NATIONAL INSTITUTE OF TECHNOLOGY CALICUT

EVS PROJECT : WEATHER FORECASTING USING ML

Done by,

P Arjun (B180454CS)

Abhimanyu M R (B180325CS)

Thanzeel Hassan (B180322CS)

Bharat Teja (B180953CS)

Eric Roshan Toppo (B180977CS)

WEATHER PREDICTION

Weather forecasting is the application of science and technology to predict the conditions of the atmosphere for a given location and time. People have attempted to predict the weather informally for millennia and formally since the 19th century. Weather forecasts are made by collecting quantitative data about the current state of the atmosphere at a given place and using meteorology to project how the atmosphere will change.

Once calculated by hand based mainly upon changes in barometric pressure, current weather conditions, and sky condition or cloud cover, weather forecasting now relies on computer-based models that take many atmospheric factors into account. Human input is still required to pick the best possible forecast model to base the forecast upon, which involves pattern recognition skills, teleconnections, knowledge of model performance, and knowledge of model biases. The inaccuracy of forecasting is due to the chaotic nature of the atmosphere, the massive computational power required to solve the equations that describe the atmosphere, the error involved in measuring the initial conditions, and an incomplete understanding of atmospheric processes. Hence, forecasts become less accurate as the difference between current time and the time for which the forecast is being made (the range of the forecast) increases. The use of ensembles and model consensus help narrow the error and pick the most likely outcome.

There is a vast variety of end uses to weather forecasts. Weather warnings are important forecasts because they are used to protect life and property. Forecasts based on temperature and precipitation are important to agriculture, and therefore to traders within commodity markets. Temperature forecasts are used by utility companies to estimate demand over coming days. On an everyday basis, many use weather forecasts to determine what to wear on a given day. Since outdoor activities are severely curtailed by heavy rain, snow and wind chill, forecasts can be used to plan activities around these events, and to plan ahead and survive them. In 2009, the US spent approximately $5.1 billion on weather forecasting.

INTELLIGENT WEATHER PREDICTIONS

Technological advancements in the 21st century have brought many improvements to weather forecasting. The growth of smartphones has brought on-the-go weather forecasting to billions of people around the world, while the location data of the devices improves the accuracy of forecasting. Another recent development, the AI revolution, has not spared weather prediction either. Developments in machine learning mean that AI can be incorporated into existing weather models to produce even more accurate forecasts. Machine learning models for weather forecasting quickly process large amounts of weather data, and they can compare data from weather stations and satellites with traditional forecasts to make highly accurate predictions.

Machine learning for more accurate forecasting

One of the main benefits of introducing machine learning to weather forecasting is more accurate predictions. Machine learning can be used to process immediate comparisons between historical weather forecasts and observations. With the use of machine learning, weather models can better account for prediction inaccuracies, such as overestimated rainfall, and produce more accurate predictions.

Expanding nowcasting with deep learning

Aside from more accurate forecasts, machine learning can also be used to improve nowcasting, which is immediate weather prediction, typically within two hours, that provides minute-by-minute precipitation forecasts. While nowcasting is technically possible through traditional forecasting using radar data, weather models based on machine learning can also take into account data from weather satellites. Integrating machine learning into weather models enables them to quickly process satellite images for nowcasting. Adding weather satellites to the tech behind nowcasting greatly expands its reach. With machine learning, potentially anyone in range of a weather satellite can use nowcasting, rather than just those living near a radar station.

jupyter Untitled Last Checkpoint: 3 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3

Run Markdown

SMARTPHONES

Aside from the introduction of AI, weather forecasting has changed with another recent technological innovation; the smartphone. People with smartphones can access detailed weather reports wherever they are, and these forecasts are more accurate thanks to their devices. Weather predictions can now take into account the specific location data from smartphones to provide users with hyperlocal forecasting. Since a city's particular weather can vary dramatically from one block to the next, the use of location data from smartphones has significantly advanced forecasting for each user and exact locations.

The future of weather forecasting

The last few decades have been transformative for the advancement of weather forecasting. Looking ahead, weather modelling stands to grow even more accurate for a greater number of people around the world.

As machine learning advances and more weather models start integrating it, weather forecasting will become increasingly accurate. There is also excellent potential for a global expansion of nowcasting, a relatively recent addition to consumer weather forecasting. Only a select few weather services include nowcasting in their forecasts, and, in the past, the tech was limited to people in areas with reliable radar coverage.

EXPLAINING LINEAR REGRESSION METHOD ¶

In simple linear regression a single independent variable is used to predict the value of a dependent variable. In multiple linear regression two or more independent variables are used to predict the value of a dependent variable. The difference between the two is the number of independent variables.

The most common method for fitting a regression line is the method of least-squares. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). Because the deviations are first squared, then summed, there are no cancellations between positive and negative values.

In [1]:

```
from statistics import mean
import numpy as np
import pandas as pd
import random
import math
import matplotlib.pyplot as plt
color='#003F72'
```

In [2]:

```
def abss(x):
    if x>=0: return x
    else: return -1*x
```

In [17]:

```
hours=[2,4,6,7,8,10,15]
marks=[30,41,60,67,73,86,97]

x=np.array(hours)
y=np.array(marks)
```

In [19]:

```
print(x)
print(y)
```

```
[ 2  4  6  7  8 10 15]
[30 41 60 67 73 86 97]
```

jupyter Linear Regression Toy (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

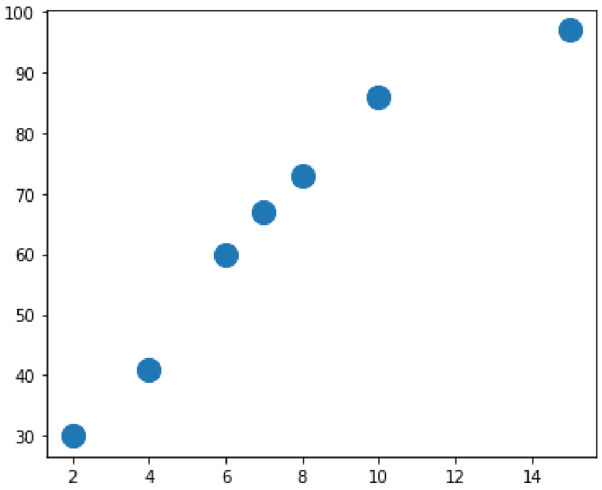
```
x=np.array(hours)
y=np.array(marks)
```

In [19]:
```
print(x)
print(y)
```

```
[ 2  4  6  7  8 10 15]
[30 41 60 67 73 86 97]
```

In [22]:
```
plt.figure(figsize=(6,5))
plt.scatter(x,y,s=200)
```

Out[22]: <matplotlib.collections.PathCollection at 0x234476f9ef0>



In [10]:
```
def best_fit_slope_and_intercept(xs,ys):
    slope=((mean(xs)*mean(ys))-mean(xs*ys))/((mean(xs)**2)-mean(xs**2))
    intercept=mean(ys)-slope*mean(xs)
    return slope,intercept
```

jupyter Linear Regression Toy (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3

Run Markdown

In [10]:
```
def best_fit_slope_and_intercept(xs,ys):
    slope=((mean(xs)*mean(ys))-mean(xs*ys))/((mean(xs)**2)-mean(xs**2))
    intercept=mean(ys)-slope*mean(xs)
    return slope,intercept
```

In [11]:
```
m,b=best_fit_slope_and_intercept(x,y)
print (m,b)
```

```
5.523809523809524 25.333333333333336
```

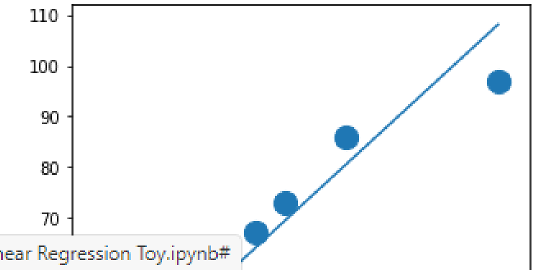
In [23]:
```
#calculating the predicted values using the newly found slope and intercept

regression_line=[]
for xi in x:
    regression_line.append((m*xi)+b)
print (regression_line)
```

```
[36.38095238095238, 47.42857142857143, 58.476190476190474, 64.0, 69.52380952380952, 80.57142857142858, 108.1904761904762]
```

In [24]:
```
plt.figure(figsize=(5,5))
#plt.plot(x,y,color='#004F72')
plt.plot(x,regression_line)
plt.scatter(x,y,s=200)
plt.show
```

Out[24]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



localhost:8888/notebooks/Downloads/Linear Regression Toy.ipynb#

jupyter Linear Regression Toy (autosaved)

Logout

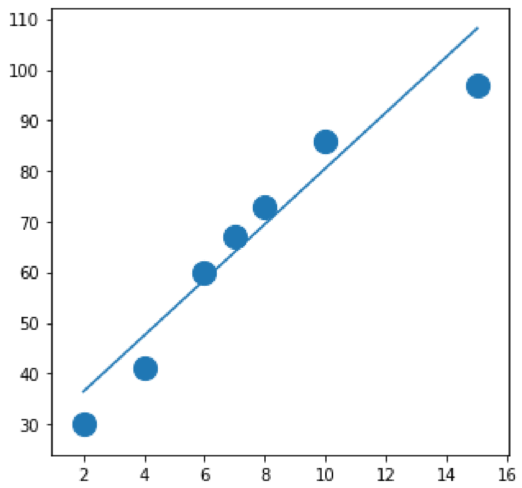
File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3

Run Markdown

Out[24]: <function matplotlib.pyplot.show(*args, **kw)>



In [25]:
```
#Accuracy of the model

correct=0
for i in range(len(x)):
    predict=(m*x[i])+b
    if abss(y[i]-predict)<5:
        correct+=1
accuracy=float(correct)/float(len(x))*100
print (accuracy)
```

42.857142857142854

In [26]:
```
print("%s correct predictions out of %s predictions" % (correct, len(y)))
```

3 correct predictions out of 7 predictions

In [ ]:

COLLECTING WEATHER DATA

In [1]:
```
import os
import pickle
import time
from collections import namedtuple
from datetime import datetime, timedelta

import pandas as pd
import requests

import matplotlib.pyplot as plt
from pyprind import ProgBar

%matplotlib inline
```

In [2]:
```
API_KEY = os.environ.get('MY_API_KEY')
BASE_URL = 'http://api.wunderground.com/api/{}/history_{}/q/TX/Round_Rock.json'
```

In [3]:
```
features = [
    "date", "meantempm", "meandewptm", "meanpressurem", "maxhumidity",
    "minhumidity", "maxtempm", "mintempm", "maxdewptm", "mindewptm",
    "maxpressurem", "minpressurem", "precipm"
]
DailySummary = namedtuple('DailySummary', features)
```

In [4]:
```
def extract_weather_data(url, api_key, target_date, days):
    """Call Wunderground API to extract weather data."""
    records = []
    bar = ProgBar(days)
    for _ in range(days):
        request = BASE_URL.format(API_KEY, target_date.strftime('%Y%m%d'))
        response = requests.get(request)
        if response.status_code == 200:
            data = response.json()['history']['dailysummary'][0]
            records.append(DailySummary(
                date=target_date,
                meantempm=data['meantempm'],
                meandewptm=data['meandewptm'],
                meanpressurem=data['meanpressurem'],
                maxhumidity=data['maxhumidity'],
                minhumidity=data['minhumidity'],
                maxtempm=data['maxtempm'],
                mintempm=data['mintempm'],
                maxdewptm=data['maxdewptm'],
                mindewptm=data['mindewptm'],
```

jupyter Collect Weather Data API Last Checkpoint: 03/12/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3

Run Markdown

In [4]:

```
def extract_weather_data(url, api_key, target_date, days):
    """Call Wunderground API to extract weather data."""
    records = []
    bar = ProgBar(days)
    for _ in range(days):
        request = BASE_URL.format(API_KEY, target_date.strftime('%Y%m%d'))
        response = requests.get(request)
        if response.status_code == 200:
            data = response.json()['history']['dailysummary'][0]
            records.append(DailySummary(
                date=target_date,
                meantempm=data['meantempm'],
                meandewptm=data['meandewptm'],
                meanpressurem=data['meanpressurem'],
                maxhumidity=data['maxhumidity'],
                minhumidity=data['minhumidity'],
                maxtempm=data['maxtempm'],
                mintempm=data['mintempm'],
                maxdewptm=data['maxdewptm'],
                mindewptm=data['mindewptm'],
                maxpressurem=data['maxpressurem'],
                minpressurem=data['minpressurem'],
                precipm=data['precipm']))
        time.sleep(6)
        bar.update()
        target_date += timedelta(days=1)
    return records
```

In [5]:

```
# Do not run this cell when collecting data on day 2
def get_target_date():
    """Return target date 1000 days prior to current date."""
    current_date = datetime.now()
    target_date = current_date - timedelta(days=1000)
    return target_date

target_date = get_target_date()
```

In [6]:

```
records = extract_weather_data(BASE_URL, API_KEY, target_date, 500)
```

```
0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:53:56
```

In [7]:

```
# Look at first five records
records[:5]
```

Out[7]:

```
[DailySummary(date=datetime.datetime(2015, 10, 3, 22, 13, 6, 559948), meantempm='21', meandewptm='6', meanpressurem='1012', max
humidity='63', minhumidity='20', maxtempm='29', mintempm='14', maxdewptm='8', mindewptm='4', maxpressurem='1014', minpressurem
='1010', precipm='0.00'),
 DailySummary(date=datetime.datetime(2015, 10, 4, 22, 13, 6, 559948), meantempm='22', meandewptm='8', meanpressurem='1015', max
humidity='63', minhumidity='25', maxtempm='29', mintempm='15', maxdewptm='10', mindewptm='7', maxpressurem='1017', minpressurem
='1012', precipm='0.00'),
```

In [7]:
```
# Look at first five records
records[:5]
```

Out[7]:
```
[DailySummary(date=datetime.datetime(2015, 10, 3, 22, 13, 6, 559948), meantempm='21', meandewptm='6', meanpressurem='1012', maxhumidity='63', minhumidity='20', maxtempm='29', mintempm='14', maxdewptm='8', mindewptm='4', maxpressurem='1014', minpressurem='1010', precipm='0.00'),
 DailySummary(date=datetime.datetime(2015, 10, 4, 22, 13, 6, 559948), meantempm='22', meandewptm='8', meanpressurem='1015', maxhumidity='63', minhumidity='25', maxtempm='29', mintempm='15', maxdewptm='10', mindewptm='7', maxpressurem='1017', minpressurem='1013', precipm='0.00'),
 DailySummary(date=datetime.datetime(2015, 10, 5, 22, 13, 6, 559948), meantempm='24', meandewptm='11', meanpressurem='1018', maxhumidity='64', minhumidity='35', maxtempm='29', mintempm='19', maxdewptm='13', mindewptm='8', maxpressurem='1020', minpressurem='1015', precipm='0.00'),
 DailySummary(date=datetime.datetime(2015, 10, 6, 22, 13, 6, 559948), meantempm='23', meandewptm='11', meanpressurem='1019', maxhumidity='73', minhumidity='25', maxtempm='30', mintempm='17', maxdewptm='14', mindewptm='8', maxpressurem='1022', minpressurem='1017', precipm='0.00'),
 DailySummary(date=datetime.datetime(2015, 10, 7, 22, 13, 6, 559948), meantempm='24', meandewptm='13', meanpressurem='1017', maxhumidity='72', minhumidity='31', maxtempm='32', mintempm='17', maxdewptm='16', mindewptm='10', maxpressurem='1020', minpressurem='1015', precipm='0.00')]
```

In [8]:
```
len(records)
```

Out[8]:
```
500
```

In [9]:
```
# save records list
with open('records_pt1.pkl', 'wb') as f:
    pickle.dump(records, f)
```

In [5]:
```
# load records list - still need to run cells 1-4
with open('records_pt1.pkl', 'rb') as fp:
    records = pickle.load(fp)
```

In [6]:
```
# Inspect last record to date; next target date should be plus one day
records[-1]
```

Out[6]:
```
DailySummary(date=datetime.datetime(2017, 2, 13, 22, 13, 6, 559948), meantempm='20', meandewptm='13', meanpressurem='1018', maxhumidity='94', minhumidity='42', maxtempm='25', mintempm='16', maxdewptm='18', mindewptm='5', maxpressurem='1022', minpressurem='1012', precipm='0.00')
```

In [7]:
```
# set new target date based on date above plus one day
target_date = datetime(2017, 2, 14)
```

In [8]:
```
records += extract_weather_data(BASE_URL, API_KEY, target_date, 500)
```

```
0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:53:38
```

In [9]:
```
len(records)
```

In [9]:
```
len(records)
```
Out[9]:
```
1000
```
In [10]:
```
# with open('records_pt2.pkl', 'wb') as f:
#     pickle.dump(records, f)
```
In [11]:
```
# load records list - still need to run cells 1 and 3
# with open('records_pt2.pkl', 'rb') as fp:
#     records = pickle.load(fp)
```
In [12]:
```
df = pd.DataFrame(records, columns=features).set_index('date')
```
In [13]:
```
tmp = df[['meantempm', 'meandewptm']].head(10)
tmp
```
Out[13]:

date	meantempm	meandewptm
2015-10-03 22:13:06.559948	21	6
2015-10-04 22:13:06.559948	22	8
2015-10-05 22:13:06.559948	24	11
2015-10-06 22:13:06.559948	23	11
2015-10-07 22:13:06.559948	24	13
2015-10-08 22:13:06.559948	26	17
2015-10-09 22:13:06.559948	26	17
2015-10-10 22:13:06.559948	24	14
2015-10-11 22:13:06.559948	26	16
2015-10-12 22:13:06.559948	28	19

In [14]:
```
# 1 day prior
N = 1

# target measurement of mean temperature
feature = 'meantempm'

# total number of rows
rows = tmp.shape[0]

# a list representing Nth prior measurements of feature
nth_prior_measurements = tmp[feature].shift(periods=N)

# makee a new column name of feature_N and add to DataFrame
col_name = f'{feature}_{N}'
```

2015-10-04 22:13:06.559948	22	8	21
2015-10-05 22:13:06.559948	24	11	22
2015-10-06 22:13:06.559948	23	11	24
2015-10-07 22:13:06.559948	24	13	23
2015-10-08 22:13:06.559948	26	17	24
2015-10-09 22:13:06.559948	26	17	26
2015-10-10 22:13:06.559948	24	14	26
2015-10-11 22:13:06.559948	26	16	24
2015-10-12 22:13:06.559948	28	19	26

In [15]:
```
def derive_nth_day_feature(df, feature, N):
    nth_prior_measurements = df[feature].shift(periods=N)
    col_name = f'{feature}_{N}'
    df[col_name] = nth_prior_measurements
```

In [16]:
```
for feature in features:
    if feature != 'date':
        for N in range(1, 4):
            derive_nth_day_feature(df, feature, N)
```

In [17]:
```
df.columns
```

Out[17]:
```
Index(['meantempm', 'meandewptm', 'meanpressurem', 'maxhumidity',
       'minhumidity', 'maxtempm', 'mintempm', 'maxdewptm', 'mindewptm',
       'maxpressurem', 'minpressurem', 'precipm', 'meantempm_1', 'meantempm_2',
       'meantempm_3', 'meandewptm_1', 'meandewptm_2', 'meandewptm_3',
       'meanpressurem_1', 'meanpressurem_2', 'meanpressurem_3',
       'maxhumidity_1', 'maxhumidity_2', 'maxhumidity_3', 'minhumidity_1',
       'minhumidity_2', 'minhumidity_3', 'maxtempm_1', 'maxtempm_2',
       'maxtempm_3', 'mintempm_1', 'mintempm_2', 'mintempm_3', 'maxdewptm_1',
       'maxdewptm_2', 'maxdewptm_3', 'mindewptm_1', 'mindewptm_2',
       'mindewptm_3', 'maxpressurem_1', 'maxpressurem_2', 'maxpressurem_3',
       'minpressurem_1', 'minpressurem_2', 'minpressurem_3', 'precipm_1',
       'precipm_2', 'precipm_3'],
      dtype='object')
```

In [18]:
```
# make list of original features without meantempm, mintempm, and maxtempm
to_remove = [feature
             for feature in features
             if feature not in ['meantempm', 'mintempm', 'maxtempm']]

# make a list of columns to keep
to_keep = [col for col in df.columns if col not in to_remove]

# select only the columns in to_keep and assign to df
df = df[to_keep]
```

jupyter Collect Weather Data API Last Checkpoint: 03/12/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

In [21]:
```python
# Call describe on df and transpose it due to the large number of columns
spread = df.describe().T

# precalculate interquartile range for ease of use in next calculation
IQR = spread['75%'] - spread['25%']

# create an outliers column which is either 3 IQRs below the first quartile or
# 3 IQRs above the third quartile
spread['outliers'] = (spread['min'] <
                      (spread['25%'] -
                       (3 * IQR))) | (spread['max'] >
                                      (spread['75%'] + 3 * IQR))

# just display the features containing extreame outliers
spread.loc[spread.outliers, ]
```

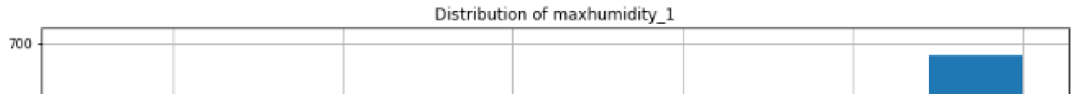
Out[21]:

	count	mean	std	min	25%	50%	75%	max	outliers
maxhumidity_1	996.0	94.326305	10.732047	45.0	94.0	100.0	100.0	100.00	True
maxhumidity_2	995.0	94.320603	10.735934	45.0	94.0	100.0	100.0	100.00	True
maxhumidity_3	994.0	94.314889	10.739825	45.0	94.0	100.0	100.0	100.00	True
minpressurem_1	994.0	1014.230382	5.858541	996.0	1011.0	1014.0	1017.0	1037.00	True
minpressurem_2	993.0	1014.231621	5.861363	996.0	1011.0	1014.0	1017.0	1037.00	True
minpressurem_3	992.0	1014.231855	5.864315	996.0	1011.0	1014.0	1017.0	1037.00	True
precipm_1	999.0	1.419109	7.958652	0.0	0.0	0.0	0.0	131.57	True
precipm_2	998.0	1.420531	7.962515	0.0	0.0	0.0	0.0	131.57	True
precipm_3	997.0	1.421956	7.966384	0.0	0.0	0.0	0.0	131.57	True

In [22]:
```python
# iterate over the precip columns
for precip_col in ['precipm_1', 'precipm_2', 'precipm_3']:
    # create a boolean array of values representing nans
    missing_vals = pd.isnull(df[precip_col])
    df[precip_col][missing_vals] = 0
```

In [23]:
```python
df = df.dropna()
```

In [24]:
```python
fig, ax = plt.subplots(figsize = (14, 8))
ax.hist(df.maxhumidity_1)
ax.set_title('Distribution of maxhumidity_1')
ax.set_xlabel('maxhumidity_1')
ax.grid()
```

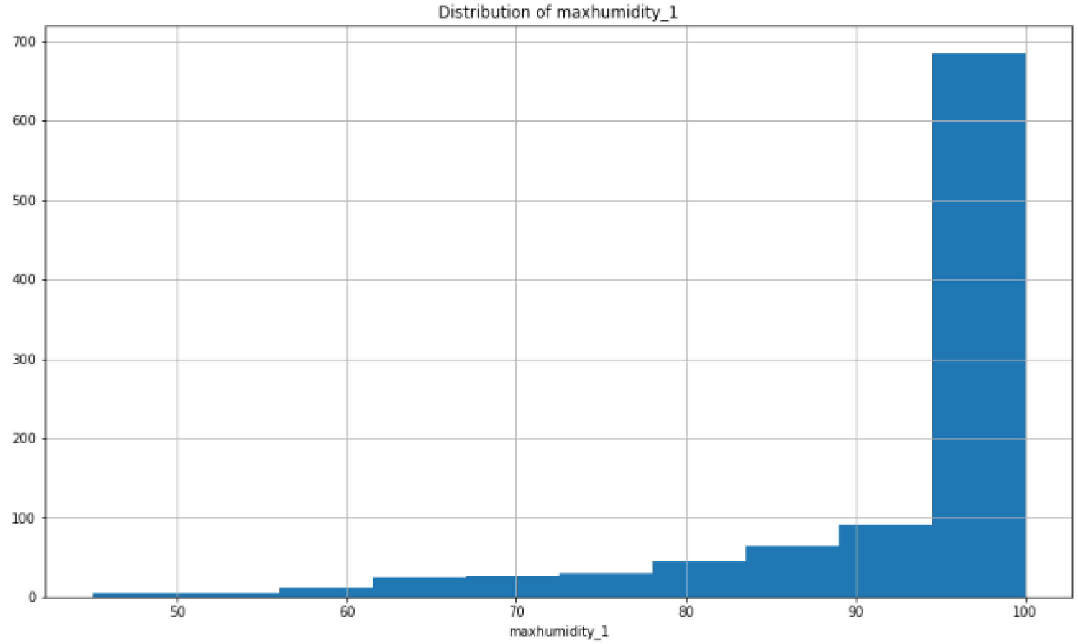


precipm_3	997.0	1.421956	7.966384	0.0	0.0	0.0	0.0	131.57	True

In [22]:
```
# iterate over the precip columns
for precip_col in ['precipm_1', 'precipm_2', 'precipm_3']:
    # create a boolean array of values representing nans
    missing_vals = pd.isnull(df[precip_col])
    df[precip_col][missing_vals] = 0
```

In [23]:
```
df = df.dropna()
```

In [24]:
```
fig, ax = plt.subplots(figsize = (14, 8))
ax.hist(df.maxhumidity_1)
ax.set_title('Distribution of maxhumidity_1')
ax.set_xlabel('maxhumidity_1')
ax.grid()
```



In [25]:
```
fig, ax = plt.subplots(figsize = (14, 8))
ax.hist(df.minpressurem_1)
ax.set_title('Distribution of minpressurem_1')
```

WEATHER FORCASTING CODE

In [1]:
```
import pickle

import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, median_absolute_error
from sklearn.model_selection import train_test_split

import matplotlib
import matplotlib.pyplot as plt
import statsmodels.api as sm

%matplotlib inline
```

In [2]:
```
with open('end-part1_df.pkl', 'rb') as fp:
    df = pickle.load(fp)
```

In [3]:
```
# df = pd.read_csv('end-part2_df.csv').set_index('date')
```

In [4]:
```
df.head()
```

Out[4]:

date	meantempm	maxtempm	mintempm	meantempm_1	meantempm_2	meantempm_3	meandewptm_1	meandewptm_2	meandewptm_3	meanpr
2015-10-06 22:13:06.559948	23.0	30.0	17.0	24.0	22.0	21.0	11.0	8.0	6.0	
2015-10-07 22:13:06.559948	24.0	32.0	17.0	23.0	24.0	22.0	11.0	11.0	8.0	
2015-10-08 22:13:06.559948	26.0	32.0	19.0	24.0	23.0	24.0	13.0	11.0	11.0	
2015-10-09 22:13:06.559948	26.0	30.0	22.0	26.0	24.0	23.0	17.0	13.0	11.0	
2015-10-10 22:13:06.559948	24.0	30.0	18.0	26.0	26.0	24.0	17.0	17.0	13.0	

5 rows × 39 columns

In [5]:
```
df_corr = df.corr()[['meantempm']].sort_values('meantempm')
```

In [6]:
```
df_corr_fil = df_corr[abs(df_corr['meantempm']) > 0.55]
```

In [7]:
```
unwanted = ['mintempm', 'maxtempm', 'meantempm']
predictors = df_corr_fil.index.tolist()
```

In [7]:
```
unwanted = ['mintempm', 'maxtempm', 'meantempm']
predictors = df_corr_fil.index.tolist()
predictors = [i for i in predictors if i not in unwanted]
```

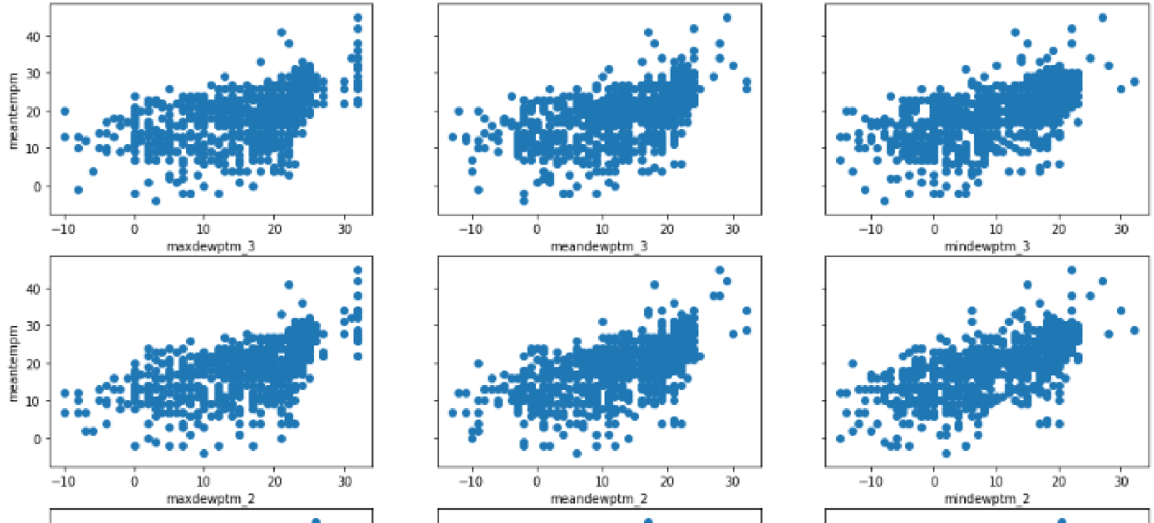
In [8]:
```
df2 = df[['meantempm'] + predictors]
```

In [9]:
```
# manually set the parameters of the figure to an appropriate size
plt.rcParams['figure.figsize'] = [16, 22]

# call subplots specifying the grid structure we desire and that
# the y axes should be shared
fig, axes = plt.subplots(nrows=6, ncols=3, sharey=True)

# Since it would be nice to loop through the features in to build this plot
# let us rearrange our data into a 2D array of 6 rows and 3 columns
arr = np.array(predictors).reshape(6, 3)

# use enumerate to loop over the arr 2D array of rows and columns
# and create scatter plots of each meantempm vs each feature
for row, col_arr in enumerate(arr):
    for col, feature in enumerate(col_arr):
        axes[row, col].scatter(df2[feature], df2['meantempm'])
        if col ==0:
            axes[row, col].set(xlabel=feature, ylabel='meantempm')
        else:
            axes[row, col].set(xlabel=feature)
plt.show()
```



In [10]:
```
# separate the predictor variables (X) from the outcome variable y
X = df2[predictors]
y = df2['meantempm']

# Add a constant to the predictor variable set to represent the Bo intercept
# X = sm.add_constant(X)
X.iloc[:5, :5]
```

Out[10]:

	maxdewptm_3	meandewptm_3	mindewptm_3	maxdewptm_2	meandewptm_2
date					
2015-10-06 22:13:06.559948	8.0	6.0	4.0	10.0	8.0
2015-10-07 22:13:06.559948	10.0	8.0	7.0	13.0	11.0
2015-10-08 22:13:06.559948	13.0	11.0	8.0	14.0	11.0
2015-10-09 22:13:06.559948	14.0	11.0	8.0	16.0	13.0
2015-10-10 22:13:06.559948	16.0	13.0	10.0	20.0	17.0

In [11]:
```
alpha = 0.01

def stepwise_selection(X, y,
                       initial_list=predictors,
                       threshold_out=alpha,
                       verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    See https://en.wikipedia.org/wiki/Stepwise_regression for the details
    """
    included = list(initial_list)
    while True:
        changed=False
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included

result = stepwise_selection(X, y)
```

```
print('resulting features:')
print(result)
```

```
Drop maxtempm_2                    with p-value 0.773688
Drop meandewptm_2                  with p-value 0.39318
Drop maxdewptm_3                   with p-value 0.348998
Drop meandewptm_3                  with p-value 0.179861
Drop mintempm_2                    with p-value 0.20512
Drop meantempm_2                   with p-value 0.223981
Drop meantempm_3                   with p-value 0.109188
Drop mintempm_3                    with p-value 0.0549466
Drop maxdewptm_1                   with p-value 0.0442908
resulting features:
['mindewptm_3', 'maxdewptm_2', 'mindewptm_2', 'maxtempm_3', 'meandewptm_1', 'mindewptm_1', 'mintempm_1', 'maxtempm_1', 'meantem
pm_1']
```

In [12]:

```
X = X[result]
model = sm.OLS(y, X).fit()
model.summary()
```

Out[12]:

OLS Regression Results

Dep. Variable:	meantempm	R-squared:	0.982
Model:	OLS	Adj. R-squared:	0.982
Method:	Least Squares	F-statistic:	5979.
Date:	Fri, 06 Jul 2018	Prob (F-statistic):	0.00
Time:	20:08:07	Log-Likelihood:	-2443.9
No. Observations:	987	AIC:	4906.
Df Residuals:	978	BIC:	4950.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
mindewptm_3	0.0821	0.022	3.665	0.000	0.038	0.126
maxdewptm_2	-0.1758	0.027	-6.533	0.000	-0.229	-0.123
mindewptm_2	-0.1459	0.029	-5.081	0.000	-0.202	-0.090
maxtempm_3	0.1630	0.021	7.908	0.000	0.123	0.203
meandewptm_1	-0.1103	0.052	-2.118	0.034	-0.212	-0.008
mindewptm_1	0.2859	0.044	6.537	0.000	0.200	0.372
mintempm_1	0.7105	0.135	5.260	0.000	0.445	0.976
maxtempm_1	0.8414	0.126	6.673	0.000	0.594	1.089
meantempm_1	-0.7120	0.254	-2.806	0.005	-1.210	-0.214

Omnibus:	102.494	Durbin-Watson:	2.049
Prob(Omnibus):	0.000	Jarque-Bera (JB):	256.945

mindewptm_3	0.0821	0.022	3.665	0.000	0.038	0.126
maxdewptm_2	-0.1758	0.027	-6.533	0.000	-0.229	-0.123
mindewptm_2	-0.1459	0.029	-5.081	0.000	-0.202	-0.090
maxtempm_3	0.1630	0.021	7.908	0.000	0.123	0.203
meandewptm_1	-0.1103	0.052	-2.118	0.034	-0.212	-0.008
mindewptm_1	0.2859	0.044	6.537	0.000	0.200	0.372
mintempm_1	0.7105	0.135	5.260	0.000	0.445	0.976
maxtempm_1	0.8414	0.126	6.673	0.000	0.594	1.089
meantempm_1	-0.7120	0.254	-2.806	0.005	-1.210	-0.214

Omnibus:	102.494	Durbin-Watson:	2.049
Prob(Omnibus):	0.000	Jarque-Bera (JB):	256.945
Skew:	-0.570	Prob(JB):	1.60e-56
Kurtosis:	5.224	Cond. No.	193.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [13]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=12)
```

In [14]:

```
# instantiate the regressor class
regressor = LinearRegression()

# fit the model by fitting the regressor to the training data
regressor.fit(X_train, y_train)

# make a prediction set using the test set
prediction = regressor.predict(X_test)

# Evaluate the prediction accuracy of the model
print('The Explained Variance: %.2f' % regressor.score(X_test, y_test))
print('The Mean Absolute Error: %.2f degrees celcius' % mean_absolute_error(
    y_test, prediction))
print('The Median Absolute Error: %.2f degrees celcius' %
      median_absolute_error(y_test, prediction))
```

```
The Explained Variance: 0.85
The Mean Absolute Error: 2.10 degrees celcius
The Median Absolute Error: 1.30 degrees celcius
```