Import libraries and define usefull constants

In [396]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import spearmanr
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
Id, MSSubClass, MSZoning, LotArea, SaleCondition, SalePrice = \
    'Id', 'MSSubClass', 'MSZoning', 'LotArea', 'SaleCondition', 'SalePrice'
train_file_path = './datasets/train.csv'
test_file_path = './datasets/test1.csv'
```

Define usefull funcitons

- **"predict" just calculates the accuracy**
  - **"runLinearRegressionModel" just creates a linear model**
  - **"runDecisionTreeModel" also creates a Decision Tree model**

In [397]:

```
def predict(features, prices, model):
    predicts = model.predict(features)
    mse = mean_squared_error(predicts, prices)
    return np.sqrt(mse)


def runLinearRegressionModel(features, prices):
    model = LinearRegression()
    model.fit(features, prices)
    return model

def runDecisionTreeModel(features, prices):
    model = DecisionTreeRegressor(random_state=0)
    model.fit(features, prices)
    return model
```

normalize categorical features

we won't use labelEncoding much because we get better accuracy with one-hot-encoding

- **in label encoding we assign a number to each category**

In [398]:

```
def lableEncode(train_df_set):
    print(f"All categories in MSZoning:", train_df_set[MSZoning].unique())
    print(f"All categories in SaleCondition:", train_df_set[SaleCondition].unique())
    MSZoning_labels = {label: value for value, label in enumerate(train_df_set[MSZoning].unique())}
    SaleCondition_lables = {label: value for value, label in enumerate(train_df_set[SaleCondition].unique())}
    print(MSZoning_labels, SaleCondition_lables)
    train_df_set[MSZoning] = train_df_set[MSZoning].replace(MSZoning_labels)
    train_df_set[SaleCondition] = train_df_set[SaleCondition].replace(SaleCondition_lables)
    return train_df_set

def oneHotEncode(train_df_set):
    prices = pd.DataFrame(train_df_set[SalePrice])
    # first MSZoning
    train_df_set.drop(SalePrice, axis=1, inplace=True)
    MSZ_one_hots = pd.get_dummies(train_df_set[MSZoning])
    train_df_set.drop(MSZoning, axis=1, inplace=True)
    train_df_set = train_df_set.join(MSZ_one_hots)
    # now SaleCondition
    SaleCond_one_hot = pd.get_dummies(train_df_set[SaleCondition])
    train_df_set.drop(SaleCondition, axis=1, inplace=True)
    train_df_set = train_df_set.join(SaleCond_one_hot)
    train_df_set = train_df_set.join(prices)
    return train_df_set
```

define a func for visualization

this funciton takes every column of the data frame and plots it with the price column

In [399]:

```
def visualizeDataFram(df: pd.DataFrame):
    for i, column_name in enumerate(df.columns[:-1]):
        new_fig = plt.figure(i)
        plt.xlabel(column_name)
        plt.ylabel(SalePrice)
        plt.plot(df[column_name], df[SalePrice], '*')
```

a funciton to remove bad datas from all columns

- **we took 60000 for lot area**
- **700000 for Sale price**
- **700000 for MSSubclass**

In [400]:

```
def removeBadDatas(train_df_set: pd.DataFrame):

    # remove bad datas from LotArea column
    bad_LotArea_datas_index = train_df_set[train_df_set[LotArea] > 60000].index
    train_df_set.drop(bad_LotArea_datas_index, inplace=True)

    # remove bad datas from SalePrices column
    bad_SaleCondition_datas_index = train_df_set[train_df_set[SalePrice] > 700000].index
    train_df_set.drop(bad_SaleCondition_datas_index, inplace=True)

    # remove bad datas from MSSubclass column
    bad_MSSubclass_datas_index = train_df_set[train_df_set[MSSubClass] > 700000].index
    train_df_set.drop(bad_MSSubclass_datas_index)
```

a function that Splits features and y's from a given pd.DataFrame

this function takes a data frame which its last column is y, and

returns the features and y column

In [401]:

```
def splitFeaturesAndPrices(train_df_set: pd.DataFrame):
    train_array = np.array(train_df_set)
    features = train_array[:, :-1]
    prices = train_array[:, -1:]
    return features, prices
```

a function that reads the dataset everytime we need the original version

in the middle of our work we may apply some changes to our data frame so this funciton help us to

read the data set and returns it as a data frame every time we want

In [402]:

```
def readDataSet(filepath: str):
    dataSet = pd.read_csv(filepath)
    return dataSet
```

describing dataset:

a quick preview of dataset

In [403]:

```
train_data_frame = readDataSet(train_file_path)
train_data_frame
```

Out[403]:

	Id	MSSubClass	MSZoning	LotArea	SaleCondition	SalePrice
0	1	60	RL	8450	Normal	208500
1	2	20	RL	9600	Normal	181500
2	3	60	RL	11250	Normal	223500
3	4	70	RL	9550	Abnorml	140000
4	5	60	RL	14260	Normal	250000
...	...	...	...	...	...	...
1455	1456	60	RL	7917	Normal	175000
1456	1457	20	RL	13175	Normal	210000
1457	1458	70	RL	9042	Normal	266500
1458	1459	20	RL	9717	Normal	142125
1459	1460	20	RL	9937	Normal	147500

1460 rows × 6 columns

as you can see; we have 5 columns(features) in which there's two categorical features

and one SalePrice column that we're going to predict

in the next step we do a label encoding

In [404]:

```
train_data_frame = lableEncode(train_data_frame)
```

```
All categories in MSZoning: ['RL' 'RM' 'C (all)' 'FV' 'RH']
All categories in SaleCondition: ['Normal' 'Abnorml' 'Partial' 'AdjLand' 'Alloca' 'Family']
{'RL': 0, 'RM': 1, 'C (all)': 2, 'FV': 3, 'RH': 4} {'Normal': 0, 'Abnorml': 1, 'Partial': 2, 'AdjLand': 3, 'Alloca': 4, 'Family': 5
}
```

then some Descriptive informations:

In [405]:

```
train_data_frame.describe()
```

Out[405]:

	Id	MSSubClass	MSZoning	LotArea	SaleCondition	SalePrice
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.00000	1460.000000
mean	730.500000	56.897260	0.340411	10516.828082	0.35000	180921.195890
std	421.610009	42.300571	0.798309	9981.264932	0.88787	79442.502883
min	1.000000	20.000000	0.000000	1300.000000	0.00000	34900.000000
25%	365.750000	20.000000	0.000000	7553.500000	0.00000	129975.000000
50%	730.500000	50.000000	0.000000	9478.500000	0.00000	163000.000000
75%	1095.250000	70.000000	0.000000	11601.500000	0.00000	214000.000000
max	1460.000000	190.000000	4.000000	215245.000000	5.00000	755000.000000

now we check the dataset for null values

In [406]:

```
train_data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   int64
 3   LotArea        1460 non-null   int64
 4   SaleCondition  1460 non-null   int64
 5   SalePrice      1460 non-null   int64
dtypes: int64(6)
memory usage: 68.6 KB
```

In [407]:

```
train_data_frame.isnull().sum()
```

Out[407]:

```
Id               0
MSSubClass       0
MSZoning         0
LotArea          0
SaleCondition    0
SalePrice        0
dtype: int64
```

Fortunately we don't have any null value

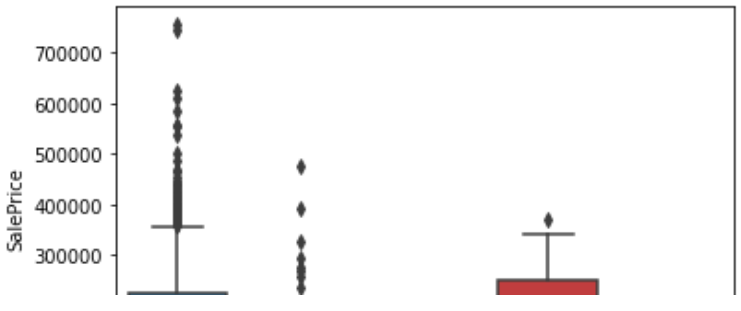
Visualize data:

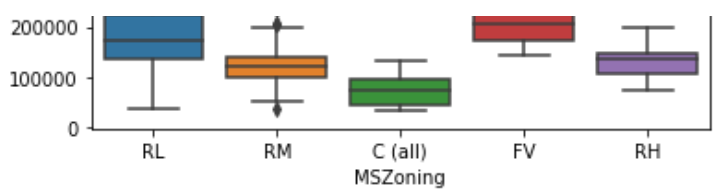
before removing bad datas:

- اینجا منظور از داده بد، داده پرته **:)**

In [408]:

```
train_data_frame = readDataSet(train_file_path)
ax = sns.boxplot(x='MSZoning', y='SalePrice', data=train_data_frame)
```



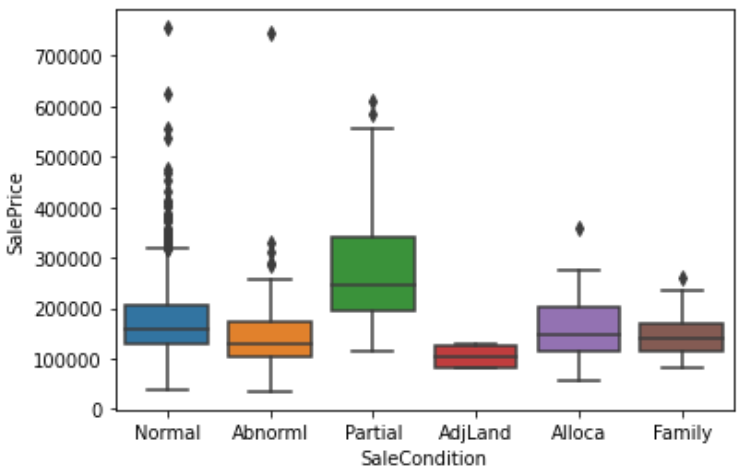


from this boxplot we can see that the houses with MSZoning of "FV" have the highest price and "c (all)" the lowest

we can also say that the RL's has the widest range of price and c(all) has the thinest range of price and is also cheap

In [409]:

```
ax = sns.boxplot(x='SaleCondition', y='SalePrice', data=train_data_frame)
```



also here we can see that houses with SaleConsition "Partial" and "AdjLand" have the highest and lowest prices

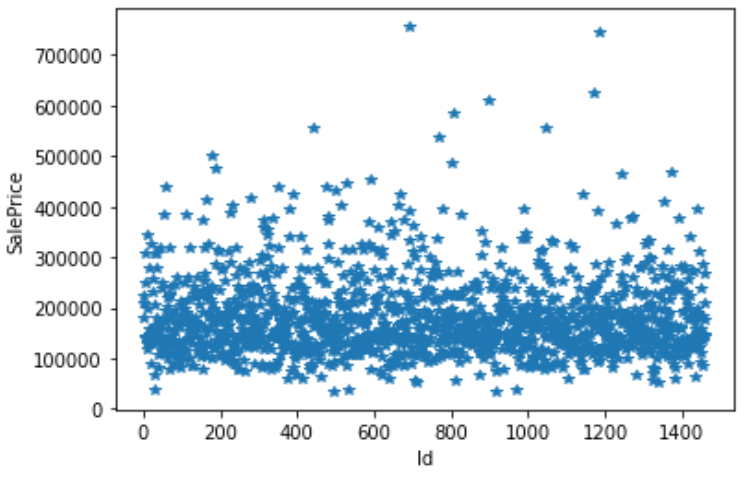
also we can see that the "partial" Condition has the widest range and Adjland has the thinnest range and is also cheap.

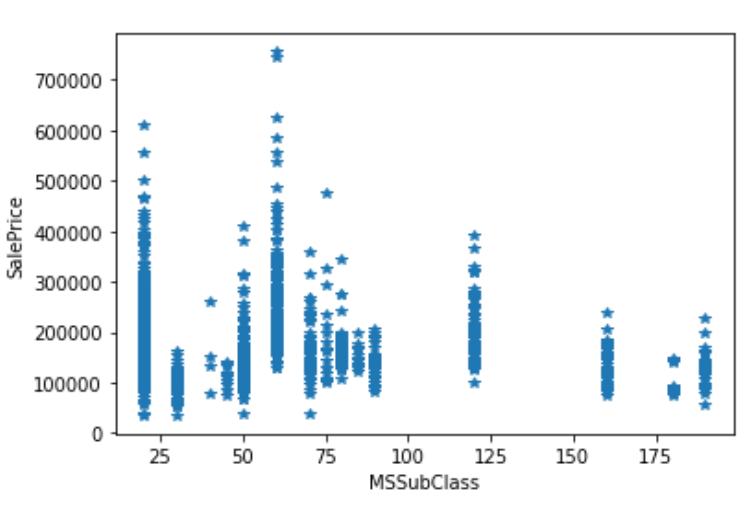
in below we plot each feature with price.

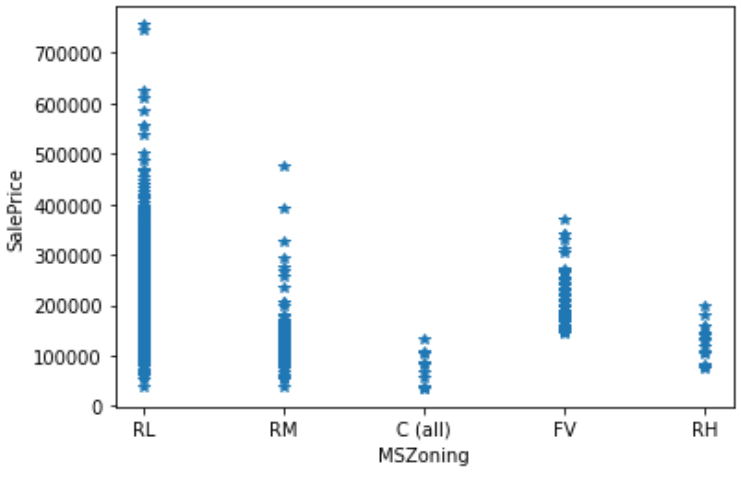
as you can see there's not a good linear relation between them (specialy for id)

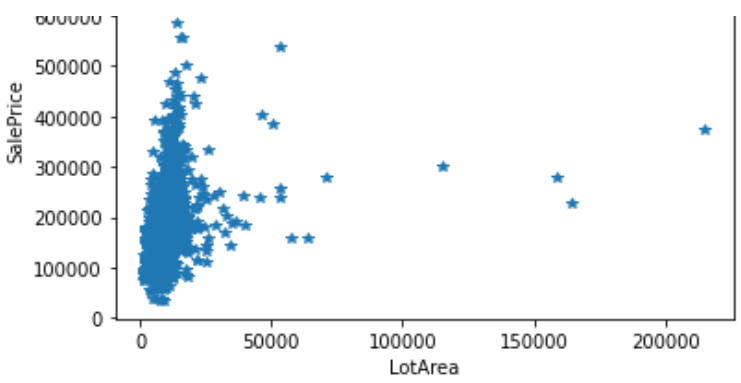
In [410]:

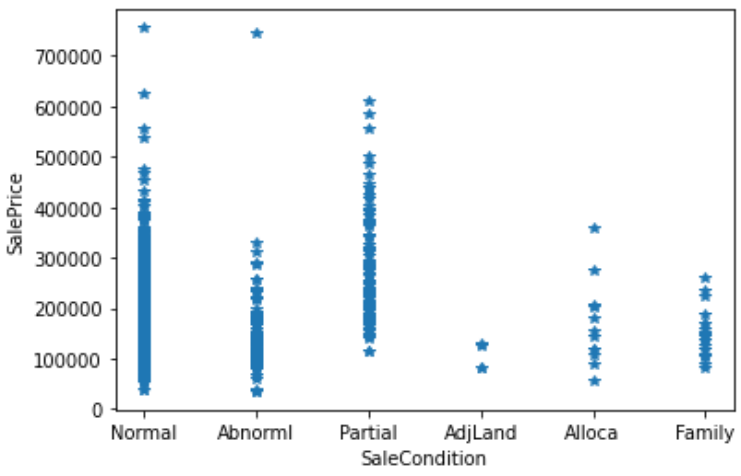
```
visualizeDataFram(train_data_frame)
```









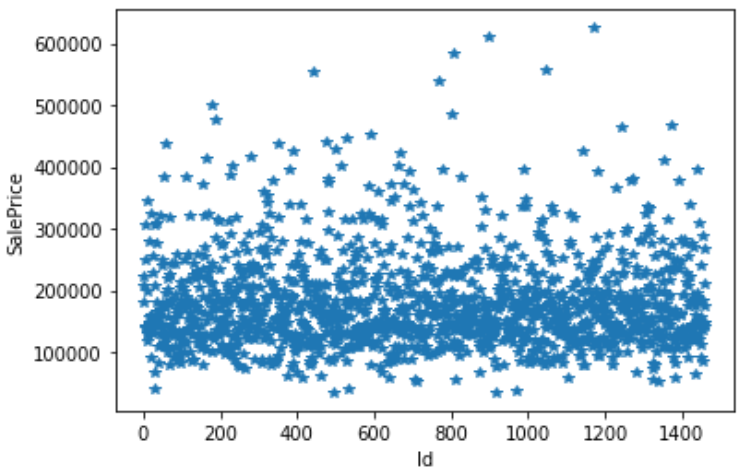


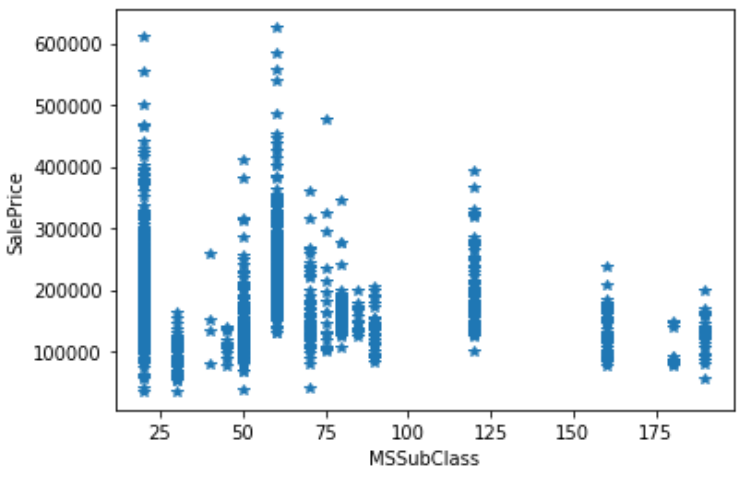
on these figs we can recognize some bad datas. that we remove them in next step

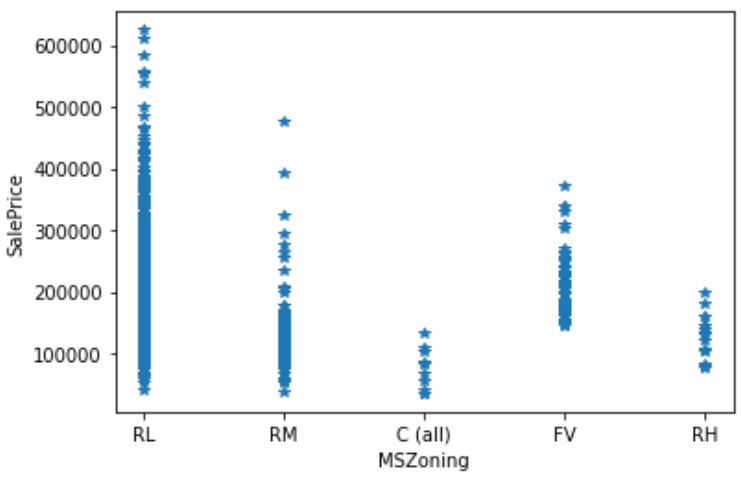
after removing bad datas:

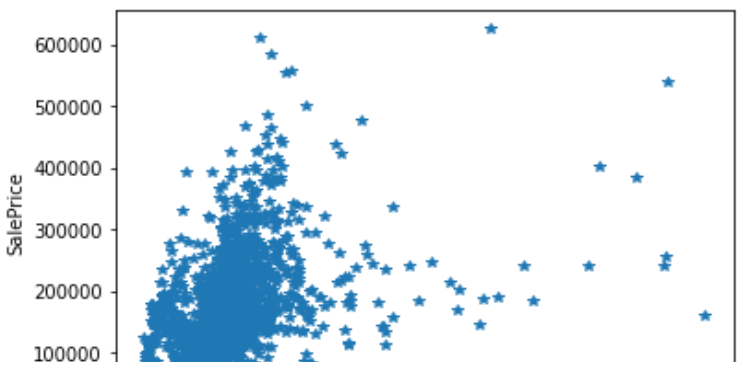
In [411]:

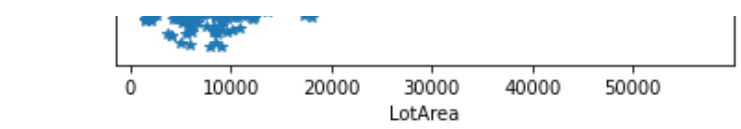
```
removeBadDatas(train_data_frame)
visualizeDataFram(train_data_frame)
```

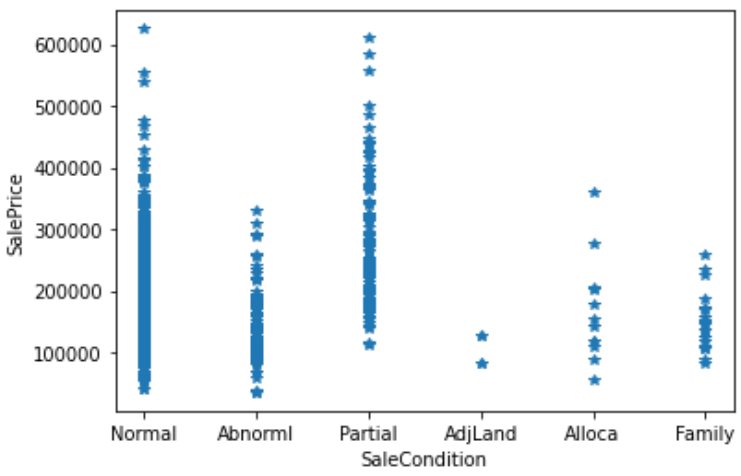












in future steps we see that removing these bad datase doesn't affect so much on our model

in come cases suprisingly we better don't remove!

getting correlations

In [412]:

```
train_data_frame = lableEncode(train_data_frame)
train_data_frame.corr()
```

```
All categories in MSZoning: ['RL' 'RM' 'C (all)' 'FV' 'RH']
All categories in SaleCondition: ['Normal' 'Abnorml' 'Partial' 'AdjLand' 'Alloca' 'Family']
{'RL': 0, 'RM': 1, 'C (all)': 2, 'FV': 3, 'RH': 4} {'Normal': 0, 'Abnorml': 1, 'Partial': 2, 'AdjLand': 3, 'Alloca': 4, 'Family': 5
}
```

Out[412]:

	Id	MSSubClass	MSZoning	LotArea	SaleCondition	SalePrice
Id	1.000000	0.011989	-0.012409	0.000234	-0.021097	-0.023828
MSSubClass	0.011989	1.000000	0.288465	-0.287639	-0.015147	-0.086883
MSZoning	-0.012409	0.288465	1.000000	-0.267702	0.068922	-0.114496
LotArea	0.000234	-0.287639	-0.267702	1.000000	0.022276	0.372709
SaleCondition	-0.021097	-0.015147	0.068922	0.022276	1.000000	0.149357
SalePrice	-0.023828	-0.086883	-0.114496	0.372709	0.149357	1.000000

as you can see the correlations between features are also low. and there's not a good linear relation between them

Run Linear Regression:

first we try linear regression without removing bad datas:

preproccess on train for running model

we use oneHotEncoding cause it gives a better accuracy

for exmaple in this case label encoding results 75440.1972959434 on train and 77784.45041973717 as accuracy on test

we also remove the Id column. it's just an id and has a uniform distribution

In [413]:

```
train_data_frame = readDataSet(train_file_path)
train_data_frame = oneHotEncode(train_data_frame)
train_data_frame.drop('Id', axis=1, inplace=True)
features, prices = splitFeaturesAndPrices(train_data_frame)
model_with_bad_datas = runLinearRegressionModel(features, prices)
accuracy = predict(features, prices, model_with_bad_datas)
accuracy
```

Out[413]:

```
68401.89031287072
```

our accuracy in train dataset is 68401.89031287072 which is not good as pridected

now we use the model on test dataset

In [414]:

```
test_data_frame = readDataSet(test_file_path)
test_data_frame = oneHotEncode(test_data_frame)
test_data_frame.drop("Id", axis=1, inplace=True)
scaler = MinMaxScaler()
test_features, test_prices = splitFeaturesAndPrices(test_data_frame)
# test_features = scaler.fit_transform(test_features)
test_accuracy = predict(test_features, test_prices, model_with_bad_datas)
print(f"accuracy on test: {test_accuracy}")
```

```
accuracy on test: 67061.46019834289
```

accuracy is still too high

with removing bad datas:

here we remove bad datas and create the model on new datasets

In [415]:

```
removeBadDatas(train_data_frame)
```

In [416]:

```
features, prices = splitFeaturesAndPrices(train_data_frame)
model_without_bad_datas = runLinearRegressionModel(features, prices)
accuracy = predict(features, prices, model_without_bad_datas)
accuracy
```

Out[416]:

```
62899.20698612004
```

the accuracy on the train gets better but ...

now on test datas

In [417]:

```
test_accuracy = predict(test_features, test_prices, model_without_bad_datas)
test_accuracy
```

Out[417]:

```
79850.81616287073
```

suprisingly on test it gets much worse

as we pridected (plots and correlations) linear regression may not be a good model for our dataset

let's try polynomial regression

Run Polynomial Regression

in polynomial regression we guess that the relation between features and prices are polynomail

so we just need to ajdust the features data set. and then feed a linear regression model with it.

In [418]:

```
train_data_frame = readDataSet(train_file_path)
train_data_frame.drop('Id', axis=1, inplace=True)
train_data_frame = oneHotEncode(train_data_frame)
features, prices = splitFeaturesAndPrices(train_data_frame)
feature_proccesor = PolynomialFeatures(degree=3)
features = feature_proccesor.fit_transform(features)
```

In [419]:

```
pol_model = runLinearRegressionModel(features, prices)
predict(features, prices, pol_model)
```

Out[419]:

```
61650.33159330268
```

in this model we get much better results:

accuracy on train for each degree:

- **2: 63795.07328396192**
- **3: 61650.33159330268 -> best conditions**
- **4: 98812.82100447727**
- **5: 73494.76229648557**
- **6: 73324.11276960478 -> not much defference but takes much longer time**

- **7: 77625.99915804443 -> on degree 7 i took 30 seconds!**
- **8: 77328.09197539145 -> on degree 8 i took 97 seconds! ### so obviously the degree with accuracy of 61650.33159330268 is the best choice**

now we use the model we made to predict the houses in test dataset houses:

In [420]:

```
test_data_frame = readDataSet(test_file_path)
test_data_frame.drop('Id', axis=1, inplace=True)
test_data_frame = oneHotEncode(test_data_frame)
test_features, test_prices = splitFeaturesAndPrices(test_data_frame)
test_features = feature_proccesor.fit_transform(test_features)
predict(test_features, test_prices, pol_model)
```

Out[420]:

```
60491.92371237629
```

on test dataset our model (with degree 3) results a accuracy of 60491.92371237629

60491.92371237629 is good, but we want something better!

now we go on Desicion tree regression

Run Desicion tree

In [421]:

```
train_data_frame = readDataSet(train_file_path)
train_data_frame.drop('Id', axis=1, inplace=True)
train_data_frame = oneHotEncode(train_data_frame)
features, prices = splitFeaturesAndPrices(train_data_frame)
tree_model = runDecisionTreeModel(features, prices)
predict(features, prices, tree_model)
```

Out[421]:

```
10025.726551100192
```

by suprise at the first try we get 10025.726551100192 as our accuracy

which is the best so far

now we try our model on test dataset:

In [422]:

```
test_data_frame = readDataSet(test_file_path)
test_data_frame.drop('Id', axis=1, inplace=True)
test_data_frame = oneHotEncode(test_data_frame)
test_features, test_prices = splitFeaturesAndPrices(test_data_frame)
predict(test_features, test_prices, tree_model)
```

Out[422]:

```
9650.22344618717
```

as you see we got 9650.22344618717 as accuracy which seems so perfect.

so we use this model to report the predicts:

In [423]:

```
test_predicts = tree_model.predict(test_features)
predictions_df = pd.DataFrame(data=test_predicts, columns=['predictions'])
predictions_df.to_csv('./predictions.csv', index=False)
```