AI4ICPS AI HACKATHON
SPRING 2023

PRESENTATION

AI4ICPS

IIT Kharagpur

Table of Contents

Presentation Outline

INTRODUCTION

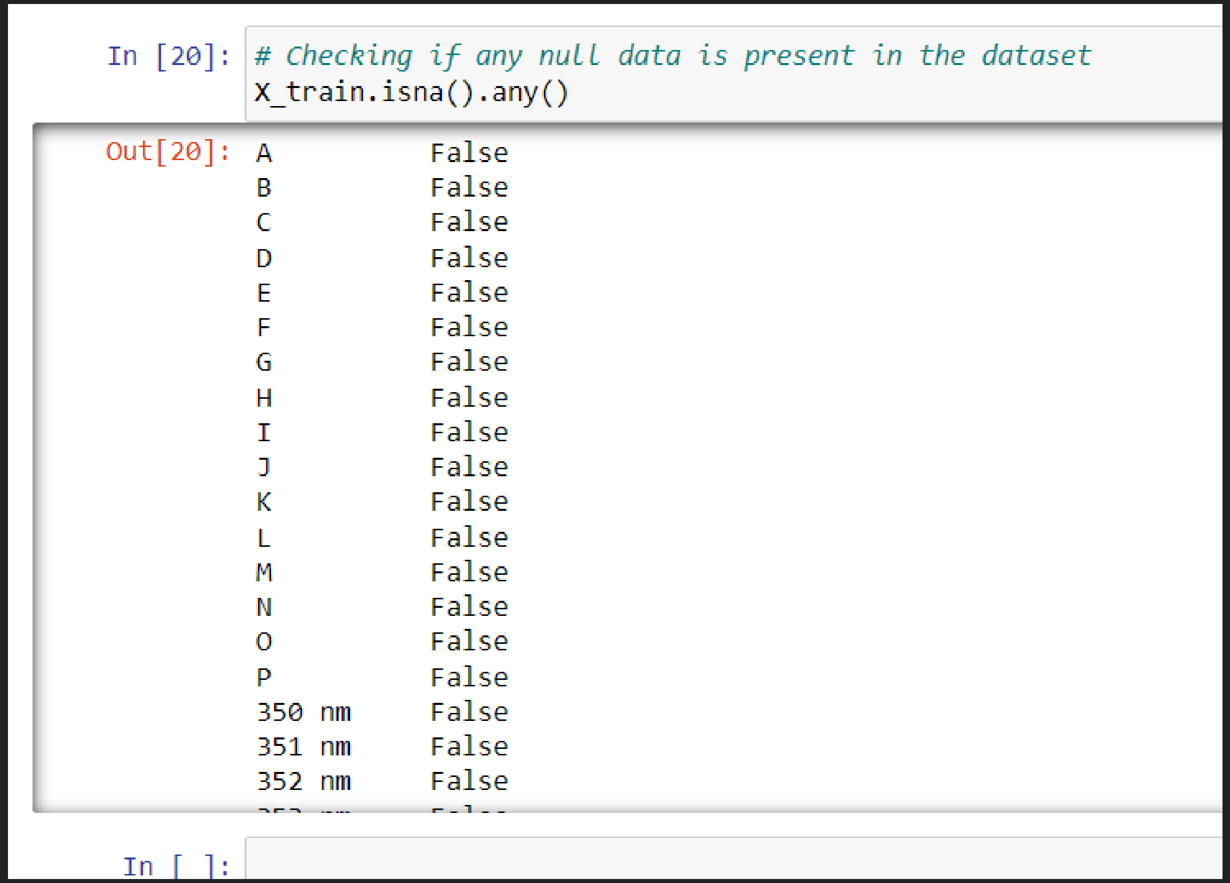
Pramod Goyal

Pre-final year

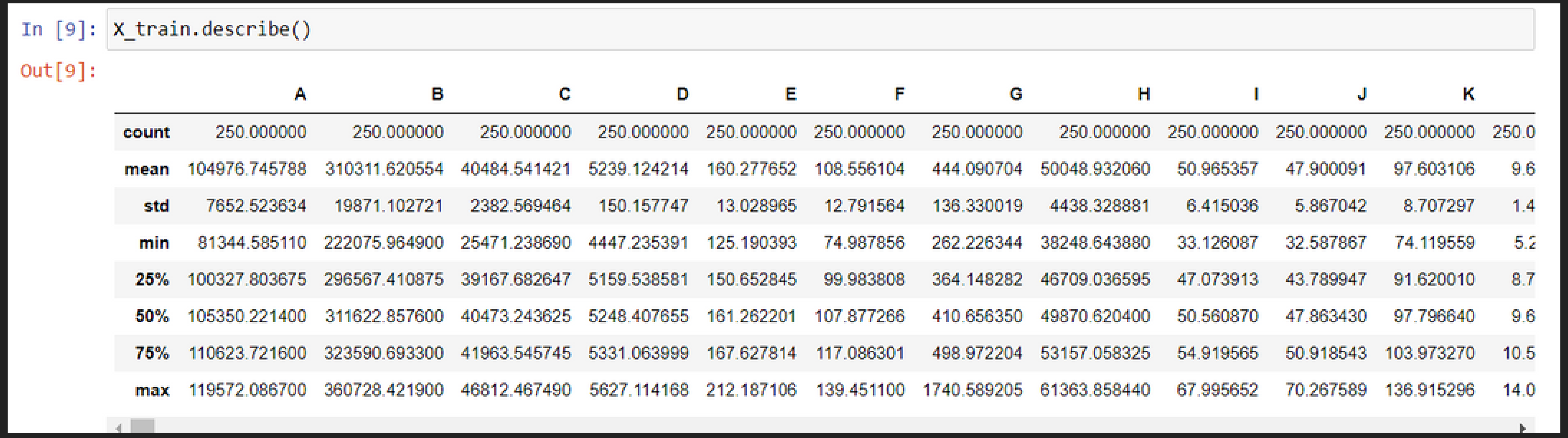
Electronics and Instrumentation Engineering

NIT Rourkela

EXPLORING THE DATA



```
In [20]: # Checking if any null data is present in the dataset
         X_train.isna().any()

Out[20]: A          False
         B          False
         C          False
         D          False
         E          False
         F          False
         G          False
         H          False
         I          False
         J          False
         K          False
         L          False
         M          False
         N          False
         O          False
         P          False
         350 nm     False
         351 nm     False
         352 nm     False

In [ ]:
```



```
In [9]: X_train.describe()

Out[9]:
```

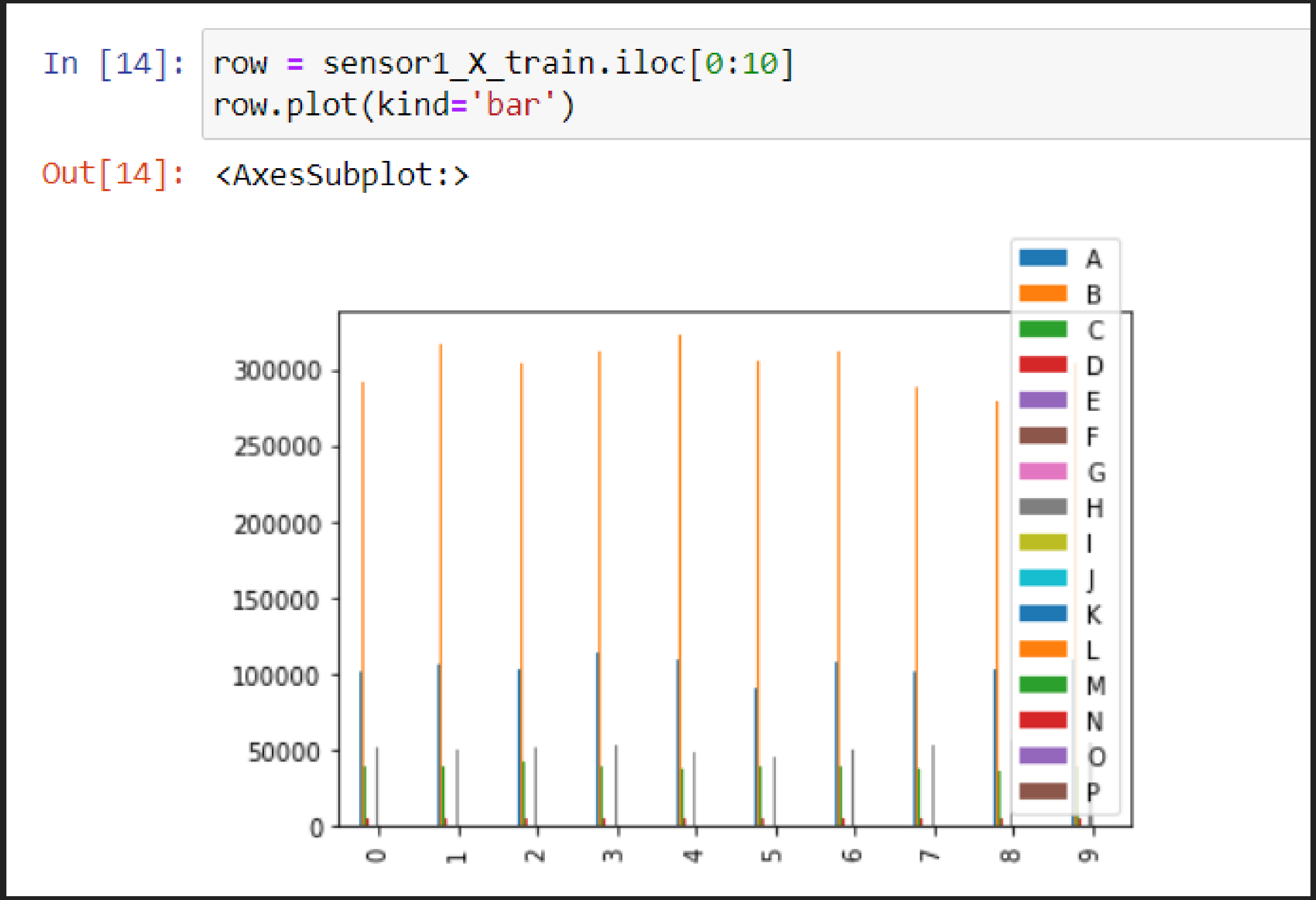
	A	B	C	D	E	F	G	H	I	J	K	
count	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.0
mean	104976.745788	310311.620554	40484.541421	5239.124214	160.277652	108.556104	444.090704	50048.932060	50.965357	47.900091	97.603106	9.6
std	7652.523634	19871.102721	2382.569464	150.157747	13.028965	12.791564	136.330019	4438.328881	6.415036	5.867042	8.707297	1.4
min	81344.585110	222075.964900	25471.238690	4447.235391	125.190393	74.987856	262.226344	38248.643880	33.126087	32.587867	74.119559	5.2
25%	100327.803675	296567.410875	39167.682647	5159.538581	150.652845	99.983808	364.148282	46709.036595	47.073913	43.789947	91.620010	8.7
50%	105350.221400	311622.857600	40473.243625	5248.407655	161.262201	107.877266	410.656350	49870.620400	50.560870	47.863430	97.796640	9.6
75%	110623.721600	323590.693300	41963.545745	5331.063999	167.627814	117.086301	498.972204	53157.058325	54.919565	50.918543	103.973270	10.5
max	119572.086700	360728.421900	46812.467490	5627.114168	212.187106	139.451100	1740.589205	61363.858440	67.995652	70.267589	136.915296	14.0

NULL DATA

First I went through the entire training data to check whether null values were present or not, and the conclusion was, no null values were present. I also went through the minimum value in each column to make sure 0 was not present in the majority of columns.

sensor 1

The data was split into sensor 1 data and sensor 2 data, as we can observe there is much variation between each property and the values from sensor 1, so it is vital to normalize it.

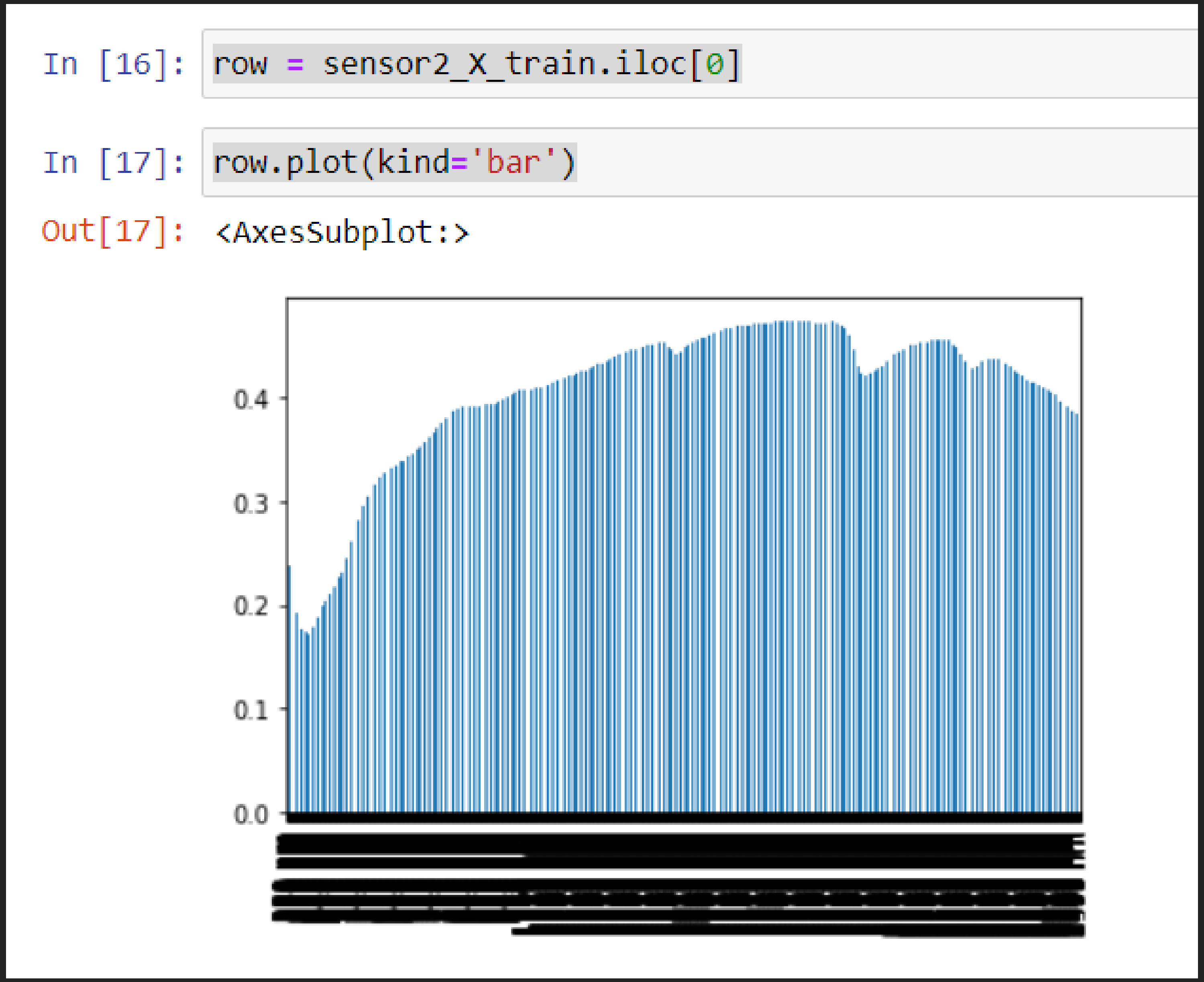


```
In [14]: row = sensor1_X_train.iloc[0:10]
         row.plot(kind='bar')

Out[14]: <AxesSubplot:>
```

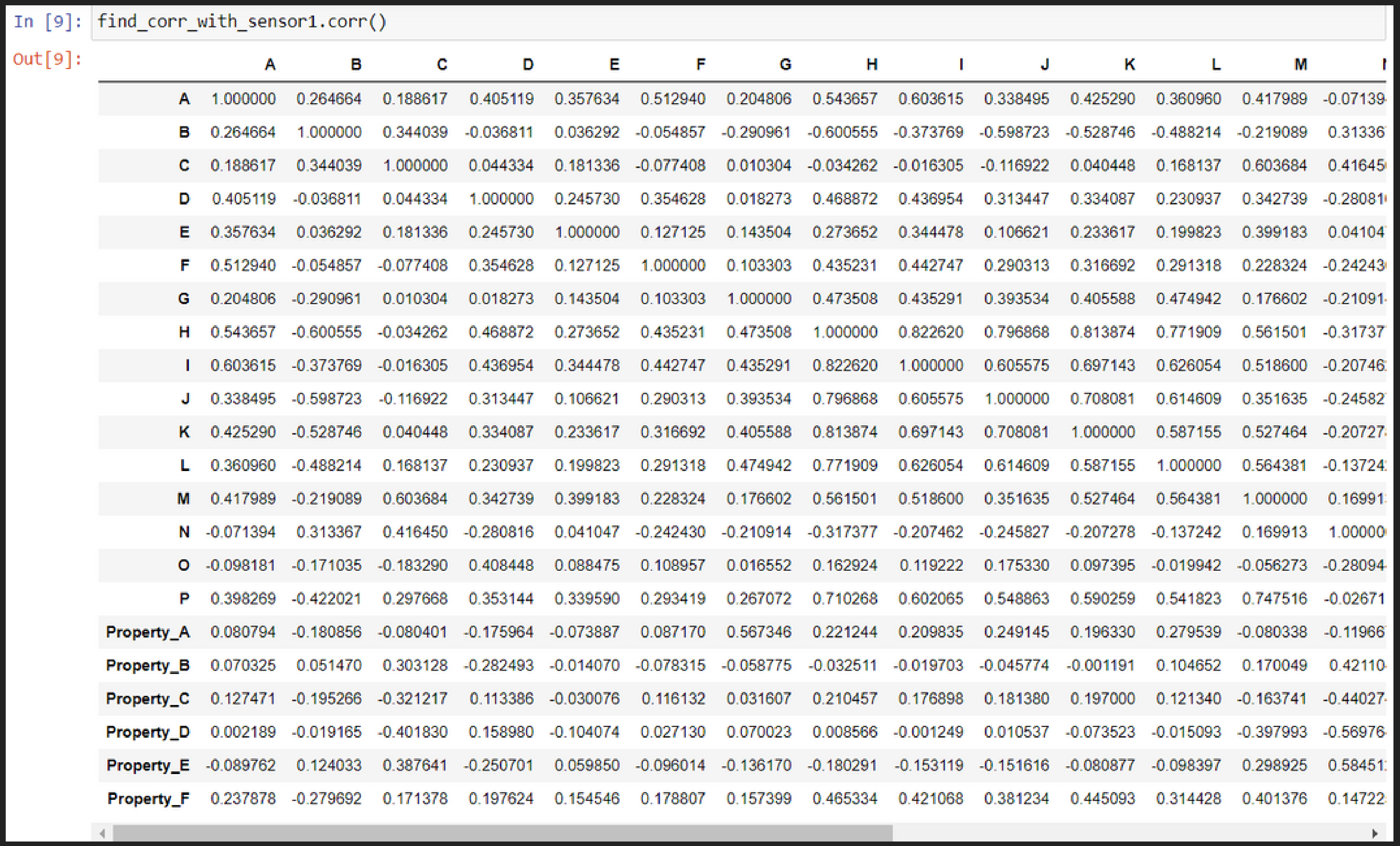
EXPLORING THE DATA

sensor 2

Sensor two had 2151 entries and we can observe there is not much variation between each value in the range of 300 values.



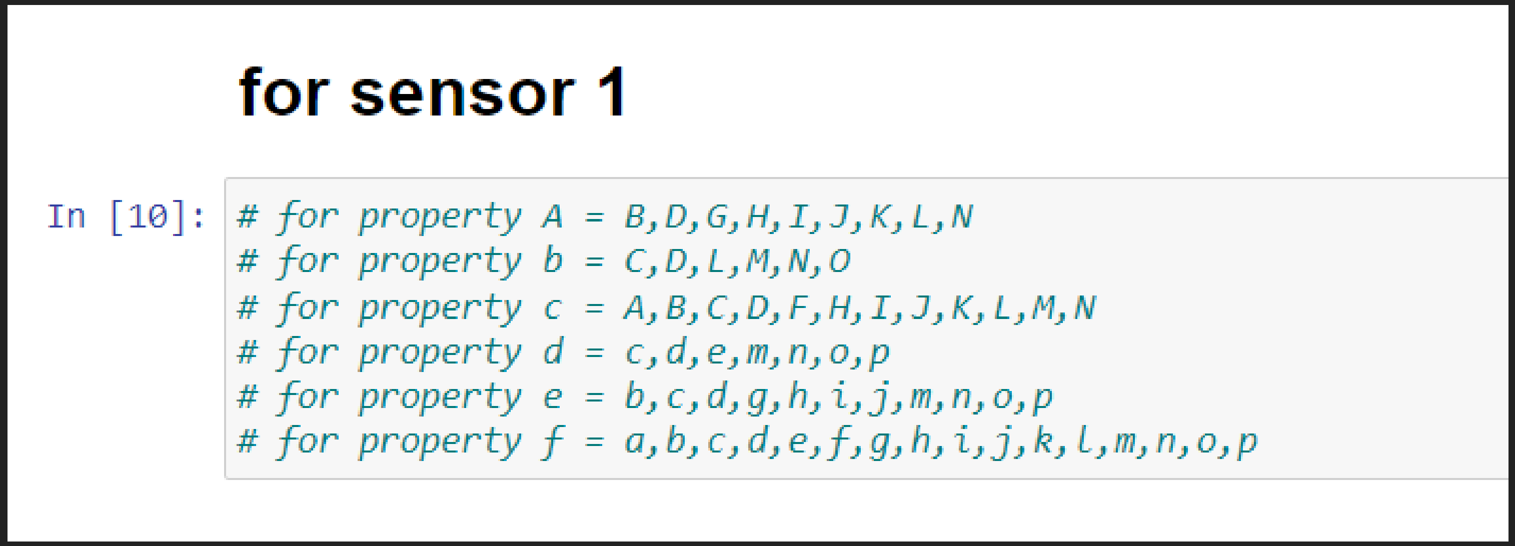
```
In [16]: row = sensor2_X_train.iloc[0]

In [17]: row.plot(kind='bar')
Out[17]: <AxesSubplot:>
```



```
In [9]: find_corr_with_sensor1.corr()
```

Out[9]:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	1.000000	0.264664	0.188617	0.405119	0.357634	0.512940	0.204806	0.543657	0.603615	0.338495	0.425290	0.360960	0.417989	-0.07139
B	0.264664	1.000000	0.344039	-0.036811	0.036292	-0.054857	-0.290961	-0.600555	-0.373769	-0.598723	-0.528746	-0.488214	-0.219089	0.31336
C	0.188617	0.344039	1.000000	0.044334	0.181336	-0.077408	0.010304	-0.034262	-0.016305	-0.116922	0.040448	0.168137	0.603684	0.41645
D	0.405119	-0.036811	0.044334	1.000000	0.245730	0.354628	0.018273	0.468872	0.436954	0.313447	0.334087	0.230937	0.342739	-0.28081
E	0.357634	0.036292	0.181336	0.245730	1.000000	0.127125	0.143504	0.273652	0.344478	0.106621	0.233617	0.199823	0.399183	0.04104
F	0.512940	-0.054857	-0.077408	0.354628	0.127125	1.000000	0.103303	0.435231	0.442747	0.290313	0.316692	0.291318	0.228324	-0.24243
G	0.204806	-0.290961	0.010304	0.018273	0.143504	0.103303	1.000000	0.473508	0.435291	0.393534	0.405588	0.474942	0.176602	-0.21091
H	0.543657	-0.600555	-0.034262	0.468872	0.273652	0.435231	0.473508	1.000000	0.822620	0.796868	0.813874	0.771909	0.561501	-0.31737
I	0.603615	-0.373769	-0.016305	0.436954	0.344478	0.442747	0.435291	0.822620	1.000000	0.605575	0.697143	0.626054	0.518600	-0.20746
J	0.338495	-0.598723	-0.116922	0.313447	0.106621	0.290313	0.393534	0.796868	0.605575	1.000000	0.708081	0.614609	0.351635	-0.24582
K	0.425290	-0.528746	0.040448	0.334087	0.233617	0.316692	0.405588	0.813874	0.697143	0.708081	1.000000	0.587155	0.527464	-0.20727
L	0.360960	-0.488214	0.168137	0.230937	0.199823	0.291318	0.474942	0.771909	0.626054	0.614609	0.587155	1.000000	0.564381	-0.13724
M	0.417989	-0.219089	0.603684	0.342739	0.399183	0.228324	0.176602	0.561501	0.518600	0.351635	0.527464	0.564381	1.000000	0.16991
N	-0.071394	0.313367	0.416450	-0.280816	0.041047	-0.242430	-0.210914	-0.317377	-0.207462	-0.245827	-0.207278	-0.137242	0.169913	1.00000
O	-0.098181	-0.171035	-0.183290	0.408448	0.088475	0.108957	0.016552	0.162924	0.119222	0.175330	0.097395	-0.019942	-0.056273	-0.28094
P	0.398269	-0.422021	0.297668	0.353144	0.339590	0.293419	0.267072	0.710268	0.602065	0.548863	0.590259	0.541823	0.747516	-0.02671
Property_A	0.080794	-0.180856	-0.080401	-0.175964	-0.073887	0.087170	0.567346	0.221244	0.209835	0.249145	0.196330	0.279539	-0.080338	-0.11966
Property_B	0.070325	0.051470	0.303128	-0.282493	-0.014070	-0.078315	-0.058775	-0.032511	-0.019703	-0.045774	-0.001191	0.104652	0.170049	0.42110
Property_C	0.127471	-0.195266	-0.321217	0.113386	-0.030076	0.116132	0.031607	0.210457	0.176898	0.181380	0.197000	0.121340	-0.163741	-0.44027
Property_D	0.002189	-0.019165	-0.401830	0.158980	-0.104074	0.027130	0.070023	0.008566	-0.001249	0.010537	-0.073523	-0.015093	-0.397993	-0.56976
Property_E	-0.089762	0.124033	0.387641	-0.250701	0.059850	-0.096014	-0.136170	-0.180291	-0.153119	-0.151616	-0.080877	-0.098397	0.298925	0.58451
Property_F	0.237878	-0.279692	0.171378	0.197624	0.154546	0.178807	0.157399	0.465334	0.421068	0.381234	0.445093	0.314428	0.401376	0.14722



for sensor 1

```
In [10]: # for property A = B,D,G,H,I,J,K,L,N
         # for property b = C,D,L,M,N,O
         # for property c = A,B,C,D,F,H,I,J,K,L,M,N
         # for property d = c,d,e,m,n,o,p
         # for property e = b,c,d,g,h,i,j,m,n,o,p
         # for property f = a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p
```

CORRELATION AND VARIANCE

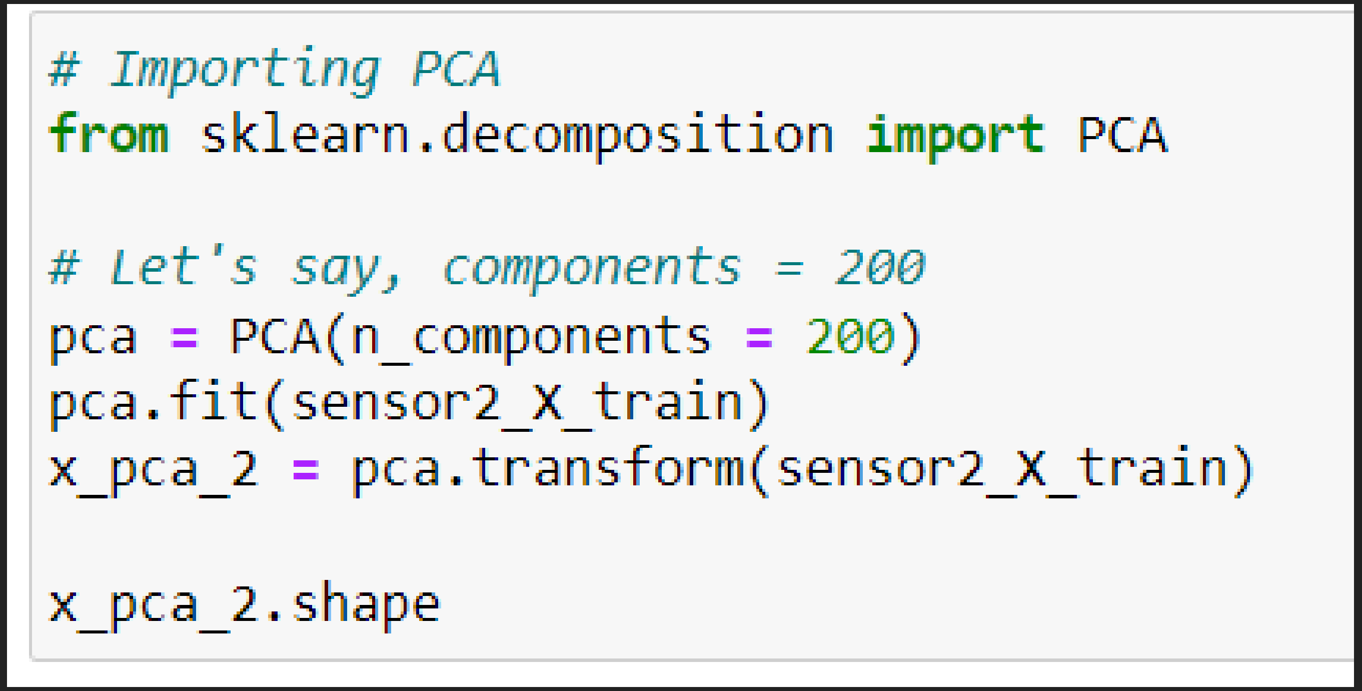
Sensor 1

It was relatively easier to find the highly correlated columns for the individual property and the ones present in sensor 1.

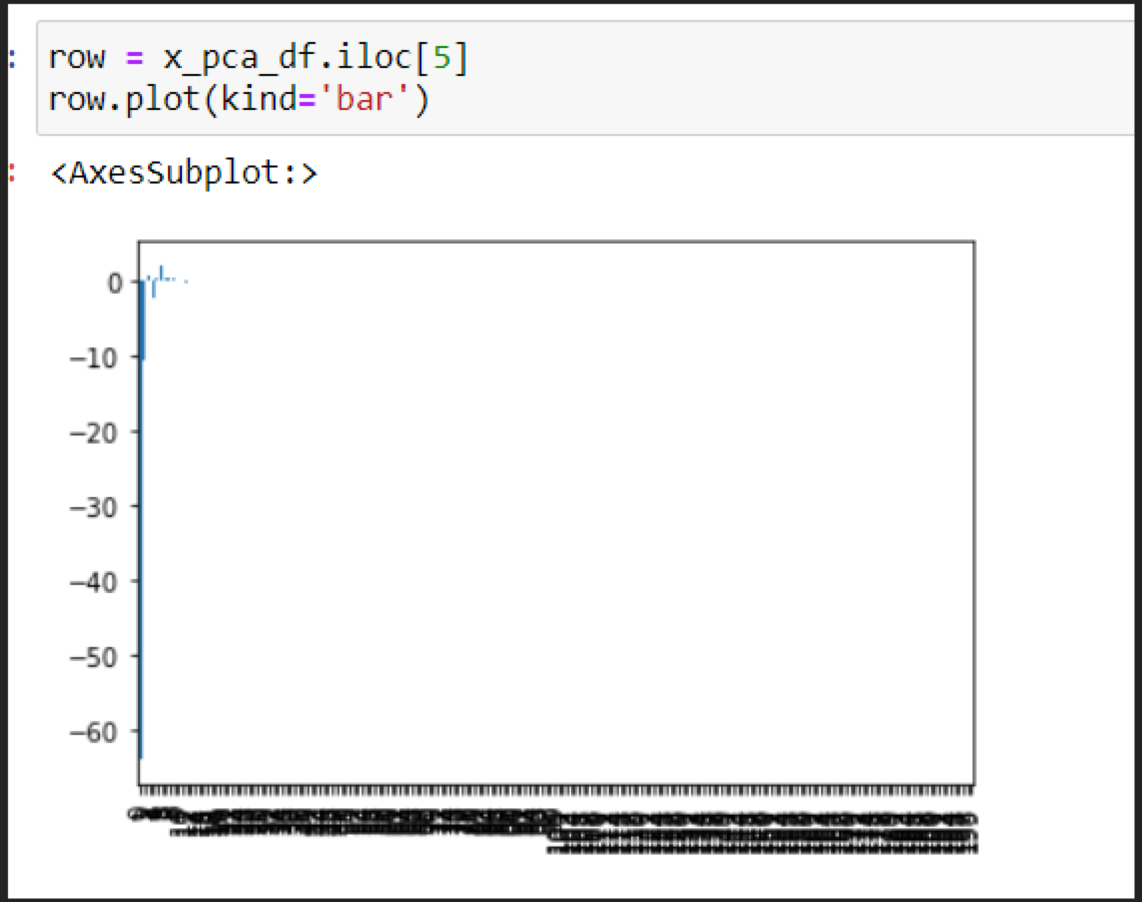
Sensor 2

As so many columns were present, PCA was first applied and tested if it presented good values, but as so much variance was found, it was concluded to not use PCA but rather manually check which subsection of sensor 2 are the most correlated

CORRELATION AND VARIANCE

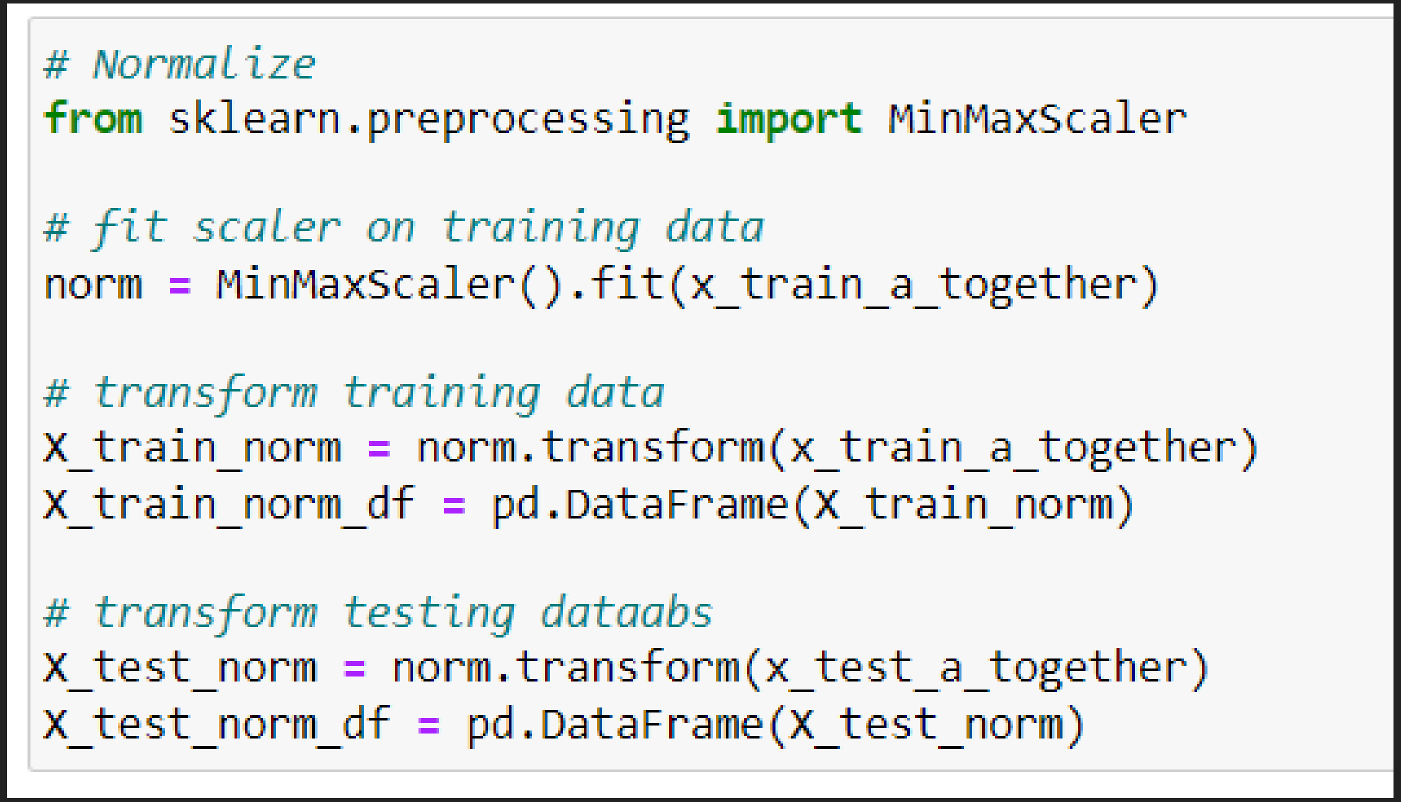


```
# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 200
pca = PCA(n_components = 200)
pca.fit(sensor2_X_train)
x_pca_2 = pca.transform(sensor2_X_train)

x_pca_2.shape
```

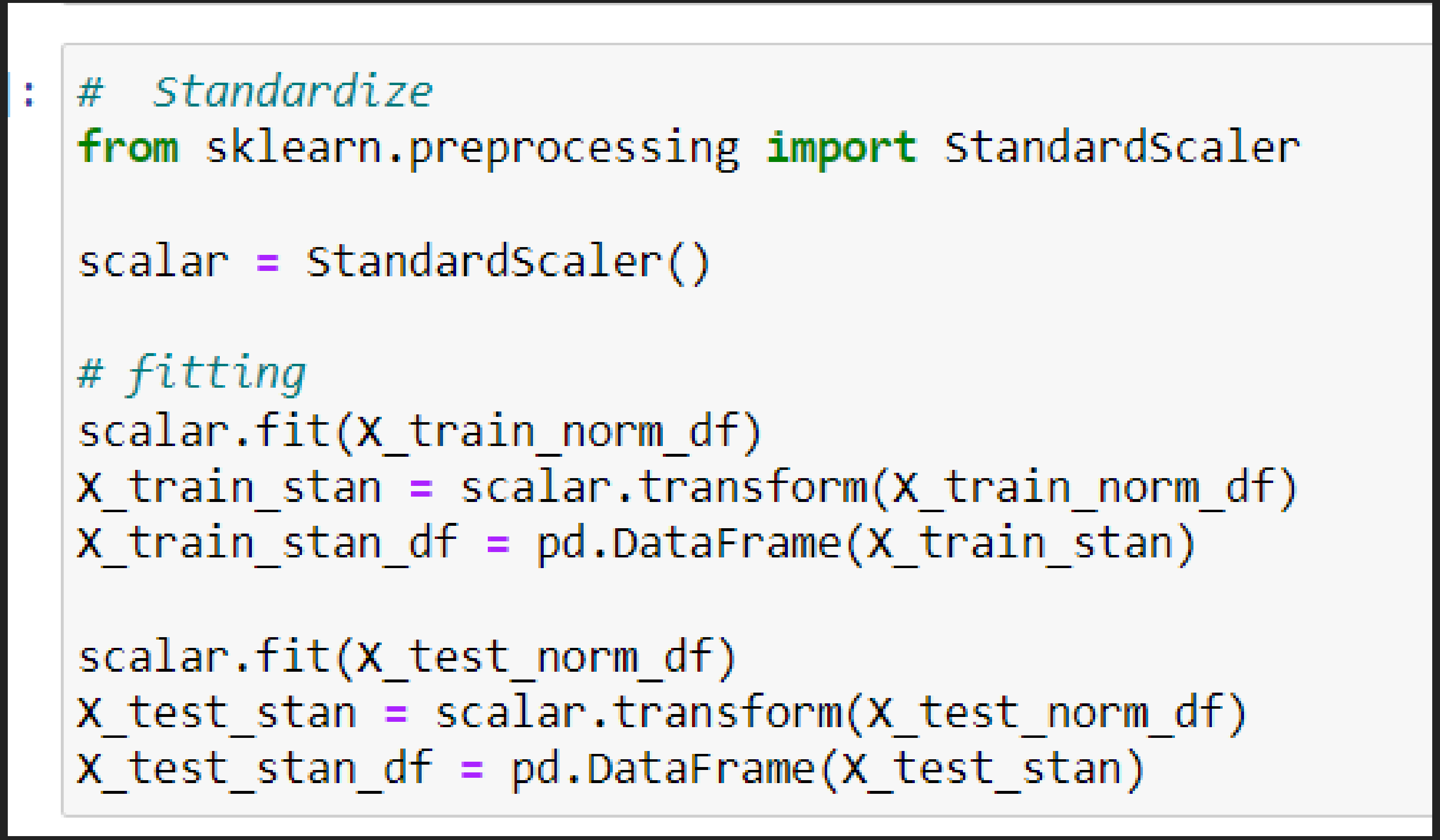


```
row = x_pca_df.iloc[5]
row.plot(kind='bar')
```

```
<AxesSubplot:>
```

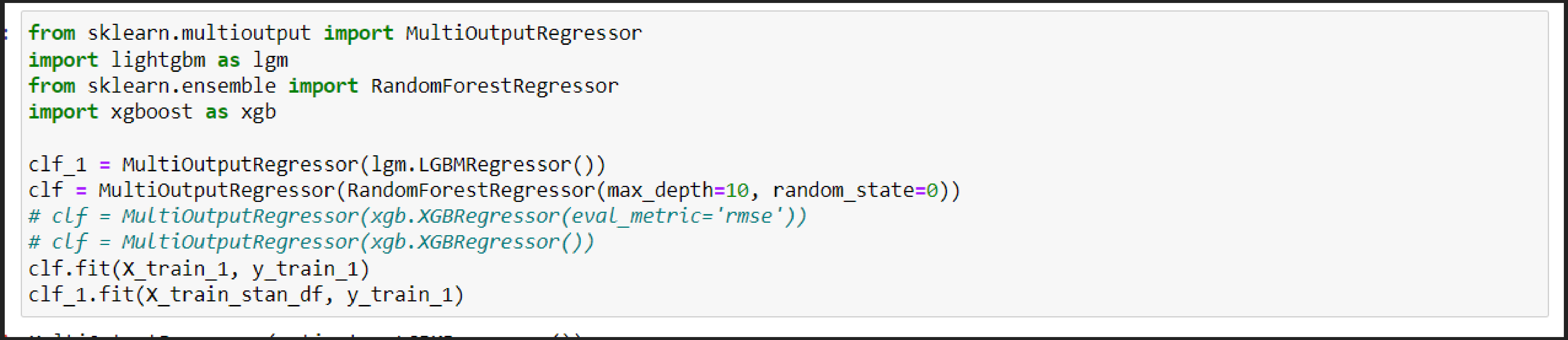


```python
# Normalize
from sklearn.preprocessing import MinMaxScaler

# fit scaler on training data
norm = MinMaxScaler().fit(x_train_a_together)

# transform training data
X_train_norm = norm.transform(x_train_a_together)
X_train_norm_df = pd.DataFrame(X_train_norm)

# transform testing dataabs
X_test_norm = norm.transform(x_test_a_together)
X_test_norm_df = pd.DataFrame(X_test_norm)
```



```python
#  Standardize
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

# fitting
scalar.fit(X_train_norm_df)
X_train_stan = scalar.transform(X_train_norm_df)
X_train_stan_df = pd.DataFrame(X_train_stan)

scalar.fit(X_test_norm_df)
X_test_stan = scalar.transform(X_test_norm_df)
X_test_stan_df = pd.DataFrame(X_test_stan)
```

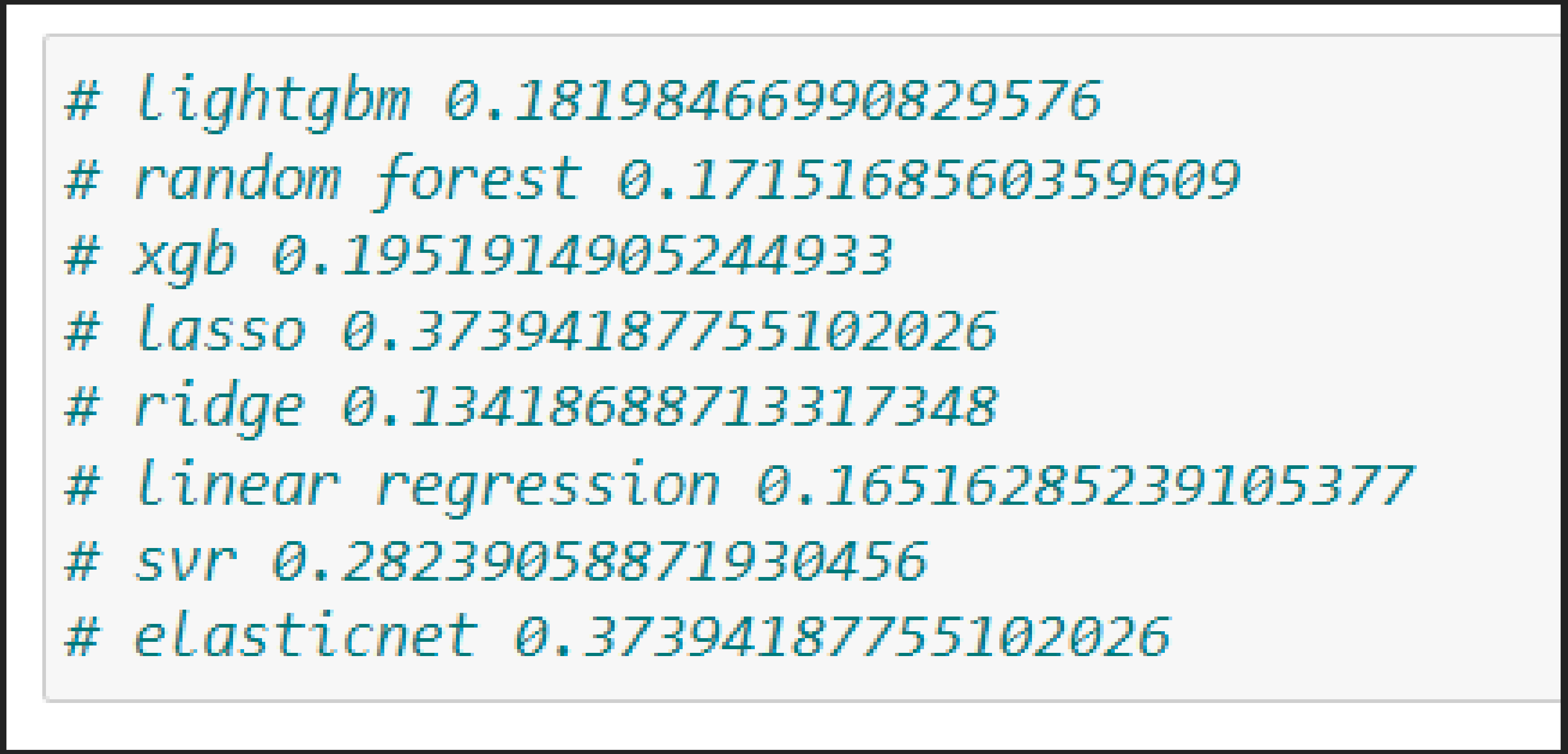
MODEL SELECTION



```
from sklearn.multioutput import MultiOutputRegressor
import lightgbm as lgm
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

clf_1 = MultiOutputRegressor(lgm.LGBMRegressor())
clf = MultiOutputRegressor(RandomForestRegressor(max_depth=10, random_state=0))
# clf = MultiOutputRegressor(xgb.XGBRegressor(eval_metric='rmse'))
# clf = MultiOutputRegressor(xgb.XGBRegressor())
clf.fit(X_train_1, y_train_1)
clf_1.fit(X_train_stan_df, y_train_1)
```

Initially, multioutput regressor was used to find the performance of various models on the dataset

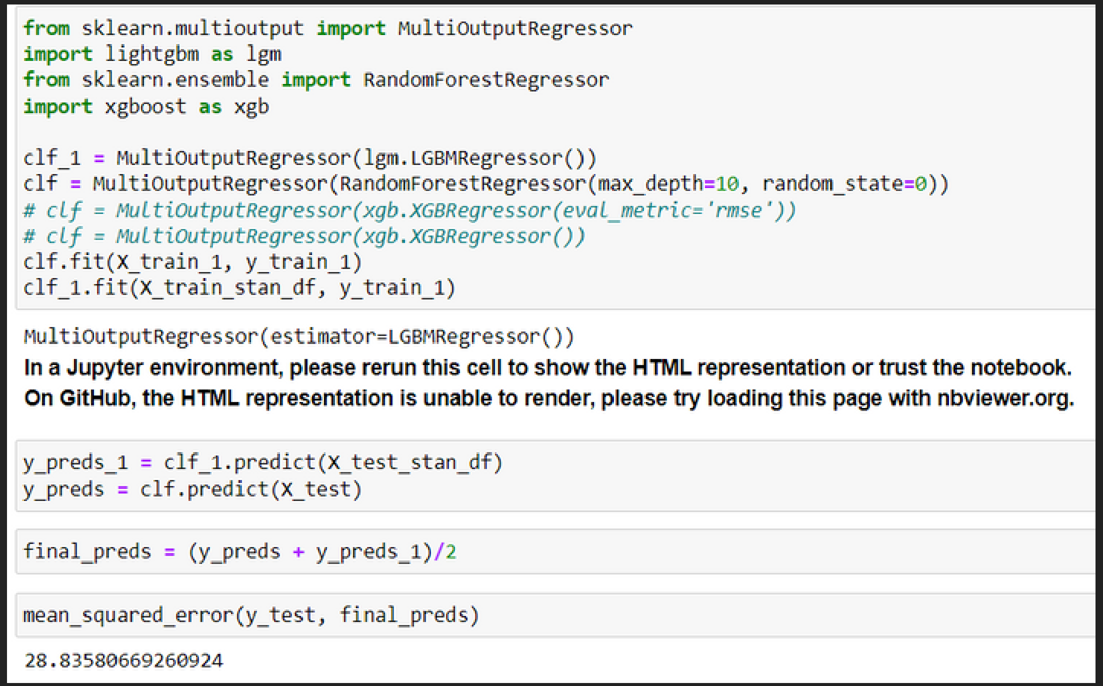
MODEL SELECTION



```
# lightgbm 0.18198466990829576
# random forest 0.1715168560359609
# xgb 0.1951914905244933
# lasso 0.37394187755102026
# ridge 0.13418688713317348
# linear regression 0.16516285239105377
# svr 0.28239058871930456
# elasticnet 0.37394187755102026
```

Multiple models were tried, linear regression was kept as the baseline

MODEL SELECTION



```
from sklearn.multioutput import MultiOutputRegressor
import lightgbm as lgm
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

clf_1 = MultiOutputRegressor(lgm.LGBMRegressor())
clf = MultiOutputRegressor(RandomForestRegressor(max_depth=10, random_state=0))
# clf = MultiOutputRegressor(xgb.XGBRegressor(eval_metric='rmse'))
# clf = MultiOutputRegressor(xgb.XGBRegressor())
clf.fit(X_train_1, y_train_1)
clf_1.fit(X_train_stan_df, y_train_1)
```

```
MultiOutputRegressor(estimator=LGBMRegressor())
```

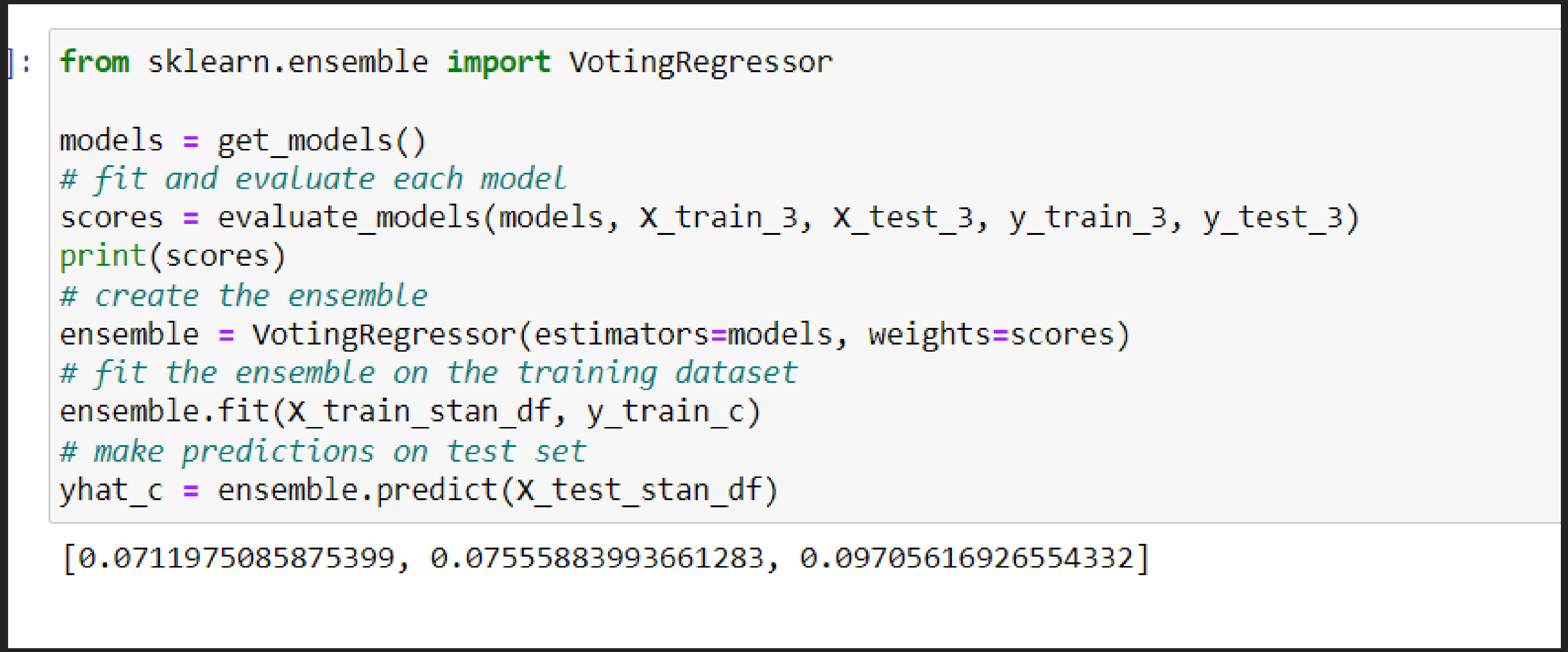
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
y_preds_1 = clf_1.predict(X_test_stan_df)
y_preds = clf.predict(X_test)
```

```
final_preds = (y_preds + y_preds_1)/2
```

```
mean_squared_error(y_test, final_preds)
```

```
28.83580669260924
```



```
from sklearn.ensemble import VotingRegressor

models = get_models()
# fit and evaluate each model
scores = evaluate_models(models, X_train_3, X_test_3, y_train_3, y_test_3)
print(scores)
# create the ensemble
ensemble = VotingRegressor(estimators=models, weights=scores)
# fit the ensemble on the training dataset
ensemble.fit(X_train_stan_df, y_train_c)
# make predictions on test set
yhat_c = ensemble.predict(X_test_stan_df)
```

```
[0.0711975085875399, 0.07555883993661283, 0.09705616926554332]
```

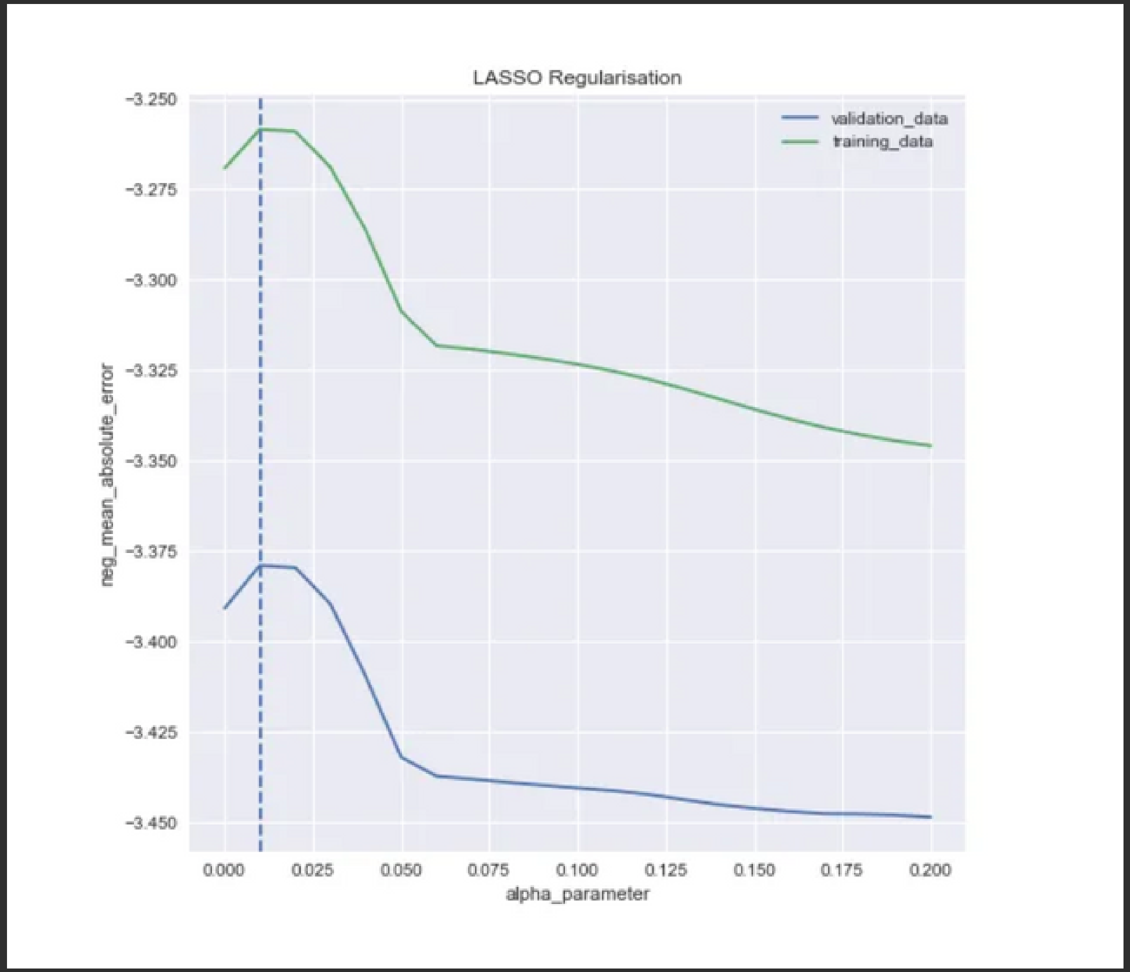
Stacking methods

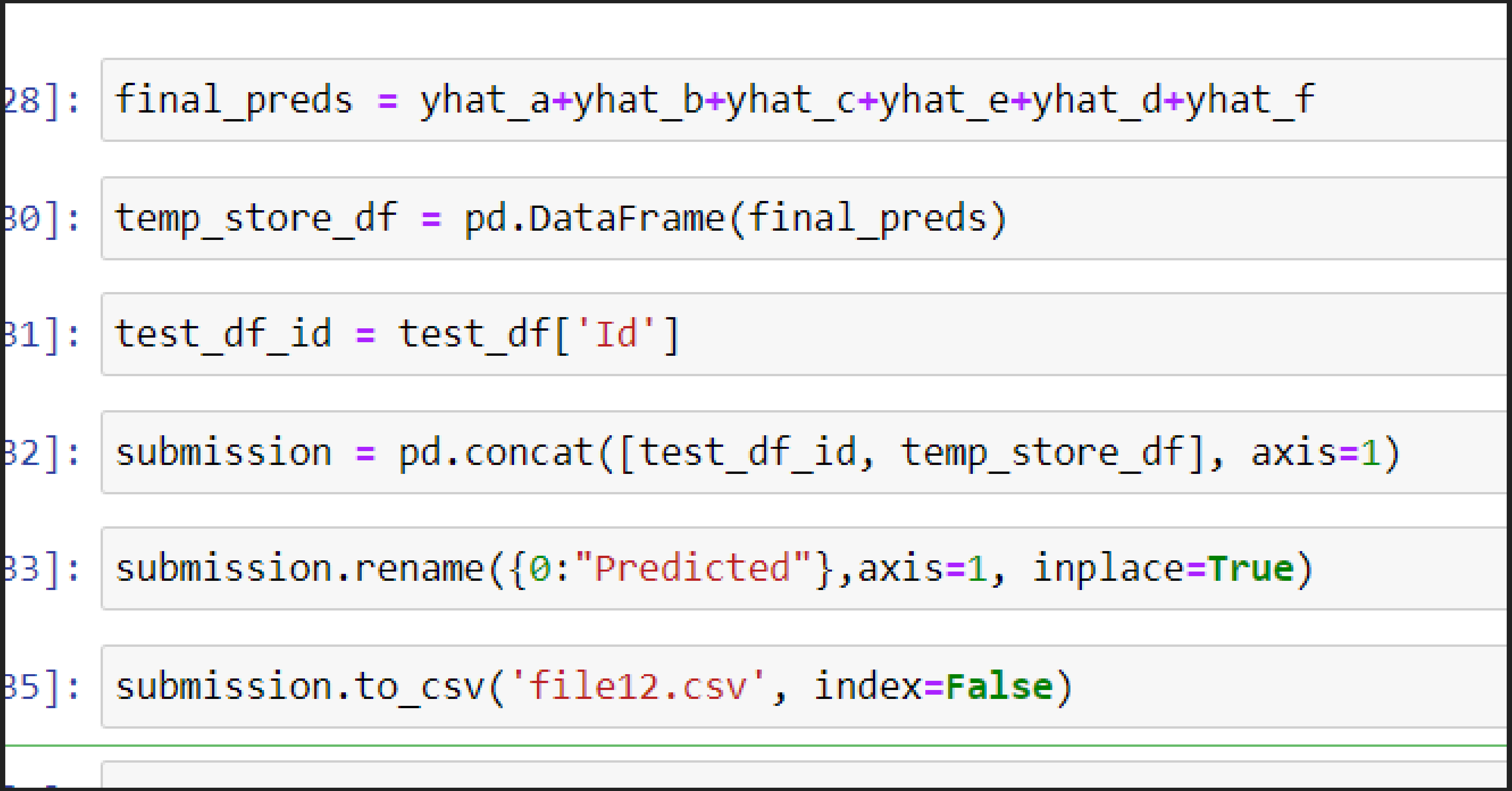
Mean stack

Weighted average stack

This presented better result as different models had different RMSE

GridSearchCV was applied to try the best parameter for various models





```
28]: final_preds = yhat_a+yhat_b+yhat_c+yhat_e+yhat_d+yhat_f

30]: temp_store_df = pd.DataFrame(final_preds)

31]: test_df_id = test_df['Id']

32]: submission = pd.concat([test_df_id, temp_store_df], axis=1)

33]: submission.rename({0:"Predicted"},axis=1, inplace=True)

35]: submission.to_csv('file12.csv', index=False)
```

Individual properties were predicted using different models with both sensor 1 and sensor 2 data

THANK YOU