Manuel Romero

Python ML with sklearn

Code bits taken from Book and Github

https://github.com/kjmazidi/Machine_Learning_2nd_edition/tree/master/Part_7_Neural_Networks

Reading the CSV file Auto.csv Part 1

> Selecting Auto.csv from local drive

Double-click (or enter) to edit

```
import pandas as pd
from google.colab import files
uploaded = files.upload()
import io
df = pd.read_csv(io.BytesIO(uploaded['Auto.csv']))
# Dataset is now stored in a Pandas Dataframe
```

Choose Files Auto.csv

- **Auto.csv**(text/csv) - 17859 bytes, last modified: 4/10/2023 - 100% done

```
Saving Auto.csv to Auto (4).csv
```

file is read and now we will output the first rows and dimensions of the table

```
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	8	302.0	140	3449	NaN	70.0	1	ford torino

```
print("The dimension of the Auto.csv data is : " , df.shape)
```

```
The dimension of the Auto.csv data is :  (392, 9)
```

Data exploration with code Part 2

MPG:

```
Range: 9 - 46.6
AVG: 23.445918
```

```
df.mpg.describe()
```

```
    count    392.000000
    mean      23.445918
    std        7.805007
    min        9.000000
    25%       17.000000
    50%       22.750000
    75%       29.000000
    max       46.600000
    Name: mpg, dtype: float64
```

Weight:

```
Range: 1613 - 5140
AVG: 2977.584184
```

```
df.weight.describe()
```

```
    count     392.000000
    mean     2977.584184
    std       849.402560
    min      1613.000000
    25%      2225.250000
    50%      2803.500000
    75%      3614.750000
    max      5140.000000
    Name: weight, dtype: float64
```

Year :

```
Range: 70 - 82
AVG: 76.010256
```

```
df.year.describe()
```

```
    count    390.000000
    mean      76.010256
```

```
std        3.668093
min       70.000000
25%       73.000000
50%       76.000000
75%       79.000000
max       82.000000
Name: year, dtype: float64
```

Explore Data types part 3

data types of columns

```
df.dtypes
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

change the cylinders column to categorical (use cat.codes)

```
df.cylinders = df.cylinders.astype("category").cat.codes
```

change the origin column to categorical (don't use cat.codes)

```
df.origin = df.origin.astype("category")
```

verifying the changes with the dtypes attribute

```
df.dtypes
```

```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

Deal with NAs part 4

```
df = df.dropna()
```

Dropped NA's

New dimensions Dimension 389 , 9

```
df.shape
```

```
(389, 9)
```

Modify columns part 5

Make a new column, mpg_high, and make it categorical

adding new blank column and then filling it in

```
##new column mpg_high set with initial value of 0
df['mpg_high'] = 0
```

Checking to see if column added

```
df.dtypes
```

```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
mpg_high           int64
dtype: object
```

```
##getting average of mpg
avg = df["mpg"].mean()
print(avg)
##filling in the column with 1 if mpg > mean mpg else is covered since we made the new column with 0s originally
df['mpg_high'][df.mpg > avg] = 1
```

```
23.490488431876607
<ipython-input-114-596124658d86>:5: 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
      df['mpg_high'][df.mpg > avg] = 1
```

```
## making mpg high a category
df.mpg_high = df.mpg_high.astype("category")
```

deleting mpg and name column and printing df.head

```
df = df.drop(columns=["mpg", "name"])
print(df.head())
```

```
   cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
1          4         350.0         165    3693          11.5  70.0      1
2          4         318.0         150    3436          11.0  70.0      1
3          4         304.0         150    3433          12.0  70.0      1
6          4         454.0         220    4354           9.0  70.0      1

  mpg_high
0        0
1        0
2        0
3        0
6        0
```

Data exploration with graphs part 6

catplot

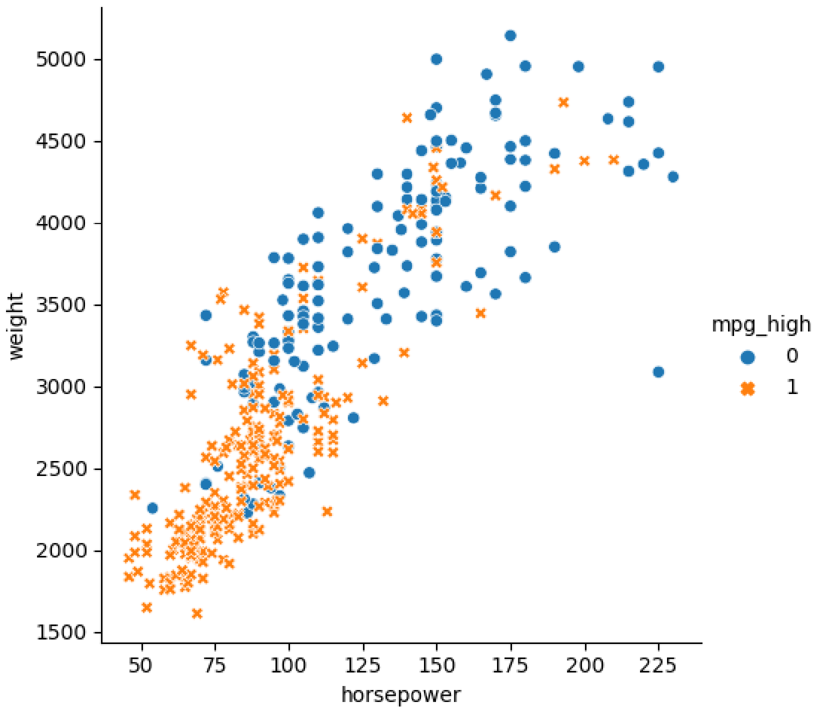
```
import seaborn as sb
from sklearn import datasets
sb.catplot(x="mpg_high", kind="count", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f983e9708b0>
```

The catplot shows us that there is more vehicles with a high mpg than those with not a high mpg.

```
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high, style=df.mpg_high)
```

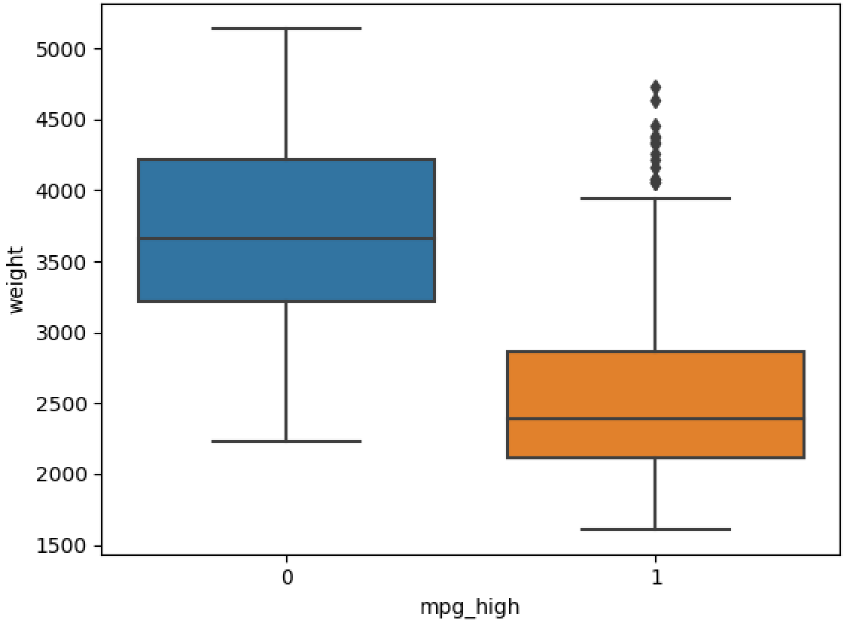
```
<seaborn.axisgrid.FacetGrid at 0x7f98463120d0>
```



We can see that the lower the weight the lower the horsepower, this makes sense since it takes less power to mave less weight also less power typically means smaller engine. Furthermore we see that most lower weight and lower powered cars are high_mpg cars this also makes sense it takes less gas to move the wight and smaller engines use less gas.

```
sb.boxplot(x = 'mpg_high', y = 'weight', data = df)
```

```
<Axes: xlabel='mpg_high', ylabel='weight'>
```



once again we see that the high mpg car are lower in weight, the mean for high mpg is much lower than that of not high mpg. We also see that there are possible outliers on the data for high mpg vehicles. Some heavy cars are still getting high mpg.

Train/test split part 7

```
df.shape
```

```
(389, 8)
```

```
df.head()
```

	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
6	4	454.0	220	4354	9.0	70.0	1	0

```
# train test split
from sklearn.model_selection import train_test_split
X = df.iloc[:, 0:6]
##7 is the mpg_high column
y = df.iloc[:, 7]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1234)
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (311, 6)
test size: (78, 6)
```

Logistic Regression pt 8

train a logistic regression model using solver lbfgs

test and evaluate

print metrics using the classification report

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.8102893890675241
```

output:

0.8102893890675241

```
# make predictions
pred = clf.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.74      0.79        39
           1       0.77      0.87      0.82        39

    accuracy                           0.81        78
```

```
       macro avg       0.81      0.81      0.81        78
    weighted avg       0.81      0.81      0.81        78
```

Decision Tree part 9

```
from sklearn import tree
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
    accuracy score:  0.7307692307692307
    precision score:  0.68
    recall score:  0.8717948717948718
    f1 score:  0.7640449438202247
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
                  precision    recall  f1-score   support

               0       0.82      0.59      0.69        39
               1       0.68      0.87      0.76        39

        accuracy                           0.73        78
       macro avg       0.75      0.73      0.73        78
    weighted avg       0.75      0.73      0.73        78
```

We see that logistic regression had better accuracy than that of decision trees.

```
tree.plot_tree(clf)
```

```
[Text(0.46629213483146065, 0.9583333333333334, 'x[1] <= 153.0\ngini = 0.447\nsamples = 311\nvalue = [105, 206]'),
 Text(0.2303370786516854, 0.875, 'x[3] <= 2822.5\ngini = 0.11\nsamples = 171\nvalue = [10, 161]'),
 Text(0.16853932584269662, 0.7916666666666666, 'x[5] <= 73.5\ngini = 0.064\nsamples = 152\nvalue = [5, 147]'),
 Text(0.14606741573033707, 0.7083333333333334, 'x[1] <= 131.0\ngini = 0.239\nsamples = 36\nvalue = [5, 31]'),
 Text(0.0898876404494382, 0.625, 'x[0] <= 0.5\ngini = 0.117\nsamples = 32\nvalue = [2, 30]'),
 Text(0.0449438202247191, 0.5416666666666666, 'x[5] <= 72.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.02247191011235955, 0.4583333333333333, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.06741573033707865, 0.4583333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.1348314606741573, 0.5416666666666666, 'x[3] <= 2377.0\ngini = 0.064\nsamples = 30\nvalue = [1, 29]'),
 Text(0.11235955056179775, 0.4583333333333333, 'gini = 0.0\nsamples = 25\nvalue = [0, 25]'),
 Text(0.15730337078651685, 0.4583333333333333, 'x[3] <= 2387.0\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.1348314606741573, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.1797752808988764, 0.375, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.20224719101123595, 0.625, 'x[3] <= 2332.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.1797752808988764, 0.5416666666666666, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.2247191011235955, 0.5416666666666666, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.19101123595505617, 0.7083333333333334, 'gini = 0.0\nsamples = 116\nvalue = [0, 116]'),
 Text(0.29213483146067415, 0.7916666666666666, 'x[5] <= 79.5\ngini = 0.388\nsamples = 19\nvalue = [5, 14]'),
 Text(0.2696629213483146, 0.7083333333333334, 'x[2] <= 87.5\ngini = 0.496\nsamples = 11\nvalue = [5, 6]'),
 Text(0.24719101123595505, 0.625, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.29213483146067415, 0.625, 'x[5] <= 75.5\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
 Text(0.2696629213483146, 0.5416666666666666, 'x[3] <= 2900.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(0.24719101123595505, 0.4583333333333333, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.29213483146067415, 0.4583333333333333, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.3146067415730337, 0.5416666666666666, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.3146067415730337, 0.7083333333333334, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
 Text(0.702247191011236, 0.875, 'x[5] <= 75.5\ngini = 0.436\nsamples = 140\nvalue = [95, 45]'),
 Text(0.5168539325842697, 0.7916666666666666, 'x[4] <= 14.75\ngini = 0.289\nsamples = 80\nvalue = [66, 14]'),
 Text(0.4044943820224719, 0.7083333333333334, 'x[2] <= 151.5\ngini = 0.113\nsamples = 50\nvalue = [47, 3]'),
 Text(0.38202247191011235, 0.625, 'x[4] <= 10.25\ngini = 0.198\nsamples = 27\nvalue = [24, 3]'),
 Text(0.3595505617977528, 0.5416666666666666, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.4044943820224719, 0.5416666666666666, 'x[3] <= 4062.0\ngini = 0.142\nsamples = 26\nvalue = [24, 2]'),
 Text(0.38202247191011235, 0.4583333333333333, 'gini = 0.0\nsamples = 15\nvalue = [15, 0]'),
 Text(0.42696629213483145, 0.4583333333333333, 'x[3] <= 4089.0\ngini = 0.298\nsamples = 11\nvalue = [9, 2]'),
 Text(0.4044943820224719, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.449438202247191, 0.375, 'x[3] <= 4448.5\ngini = 0.18\nsamples = 10\nvalue = [9, 1]'),
 Text(0.42696629213483145, 0.2916666666666667, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.47191011235955055, 0.2916666666666667, 'x[3] <= 4460.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.449438202247191, 0.20833333333333334, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.4943820224719101, 0.20833333333333334, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.42696629213483145, 0.625, 'gini = 0.0\nsamples = 23\nvalue = [23, 0]'),
 Text(0.6292134831460674, 0.7083333333333334, 'x[2] <= 125.0\ngini = 0.464\nsamples = 30\nvalue = [19, 11]'),
 Text(0.6067415730337079, 0.625, 'x[3] <= 3111.5\ngini = 0.393\nsamples = 26\nvalue = [19, 7]'),
 Text(0.5393258426966292, 0.5416666666666666, 'x[2] <= 92.5\ngini = 0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.5168539325842697, 0.4583333333333333, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.5617977528089888, 0.4583333333333333, 'x[4] <= 15.25\ngini = 0.245\nsamples = 7\nvalue = [1, 6]'),
 Text(0.5393258426966292, 0.375, 'x[2] <= 105.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.5168539325842697, 0.2916666666666667, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5617977528089888, 0.2916666666666667, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5842696629213483, 0.375, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.6741573033707865, 0.5416666666666666, 'x[3] <= 3384.0\ngini = 0.111\nsamples = 17\nvalue = [16, 1]'),
 Text(0.651685393258427, 0.4583333333333333, 'x[3] <= 3332.5\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
 Text(0.6292134831460674, 0.375, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
 Text(0.6741573033707865, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6966292134831461, 0.4583333333333333, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.651685393258427, 0.625, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.8876404494382022, 0.7916666666666666, 'x[5] <= 79.5\ngini = 0.499\nsamples = 60\nvalue = [29, 31]'),
```

```
Text(0.8426966292134831, 0.7083333333333334, 'x[4] <= 18.95\ngini = 0.493\nsamples = 50\nvalue = [28, 22]'),
```

Here is the decision Tree plotted

```
Text(0.7415730337078652, 0.4583333333333333, 'x[3] <= 3507.5\ngini = 0.498\nsamples = 15\nvalue = [8, 7]'),
```

Neural Network Part 10

```
Text(0.7415730337078652, 0.2916666666666667, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
```

Neural Network

```
Text(0.7865168539325843, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
```

```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
Text(0.8539325842696629, 0.375, 'x[1] <= 237.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
```

```
# train
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234)
clf.fit(X_train_scaled, y_train)
```

```
▾ MLPClassifier
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

```
Text(0.9775280898876404, 0.5416666666666666, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')]
```

```
pred = clf.predict(X_test_scaled)
# output results

print('accuracy = ', accuracy_score(y_test, pred))

confusion_matrix(y_test, pred)
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
accuracy =  0.8333333333333334
              precision    recall  f1-score   support

           0       0.80      0.90      0.84        39
           1       0.88      0.77      0.82        39

    accuracy                           0.83        78
   macro avg       0.84      0.83      0.83        78
weighted avg       0.84      0.83      0.83        78
```

different settings

```
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(4,6), max_iter=1500, random_state=1234)
clf.fit(X_train_scaled, y_train)
# make predictions
pred = clf.predict(X_test_scaled)

print('accuracy = ', accuracy_score(y_test, pred))

# confusion matrix
confusion_matrix(y_test, pred)
print(classification_report(y_test, pred))
```

```
accuracy =  0.8333333333333334
              precision    recall  f1-score   support

           0       0.86      0.79      0.83        39
           1       0.81      0.87      0.84        39

    accuracy                           0.83        78
   macro avg       0.84      0.83      0.83        78
weighted avg       0.84      0.83      0.83        78
```

using the neural network with clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234) provides a much better accuracy than logistic regression and about the same for second model of neural netwroks. However the first model works better for precission on high mpg vs the second one being better for not high mpg classification. The reason for this is the change in hidden layers.

```
My prefrence was python it feels much better to use and much faster. One
issue was that my computer isnt updating my pandas and therefore is nt
working great but with google colab that was solved and everything went
smoothly after that. On the other hand R sometimes installs packages
succesfully and others it wont not sure why, but probably user error or
machine error that i need to look into.
```

My prefrence was python it feels much better to use and much faster. One issue was that my computer isnt updating my pandas and therefore is nt working great but with google colab that was solved and everything went smoothly after that. On the other hand R sometimes installs packages succesfully and others it wont not sure why, but probably user error or machine error that i need to look into.

1s completed at 4:17 PM