# USED CARS PRICE PREDICTION

Prepared by:
Kateryna Kondratovych
Claudia Słaboń

# INTRODUCTION

Out[6]:

| ID | Name | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | Engine | Power | Seats | New_Price | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3509 | Hyundai i20 Asta Option 1.4 CRDi | Coimbatore | 2017 | 24153 | Diesel | Manual | Second | 22.54 | 1396.0 | 88.73 | 5.0 | NaN | 9.18 |
| 3332 | Mahindra Quanto C6 | Mumbai | 2013 | 35000 | Diesel | Manual | First | 17.21 | 1493.0 | 100.00 | 7.0 | NaN | 3.49 |
| 5383 | Tata Indigo LX BSII | Hyderabad | 2005 | 92000 | Diesel | Manual | Second | 16.10 | 1405.0 | 70.00 | 5.0 | NaN | 1.10 |
| 1891 | Maruti Swift Ldi BSIV | Delhi | 2014 | 62000 | Diesel | Manual | First | 17.80 | 1248.0 | 75.00 | 5.0 | NaN | 3.75 |
| 5757 | Toyota Innova Crysta 2.4 VX MT | Kolkata | 2017 | 31000 | Diesel | Manual | First | 13.68 | 2393.0 | 147.80 | 7.0 | 21.36 Lakh | 16.95 |

In [7]:

```
X_train.describe()
```

Out[7]:

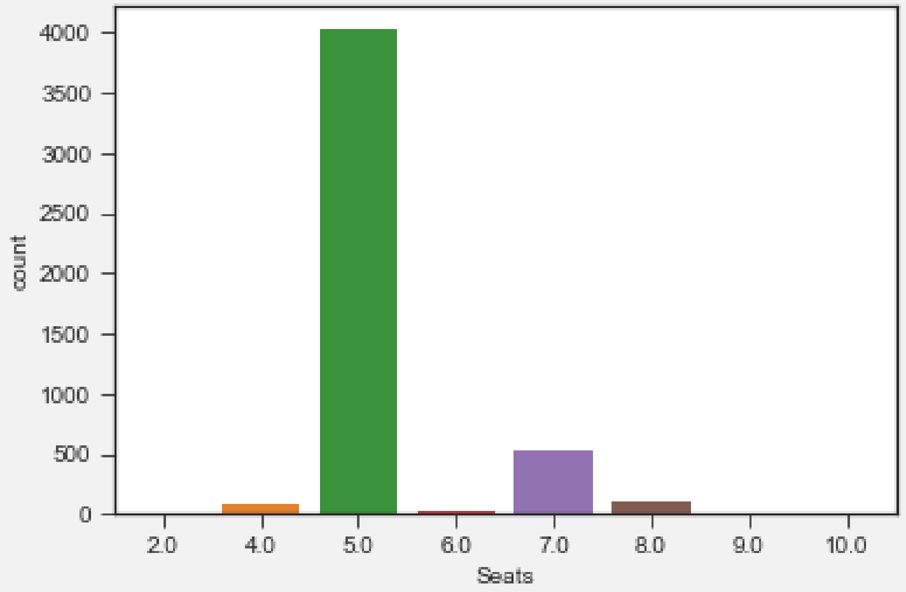
| | Year | Kilometers_Driven | Mileage | Engine | Power | Seats | Price |
|---|---|---|---|---|---|---|---|
| count | 4815.000000 | 4.815000e+03 | 4814.000000 | 4790.000000 | 4702.000000 | 4785.000000 | 4815.000000 |
| mean | 2013.369055 | 5.915726e+04 | 18.122254 | 1619.069102 | 112.886918 | 5.274399 | 9.429570 |
| std | 3.286841 | 1.005633e+05 | 4.576851 | 594.356557 | 52.919245 | 0.804034 | 11.246342 |
| min | 1998.000000 | 1.710000e+02 | 0.000000 | 624.000000 | 34.200000 | 0.000000 | 0.440000 |
| 25% | 2011.000000 | 3.349050e+04 | 15.170000 | 1198.000000 | 76.000000 | 5.000000 | 3.500000 |
| 50% | 2014.000000 | 5.308000e+04 | 18.120000 | 1493.000000 | 94.000000 | 5.000000 | 5.600000 |
| 75% | 2016.000000 | 7.300000e+04 | 21.100000 | 1989.500000 | 138.100000 | 5.000000 | 9.775000 |
| max | 2019.000000 | 6.500000e+06 | 33.540000 | 5461.000000 | 560.000000 | 10.000000 | 160.000000 |

Now it is looking much bettter, but still we have some 0 values in variables, where we cannot have them (for example 0 Seats) and NA. In next section we will work with this problem

PLAN:
Introduction
Missing Values
Descriptive Analysis
Variable Selection and Transformation
Training/test data division, cross-validation
Usage of Chosen Method on test data
Summary and Conclusions

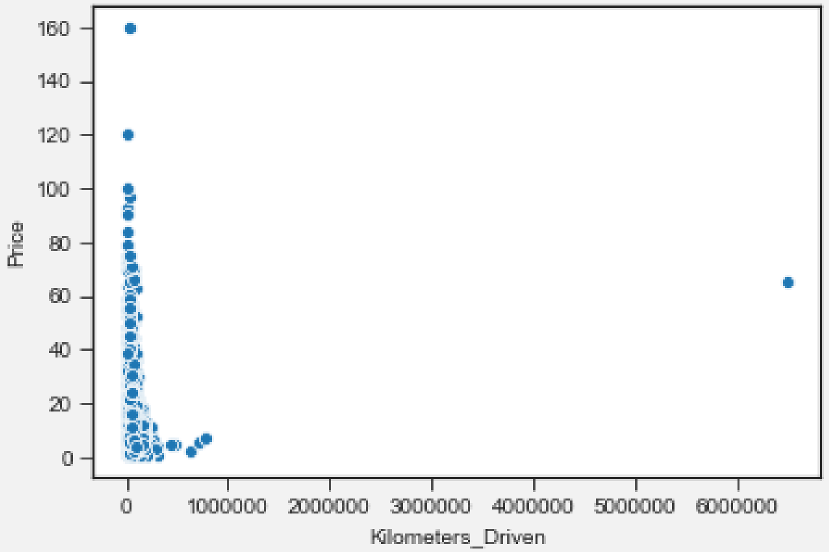
# MISSING VALUES

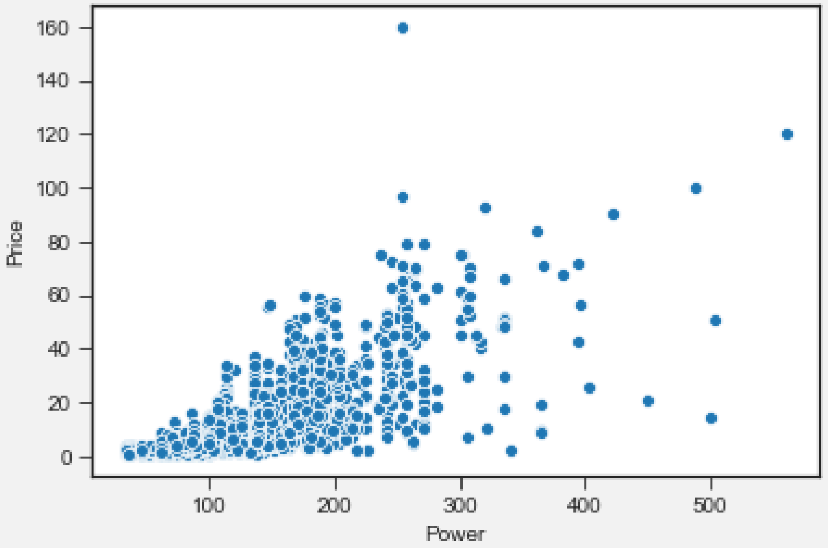
```
Name                  0
Location              0
Year                  0
Kilometers_Driven     0
Fuel_Type             0
Transmission          0
Owner_Type            0
Mileage               1
Engine               25
Power               113
Seats                30
New_Price          4155
Price                 0
dtype: int64

Percentage of NA's:
Name                0.00
Location            0.00
Year                0.00
Kilometers_Driven   0.00
Fuel_Type           0.00
Transmission        0.00
Owner_Type          0.00
Mileage             0.02
Engine              0.52
Power               2.35
Seats               0.62
New_Price          86.29
Price               0.00
dtype: float64
```

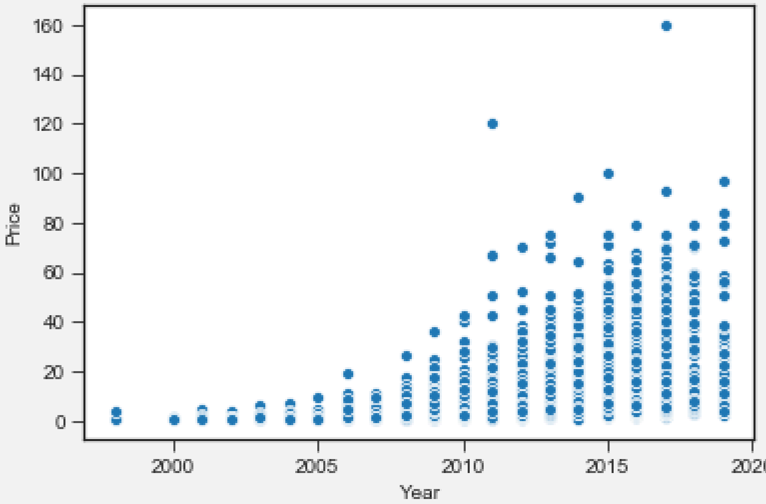


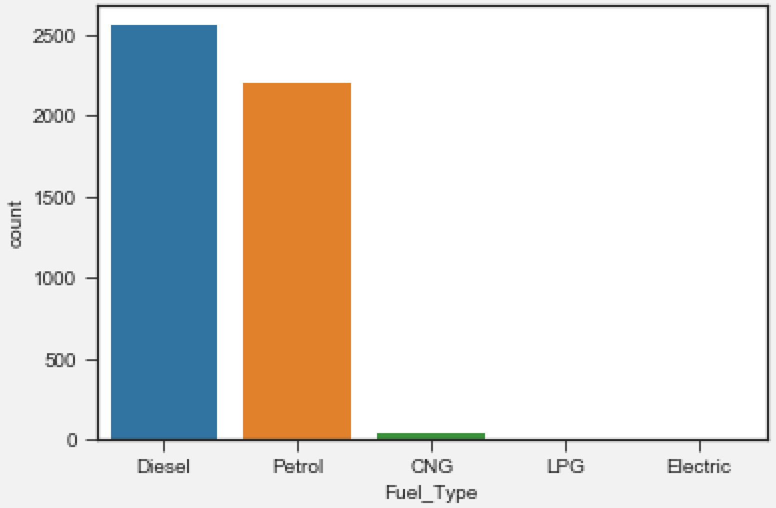

```
by_brand3 = X_train.groupby(['Brand','Brand2','Brand3'])
by_brand2 = X_train.groupby(['Brand','Brand2'])
by_brand1 = X_train.groupby(['Brand'])
def impute_mean(series):
    return series.fillna(series.mean())
X_train.Mileage = by_brand3.Mileage.transform(impute_mean)
X_train[X_train['Mileage'].isnull()]
```

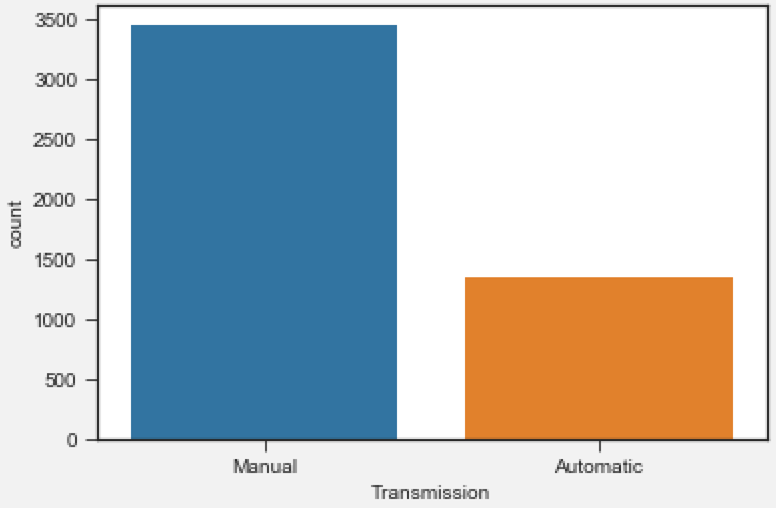
# DESCRIPTIVE ANALYSIS



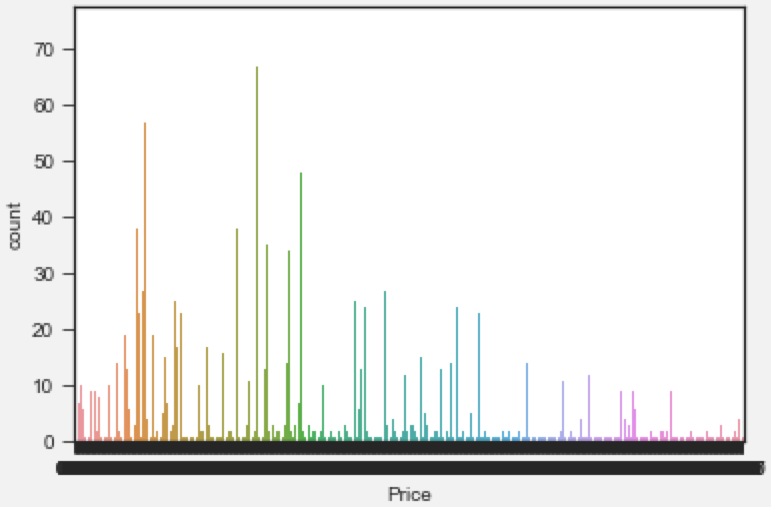



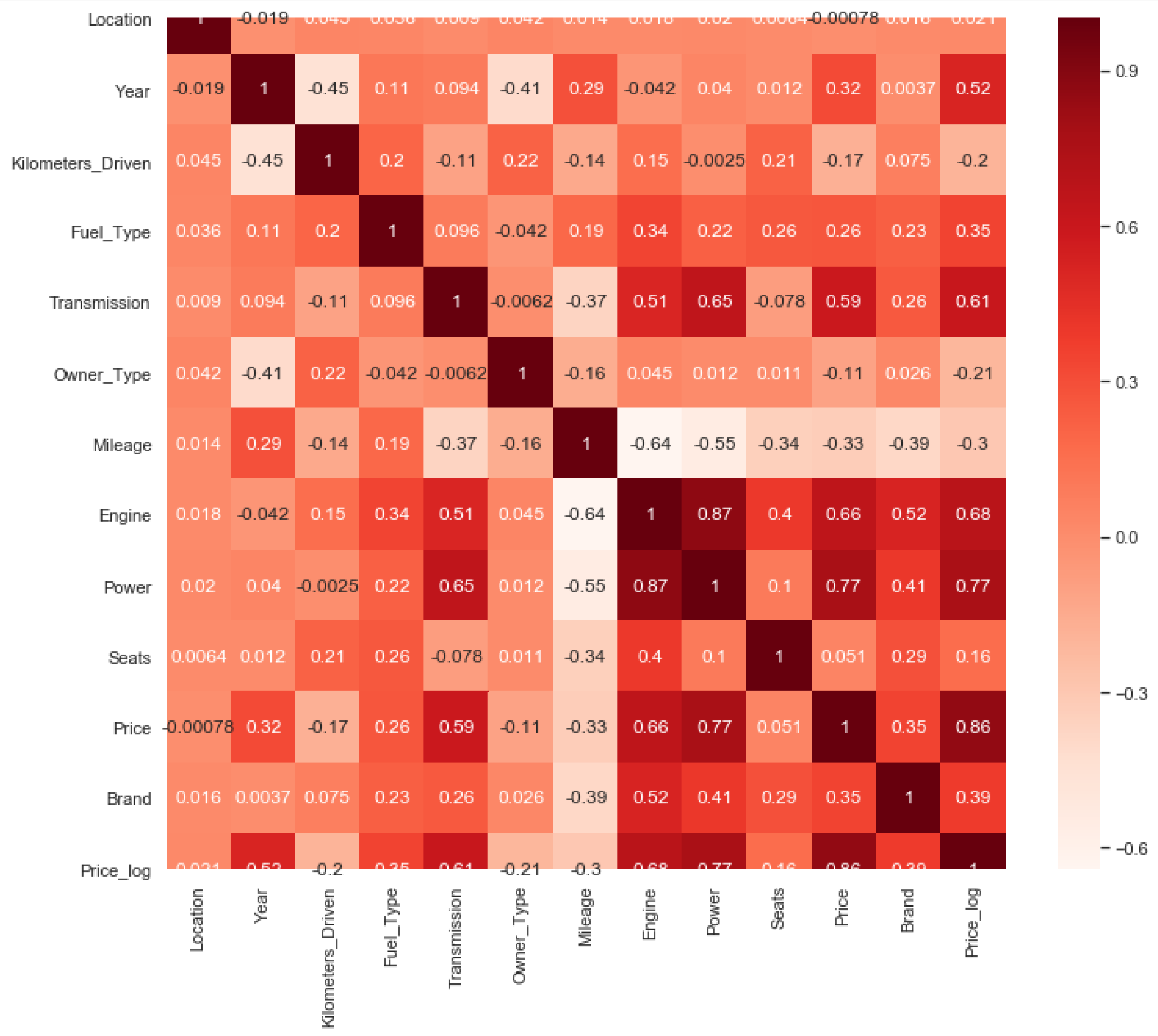



# VARIABLE SELECTION AND TRANSFORMATION




Skewness: 3.0255877459369818

Year 0.515176
Transmission 0.613594
Engine 0.684915
Power 0.766890
Price 0.856055
Brand 0.387337
Price_log 1.000000



# LINEAR REGRESSION

In [57]:

```python
import random
from sklearn.model_selection import KFold
for z in range(10):
    trainRes = []
    valRes = []
    kf = KFold(n_splits=5, shuffle=True, random_state=random.randint(0,10000))
    for train, test in kf.split(X_train2.index.values):
        mod = sm.GLM.from_formula(formula="Price_log ~ Transmission + Power +Year", data=X_train_f.iloc[train], family=sm.famili
        res = mod.fit()
        predsTrain = res.predict()
        preds = res.predict(X_train_f.iloc[test])
        trainRes.append(metrics.mean_squared_error(X_train_f.iloc[train].Price_log, predsTrain))
        valRes.append(metrics.mean_squared_error(X_train_f.iloc[test].Price_log, preds))
    print("Train AUC:", np.mean(trainRes), "Valid MSE:", np.mean(valRes))
```

```
Train AUC: 0.1275945720645234 Valid MSE: 0.12831727597437734
Train AUC: 0.12761032812077613 Valid MSE: 0.12818194157573876
Train AUC: 0.1276075343983899 Valid MSE: 0.12821580511469835
Train AUC: 0.12756324668486982 Valid MSE: 0.1285784132328012
Train AUC: 0.12762263267644886 Valid MSE: 0.12806959913901234
Train AUC: 0.12757056249161258 Valid MSE: 0.1285375786235907
Train AUC: 0.12757353786277217 Valid MSE: 0.1285156784452455
Train AUC: 0.1276057067005062 Valid MSE: 0.12821169445385755
Train AUC: 0.1275354483536078 Valid MSE: 0.12885624213058575
Train AUC: 0.12761289075293686 Valid MSE: 0.12815004727379206
```

```python
import random
from sklearn.model_selection import KFold
for z in range(10):
    trainRes = []
    valRes = []
    kf = KFold(n_splits=5, shuffle=True, random_state=random.randint(0,10000))
    for train, test in kf.split(X_train2.index.values):
        mod = sm.GLM.from_formula(formula="Price_log ~ Transmission + Power +Year+Fuel_Type+Brand", data=X_train_f.iloc[train],
        res = mod.fit()
        predsTrain = res.predict()
        preds = res.predict(X_train_f.iloc[test])
        trainRes.append(metrics.mean_squared_error(X_train_f.iloc[train].Price_log, predsTrain))
        valRes.append(metrics.mean_squared_error(X_train_f.iloc[test].Price_log, preds))
    print("Train AUC:", np.mean(trainRes), "Valid MSE:", np.mean(valRes))
```

```
Train AUC: 0.07302763553890937 Valid MSE: 0.1011634980011999
Train AUC: 0.07298551321078126 Valid MSE: 0.10160697994625134
Train AUC: 0.07294813595158618 Valid MSE: 0.10217944142961957
Train AUC: 0.0730209122767681 Valid MSE: 0.10208639132572044
Train AUC: 0.07295204710083966 Valid MSE: 0.10187526789246867
Train AUC: 0.07300728947394422 Valid MSE: 0.10162813289375865
Train AUC: 0.07297449342117264 Valid MSE: 0.1016724216548834
Train AUC: 0.07293803779894117 Valid MSE: 0.10204718696630803
Train AUC: 0.07294829575979596 Valid MSE: 0.18082738300318998
Train AUC: 0.07296784463098271 Valid MSE: 0.10241988786548735
```

# KNN

```
kf = KFold(n_splits=5, shuffle=True, random_state=2018)
probs = []
indicies = []
mse = []
r=[]

n_neighbors = 25
clf = neighbors.KNeighborsRegressor(n_neighbors, n_jobs=-1, p=2) #p=2 means Euclidean metrics
for train, test in kf.split(X_train2.index.values):
    clf.fit(X_train2.iloc[train][features].values, X_train_f.iloc[train]["Price_log"].values)
    prob = clf.predict(X_train2.iloc[test][features].values)
    probs.append(prob)
    indicies.append(test)
    mse.append(metrics.mean_squared_error(X_train_f.iloc[test]["Price_log"].values, prob))
    r.append(metrics.r2_score(X_train_f.iloc[test]["Price_log"].values, prob))
print("MSE:")
print(np.mean(mse))
print(mse)
print("\nR_squared:")
print(np.mean(r))
print(r)
```

```
MSE:
0.1007171235197765
[0.09273168946386275, 0.10477214524931174, 0.09160631982743386, 0.10476296344670001, 0.10971249961157416]

R_squared:
0.8664663862072419
[0.8786857079752665, 0.8622284346216833, 0.8825862757616265, 0.8664687179029944, 0.8423627947746393]
```

Also, we tried n_neighbors =50

```
kf = KFold(n_splits=5, shuffle=True, random_state=2018)
probs = []
indicies = []
mse = []
r=[]

n_neighbors = 50
clf = neighbors.KNeighborsRegressor(n_neighbors, n_jobs=-1, p=2)
for train, test in kf.split(X_train2.index.values):
    clf.fit(X_train2.iloc[train][features].values, X_train_f.iloc[train]["Price_log"].values)
    prob = clf.predict(X_train2.iloc[test][features].values)
    probs.append(prob)
    indicies.append(test)
    mse.append(metrics.mean_squared_error(X_train_f.iloc[test]["Price_log"].values, prob))
    r.append(metrics.r2_score(X_train_f.iloc[test]["Price_log"].values, prob))
print("MSE:")
print(np.mean(mse))
print(mse)
print("\nR_squared:")
print(np.mean(r))
print(r)
```

```
MSE:
0.1069828207847455
[0.09982755615754128, 0.10968818439624384, 0.09690989012821222, 0.113210916690616, 0.11527755655111413]

R_squared:
0.8582046037599147
[0.8694026888776691, 0.8557640217080957, 0.8757885794678831, 0.8557009237260992, 0.8343668050198262]
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=2018)
probs = []
indicies = []
mse = []
r=[]

n_neighbors = 25
clf = neighbors.KNeighborsRegressor(n_neighbors, n_jobs=-1, p=1) #p=1 means Manhatan metrics
for train, test in kf.split(X_train2.index.values):
    clf.fit(X_train2.iloc[train][features].values, X_train_f.iloc[train]["Price_log"].values)
    prob = clf.predict(X_train2.iloc[test][features].values)
    probs.append(prob)
    indicies.append(test)
    mse.append(metrics.mean_squared_error(X_train_f.iloc[test]["Price_log"].values, prob))
    r.append(metrics.r2_score(X_train_f.iloc[test]["Price_log"].values, prob))
print("MSE:")
print(np.mean(mse))
print(mse)
print("\nR_squared:")
print(np.mean(r))
print(r)
```

```
MSE:
0.1004278068984125
[0.09387832418601454, 0.10391610765212066, 0.09025277473552533, 0.1048334485134056, 0.10925837940499644]

R_squared:
0.8668510088432976
[0.8771856470971237, 0.8633540929682512, 0.8843211426405158, 0.8663788773618696, 0.8430152841487281]
```

# SVR

LINEAR

```
from sklearn.svm import SVR
#data have been previously normalized
features = X_train2.columns.tolist()
X_train2.iloc[test][features]
kf = KFold(n_splits=5)
mses = []
r=[]
clf = SVR(C=1, cache_size=500, kernel='linear',
    max_iter=-1, tol=0.001, verbose=False)

for train, test in kf.split(X_train2.index.values):

    clf.fit(X_train2.iloc[train][features].values, X_train_f.iloc[train]["Price_log"].values)
    prob = clf.predict(X_train2.iloc[test][features].values)
    mses.append(metrics.mean_squared_error(X_train_f.iloc[test]["Price_log"].values, prob))
    r.append(metrics.r2_score(X_train_f.iloc[test]["Price_log"].values, prob))

print("MSE:")
print(np.mean(mses))
print(mses)
print("\nR_squared:")
print(np.mean(r))
print(r)
```

```
MSE:
0.12948012299206738
[0.15874197860758177, 0.11977196275956149, 0.11974562704065317, 0.12674426973800326, 0.12239677681453726]

R_squared:
0.8284736793502641
[0.7797701635591572, 0.838381445566693, 0.8500766896431577, 0.8339830355795458, 0.8401570624027663]
```

POLYNOMINAL

```
features = X_train2.columns.tolist()
X_train2.iloc[test][features]
kf = KFold(n_splits=5)
mses = []
r=[]
clf = SVR(C=1, cache_size=500, degree=2,kernel='poly',
    max_iter=-1, tol=0.001, verbose=False)

for train, test in kf.split(X_train2.index.values):

    clf.fit(X_train2.iloc[train][features].values, X_train_f.iloc[train]["Price_log"].values)
    prob = clf.predict(X_train2.iloc[test][features].values)
    mses.append(metrics.mean_squared_error(X_train_f.iloc[test]["Price_log"].values, prob))
    r.append(metrics.r2_score(X_train_f.iloc[test]["Price_log"].values, prob))

print("MSE:")
print(np.mean(mses))
print(mses)
print("\nR_squared:")
print(np.mean(r))
print(r)
```

```
MSE:
0.10769943339359962
[0.11920064214864541, 0.09910466250460317, 0.099796650983007, 0.11543015550171343, 0.10496505583300291]

R_squared:
0.8575350210376651
[0.8346276255700784, 0.8662696016450139, 0.8750531050890122, 0.8488029158352595, 0.8629218570489615]
```

RBF

```
features = X_train2.columns.tolist()
X_train2.iloc[test][features]
kf = KFold(n_splits=5)
mses = []
r=[]
clf = SVR(C=1, cache_size=500, degree=3,kernel='rbf',
    max_iter=-1, tol=0.001, verbose=False)

for train, test in kf.split(X_train2.index.values):

    clf.fit(X_train2.iloc[train][features].values, X_train_f.iloc[train]["Price_log"].values)
    prob = clf.predict(X_train2.iloc[test][features].values)
    mses.append(metrics.mean_squared_error(X_train_f.iloc[test]["Price_log"].values, prob))
    r.append(metrics.r2_score(X_train_f.iloc[test]["Price_log"].values, prob))

print("MSE:")
print(np.mean(mses))
print(mses)
print("\nR_squared:")
print(np.mean(r))
print(r)
```

```
MSE:
0.10472341869806204
[0.11038363397901829, 0.09866005222685807, 0.09694393518208066, 0.11520769738031583, 0.10242177472203728]

R_squared:
0.8615383381404491
[0.8468598547770401, 0.866869552323952, 0.8786247478032547, 0.8490943043303879, 0.8662432314676111]
```

RBF 2

MSE:
0.10451219617870751
[0.11117407194649125, 0.09760624066298232, 0.09637780353971089, 0.11502580685946663, 0.10237705788848865]

R_squared:
0.8618045064796229
[0.8457632448832985, 0.8682915504082667, 0.8793335530600264, 0.8493325550393995, 0.8663016290071237]

# USAGE OF CHOSEN METHOD ON TEST DATA

MSE: 0.0975013790102812
$R^2$: 0.8727525123577311

```
n_neighbors = 25
clf = neighbors.KNeighborsRegressor(n_neighbors, n_jobs=-1, p=1)

clf.fit(X_test2,Y)
y_pred=clf.predict(X_test2)
print(metrics.mean_squared_error(Y,y_pred))
print(metrics.r2_score(Y,y_pred))
```

Dependent variable: Price_log - logarithm of Price Variable

Independent variables: Transmission, Year, Power

By R^2 and MSE we chose kNN with n_neighbors = 25 and Manhatan metric

# CONCLUSIONS

- Best Model: kNN with n_neighbors = 25 and Manhatan metric
- Validation:  MSE: 0.1004278068984125 R_squared:0.8668510088432976
- Final:            MSE: 0.0975013790102812 R_squared: 0.8727525123577311