E-14a: Lab 3

Numpy | Pandas | Sklearn

Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

For more information, check this page: http://www.numpy.org/ (http://www.numpy.org/)

In [2]:
```
# Load library
import numpy as np
```

Let us explore numpy through examples.

Arrays

In [3]:
```
a = np.array([1, 2, 3])
print(a.shape)
print(a[0], a[1], a[2])

a[0] = 5
print(a)
```

```
(3,)
1 2 3
[5 2 3]
```

In [ ]:
```
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b.shape)
print(b[0, 0], b[0, 1], b[1, 0])
```

In [4]:
```
a = np.zeros((2, 2))
print(a)
```

```
[[0. 0.]
 [0. 0.]]
```

In [5]:
```
b = np.ones((1, 2))
print(b)
```

```
[[1. 1.]]
```

In [6]:
```
e = np.random.random((2, 2))
print(e)
```

```
[[0.04502328 0.85775263]
 [0.99025505 0.58551765]]
```

Array indexing

In [7]:
```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(a)
print(a[0, 1])
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
2
```

In [8]:
```
b = a[:2, 1:3]
print(b)
```

```
[[2 3]
 [6 7]]
```

In [9]:
```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
row_r1 = a[1, :]
row_r2 = a[1:2, :]
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
```

Array math

In [10]:
```
x = np.array([[1, 2], [3, 4]], dtype=np.float64)
y = np.array([[5, 6], [7, 8]], dtype=np.float64)
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

In [11]:
```
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

In [12]:
```
print(x * y)
print(np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

In [13]:
```
print(x / y)
print(np.divide(x, y))
```

```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

In [14]:
```
print(np.sqrt(x))
```

```
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

In [15]:
```
x = np.array([[1, 2], [3, 4]])
print(np.sum(x))
print(np.sum(x, axis=0))
print(np.sum(x, axis=1))
```

```
10
[4 6]
[3 7]
```

In [16]:
```
x = np.array([[1, 2], [3, 4]])
print(x)
print(x.T)
```

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
```

In [17]:
```
v = np.array([1, 2, 3])
print(v)
print(v.T)
```

```
[1 2 3]
[1 2 3]
```

Broadcasting

In [18]:
```
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)

for i in range(4):
    y[i, :] = x[i, :] + v

print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

Plotting

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

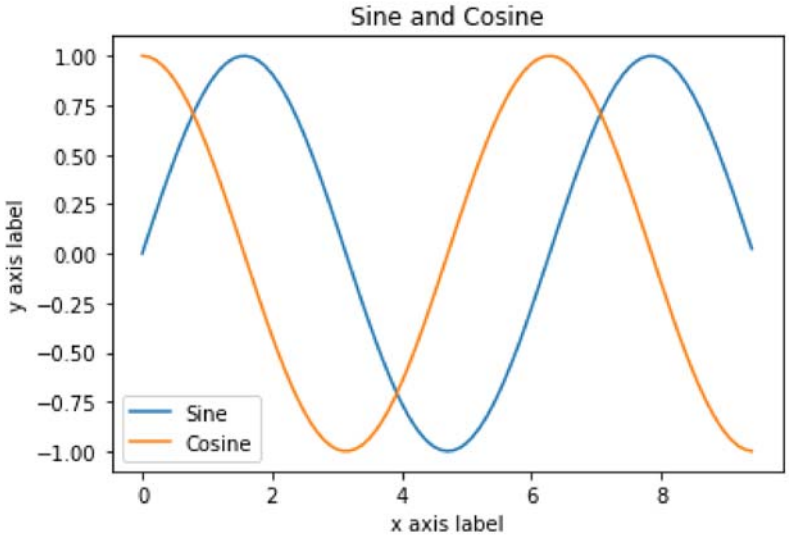
Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery: https://matplotlib.org/index.html (https://matplotlib.org/index.html)

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

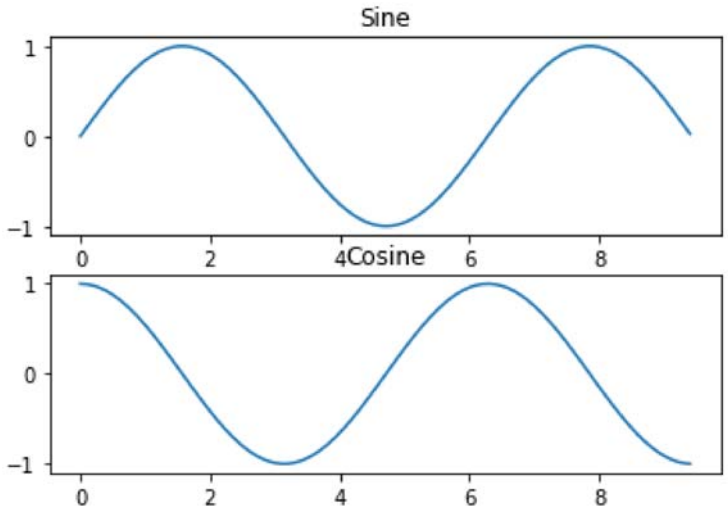
In [19]:
```
import matplotlib.pyplot as plt

x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

plt.plot(x, y)
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

In [20]:
```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



In [21]:
```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.subplot(2, 1, 1)
plt.plot(x, y_sin)
plt.title('Sine')

plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')
plt.show()
```



Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NumFOCUS sponsored project. This will help ensure the success of development of pandas as a world-class open-source project, and makes it possible to donate to the project.

For more information, check this page: https://pandas.pydata.org/ (https://pandas.pydata.org/)

In [22]:
```
# Load library
import pandas as pd
```

In [23]:
```
# Import dataset
dataset = pd.read_csv("Boston_March2018.csv")
```

In [24]:
```
dataset.head()
```

Out[24]:

	MLSNUM	STATUS	LISTPRICE	SOLDPRICE	LISTDATE	SOLDDATE	EXPIREDDATE	DOM	DTO
0	72049670	SLD	239900.0	247000.0	8/6/2016	3/1/2018	NaN	295	295
1	72056522	SLD	338000.0	338000.0	8/22/2016	3/1/2018	NaN	545	515
2	72080286	SLD	2999999.0	2950000.0	10/12/2016	3/1/2018	NaN	504	493
3	72118879	SLD	2600000.0	2600000.0	2/14/2017	3/1/2018	NaN	28	14
4	72124101	SLD	525000.0	525000.0	2/28/2017	3/1/2018	NaN	867	287

5 rows × 38 columns

In [25]:
```
dataset.columns
```

Out[25]:
```
Index(['MLSNUM', 'STATUS', 'LISTPRICE', 'SOLDPRICE', 'LISTDATE', 'SOLDDATE',
       'EXPIREDDATE', 'DOM', 'DTO', 'ADDRESS', 'CITY', 'STATE', 'ZIP', 'AREA',
       'BEDS', 'BATHS', 'SQFT', 'AGE', 'LOTSIZE', 'AGENTNAME', 'OFFICENAME',
       'OFFICEPHONE', 'SHOWINGINSTRUCTIONS', 'REMARKS', 'STYLE', 'LEVEL',
       'GARAGE', 'HEATING', 'COOLING', 'ELEMENTARYSCHOOL', 'JUNIORHIGHSCHOOL',
       'HIGHSCHOOL', 'OTHERFEATURES', 'PROPTYPE', 'STREETNAME', 'HOUSENUM1',
       'HOUSENUM2', 'PHOTOURL'],
      dtype='object')
```

In [26]:
```
dataset.LISTPRICE.head()
```

Out[26]:
```
0     239900.0
1     338000.0
2    2999999.0
3    2600000.0
4     525000.0
Name: LISTPRICE, dtype: float64
```

In [27]:
```
dataset['LISTPRICE'].head()
```

Out[27]:
```
0     239900.0
1     338000.0
2    2999999.0
3    2600000.0
4     525000.0
Name: LISTPRICE, dtype: float64
```

In [28]:
```
columns_i_want=['LISTPRICE', 'SOLDPRICE']
dataset[columns_i_want].head()
```

Out[28]:

	LISTPRICE	SOLDPRICE
0	239900.0	247000.0
1	338000.0	338000.0
2	2999999.0	2950000.0
3	2600000.0	2600000.0
4	525000.0	525000.0

In [29]:
```
dataset.PROPTYPE.unique()
```

Out[29]:
```
array(['MF', 'CC', 'SF'], dtype=object)
```

In [30]:
```
dataset.iloc[:2]
```

Out[30]:

	MLSNUM	STATUS	LISTPRICE	SOLDPRICE	LISTDATE	SOLDDATE	EXPIREDDATE	DOM	DTO
0	72049670	SLD	239900.0	247000.0	8/6/2016	3/1/2018	NaN	295	295
1	72056522	SLD	338000.0	338000.0	8/22/2016	3/1/2018	NaN	545	515

2 rows × 38 columns

In [31]:
```
dataset.iloc[:2, :2]
```

Out[31]:

	MLSNUM	STATUS
0	72049670	SLD
1	72056522	SLD

In [32]:
```
dataset.iloc[:, 4:6].head()
```

Out[32]:

	LISTDATE	SOLDDATE
0	8/6/2016	3/1/2018
1	8/22/2016	3/1/2018
2	10/12/2016	3/1/2018
3	2/14/2017	3/1/2018
4	2/28/2017	3/1/2018

In [33]:
```
dataset.SOLDPRICE < 1000000
```

Out[33]:
```
0        True
1        True
2       False
3       False
4        True
5        True
6        True
7        True
8       False
9        True
10       True
11       True
12      False
13      False
14      False
15       True
16      False
17       True
18       True
```

In [34]:
```
dataset[dataset.SOLDPRICE < 1000000].head()
```

Out[34]:

	MLSNUM	STATUS	LISTPRICE	SOLDPRICE	LISTDATE	SOLDDATE	EXPIREDDATE	DOM	DTO
0	72049670	SLD	239900.0	247000.0	8/6/2016	3/1/2018	NaN	295	295
1	72056522	SLD	338000.0	338000.0	8/22/2016	3/1/2018	NaN	545	515
4	72124101	SLD	525000.0	525000.0	2/28/2017	3/1/2018	NaN	867	287
5	72133120	SLD	468000.0	479000.0	3/20/2017	3/1/2018	NaN	273	273
6	72148511	SLD	209900.0	176505.0	4/19/2017	3/1/2018	NaN	311	287

5 rows × 38 columns

In [35]:
```
np.sum(dataset.SOLDPRICE), np.mean(dataset.SOLDPRICE)
```

Out[35]:
```
(2464492263.65, 476598.7746374008)
```

In [36]:
```
dataset[(dataset.SOLDPRICE < 1000000) & (dataset.PROPTYPE == "SF")]
```

Out[36]:

	MLSNUM	STATUS	LISTPRICE	SOLDPRICE	LISTDATE	SOLDDATE	EXPIREDDATE	DOM
4	72124101	SLD	525000.0	525000.0	2/28/2017	3/1/2018	NaN	867
5	72133120	SLD	468000.0	479000.0	3/20/2017	3/1/2018	NaN	273
6	72148511	SLD	209900.0	176505.0	4/19/2017	3/1/2018	NaN	311
7	72153413	SLD	549900.0	530000.0	4/26/2017	3/1/2018	NaN	261
9	72166889	SLD	130000.0	130000.0	5/16/2017	3/1/2018	NaN	185
10	72166942	SLD	1150000.0	945000.0	5/18/2017	3/1/2018	NaN	194

In [37]:
```
dataset.describe()
```

Out[37]:

	MLSNUM	LISTPRICE	SOLDPRICE	EXPIREDDATE	DOM	DTO	
count	5.171000e+03	5.171000e+03	5.171000e+03	0.0	5171.000000	5171.000000	5171.
mean	7.225305e+07	4.807745e+05	4.765988e+05	NaN	69.591762	48.977374	2062.
std	4.473714e+04	4.945971e+05	5.016361e+05	NaN	92.763384	68.484702	1119.
min	7.115216e+07	1.990000e+04	2.300000e+04	NaN	0.000000	0.000000	101.
25%	7.224754e+07	2.499000e+05	2.450000e+05	NaN	18.000000	6.000000	1747.
50%	7.226724e+07	3.748000e+05	3.700000e+05	NaN	35.000000	20.000000	2048.
75%	7.227533e+07	5.499000e+05	5.515000e+05	NaN	94.000000	73.000000	2360.
max	7.230167e+07	1.050000e+07	1.511000e+07	NaN	1562.000000	938.000000	34452.

In [41]:
```
grouped_by_type = dataset.groupby("PROPTYPE")
grouped_by_type
```

Out[41]:
```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000154BEBE6630>
```

In [43]:
```
dataset.groupby("PROPTYPE").describe()
```

Out[43]:

	AGE								BATHS		...	S
	count	mean	std	min	25%	50%	75%	max	count	mean	...	7
PROPTYPE												
CC	1444.0	49.587950	53.831568	0.0	11.0	32.0	93.0	1019.0	1444.0	1.745845	...	1
MF	484.0	108.140496	29.322242	5.0	98.0	113.0	118.0	283.0	484.0	2.927686	...	3
SF	3243.0	60.889608	53.487955	0.0	30.0	57.0	82.0	1863.0	3243.0	2.024206	...	2

3 rows × 112 columns

In [40]:
```
dataset.groupby("PROPTYPE").mean()
```

Out[40]:

	MLSNUM	LISTPRICE	SOLDPRICE	EXPIREDDATE	DOM	DTO	ZIP
PROPTYPE							
CC	7.225364e+07	512072.209661	515437.729363	NaN	54.799169	39.804017	210
MF	7.225119e+07	498397.169421	493278.811983	NaN	55.506198	41.280992	199
SF	7.225306e+07	464208.570802	456815.676056	NaN	78.280604	54.210607	205

TODO: http://pandas.pydata.org/pandas-docs/stable/merging.html (http://pandas.pydata.org/pandas-docs/stable/merging.html)

Sklearn

Sklearn provides simple and efficient tools for data mining and data analysis. It is accessible to everybody, and reusable in various contexts. Built on NumPy, SciPy, and matplotlib.

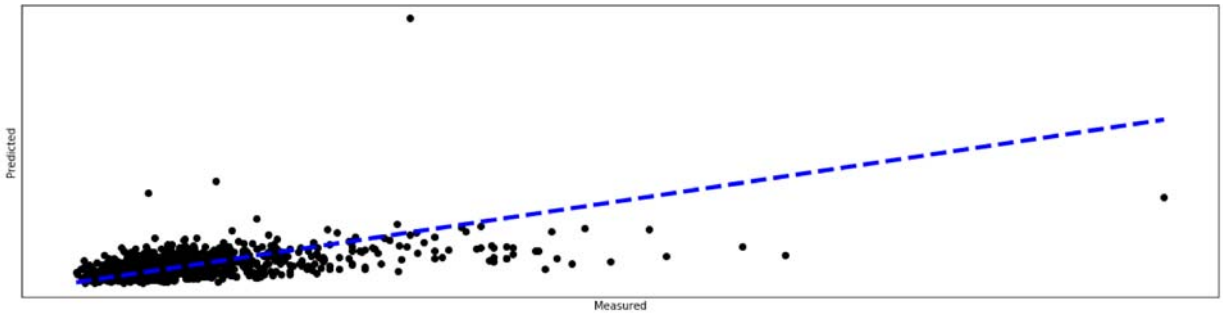
Open source, commercially usable - BSD license. Take a look at: http://scikit-learn.org/stable/ (http://scikit-learn.org/stable/)

In [49]:
```
# Load tools from sklearn library

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

In [63]:
```
dataset=dataset.fillna(-9999)
```

In [64]:
```
# Run Linear Regresison
# Use features: 'BEDS', 'BATHS', 'SQFT', 'AGE','lOTSIZE'

prices = dataset['SOLDPRICE']
features = dataset[['BEDS', 'BATHS', 'SQFT', 'AGE', 'LOTSIZE']]

print ("Orginal shape")
print (features.shape, prices.shape)
print ()

features = features.fillna(-9999)

# Split data using the above example
features_train = features[:4000]
features_test = features[4000:]

# Split the targets into training/testing sets
prices_train = prices[:4000]
prices_test = prices[4000:]

print ("Train shape")
print (features_train.shape, prices_train.shape)
print ()

print ("Test shape")
print (features_test.shape, prices_test.shape)

regr.fit(features_train, prices_train)

# Make predictions using the testing set
prices_pred = regr.predict(features_test)

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(prices_test, prices_pred))

# Plot outputs
plt.figure(figsize=(20, 5))
plt.scatter(prices_test, prices_pred, color='black')
plt.plot([prices_test.min(), prices_test.max()], [prices_test.min(), prices_test.
plt.ylabel('Predicted')
plt.xlabel('Measured')

plt.xticks(())
plt.yticks(())

plt.show()
```

```
Orginal shape
(5171, 5) (5171,)

Train shape
(4000, 5) (4000,)

Test shape
(1171, 5) (1171,)
Variance score: 0.02
```

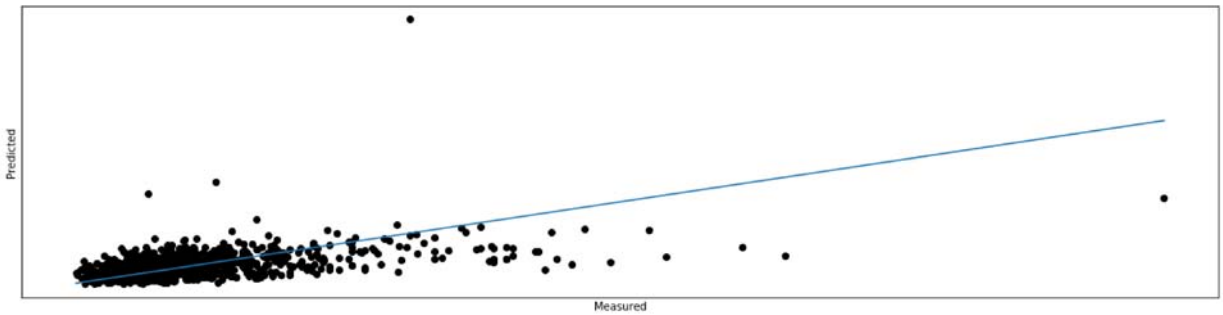


In [53]:
```
#Load tools from sklearn library

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Run Linear Regresssion
# Use feature: 'BEDS', 'BATHS', 'SQFT', 'AGE'
prices=dataset['SOLDPRICE']
features=dataset[['BEDS','BATHS','SQFT','AGE']]
print("Original shape")
print(features.shape, prices.shape)
print()
```

```
Original shape
(5171, 4) (5171,)
```

In [61]:
```
# Split data using the above example
features_train=features[:4000]
features_test=features[4000:]

#Split the targets into training/testing sets
prices_train=prices[:4000]
prices_test=prices[4000:]

regr=linear_model.LinearRegression()

print ("Train shape")
print(features_train.shape, prices_train.shape)
print ()

print ("Test shape")
print (features_test.shape, prices_test.shape)

regr.fit(features_train, prices_train)

#Make predictions using the testing set
regr.fit(features_train, prices_train)
prices_pred=regr.predict(features_test)

#Explain variance score: 1 is perfect prediction
print('Variance Score: %.2f' % r2_score(prices_test, prices_pred))

#Plot outputs
plt.figure(figsize=(20,5))
plt.scatter(prices_test, prices_pred, color='black')
plt.plot([prices_test.min(), prices_test.max()], [prices_test.min(),prices_test.m

plt.ylabel('Predicted')
plt.xlabel('Measured')

plt.xticks(())
plt.yticks(())

plt.show()
```

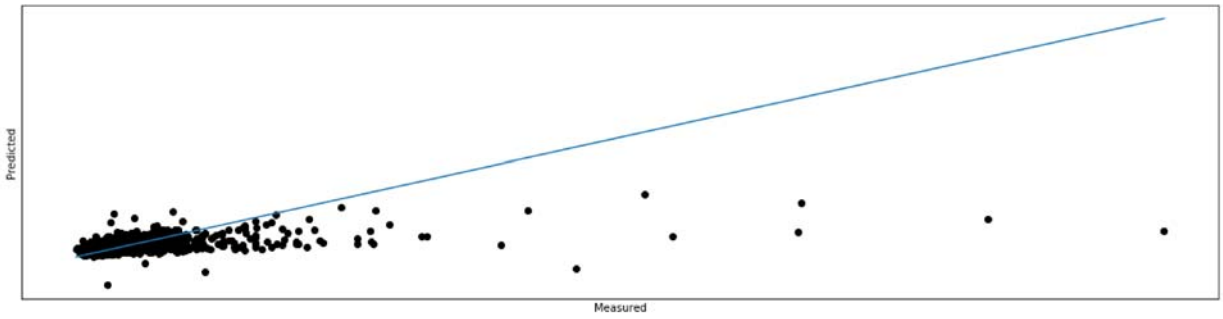
```
Train shape
(4000, 4) (4000,)

Test shape
(1171, 4) (1171,)
Variance Score: 0.02
```



In [71]:
```
target_df=dataset['SOLDPRICE']
features_df=dataset[['BEDS','BATHS','SQFT','AGE','LOTSIZE']]
```

In [84]:
```
# Use sklearn train/test/split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train,y_test = train_test_split(dataset[['BEDS','BATHS','SQFT'

regr.fit(features_train, prices_train)

#Make predictions using the testing set
regr.fit(x_train,y_train)
y_pred=regr.predict(x_test)

#Explain variance score: 1 is perfect prediction
print('Variance Score: %.2f' % r2_score(y_test, y_pred))

#Plot outputs
plt.figure(figsize=(20,5))
plt.scatter(y_test, y_pred, color='black')
plt.plot([y_test.min(), y_test.max()], [y_test.min(),y_test.max()])

plt.ylabel('Predicted')
plt.xlabel('Measured')

plt.xticks(())
plt.yticks(())

plt.show()
```

```
Variance Score: 0.25
```



In [75]:
```
features_train
```

Out[75]:

	BEDS	BATHS	SQFT	AGE	LOTSIZE
3945	3	3.5	1467	119	1690.0
1964	5	2.0	1842	118	4356.0
605	4	2.0	1846	65	10000.0
3046	3	1.0	1226	118	3244.0
4655	7	4.0	3468	98	3684.0
184	3	1.5	1300	33	4438.0
750	4	1.5	2158	58	10018.0
3228	4	3.0	1892	59	10635.0
696	4	2.0	1880	118	9583.0
3237	2	1.5	1116	36	-9999.0
3421	4	3.0	4265	38	43560.0
4838	2	1.0	760	54	-9999.0
2930	2	1.5	2951	67	12593.0
3262	2	1.0	1129	220	-9999.0
2902	3	1.0	1104	118	10000.0
3507	1	1.0	348	45	0.0
2008	4	2.5	2171	46	95396.0
3030	5	2.0	1832	29	8255.0
3388	4	2.0	2354	87	27007.0
2274	3	2.0	1631	119	-9999.0
4798	4	2.0	2499	65	12720.0
1341	3	3.0	2262	24	-9999.0
3485	3	1.0	1056	64	15002.0
465	3	1.0	1333	78	6476.0
4248	3	1.0	1716	58	26943.0
3298	3	2.0	1440	32	5000.0
1589	3	2.5	1632	0	33247.0
84	3	1.0	960	61	10018.0
4058	2	2.5	1901	1	-9999.0
2320	3	2.0	1414	63	18731.0
...	...	...	...	...	...
4504	4	2.5	2624	25	30056.0
1681	2	2.0	1300	12	-9999.0
3755	2	2.0	1192	12	-9999.0

	BEDS	BATHS	SQFT	AGE	LOTSIZE
3210	3	2.0	1533	96	5181.0
243	4	3.5	3225	2	75386.0
2983	3	2.5	1836	38	22178.0
1494	1	1.0	843	1	-9999.0
633	4	2.0	1382	52	30000.0
3489	4	2.0	2500	33	7333.0
4211	4	2.5	2452	100	611147.0
2585	3	2.0	1645	61	10001.0
3647	3	2.0	2402	48	30502.0
3331	4	1.0	1338	92	19642.0
1097	3	3.5	1790	118	-9999.0
2106	5	3.0	2207	98	1380.0
772	4	2.5	3256	23	11646.0
2107	2	2.5	2003	133	217965.0
2349	2	1.0	480	58	4792.0
3950	4	1.5	1536	98	9638.0
329	4	2.5	3068	19	8962.0
1650	4	2.0	1440	25	22893.0
2812	2	2.0	1926	65	88427.0
688	3	2.5	1952	98	15718.0
1883	2	2.0	948	34	-9999.0
3310	2	2.0	1301	1	-9999.0
4746	2	1.0	594	118	-9999.0
3978	4	2.0	1608	58	47916.0
314	4	2.5	2800	59	46500.0
3395	1	1.0	378	48	-9999.0
1793	2	2.0	1600	48	-9999.0

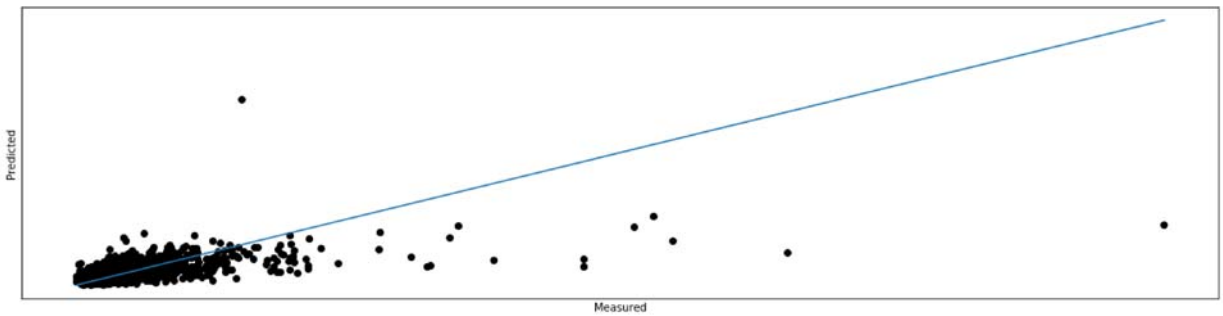
3464 rows × 5 columns

In [90]:
```
from sklearn.model_selection import train_test_split
```

In [92]:

```
# YOUR TURN
# Use categorical variables

# new_dataset_name = pandas_library.get_dummies(dataset, columns=[list_columns_na
df_dummies = pd.get_dummies(dataset, columns=['PROPTYPE'])

features=['BEDS','BATHS','SQFT','LOTSIZE','PROPTYPE_CC', 'PROPTYPE_MF', 'PROPTYPE_
target='SOLDPRICE'
x_train, x_test, y_train,y_test = train_test_split(df_dummies[features],df_dummie

regr.fit(features_train, prices_train)

#Make predictions using the testing set
regr.fit(x_train,y_train)
y_pred=regr.predict(x_test)

#Explain variance score: 1 is perfect prediction
print('Variance Score: %.2f' % r2_score(y_test, y_pred))

#Plot outputs
plt.figure(figsize=(20,5))
plt.scatter(y_test, y_pred, color='black')
plt.plot([y_test.min(), y_test.max()], [y_test.min(),y_test.max()])

plt.ylabel('Predicted')
plt.xlabel('Measured')

plt.xticks(())
plt.yticks(())

plt.show()
```

```
Variance Score: 0.27
```



In [ ]:
```
plt.figure(figsize=(10, 7))

resid = y_test - regr.predict(X_test)

plt.axhline(y=0, linestyle='-', linewidth=2, color="r")
plt.scatter(x=y_pred, y=resid, alpha=0.5, s=3)

plt.title("Residual plot")
plt.ylabel(r"Y-$\hat Y$")
plt.xlabel(r'$\hat Y$')
plt.tight_layout()
plt.show()
```