Submission Number: 1

Group Number: 17

Group Members:

Full Legal Name	Location (Country)	E-Mail Address	Non-Contributing Member (X)
Samyak Jain	India	samyakjain.silverline@gmail.com	
Korede Samuel Olaosun	Nigeria	olaosunkoredesamuel@yahoo.com	

Statement of integrity: By typing the names of all group members in the text box below, you confirm that the assignment submitted is original work produced by the group (*excluding any non-contributing members identified with an "X" above*).

Samyak Jain, Korede Samuel Olaosun

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.

** Note, you may be required to provide proof of your outreach to non-contributing members upon request.*

1.1 What is an Exchange-Traded Fund (ETF)?

An ETF is a collection of assets whose shares are traded on a stock market. They combine the characteristics and potential benefits of stocks, mutual funds, and bonds. ETF shares, like individual stocks, are traded throughout the day at varying prices based on supply and demand.

1.2 Pick 1 of the funds in the data set, and find the weightings. Show in Python table

In [36]:
```
# weighted average for LUXXX

weighted = dframe.groupby("Date")["LUXXX"].mean()
print (weighted)
```

```
Date
1-Apr-16    1460.223
1-Dec-17    1631.147
1-Feb-19    1437.316
1-Jan-16    1390.716
1-Jul-16    1454.413
              ...
9-Jun-17    1721.569
9-Mar-18    1585.330
9-Nov-18    1533.675
9-Oct-20    1075.670
9-Sep-16    1583.787
Name: LUXXX, Length: 252, dtype: float64
```

2.1 Import the data from the csv file

In [164]:
```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
```

In [19]:
```
dframe = pd.read_csv('MScFE 650 MLF GWP Data.csv')
dframe.head()
```

Out[19]:

	Date	LUXXX	MSCI ARGENTINA	BLP ORIENTE MEDIO	MSCI AUSTRALIA	MSCI AUSTRIA	MSCI BELGIUM	MSCI BRAZIL	MSCI CANADA
0	1-Jan-16	1390.716	2376.29	3525.9150	1068.79	106.70	105.38	1036.23	1663.27
1	8-Jan-16	1291.267	2260.85	3280.6683	1005.56	97.66	99.35	952.01	1586.18
2	15-Jan-16	1257.086	2217.50	3118.2981	985.38	93.54	97.32	904.64	1541.08
3	22-Jan-16	1254.167	2281.98	2935.0677	985.87	95.79	100.73	879.17	1582.10
4	29-Jan-16	1298.240	2462.19	3134.0840	1005.56	96.93	103.05	958.97	1638.84

5 rows × 36 columns

3.1 Summarize the min, max, mean, median, and standard deviation of each column

In [21]:
```
dframe.describe()
```

Out[21]:

	LUXXX	MSCI ARGENTINA	BLP ORIENTE MEDIO	MSCI AUSTRALIA	MSCI AUSTRIA	MSCI BELGIUM	MSCI BRAZIL
count	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000	252.000000
mean	1457.231905	2526.652262	3029.482978	1183.940159	127.418889	91.290238	1813.989167
std	238.611226	899.378857	516.678825	101.261295	24.770889	12.106033	354.986912
min	871.500000	844.090000	1722.870000	957.150000	78.290000	57.440000	879.170000
25%	1320.741000	1773.735000	2730.897500	1125.265000	102.820000	86.830000	1589.772500
50%	1491.081000	2541.975000	3113.414050	1177.375000	130.635000	95.015000	1860.960000
75%	1656.015500	3138.222500	3460.390000	1241.797500	146.950000	100.512500	2116.162500
max	1812.010000	4467.410000	3750.865500	1431.460000	177.580000	107.340000	2404.740000

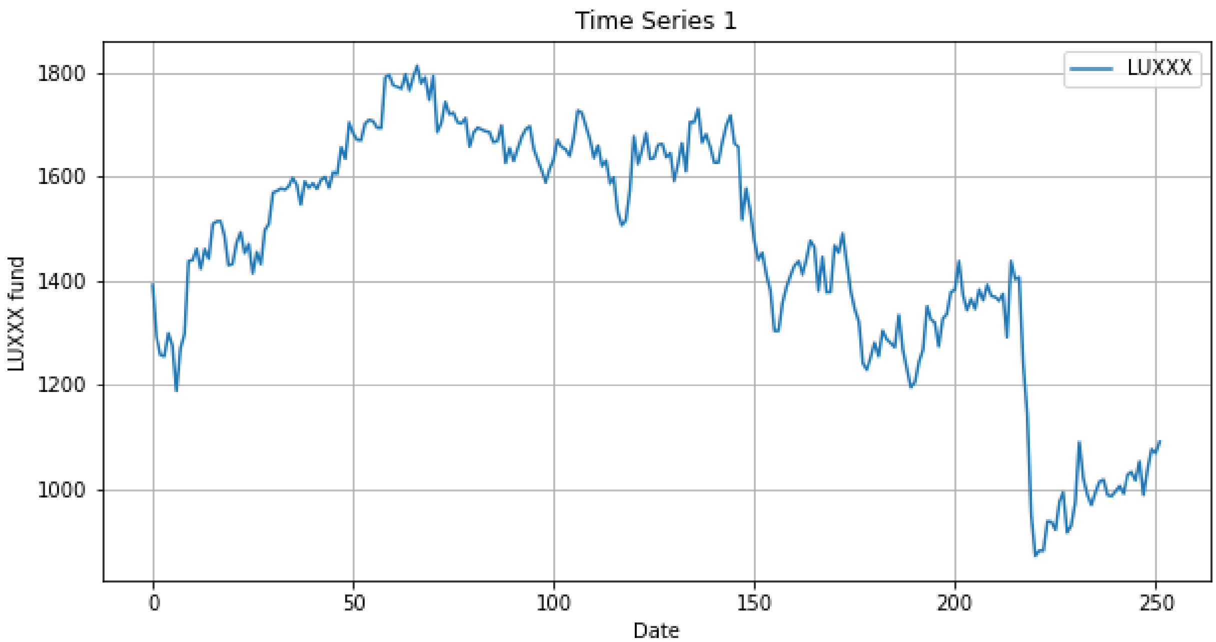
8 rows × 35 columns

4.1 Write a Python function that graphs 1 time series with appropriate time labels

4.1 Write a Python function that graphs 1 time series with appropriate time labels

In [49]:
```
time_series1 = dframe[['LUXXX']]
time_series1.plot(figsize=(10,5), grid=True)
plt.xlabel('Date')
plt.ylabel('LUXXX fund')
plt.title('Time Series 1')
```

Out[49]:
```
Text(0.5, 1.0, 'Time Series 1')
```

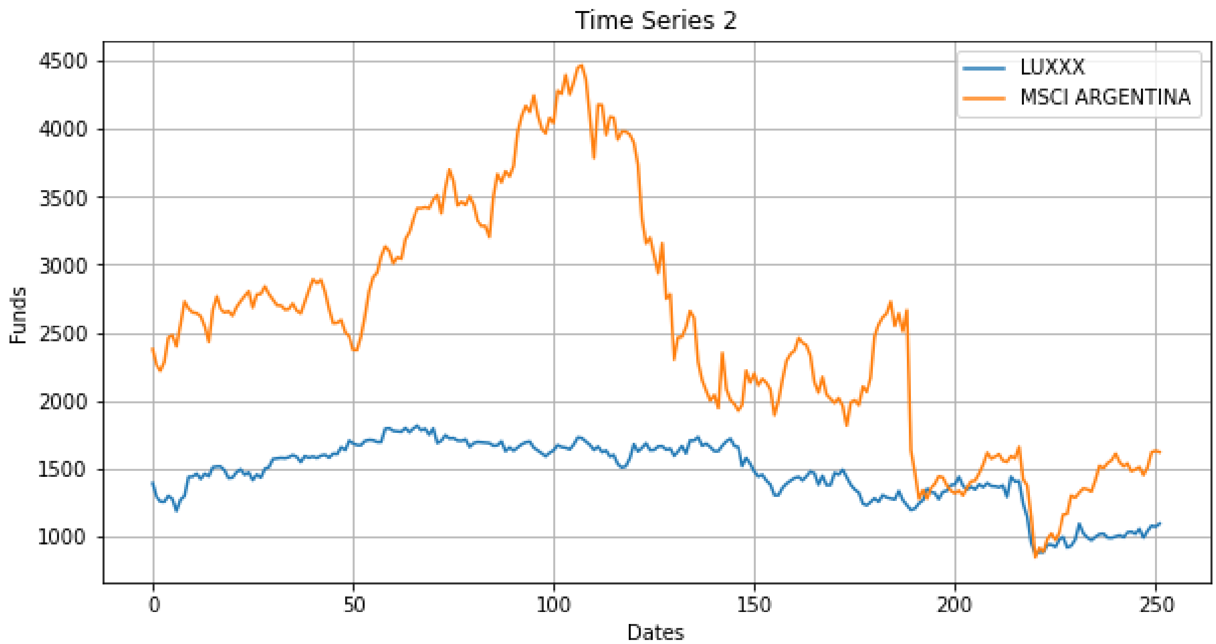


In [ ]:
```
time_series2 = dframe[['LUXXX', 'MSCI ARGENTINA']]
one_series.plot()

two_series=df[['LUXXX','MSCI BRAZIL']]
two_series.plot()

two_series['LUXXX_daily_return']= two_series['LUXXX'].pct_change()*100
two_series['MSCI BRAZIL_daily_return']= two_series['MSCI BRAZIL'].pct_change()*1(
two_series[['LUXXX_daily_return','MSCI BRAZIL_daily_return']].plot()
```

4.2 Write a Python function that graphs 2 time series on the same plot, with labels

In [48]:
```
time_series2 = dframe[['LUXXX', 'MSCI ARGENTINA']]
time_series2.plot(figsize=(10,5), grid=True)
plt.xlabel('Dates')
plt.ylabel('Funds')
plt.title('Time Series 2')
```

Out[48]: Text(0.5, 1.0, 'Time Series 2')

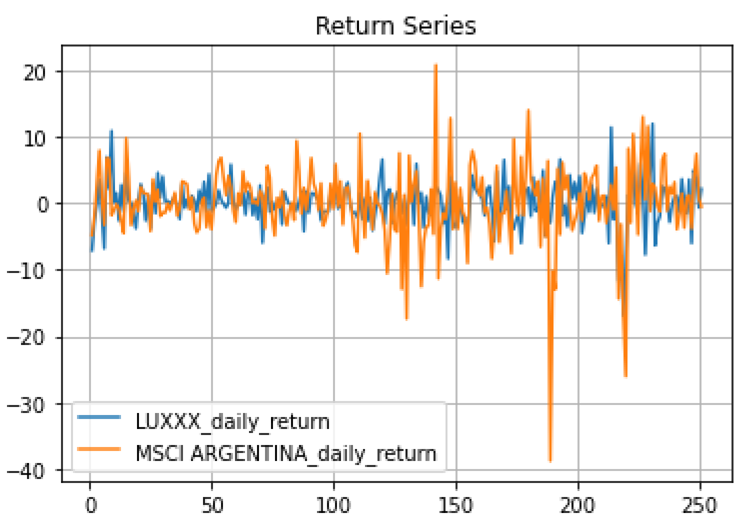


4.3 Write a Python function that compares the 2 return series

In [54]:
```
time_series2['LUXXX_daily_return']= time_series2['LUXXX'].pct_change()*100
time_series2['MSCI ARGENTINA_daily_return']= time_series2['MSCI ARGENTINA'].pct_c
time_series2[['LUXXX_daily_return','MSCI ARGENTINA_daily_return']].plot(grid=True
plt.title('Return Series')
```

```
/home/fabulouskorex/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.p
y:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  """Entry point for launching an IPython kernel.
/home/fabulouskorex/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.p
y:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

Out[54]: Text(0.5, 1.0, 'Return Series')



5.1 Compute the correlation using Pearson correlation

In [56]:
```
pearson_corr = dframe.corr(method = 'pearson')
pearson_corr.head()
```

Out[56]:

	LUXXX	MSCI ARGENTINA	BLP ORIENTE MEDIO	MSCI AUSTRALIA	MSCI AUSTRIA	MSCI BELGIUM	MSCI BRAZIL	MS CANA
LUXXX	1.000000	0.754584	0.823881	0.046895	0.674894	0.850111	0.337350	0.133
MSCI ARGENTINA	0.754584	1.000000	0.740835	-0.160676	0.604493	0.747766	0.260126	-0.047
BLP ORIENTE MEDIO	0.823881	0.740835	1.000000	-0.124534	0.387526	0.925401	0.144240	-0.114
MSCI AUSTRALIA	0.046895	-0.160676	-0.124534	1.000000	0.516601	-0.027924	0.755042	0.916
MSCI AUSTRIA	0.674894	0.604493	0.387526	0.516601	1.000000	0.495019	0.731040	0.562

5 rows × 35 columns

5.2 Recompute the calculation, instead of using Spearman correlation

In [59]:
```
spearman_corr = dframe.corr(method = 'spearman')
spearman_corr.head()
```

Out[59]:

	LUXXX	MSCI ARGENTINA	BLP ORIENTE MEDIO	MSCI AUSTRALIA	MSCI AUSTRIA	MSCI BELGIUM	MSCI BRAZIL	MS CANA
LUXXX	1.000000	0.760165	0.763908	0.008914	0.630077	0.728380	0.171258	0.014
MSCI ARGENTINA	0.760165	1.000000	0.762681	-0.170967	0.543215	0.815686	0.189584	-0.174
BLP ORIENTE MEDIO	0.763908	0.762681	1.000000	-0.225116	0.305152	0.881598	0.022868	-0.222
MSCI AUSTRALIA	0.008914	-0.170967	-0.225116	1.000000	0.522231	-0.222302	0.702757	0.936
MSCI AUSTRIA	0.630077	0.543215	0.305152	0.522231	1.000000	0.368606	0.674107	0.472

5 rows × 35 columns

5.3 Recompute the calculation, instead of using Kendall correlation

In [60]:
```
kendall_corr = dframe.corr(method = 'kendall')
kendall_corr.head()
```

Out[60]:

	LUXXX	MSCI ARGENTINA	BLP ORIENTE MEDIO	MSCI AUSTRALIA	MSCI AUSTRIA	MSCI BELGIUM	MSCI BRAZIL	MSCI CANA
LUXXX	1.000000	0.558401	0.580535	0.026813	0.457345	0.533548	0.110099	0.052
MSCI ARGENTINA	0.558401	1.000000	0.547587	-0.091444	0.381774	0.600645	0.158351	-0.084
BLP ORIENTE MEDIO	0.580535	0.547587	1.000000	-0.117372	0.233858	0.704484	0.030228	-0.115
MSCI AUSTRALIA	0.026813	-0.091444	-0.117372	1.000000	0.340669	-0.131601	0.522165	0.784
MSCI AUSTRIA	0.457345	0.381774	0.233858	0.340669	1.000000	0.273319	0.485170	0.324

5 rows × 35 columns

6.1 Choose one of the 35 variables to serve as your response variable (e.g. LUXXX)

In [142]:
```
# We chose LUXXX as our response variable
Y = dframe.LUXXX
Y
```

Out[142]:
```
0      1390.716
1      1291.267
2      1257.086
3      1254.167
4      1298.240
         ...
247     988.345
248    1037.211
249    1075.670
250    1068.089
251    1090.573
Name: LUXXX, Length: 252, dtype: float64
```

7.1 Use the Pearson correlation matrix

In [145]:
```
# We chose LUXXX as our response variable

X=dframe.copy()
X.drop(columns=['Date', 'LUXXX'], inplace=True)
X.shape
```

Out[145]:
```
(252, 34)
```

7.2 Show the amount of variation explained by the first 5 components

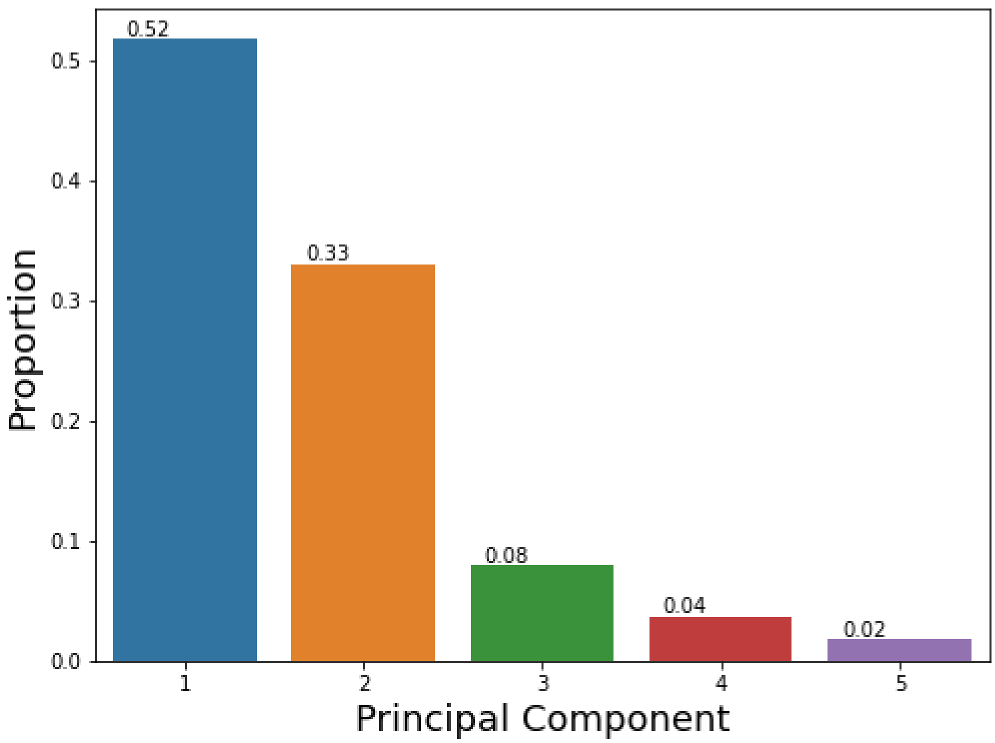
In [153]:
```
from sklearn.preprocessing import StandardScaler
X_normalized=StandardScaler().fit_transform(X)


pca = PCA(n_components=5)
pca.fit(X)

print("Percentage of variance explained by each of the selected components:")
print(pca.explained_variance_ratio_)
print("Total variance explained by the first 5 components")
print(pca.explained_variance_ratio_.sum())
```

```
Percentage of variance explained by each of the selected components:
[0.51756363 0.33009646 0.07894327 0.03678651 0.01779009]
Total variance explained by the first 5 components
0.9811799501538664
```

7.3 How many components are needed to express 80% of the variation of the data?

In [151]:

```
import seaborn as sns
dset3 = pd.DataFrame()
dset3['pca'] = range(1,6)
dset3['vari'] = pd.DataFrame(pca.explained_variance_ratio_)
plt.figure(figsize=(8,6))
graph = sns.barplot(x='pca', y='vari', data=dset3)
for p in graph.patches:
    graph.annotate('{:.2f}'.format(p.get_height()), (p.get_x()+0.2, p.get_height(
                   ha='center', va='bottom',
                   color= 'black')
plt.ylabel('Proportion', fontsize=18)
plt.xlabel('Principal Component', fontsize=18)
plt.show()
```



The first two components expressed 80% variation of the data.

In [ ]:

```
Show the amount of variation explained by the first 5 components
How many components are needed to express 80% of the variation of the data?
What is your interpretation of the 1st component?
```

7.4 What is your interpretation of the 1st component?

The first component accounts for more than half of the variance. This indicates that the data is widely disseminated or spread.

Q8. We will use PCA and the Lasso

regression

What do these 2 methodologies have in common? How do they differ?

PCA can be used as a dimensionality reduction technique if you drop Principal Components based on a heuristic, but it offers no feature selection, as the Principal Components are retained instead of the original features. However, tuning the number of Principal Components retained should work better than using heuristics, unless there are many low variance components and you are simply interested in filtering them.

LASSO on the other hand can, intrinsically, perform feature selection as the coefficients of predictors are shrunk towards zero. It still requires hyperparameter tuning because there's a regularization coefficient that weights how severe is the regularization of the loss function.

9.1 Run a regression of Y versus the PCA scores

In [154]:
```
X_r = pca.fit(X_normalized).transform(X_normalized)
X_r.shape
```

Out[154]: (252, 5)

In [158]:
```
# Regression score
regr_PCA = LinearRegression()
regr_PCA.fit(X_r, Y)
print("Regression score")
regr_PCA.score(X_r, Y)
```

```
Regression score
```

Out[158]: 0.8737184106725846

10.1 Run a linear regression of Y versus the other predictors

In [160]:
```
regres = LinearRegression()
regres.fit(X, Y)
print("Regression score")
regres.score(X, Y)
```

```
Regression score
```

Out[160]: 0.9806055317363801

10.2 For the lasso, use at least 1000 different values of the penalty parameter

In [161]:
```
# We created 1000 alphas
alphas = np.arange(0.001, 1.001, 0.001)
len(alphas)
```

Out[161]: 1000

10.3 Split the data into testing and training, with 2/3 for training and 1/3 for testing

In [163]:
```
# Split testing and training
X_train, X_test, Y_train, Y_test = train_test_split(X_normalized, Y, test_size=1/
```

10.4 Graph the overall model mismatch for each of the 1000 values of the paramter

In [165]:
```
# Fit Lasso model for each alpha value

test_errors=[]

for alpha in alphas:

    model=Lasso(alpha=alpha)
    model.fit(X_train, Y_train)

    Y_test_predict = model.predict(X_test)
    test_errors.append(mean_squared_error(Y_test, Y_test_predict))
```

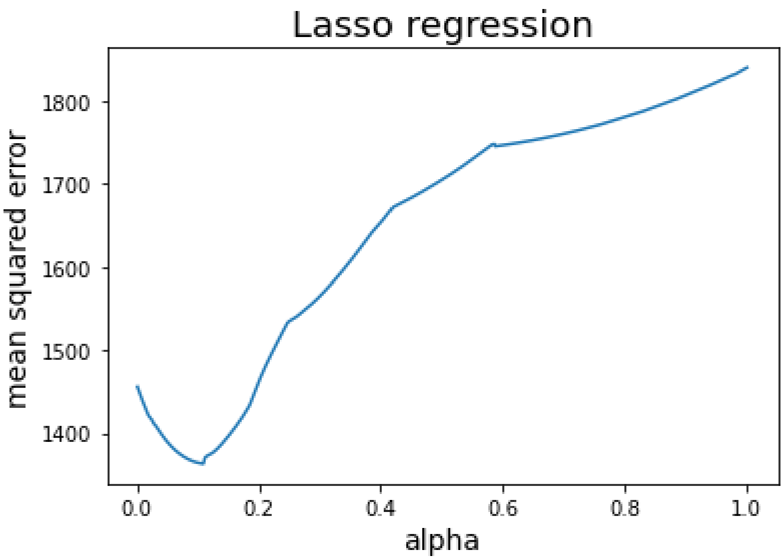
```
/home/fabulouskorex/anaconda3/lib/python3.7/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:532: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations. Duality gap: 91909.15532
545005, tolerance: 948.460388262328
  positive)
/home/fabulouskorex/anaconda3/lib/python3.7/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:532: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations. Duality gap: 90881.49340
177038, tolerance: 948.460388262328
  positive)
/home/fabulouskorex/anaconda3/lib/python3.7/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:532: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations. Duality gap: 89825.58527
082701, tolerance: 948.460388262328
  positive)
/home/fabulouskorex/anaconda3/lib/python3.7/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:532: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations. Duality gap: 88740.09877
728231, tolerance: 948.460388262328
```

In [167]:
```
print("lowest mean squared error on test data")
alphas[np.argmin(test_errors)]
```

```
lowest mean squared error on test data
```

Out[167]:
```
0.109
```

In [175]:
```
plt.plot(alphas, test_errors)
plt.xlabel("alpha", fontsize=14)
plt.ylabel("mean squared error", fontsize=14)
plt.title("Lasso regression", fontsize=18);
```



10.5 Find a lasso model that includes no more than 7 predictors

In [203]:
```
best_reg=Lasso(alpha=alphas[np.argmin(test_errors)])
best_reg.fit(X_train, Y_train)
best_features = dframe.iloc[:,2:].columns[best_coef !=0]
best_model = pd.DataFrame({"Features":best_features, "Coefficients":best_reg.coe
best_model
```

```
/home/fabulouskorex/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/
_coordinate_descent.py:532: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations. Duality gap: 28595.91092284767
4, tolerance: 948.460388262328
  positive)
```

Out[203]:

	Features	Coefficients
0	MSCI ARGENTINA	-46.734333
1	BLP ORIENTE MEDIO	3.914027
2	MSCI AUSTRALIA	-5.195193
3	MSCI AUSTRIA	25.336341
4	MSCI BELGIUM	41.095624
5	MSCI BRAZIL	0.000000
6	MSCI CANADA	61.697362
7	MSCI CHINA	-0.000000
8	MSCI DENMARK	27.490842
9	MSCI EM ASIA	50.657815
10	MSCI EM EU-MIDE-AFRICA	37.575869
11	MSCI EM EUROPE	-34.281659
12	MSCI EM LATIN AMERICA	27.787135
13	MSCI FINLAND	5.363932
14	MSCI FRANCE	-23.046115
15	MSCI GERMANY	84.754895
16	MSCI HONG KONG	-43.734658
17	MSCI INDIA	-54.785163
18	MSCI ITALY	-84.246710
19	MSCI JAPAN	36.568381
20	MSCI KOREA	-38.721027
21	MSCI MEXICO	46.084003
22	MSCI NETHERLANDS	-66.656007
23	MSCI NEW ZEALAND	14.391293
24	MSCI NORWAY	1.451506
25	MSCI PERU	48.636250
26	MSCI RUSSIA	0.184021
27	MSCI SINGAPORE	27.853924

	Features	Coefficients
28	MSCI SOUTH AFRICA	-34.875563
29	MSCI SPAIN	95.604993
30	MSCI SWEDEN	35.520793
31	MSCI SWITZERLAND	-95.979310
32	MSCI UK	0.000000
33	MSCI USA	1.200671

In [197]:
```
# Get absolute value as importance
coefficients=best_reg.coef_
importance = np.abs(coefficients)
print("Importance")
importance
```

```
Importance
```

Out[197]:
```
array([46.73433304,  3.914027  ,  5.19519264, 25.33634115, 41.09562389,
        0.        , 61.69736167,  0.        , 27.4908418 , 50.65781512,
       37.5758689 , 34.28165884, 27.78713479,  5.36393217, 23.04611511,
       84.75489539, 43.73465793, 54.78516346, 84.24671033, 36.56838114,
       38.7210266 , 46.08400299, 66.65600722, 14.39129275,  1.45150641,
       48.63625012,  0.18402056, 27.8539243 , 34.87556272, 95.60499348,
       35.52079257, 95.97931032,  0.        ,  1.20067103])
```

In [198]:
```
# We sort and select the top 7 predictors
features_importance = {'Features':X.columns,'Importance':importance}

df_features=pd.DataFrame(features_importance)
selected_7_predictors=df_features.sort_values(by=['Importance'], ascending=False)
print("Selected predictors")
selected_7_predictors
```

```
Selected predictors
```

Out[198]:
```
array(['MSCI SWITZERLAND', 'MSCI SPAIN', 'MSCI GERMANY', 'MSCI ITALY',
       'MSCI NETHERLANDS', 'MSCI CANADA', 'MSCI INDIA'], dtype=object)
```

In [202]:
```
# columns selection for predictors
X_selected=X[selected_7_predictors]
X_selected.shape
```

Out[202]:
```
(252, 7)
```

In [201]:
```
# fitting of the final model to the selected predictors
X_train_selected, X_test_selected, Y_train, Y_test = train_test_split(X_selected,
best_reg_selected=Lasso(alpha=alphas[np.argmin(test_errors)])
best_reg_selected.fit(X_train_selected, Y_train)
print("The seven coefficients of the model:")
best_reg_selected.coef_
```

```
The seven coefficients of the model are:
```

Out[201]:
```
array([-1.0143132 , 10.15652002,  7.64310851, -8.72945924, -4.35986131,
        0.6617353 , -0.03532224])
```

11. Which model provides a better fit to the data and why?

PCA provides a better fit to the data due to its dimensionality reduction technique. PCA, while reducing the number of features, does not care about the interpretability of features. The only thing that it cares about is preserving the maximum variance, thereby resulting in a better fit.

12. Which model provides better interpretation of the results?

Lasso provides more interpretability of results and performs feature selection, as compared to PCA. PCA while performing the dimensionality reduction in regression ignores the relationship between X and Y variables. Therefore, dropping low variance components while ignoring their relationship to Y loses interpretability.

13. How did your group divide the work?

Unfortunately, we were left with just two members in the group as one of them un-enrolled. Thus, we both did whatever we could and collated our respective parts at the end. One person was more responsible for the theory and basic questions, while the other did the entire regression and PCA parts.