In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing,svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [2]:
```
df=pd.read_csv(r"C:\Users\pucha\Downloads\fiat500_VehicleSelection_Dataset.csv")
df
```

Out[2]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	lounge	51	882	25000	1	44.907242	8.611560	8900
1	2	pop	51	1186	32500	1	45.666359	12.241890	8800
2	3	sport	74	4658	142228	1	45.503300	11.417840	4200
3	4	lounge	51	2739	160000	1	40.633171	17.634609	6000
4	5	pop	73	3074	106880	1	41.903221	12.495650	5700
...	...	...	...	...	...	...	...	...	...
1533	1534	sport	51	3712	115280	1	45.069679	7.704920	5200
1534	1535	lounge	74	3835	112000	1	45.845692	8.666870	4600
1535	1536	pop	51	2223	60457	1	45.481541	9.413480	7500
1536	1537	lounge	51	2557	80750	1	45.000702	7.682270	5990
1537	1538	pop	51	1766	54276	1	40.323410	17.568270	7900

1538 rows × 9 columns

In [7]:
```
df=df[['km','lat']]
df.columns=['km','lat']
df.head(10)
```

Out[7]:

	km	lat
0	25000	44.907242
1	32500	45.666359
2	142228	45.503300
3	160000	40.633171
4	106880	41.903221
5	70225	45.000702
6	11600	44.907242
7	49076	41.903221
8	76000	45.548000
9	89000	45.438301

In [8]:
```
df.head()
```

Out[8]:

	km	lat
0	25000	44.907242
1	32500	45.666359
2	142228	45.503300
3	160000	40.633171
4	106880	41.903221

In [9]:
```
df.tail()
```

Out[9]:

	km	lat
1533	115280	45.069679
1534	112000	45.845692
1535	60457	45.481541
1536	80750	45.000702
1537	54276	40.323410

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1538 entries, 0 to 1537
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   km      1538 non-null   int64  
 1   lat     1538 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 24.2 KB
```

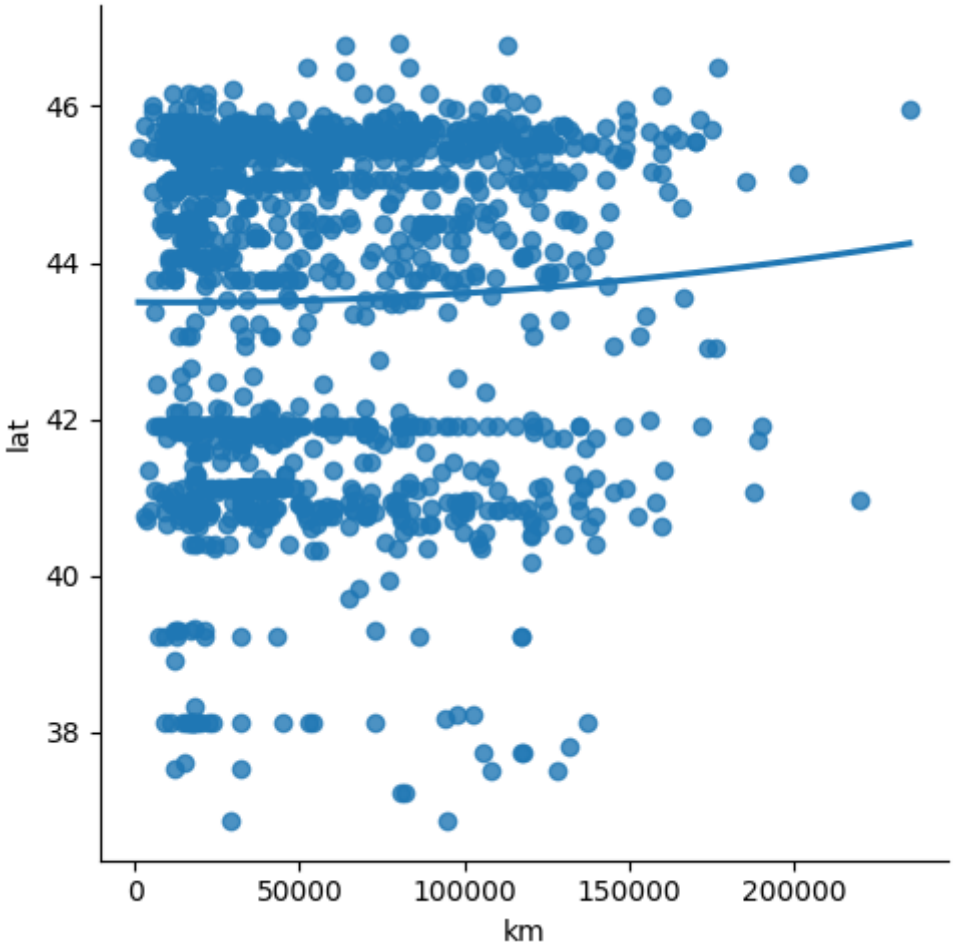
In [11]:
```
df.describe()
```

Out[11]:

	km	lat
count	1538.000000	1538.000000
mean	53396.011704	43.541361
std	40046.830723	2.133518
min	1232.000000	36.855839
25%	20006.250000	41.802990
50%	39031.000000	44.394096
75%	79667.750000	45.467960
max	235000.000000	46.795612

In [12]:
```
sns.lmplot(x="km",y="lat",data=df,order=2,ci=None)
```

Out[12]:
```
<seaborn.axisgrid.FacetGrid at 0x175cbd845b0>
```



In [13]:
```
df.fillna(method = 'ffill',inplace = True)
```

```
C:\Users\pucha\AppData\Local\Temp\ipykernel_21040\3028625988.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df.fillna(method = 'ffill',inplace = True)
```

In [14]:
```
x=np.array(df['km']).reshape(-1,1)
y=np.array(df['lat']).reshape(-1,1)
```

In [15]:
```
df.dropna(inplace = True)
```

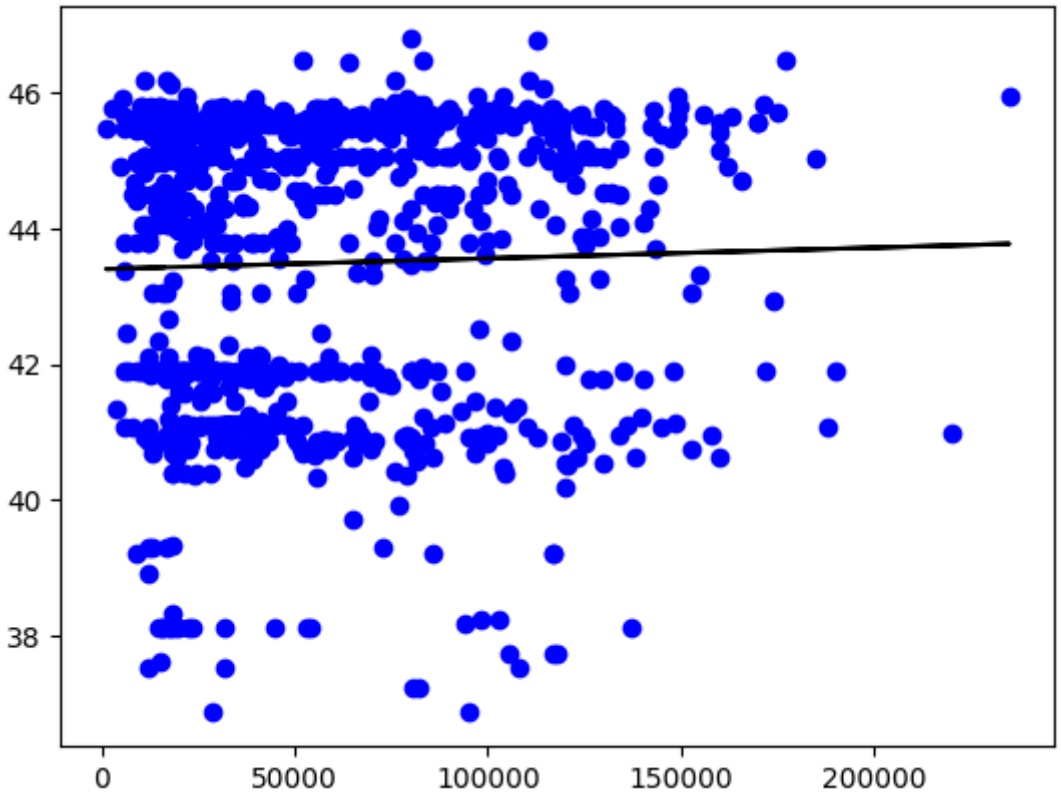
```
C:\Users\pucha\AppData\Local\Temp\ipykernel_21040\1791587065.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df.dropna(inplace = True)
```

In [18]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.7)
```

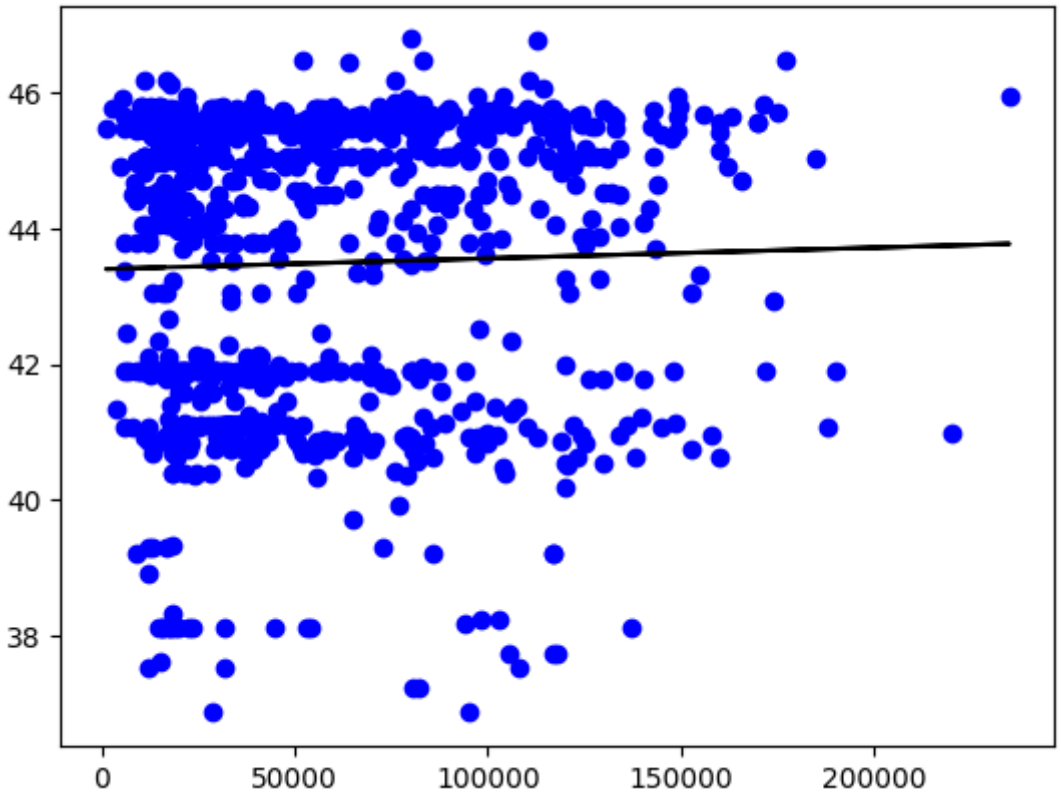
In [19]:
```
regr=LinearRegression()
regr.fit(x_train,y_train)
print(regr.score(x_test,y_test))
```

```
2.5409409364351987e-05
```

In [20]:
```
 y_pred=regr.predict(x_test)
plt.scatter(x_test,y_test,color='b')
plt.plot(x_test,y_pred,color='k')
plt.show()
```

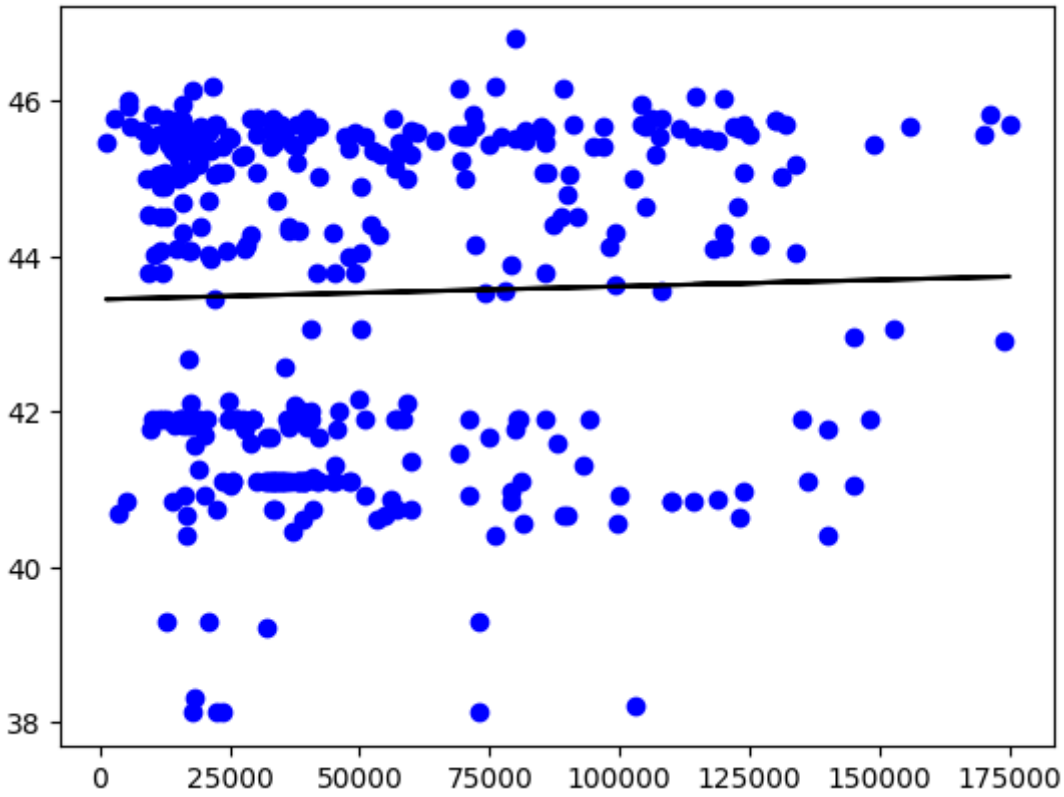


In [21]:
```
 y_pred=regr.predict(x_test)
plt.scatter(x_test,y_test,color='b')
plt.plot(x_test,y_pred,color='k')
plt.show()
```



In [23]:
```
df.dropna(inplace=True)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
regr=LinearRegression()
regr.fit(x_train,y_train)
print("Regression:",regr.score(x_test,y_test))
y_pred=regr.predict(x_test)
plt.scatter(x_test,y_test,color='b')
plt.plot(x_test,y_pred,color='k')
plt.show()
```

```
C:\Users\pucha\AppData\Local\Temp\ipykernel_21040\3772379865.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df.dropna(inplace=True)

Regression: 0.0019025008972813895
```



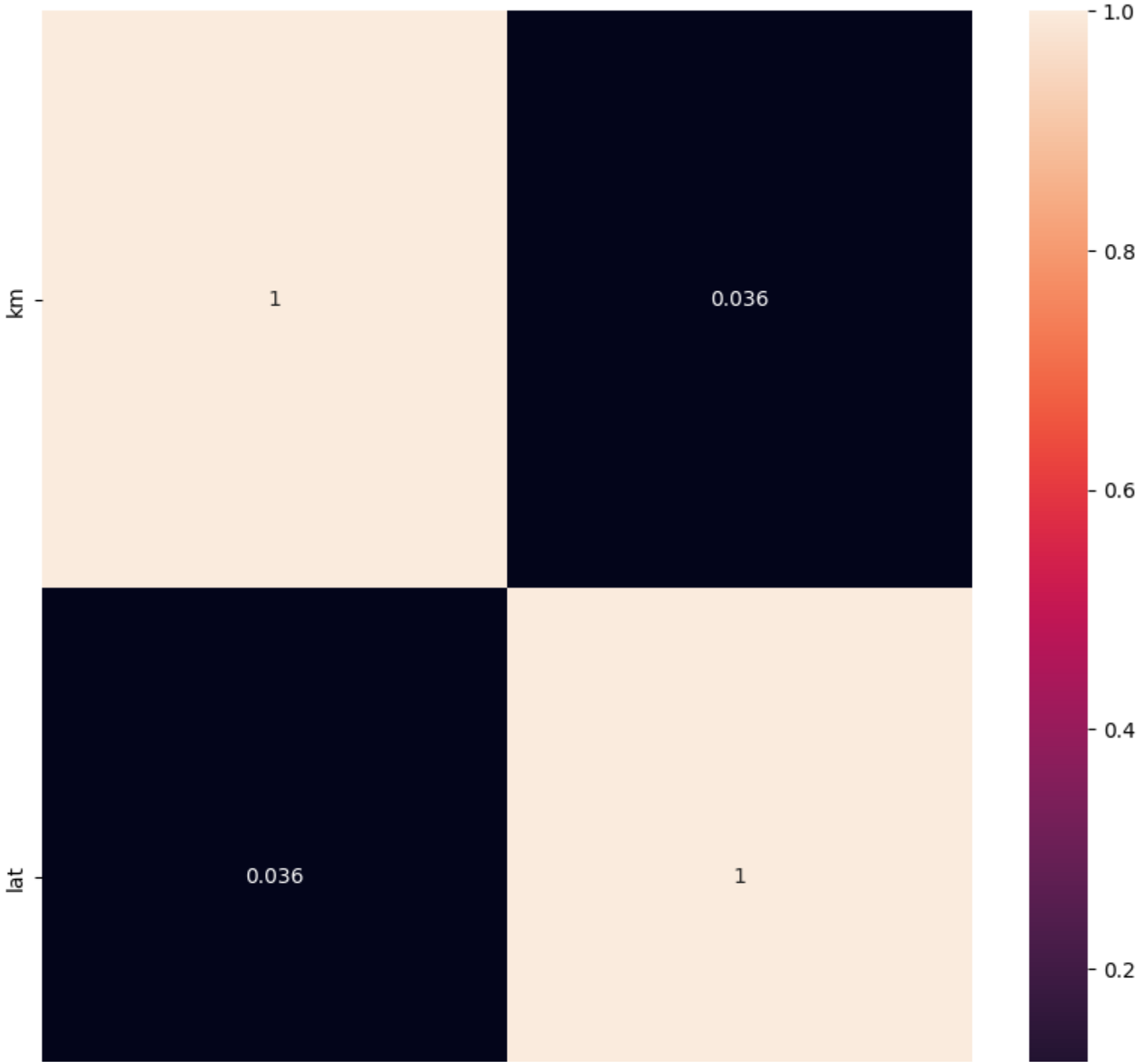
In [24]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
model=LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
r2=r2_score(y_test,y_pred)
print("R2.score:",r2)
```

```
R2.score: 0.0019025008972813895
```

In [25]:
```
from sklearn.linear_model import Ridge,RidgeCV,Lasso
from sklearn.preprocessing import StandardScaler
```

In [26]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True)
```

Out[26]: <Axes: >


km
lat
1
0.036
0.036
1
1.0
0.8
0.6
0.4
0.2

In [27]:
```
features=df.columns[0:2]
target=df.columns[-1]
#x and y values
x=df[features].values
y=df[target].values
#splot
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
print("The dimension of x_train is {}".format(x_train.shape))
print("The dimension of x_test is {}".format(x_test.shape))
#scale features
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

```
The dimension of x_train is (1076, 2)
The dimension of x_test is (462, 2)
```

In [28]:
```
#model
lr=LinearRegression()
#fit model
lr.fit(x_train,y_train)
#predict
#prediction=lr.predict(x_test)
#actual
actual=y_test
train_score_lr=lr.score(x_train,y_train)
test_score_lr=lr.score(x_test,y_test)
print("\nLinear Regression Model:\n")
print("The train score for lr model is {}".format(train_score_lr))
print("The test score for lr model is {}".format(test_score_lr))
```

```
Linear Regression Model:

The train score for lr model is 1.0
The test score for lr model is 1.0
```

In [29]:
```
ridgeReg=Ridge(alpha=10)
ridgeReg.fit(x_train,y_train)
#train and test scorefor ridge regression
train_score_ridge=ridgeReg.score(x_train,y_train)
test_score_ridge=ridgeReg.score(x_test,y_test)
print("\nRidge Model:\n")
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(test_score_ridge))
```

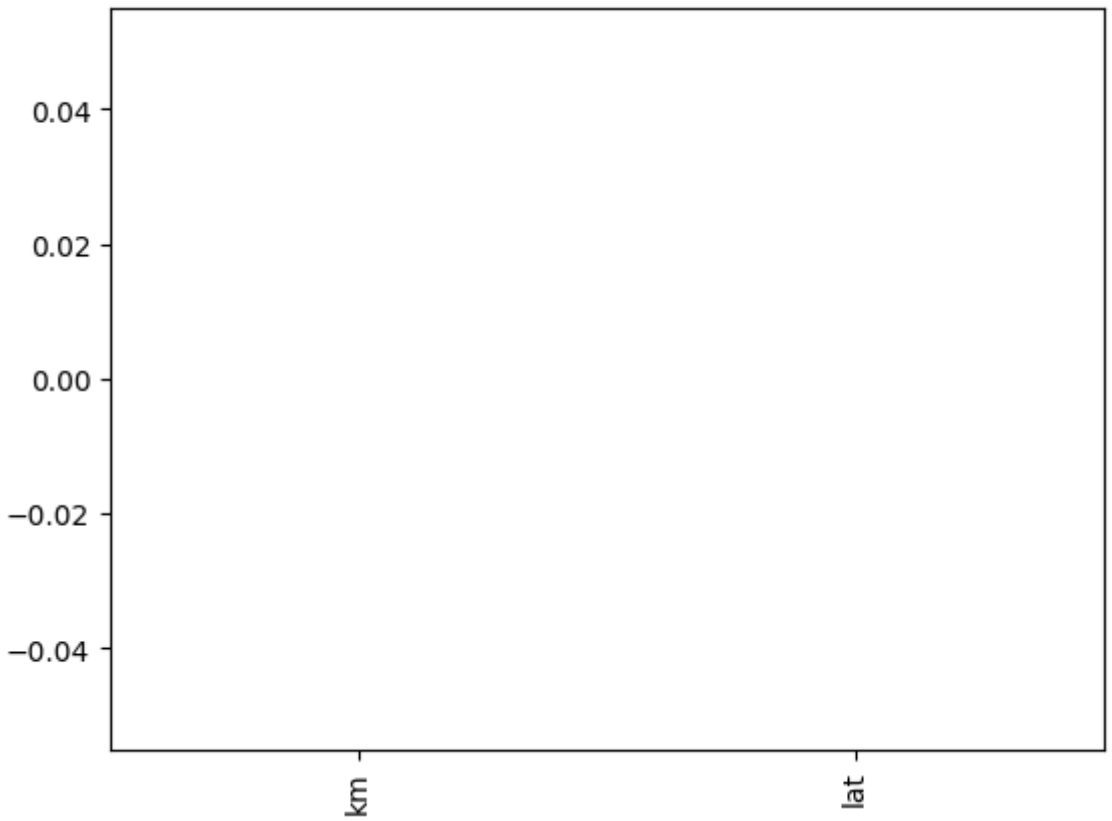
```
Ridge Model:

The train score for ridge model is 0.9999149781117884
The test score for ridge model is 0.9999142154121183
```

In [30]:
```
print("\nLasso Model:\n")
lasso=Lasso(alpha=10)
lasso.fit(x_train,y_train)
train_score_ls=lasso.score(x_train,y_train)
test_score_ls=lasso.score(x_test,y_test)
print("The train score for ls model is {}".format(train_score_ls))
print("The test score for ls model is {}".format(test_score_ls))
```

```
Lasso Model:

The train score for ls model is 0.0
The test score for ls model is -0.0027944198857072777
```

In [31]:
```
pd.Series(lasso.coef_,features).sort_values(ascending=True).plot(kind="bar")
```

Out[31]: <Axes: >



In [32]:
```
from sklearn.linear_model import LassoCV
#Lasso Cross Validation
lasso_cv=LassoCV(alphas=[0.0001,0.001,0.01,0.1,1,10],random_state=0).fit(x_train,y_train)
#score
print(lasso_cv.score(x_train,y_train))
print(lasso_cv.score(x_test,y_test))
```

```
0.999999997786743
0.9999999977805583
```

In [33]:
```
#using the linear CV model
from sklearn.linear_model import RidgeCV
#ridge Cross Validation
ridge_cv=RidgeCV(alphas=[0.0001,0.001,0.01,0.1,1,10]).fit(x_train,y_train)
#score
print("The train score for ridge model is {}".format(ridge_cv.score(x_train,y_train)))
print("The test score for ridge model is {}".format(ridge_cv.score(x_test,y_test)))
```

```
The train score for ridge model is 0.9999999999999918
The test score for ridge model is 0.9999999999999917
```

In [34]:
```
from sklearn.linear_model import ElasticNet
regr=ElasticNet()
regr.fit(x,y)
print(regr.coef_)
print(regr.intercept_)
```

```
[3.74911416e-07 8.01713369e-01]
8.613651062148541
```

In [35]:
```
y_pred_elastic=regr.predict(x_train)
```

In [36]:
```
mean_squared_error=np.mean((y_pred_elastic-y_train)**2)
print("Mean Squared Error on test set",mean_squared_error)
```

```
Mean Squared Error on test set 1219.3167691435765
```

In [ ]:
```

```