Final_Turn_in_Arrow

March 29, 2021

[1]:
```python
from sklearn.preprocessing import LabelEncoder
gle = LabelEncoder
import seaborn as sbs
from sklearn.metrics import r2_score
```

[2]:
```python
#imports
import pandas as pd
#read csv, create dataframes

import numpy as np
#used for vector values

import matplotlib.pyplot as plt
#Each pyplot function makes some change to a figure: e.g., creates a figure,␣
↪creates a plotting area in a figure,
#plots some lines in a plotting area, decorates the plot with labels, etc.

from sklearn.linear_model import LinearRegression
#check features for best fit

from sklearn.linear_model import LogisticRegression
#check features for best fit

from sklearn.metrics import mean_absolute_error
#check accuracy of model

from sklearn.model_selection import train_test_split
#used to create training data and validation data

from sklearn.tree import DecisionTreeRegressor
#Decision tree builds regression or classification models in the form of a tree␣
↪structure. It breaks down a dataset into
#smaller and smaller subsets while at the same time an associated decision tree␣
↪is incrementally developed. The final
#result is a tree with decision nodes and leaf nodes.

from sklearn.ensemble import RandomForestRegressor
```

```
#Random forest builds multiple decision trees and merges them together to get a␣
↪more accurate and stable prediction.
#Random forest has nearly the same hyperparameters as a decision tree or a␣
↪bagging classifier. ... Random forest adds additional randomness to the␣
↪model, while growing the trees.

from sklearn.preprocessing import StandardScaler
#used to Standardize features by removing the mean and scaling to unit variance

from random import sample
from random import shuffle
from random import Random
print("Ready to Roll")
```

Ready to Roll

[67]:
```
#import data and create dataframe
#sample set to a set random sample of 1000 instances
df = pd.read_csv('**********')
df = df.sample(1000, random_state=42)

#label encoder to allow use of the mpn column
gle = LabelEncoder()

#adding lable column which groups the mfr by labels
df['label'] = df.groupby(pd.Grouper(key='mfr')).ngroup()

#creating labels for mpn
mpn_labels = np.unique(df['mpn'])
mpn_labels = gle.fit_transform(df['mpn'])
mpn_mapping = {index: label for index, label in
                  enumerate(gle.classes_)}

#adding mpn labels to the dataframe
df['mpn_labels']= mpn_labels

#features to be used in the model after analyzing all features to find the␣
↪optimal result
clean_df_cols = [***************]

#creating data frame from the features
clean_df = df[clean_df_cols]

data = df
#simplifying the X and y variables
X = clean_df
```

```
y = df.expected_leadtime


#function of how labels for the mfr column are grouped
def label(DF,label):
        Y = DF.groupby(["label"]).get_group(label)
        Input = pd.DataFrame(Y)
        return Input
```

[13]:
```
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 42,␣
 ↪test_size=0.20)
# split data into training and validation data, for both features and target
# The split is based on a random number generator. Supplying a numeric value to
# the random_state argument guarantees we get the same split every time we
# run this script.

forest_model = RandomForestRegressor(random_state=42)
forest_model.fit(train_X, train_y)
Expect_Time_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, Expect_Time_preds))
```

```
6.452105332413997
```

[14]:
```
# Define the models
model_1 = RandomForestRegressor(n_estimators=50, random_state=42)
model_2 = RandomForestRegressor(n_estimators=100, random_state=42)
model_3 = RandomForestRegressor(n_estimators=100, criterion='mae',␣
 ↪random_state=42)
model_4 = RandomForestRegressor(n_estimators=200, min_samples_split=20,␣
 ↪random_state=42)
model_5 = RandomForestRegressor(n_estimators=100, max_depth=7, random_state=42)
# model selection used as standard from kaggle

models = [model_1, model_2, model_3, model_4, model_5]

#function to calculate the MAE of each model to find the model of best fit
def score_model(model, X_t=train_X, X_v=val_X, y_t=train_y, y_v=val_y):
        model.fit(X_t, y_t)
        preds = model.predict(X_v)
        return mean_absolute_error(y_v, preds)

#for loop to run function on all model options
for i in range(0, len(models)):
    mae = score_model(models[i])
    print("Model",(i+1, mae))
```

```
Model (1, 6.478759144918404)
```

```
Model (2, 6.452105332413997)
Model (3, 6.25325)
Model (4, 6.475031567773015)
Model (5, 6.303856369783222)
```

[133]:

```
model = model_3
Team_11_model = model
# Fit the model to the training data
Team_11_model.fit(X, y)

# Generate test predictions
Team_11_preds_test = Team_11_model.predict(X)

print(mean_absolute_error(y, Team_11_preds_test))

data = pd.DataFrame({'label':df.label,'mfr':df.mfr, 'Expected_Time':
 ↪Team_11_preds_test,
                     'Actual_Time': df.actualleadtime,
                     'Time_Diff':Team_11_preds_test - df.actualleadtime})
data['MAE']= mean_absolute_error(y, Team_11_preds_test)

#function to run label groups through the model
def model(DF):

    model = model_3
    Team_11_model = model

# Fit the model to the training data
    Team_11_model.fit(X, y)

# Generate test predictions
    Team_11_preds_test = Team_11_model.predict(X)
    DF['MAE']= mean_absolute_error(y, Team_11_preds_test)


    output = pd.DataFrame({'label':df.label, 'mfr':df.mfr, 'Expected_Time':␣
 ↪Team_11_preds_test,
                           'Actual_Time': df.actualleadtime,
                           'Time_Diff':Team_11_preds_test - df.actualleadtime,␣
 ↪'MAE': data.MAE})
    Results = output
    return Results
```

```
2.9710649999999994
```

[123]:

```
#r squared for model confidence
r2_score(y, Team_11_preds_test)
```

[123]: 0.7363091130316468

[134]:
```
#adding the value MAE to the data frame
data = pd.DataFrame({'label':df.label,'mfr':df.mfr, 'Expected_Time':
↪Team_11_preds_test,
                    'Actual_Time': df.actualleadtime,
                    'Time_Diff':Team_11_preds_test - df.actualleadtime, 'MAE':␣
↪data.MAE})
data['MAE']= mean_absolute_error(y, Team_11_preds_test)
```

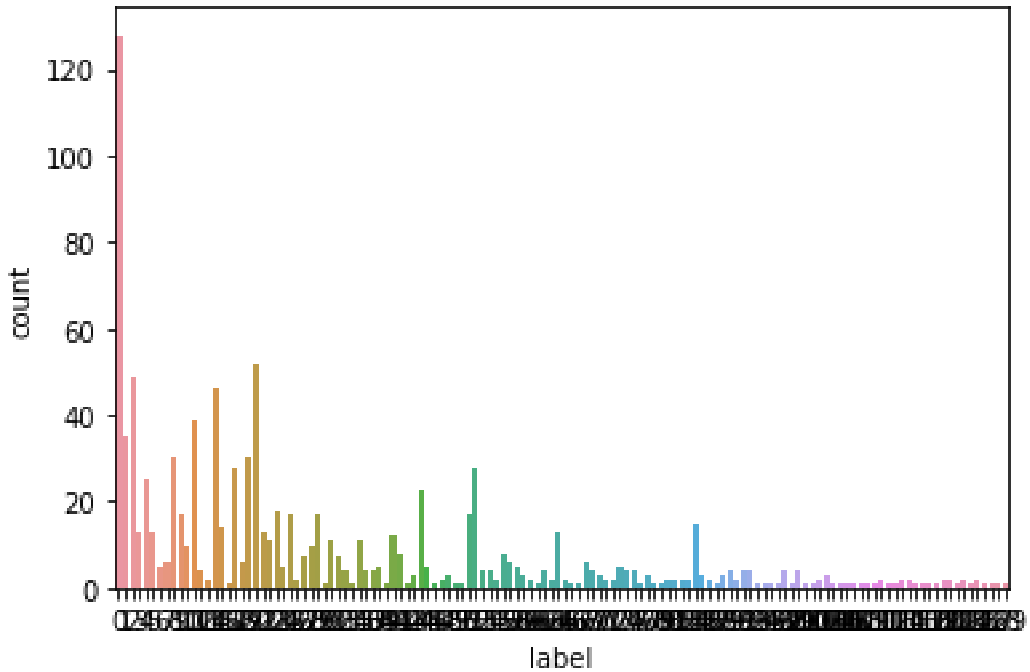
[183]:
```
#function that will repeat the lables through the model
def repeater(arg):
    for i in range(0,336):
        W = label(arg, i)
        v = pd.DataFrame(W.groupby('mfr').mean())
        return v
```

[187]:
```
Final_csv = pd.DataFrame({'ID': data.label, 'mfr': data.mfr, 'Expected_Time':
↪Team_11_preds_test,'Actual_Time':df.actualleadtime,
                         'Time_Difference':df.actualleadtime -␣
↪Team_11_preds_test})

#Function that brings everything together
def main():
    V= repeater(data)
    P= model(V)
    Results = P.groupby('mfr').mean()
    return Results
#creating the data frame columns and information to represent in the csv file



#Returning results to a csv
Final_csv.to_csv('results.csv', index = False)
```

[ ]:
```
#checking to see if everything works as planned
main()
```

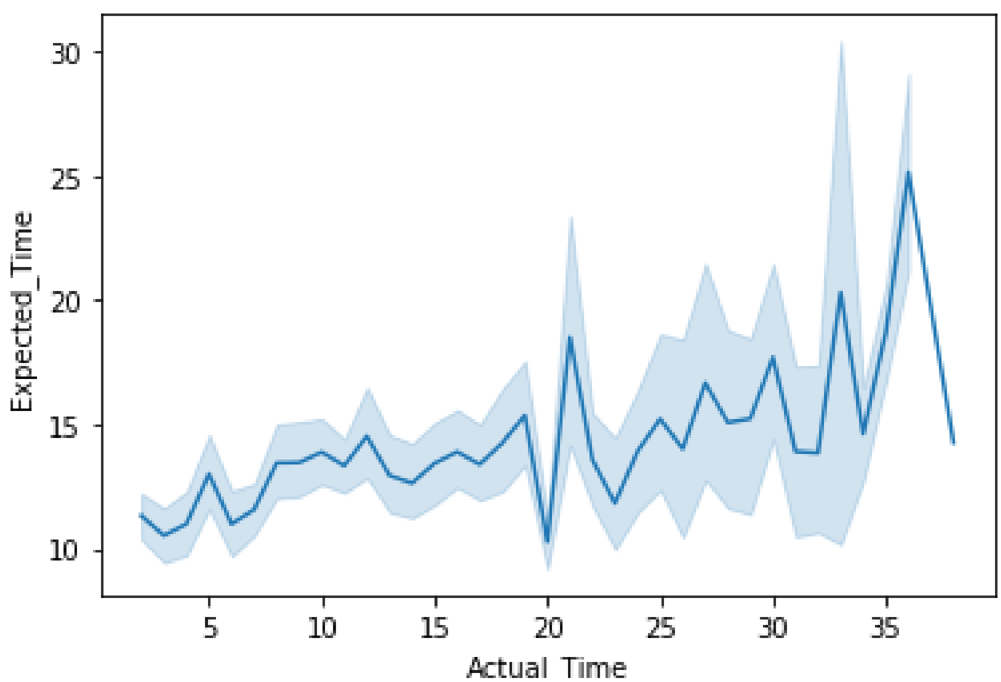
[190]:
```
#laying out the results to get a starting idea of what we are looking at
sbs.countplot(x= data.label, data=data)
```

[190]: <matplotlib.axes._subplots.AxesSubplot at 0x2100011ce48>



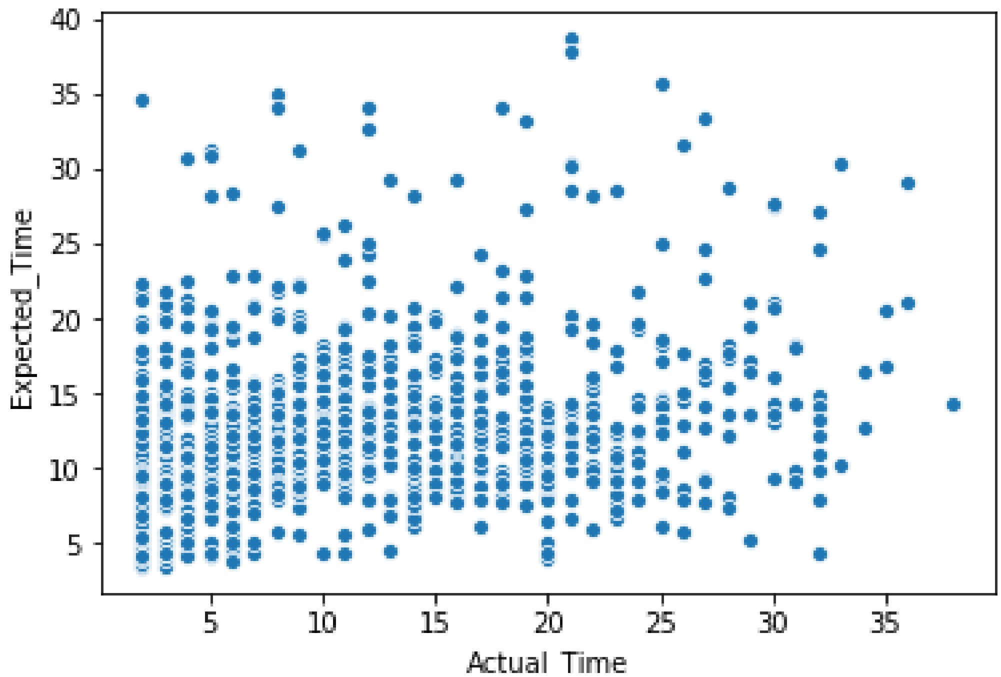
[191]:
```
#looking for trends in the data
sbs.lineplot(x= data.Actual_Time, y= data.Expected_Time, data=data)
```

[191]: <matplotlib.axes._subplots.AxesSubplot at 0x2107cad7088>

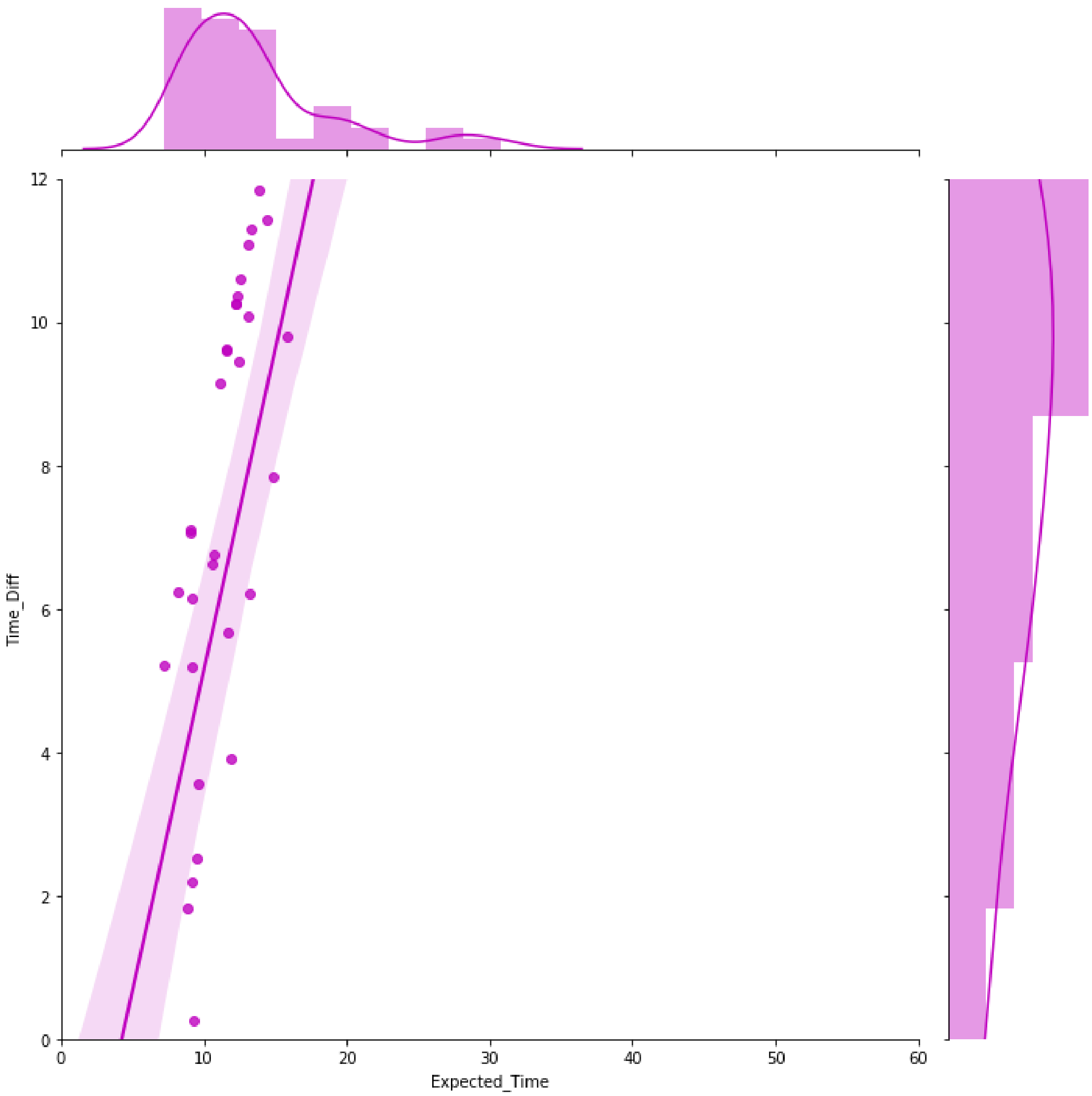


[192]:
```
#looking for groupings and consistency as well as outliers
sbs.scatterplot(x= 'Actual_Time', y= 'Expected_Time', data=data)
```

[192]: <matplotlib.axes._subplots.AxesSubplot at 0x210004f21c8>



[193]:
```
#finding distribution and skewness
tips = label(data, 14)

g = sbs.jointplot(x="Expected_Time", y="Time_Diff", data=tips,
                  kind="reg", truncate=False,
                  xlim=(0, 60), ylim=(0, 12),
                  color="m", height=10)
```

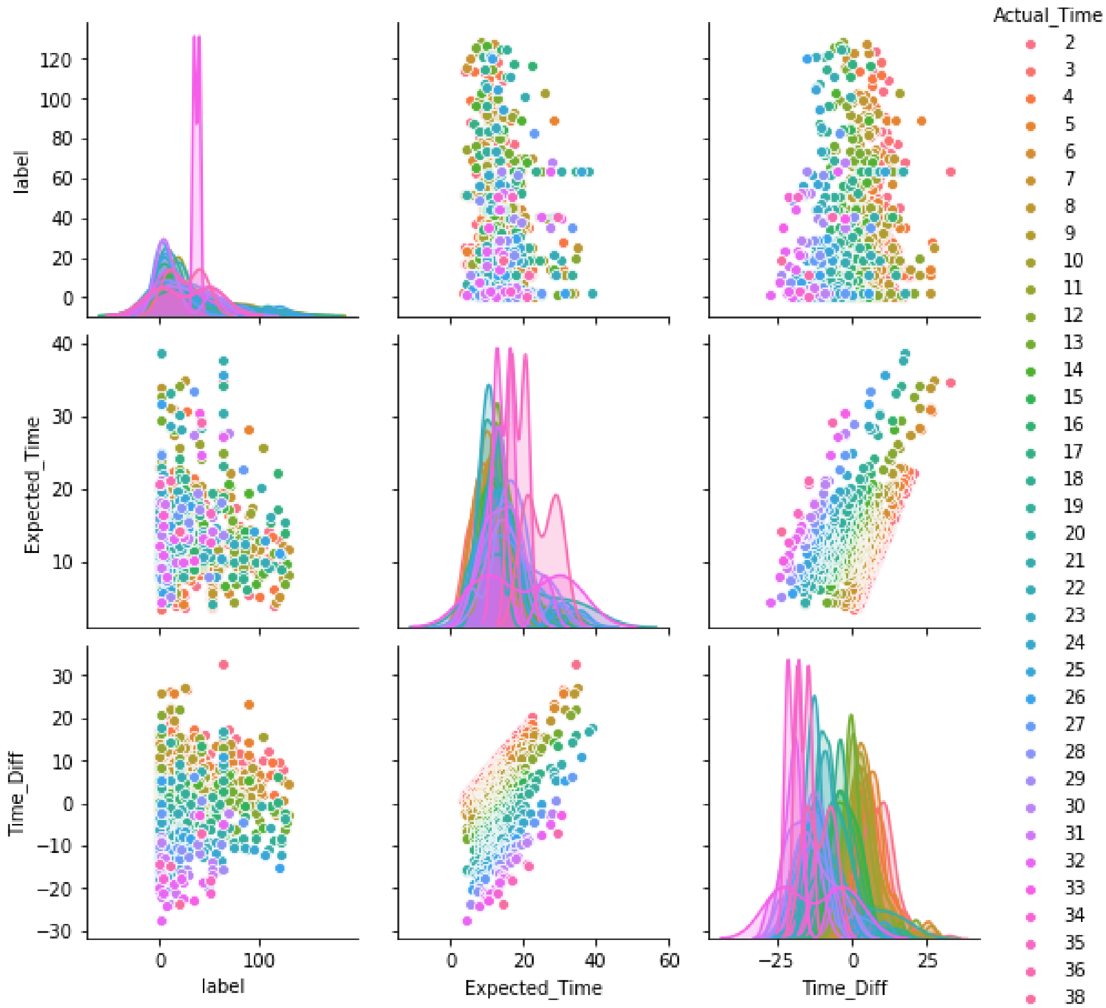


[194]:
```
#comparing actual time against label, expected and time diff
data = pd.DataFrame({'label':df.label,'Expected_Time':
↪Team_11_preds_test,'Actual_Time': df.actualleadtime,
                     'Time_Diff':Team_11_preds_test - df.actualleadtime})
sbs.pairplot(data, hue='Actual_Time')
```

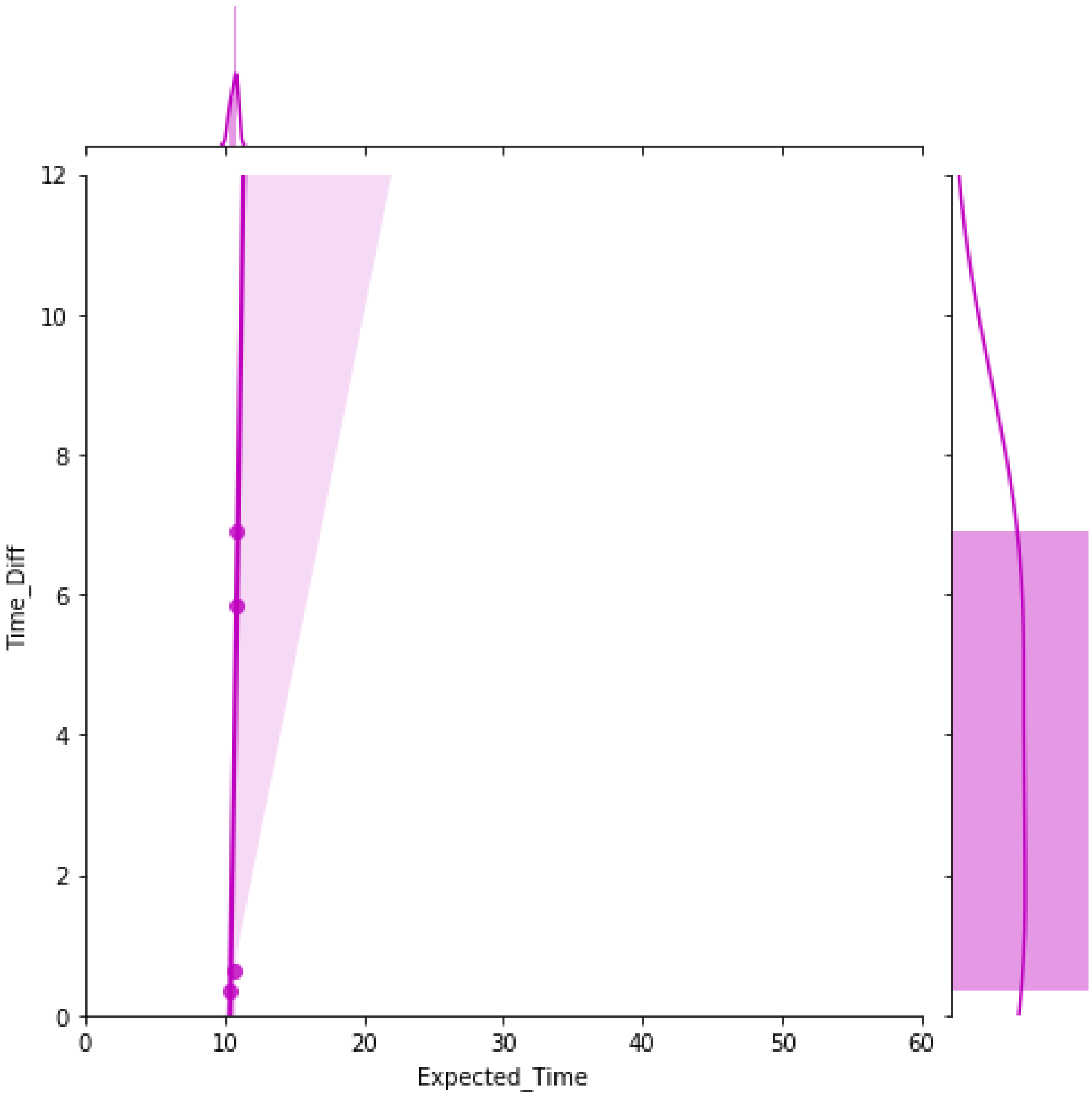
```
C:\Users\nmill\anaconda3\lib\site-packages\seaborn\distributions.py:288:
UserWarning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\nmill\anaconda3\lib\site-packages\seaborn\distributions.py:288:
UserWarning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\nmill\anaconda3\lib\site-packages\seaborn\distributions.py:288:
```

```
UserWarning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
```

[194]: `<seaborn.axisgrid.PairGrid at 0x210008a6948>`



[195]:
```
#displaying one label for reference
tips = label(data, 12)

g = sbs.jointplot(x="Expected_Time", y="Time_Diff", data=tips,
                  kind="reg", truncate=False,
                  xlim=(0, 60), ylim=(0, 12),
                  color="m", height=7)
```



12
10
8
6
4
2
0
Time_Diff
0
10
20
30
40
50
60
Expected_Time

[ ]: