All necessary code is in `order_pay_model.py`
You should be able to run it out of the box with `requirements.txt` installed and `datafile` changed the location of your '`datafile.tsv`' The only other things that you may want to change are `test_size, sample, scale` in my preprocessing function. I've included all of my scratch code in case you want to look through how I got to my solution, but I haven't been back through any of them to clean them up.

3 – Proposing a New Shopper Pay Structure

1. After exploring the dataset, what potential issues do you see with the underlying data provided? How would you go about doing some basic cleaning on this dataset?

The data contains some NaN values. I'm again going to drop these for simplicity, but they could be imputed easily. This is around 20% of our data dropped—but we should still have plenty to work it.

There are some outliers—some of which don't make any sense, like actual shopping time of 6000 minutes and driving time of 9000 minutes. I again take the simplest approach to just drop any row with a value > or < 3 standard deviations.

2. Our first pass at a shopper pay model might pay shoppers based on the cost of the order. What do you think of that proposal?

I would think that this should maybe have some bearing on the shopper pay, but shouldn't be the only factor. Shoppers who get a low-cost order may be frustrated if they don't make enough or if their pay is highly inconsistent compared to their time spent. It could also cause problems on items where they pick what it is as they may intentionally pick something more expensive so that they get paid more (though this may be a good thing for Shipt's bottom line in the end :)

3. Given the data provided, what evidence do you see for or against the proposal in (2)?

```
df['actual_order_cost'].describe()

count    207147.000000
mean         93.180852
std          58.791031
min           0.020000
25%          50.820000
50%          78.640000
75%         119.680000
max        1547.280000
```

Since we have such a large range and high standard deviation relative to our quartiles I think basing actual payment on cost of the order would lead to highly inconsistent payments.

4.Instead of paying shoppers based on the order cost, we'd like to explore using a formula based off of the estimated time to fulfill the order (`order_estimated_driving_time_min` and `order_estimated_shopping_time_min`) while keeping the following goals in mind:

- We'd like to set the target average hourly wage at $X per hour (say $15 for example).

- We'd like to reduce some of the variance we see in the current shopper pay numbers (`actual_shopper_pay`). How would you come up with such a formula? (*)

You can see my model in `order_pay_model.py`

First, in my preprocessing function, I create a new column called `'total_time_min'` which is the sum of `'order_estimated_driving_time_min'` and `'order_estimated_shopping_time_min'`.

Next, I create a new column called `'time_in_hours'` which is `'total_time_min'` divided by 60.

Next, I create a new DataFrame, `target`, which is `'time_in_hours'` multiplied by 15. This is our target payment if we pay strictly based on $15 per hour.

Next, I want to transform `target` to have a mean of 15 and a standard deviation of 7.5 (this could be any number we wanted, but 7.5 seemed to work well with both higher and lower than normal inputs)

This is done by `target = m2 + (target - m1) * s2/s1`
Where:

- `s1` = standard deviation of `target`
- `s2` = 7.5 (our chosen standard deviation)
- `m1` = mean of `target`
- `m2` = 15 (our chosen mean)

Now that our target data has been transformed we should be able to fit a model to obtain our desired output of a reduction in variance and a mean payment of $15 per hour.

I then fit a simple linear regression model to our two independent variables and I am able to use predict to determine what payment should be. I applied a 5th degree polynomial transformation because it seemed to get desired results (this may not be totally necessary.) A production version would need to be more complex to handle outliers and other non-expected values, but I think the basic principles of a coefficient multiplier on each of our features would remain the same.

As we can see, the model gets the desired output even at higher or lower than normal inputs

```
>>>newx = poly.transform([[60,60]])
>>>lm.predict(newx)
array([31.31572474])

>>>newx = poly.transform([[120,120]])
>>>lm.predict(newx)
array([62.57372587])

>>>newx = poly.transform([[1,1]])
>>>lm.predict(newx)
array([0.5786903])
```

At the risk of discrediting my own work or showing that I don't understand the question—it seems to me that if you want to build a model on how much to pay people, base it only on their time spent, and with the goal of paying them $15 an hour, it would be simpler just to take their total time spent in hours and multiply that by 15. This is essentially what my model is doing anyways, just in a more complicated way