int), cast(MetColor as int), cast(Automatic as int), cast(CC as int), cast(Doors as
int), cast(Weight as int) FROM usedcars_")
df_typed
```

df_typed: pyspark.sql.dataframe.DataFrame = [Price: integer, Age: integer ... 8 more fields]

```
Out[8]: DataFrame[Price: int, Age: int, KM: int, FuelType: string, HP: int, MetColor: int, Automatic:
int, CC: int, Doors: int, Weight: int]
```

```
# FuelType 칼럼의 값들중 중복제거하여 뽑아보기
display(df_typed.select("FuelType").distinct())
```

▸ (5) Spark Jobs

	FuelType
1	Diesel
2	diesel
3	petrol
4	CNG
5	methane
6	Petrol
7	CompressedNaturalGas

Showing all 7 rows.

- replace함수 써서 FuelType칼럼의 값들의 앞글자를 대문자에서 소문자로 바꾸기
- "CompressedNaturalGas","methane","CNG" 값들은 "cng"로 바꿔주기

```
1  df_cleaned_fueltype =
   df_typed.na.replace(["Diesel","Petrol","CompressedNaturalGas","methane","CNG"],
   ["diesel","petrol","cng","cng","cng"],"FuelType")
2  display(df_cleaned_fueltype.select("FuelType").distinct())
```

▸ (5) Spark Jobs
▸ df_cleaned_fueltype: pyspark.sql.dataframe.DataFrame = [Price: integer, Age: integer ... 8 more fields]

	FuelType
1	diesel
2	petrol
3	cng

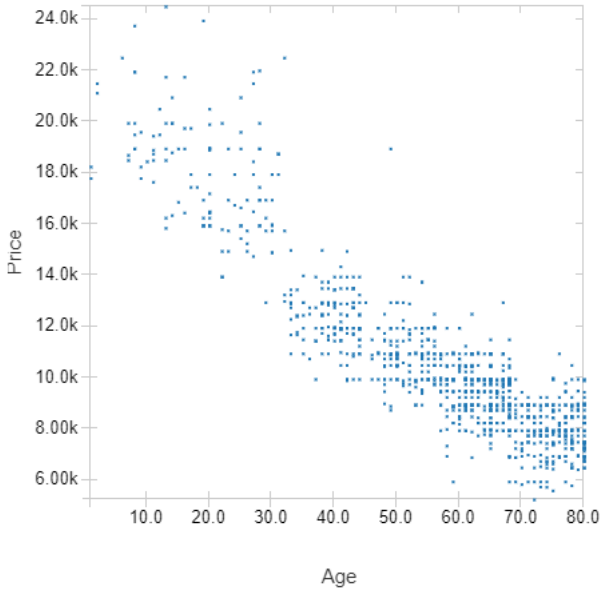
Showing all 3 rows.

```
# 깔끔하게 전처리가 끝난 파일을 새로운 이름의 파일로 만들기
df_cleaned_of_nulls.write.mode("overwrite").saveAsTable("usedcars_clean_csv")
```

- 그렇게 저장한 깔끔한 파일 불러내서 Price와 Age로 상관관계 시각화 하기

```sql
%sql
SELECT Price, Age FROM usedcars_clean_csv WHERE FuelType = 'petrol'
```

▸ (1) Spark Jobs



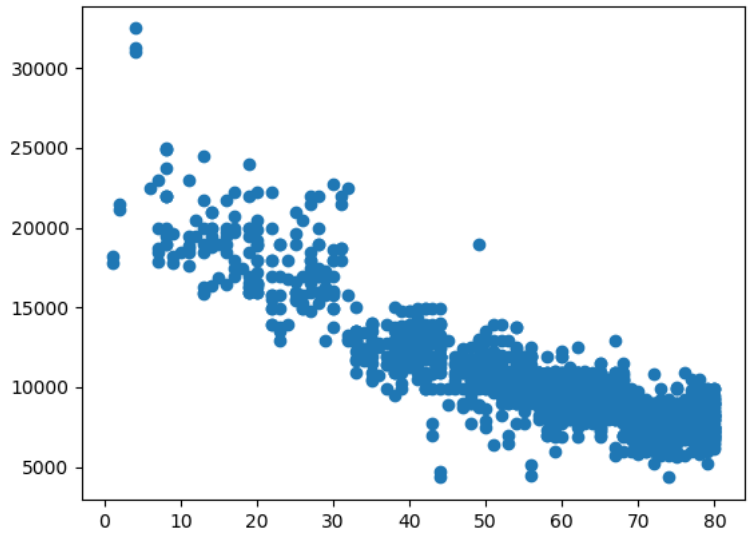
Showing sample based on the first 1000 rows.

- pandas 스타일로 출력해보기

  (더 후져보이는 건 기분탓....?)

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

fig, ax = plt.subplots()
pdf = df_cleaned_of_nulls.toPandas()
ax.scatter(pdf.Age, pdf.Price)

display(fig)
```

▸ (1) Spark Jobs



▼ 4) Advanced EDA with Azure Databricks

```
# 자 시작

import numpy as np
import pandas as pd

df = spark.sql("SELECT * FROM usedcars_clean_csv")
df_affordability = df.selectExpr("Age","KM", "CASE WHEN Price < 12000 THEN 1 ELSE 0 END as Affordable")
display(df_affordability)
```

▸ (1) Spark Jobs
▸ df_affordability: pyspark.sql.dataframe.DataFrame = [Age: integer, KM: integer ... 1 more fields]

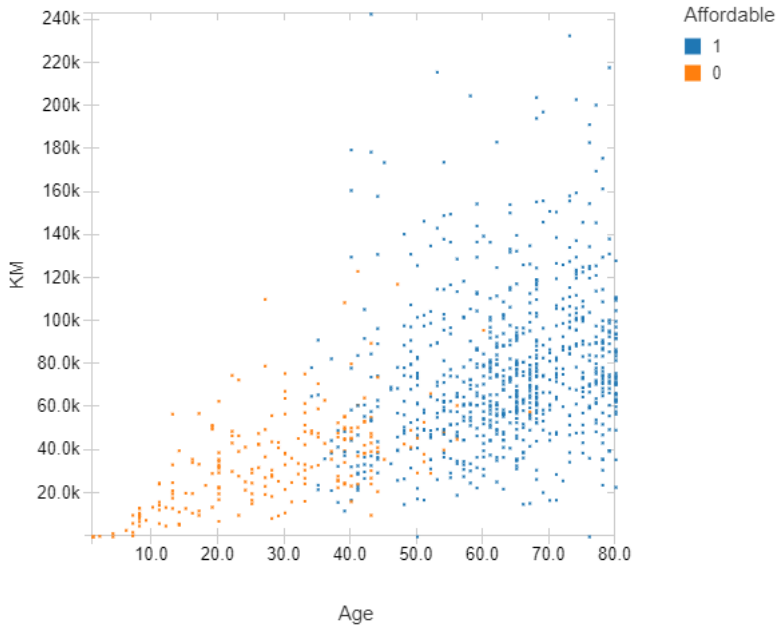
	Age	KM	Affordable
1	65	82000	1
2	74	130025	1
3	80	64000	1
4	54	62987	1
5	80	62581	1
6	70	59017	1
7	44	41499	0

Showing the first 1000 rows.

- df_affordability의 Scatter 살펴보기

```
1  display(df_affordability)
```

▸ (1) Spark Jobs



Showing sample based on the first 1000 rows.

```
# x값 y값 나눠주기
X = df_affordability.select("Age", "KM").toPandas().values
y = df_affordability.select("Affordable").toPandas().values
```

```
1  X
```

```
Out[5]: array([[    65,  82000],
       [    74, 130025],
       [    80,  64000],
       ...,
       [    80,  44444],
       [    54,  46230],
       [    77,  62285]], dtype=int32)
```

Command took 0.02 seconds -- by kmhkmh7749@hyuscec.onmicrosoft.com at 2021. 5. 17. 오후 11:47:46 on DBRCLS5

Cmd 16

```
1  y
```

```
Out[6]: array([[1],
       [1],
       [1],
       ...,
       [1],
       [0],
       [1]], dtype=int32)
```

Command took 0.01 seconds -- by kmhkmh7749@hyuscec.onmicrosoft.com at 2021. 5. 17. 오후 11:47:46 on DBRCLS5

```
# normalize를 위해 StandardScaler사용하기
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# scaling 결과 살펴보기
print(pd.DataFrame(X).describe().round(2))
print(pd.DataFrame(X_scaled).describe().round(2))
```

```
print(pd.DataFrame(X).describe().round(2))
print(pd.DataFrame(X_scaled).describe().round(2))
```

```
             0          1
count  1436.00    1436.00
mean     55.95   68533.26
std      18.60   37506.45
min       1.00       1.00
25%      44.00   43000.00
50%      61.00   63389.50
75%      70.00   87020.75
max      80.00  243000.00
             0        1
count  1436.00  1436.00
mean     -0.00     0.00
std       1.00     1.00
min      -2.96    -1.83
25%      -0.64    -0.68
50%       0.27    -0.14
75%       0.76     0.49
max       1.29     4.65
```

```
from sklearn import linear_model
# Create a linear model for Logistic Regression
clf = linear_model.LogisticRegression(C=1)

# we create an instance of Neighbours Classifier and fit the data.
clf.fit(X_scaled, y)

# 위젯에 Age랑 Distance값 넣어놓기
dbutils.widgets.text("Age","40", "Age (months)")
dbutils.widgets.text("Distance Driven", "40000","Distance Driven (KM)")
```


widgets : https://docs.microsoft.com/ko-kr/azure/databricks/notebooks/widgets

```
# 위젯을 써보자. 다른 매개변수를 가져올 수 있쥬
age = int(dbutils.widgets.get("Age"))
km = int(dbutils.widgets.get("Distance Driven"))

scaled_input = scaler.transform([[age, km]])

prediction = clf.predict(scaled_input)

print("Can I afford a car that is {} month(s) old with {} KM's on it?".format(age,km))
print("Yes (1)" if prediction[0] == 1 else "No (1)")
```

```
age = int(dbutils.widgets.get("Age"))
km = int(dbutils.widgets.get("Distance Driven"))

scaled_input = scaler.transform([[age, km]])

prediction = clf.predict(scaled_input)

print("Can I afford a car that is {} month(s) old with {} KM's on it?".format(age,km))
print("Yes (1)" if prediction[0] == 1 else "No (1)")
```

```
Can I afford a car that is 40 month(s) old with 40000 KM's on it?
No (1)
```

```
# 예측한 값들 살펴보기
scaled_inputs = scaler.transform(X)
predictions = clf.predict(scaled_inputs)
print(predictions)
```

```
/databricks/python/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning:
Data with input dtype int32 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
[1 1 1 ... 1 1 1]
```

- 모델 정확도 확인

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y, predictions)
print("Model Accuracy: {}".format(score.round(3)))
```

```
Model Accuracy: 0.926
```

- One hot encoding and feature scaling

```
df_ohe = df.toPandas().copy(deep=True)
df_ohe['FuelType'] = df_ohe['FuelType'].astype('category')
df_ohe = pd.get_dummies(df_ohe)

df_ohe.head(15)
```

▸ (1) Spark Jobs

Out[14]:

	Price	Age	KM	HP	MetColor	Automatic	CC	Doors	Weight	FuelType_cng	FuelType_diesel	FuelType_petrol
0	7450	65	82000	86	1	0	1300	3	1015	0	0	1
1	7250	74	130025	110	1	0	1600	3	1050	0	0	1
2	8950	80	64000	110	0	0	1600	3	1055	0	0	1
3	11450	54	62987	110	0	0	1600	5	1080	0	0	1
4	6950	80	62581	110	0	0	1600	5	1075	0	0	1
5	8250	70	59017	107	1	1	1600	3	1080	0	0	1
6	12950	44	41499	110	1	0	1600	5	1103	1	0	0
7	9950	65	65513	110	1	1	1600	4	1070	0	0	1
8	7900	75	125400	110	0	0	1600	3	1050	0	0	1
9	6495	74	96302	86	1	0	1300	3	1015	0	0	1
10	9500	66	74963	107	0	1	1600	3	1085	0	0	1
11	6490	80	100123	110	1	0	1600	3	1050	0	0	1
12	7250	80	110887	110	1	0	1600	3	1055	0	0	1
13	9450	80	66843	110	1	0	1600	5	1075	0	0	1
14	16500	27	37177	110	0	0	1600	5	1130	0	0	1

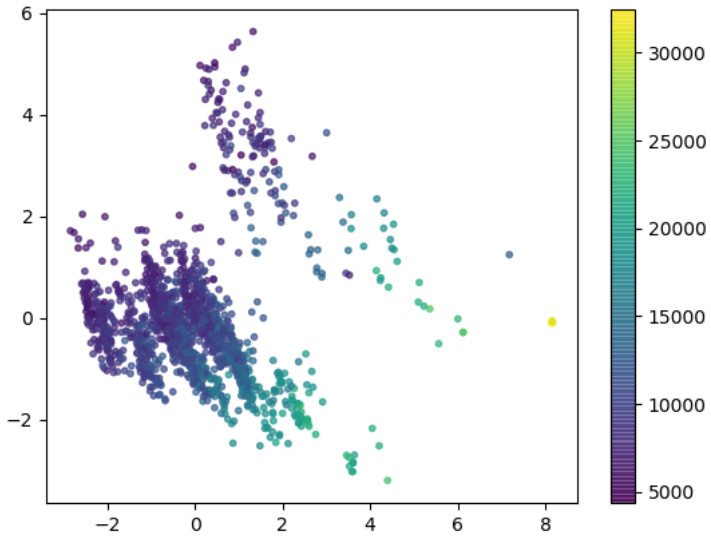
- df_ohe_scaled 라는 새로운 데이터 프레임 만들기

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
columns_to_scale = ['Age', 'KM', 'HP', 'CC','Weight']
df_ohe_scaled = df_ohe.dropna().copy()
df_ohe_scaled[columns_to_scale] = scaler.fit_transform(df_ohe.dropna()[columns_to_scale])

df_ohe_scaled.head(15)
```

```
/databricks/python/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: DataConversionWarnin
g: Data with input dtype int32 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/databricks/python/lib/python3.6/site-packages/sklearn/base.py:464: DataConversionWarning: Data with i
nput dtype int32 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
Out[15]:
```

	Price	Age	KM	HP	MetColor	Automatic	CC	Doors	Weight	FuelType_cng	FuelType_diesel
0	7450	0.486886	0.359176	-1.035138	1	0	-1.425994	3	-1.091915	0	0
1	7250	0.970926	1.640069	0.567440	1	0	0.177279	3	-0.426804	0	0
2	8950	1.293619	-0.120908	0.567440	0	0	0.177279	3	-0.331788	0	0
3	11450	-0.104718	-0.147926	0.567440	0	0	0.177279	5	0.143291	0	0
4	6950	1.293619	-0.158755	0.567440	0	0	0.177279	5	0.048275	0	0
5	8250	0.755797	-0.253812	0.367118	1	1	0.177279	3	0.143291	0	0
6	12950	-0.642540	-0.721041	0.567440	1	0	0.177279	5	0.580364	1	0
7	9950	0.486886	-0.080554	0.567440	1	1	0.177279	4	-0.046740	0	0
8	7900	1.024708	1.516714	0.567440	0	0	0.177279	3	-0.426804	0	0
9	6495	0.970926	0.740630	-1.035138	1	0	-1.425994	3	-1.091915	0	0
10	9500	0.540668	0.171490	0.367118	0	1	0.177279	3	0.238307	0	0
11	6490	1.293619	0.842542	0.567440	1	0	0.177279	3	-0.426804	0	0
12	7250	1.293619	1.129632	0.567440	1	0	0.177279	3	-0.331788	0	0
13	9450	1.293619	-0.045082	0.567440	1	0	0.177279	5	0.048275	0	0
14	16500	-1.556838	-0.836314	0.567440	0	0	0.177279	5	1.093450	0	0

```
# 차원축소(Dimensionality reduction) 살펴보기
# 데이터 양 줄이기

from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

fig, ax = plt.subplots()

features = ['Age', 'KM', 'HP', 'Weight', 'CC', 'Doors',  'Automatic', 'MetColor', 'FuelType_cng', 'FuelType_diesel', 'FuelTyp

x_2d = PCA(n_components=2).fit_transform(df_ohe_scaled[features])
sc = plt.scatter(x_2d[:,0], x_2d[:,1], c=df_ohe_scaled['Price'], s=10, alpha=0.7)
plt.colorbar(sc)

display(fig)
```




```
#

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

fig, ax = plt.subplots()

features_RFR = ['Age', 'KM', 'HP', 'Weight', 'CC', 'Doors', 'Automatic', 'MetColor', 'FuelType_cng', 'FuelType_diesel', 'Fuel

# Create train and test data
X = df_ohe[features_RFR].as_matrix()
y = df.toPandas()['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =0)

# Initialize  a random forest regressor
# 'Train' the model
RandomForestReg = RandomForestRegressor()
RandomForestReg.fit(X_train, y_train)

imp = pd.DataFrame(
        RandomForestReg.feature_importances_ ,
        columns = ['Importance'] ,
        index = features_RFR
    )
imp = imp.sort_values( [ 'Importance' ] , ascending = True )
imp['Importance'].plot(kind='barh')

display(fig)
```