Before you turn in the homework, make sure everything runs as expected. To do so, select **Kernel→Restart & Run All** in the toolbar above. Remember to submit both on **DataHub** and **Gradescope**.

Please fill in your name and include a list of your collaborators below.

In [1]:
```
NAME = "Junsheng Pei"
COLLABORATORS = ""
```

Project 2: NYC Taxi Rides

Extras

Put all of your extra work in here. Feel free to save figures to use when completing Part 4.

In [2]:
```
import os
import pandas as pd
import numpy as np
from pathlib import Path
from sqlalchemy import create_engine
from utils import timeit
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
!ls -lh /srv/db/taxi_2016_student_small.sqlite
```

```
-rw-r--r-- 1 root root 2.1G Nov  7 04:44 /srv/db/taxi_2016_student_small.sqlite
```

In [4]:
```
DB_URI = "sqlite:////srv/db/taxi_2016_student_small.sqlite"
TABLE_NAME = "taxi"

sql_engine = create_engine(DB_URI)
with timeit():
    print(f"Table {TABLE_NAME} has {sql_engine.execute(f'SELECT COUNT(*) FROM {TABLE_NAME}').first()[0]} rows!")
```

```
Table taxi has 15000000 rows!
1.15 s elapsed
```

Data Selection:

we chose data from April instead of January, and then we export the data

In [5]:
```
query = f"""
            SELECT *
            FROM (
            SELECT *
            FROM (
            SELECT *
            FROM (
            SELECT *
            FROM taxi
            WHERE tpep_pickup_datetime
                BETWEEN '2016-04-01' AND '2016-04-30'
                AND record_id % 100 == 0
            ORDER BY tpep_pickup_datetime
            )
            WHERE (julianday(tpep_dropoff_datetime) - julianday(tpep_pickup_datetime)) < 0.5
            )
            )
            WHERE passenger_count > 0
            """

with timeit(): # this query should take less than a second
    cleaned_df = pd.read_sql(query, sql_engine)
cleaned_df['tpep_pickup_datetime'] = pd.to_datetime(cleaned_df['tpep_pickup_datetime'])
cleaned_df['tpep_dropoff_datetime'] = pd.to_datetime(cleaned_df['tpep_dropoff_datetime'])
cleaned_df['duration'] = cleaned_df["tpep_dropoff_datetime"]-cleaned_df["tpep_pickup_datetime"]
cleaned_df['duration'] = cleaned_df['duration'].dt.total_seconds()
```

```
2.62 s elapsed
```

In [6]:
```
from sklearn.model_selection import train_test_split
train_df, val_df = train_test_split(cleaned_df, test_size=0.2, random_state=42)
data_file = Path("./", "cleaned_data_2016.hdf") # Path of hdf file
train_df.to_hdf(data_file, "train") # Train data of hdf file
val_df.to_hdf(data_file, "val") # Val data of hdf file
```

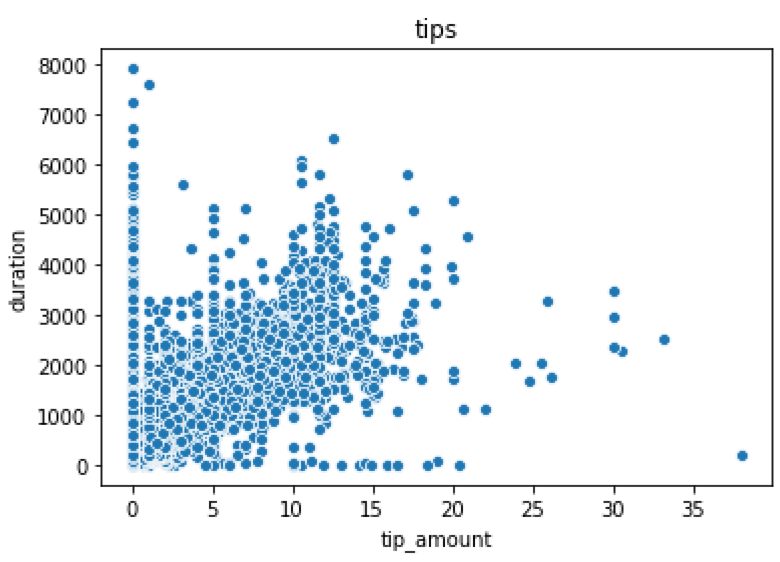
Better features

We test the the following features: 'tip_amount','trip_distance','ifdaytime','ifweekday','total_amount','fare_amount.

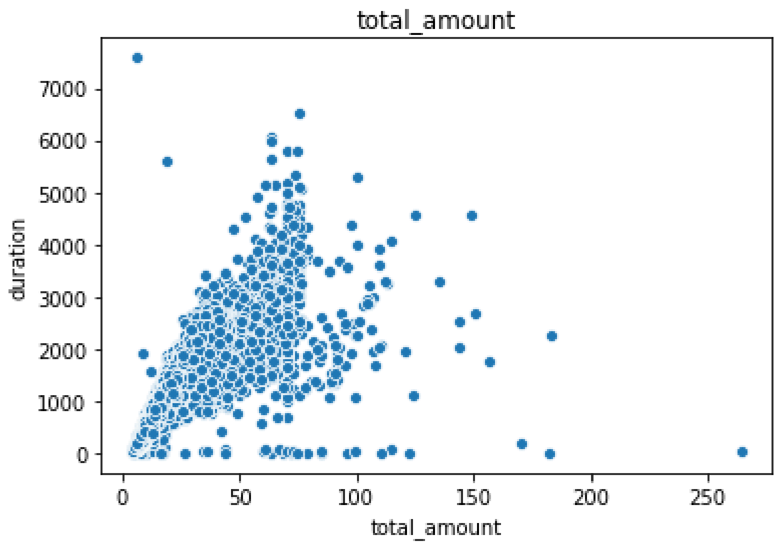
And we found those feature can help us predict duration

In [7]:
```
cleaned_df = cleaned_df[cleaned_df['duration'] < 10000]
```

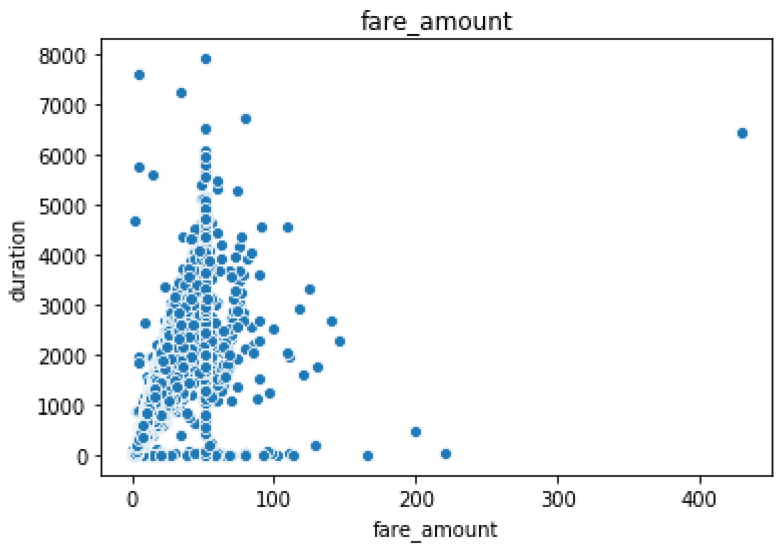
In [8]:
```
cleaned_df_feature = cleaned_df[cleaned_df['tip_amount'] < 40]
sns.scatterplot('tip_amount','duration', data = cleaned_df_feature)
plt.title('tips')
plt.show()
```



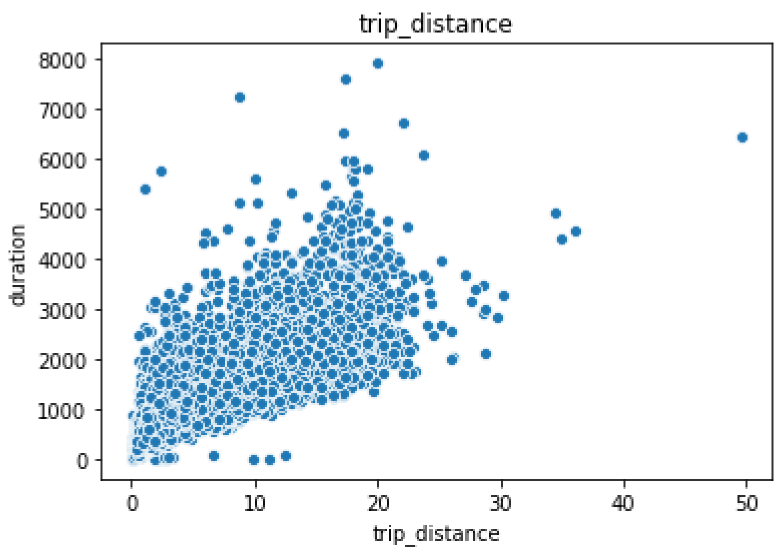
In [9]:
```
cleaned_df_feature = cleaned_df[cleaned_df['tip_amount'] > 0]
sns.scatterplot('total_amount','duration', data = cleaned_df_feature)
plt.title('total_amount')
plt.show()
```



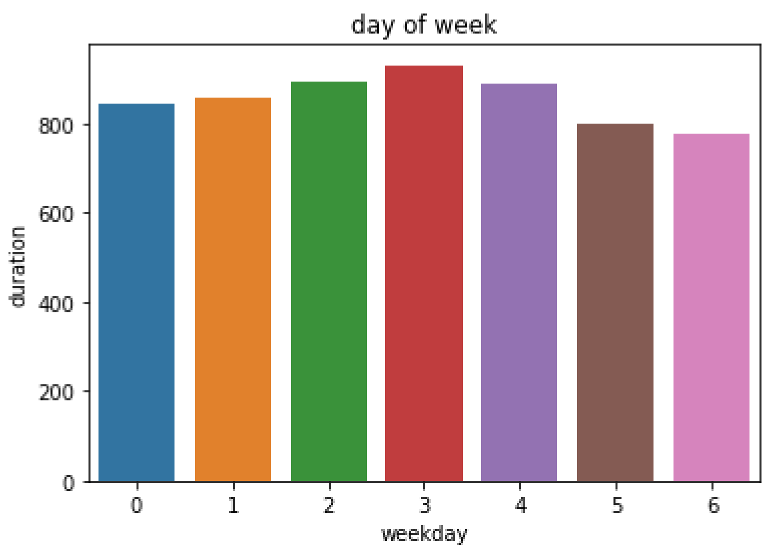
In [10]:
```
cleaned_df_feature = cleaned_df[cleaned_df['fare_amount'] > 0]
sns.scatterplot('fare_amount','duration', data = cleaned_df_feature)
plt.title('fare_amount')
plt.show()
```



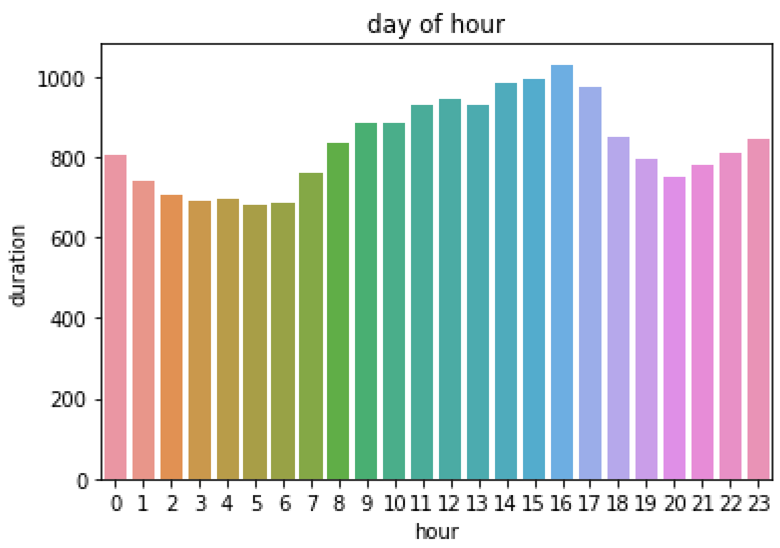
In [11]:
```
cleaned_df_feature = cleaned_df[cleaned_df['trip_distance'] > 0]
sns.scatterplot('trip_distance','duration', data = cleaned_df_feature)
plt.title('trip_distance')
plt.show()
```



In [12]:
```
cleaned_df['weekday'] =  cleaned_df['tpep_pickup_datetime'].dt.dayofweek
groupByweekday = cleaned_df.groupby('weekday')['duration'].mean()
sns.barplot(x =groupByweekday.index, y = groupByweekday)
plt.title('day of week')
plt.show()
```



In [13]:
```
cleaned_df['hour'] =  cleaned_df['tpep_pickup_datetime'].dt.hour
groupByhour = cleaned_df.groupby('hour')['duration'].mean()
sns.barplot(x =groupByhour.index, y = groupByhour)
plt.title('day of hour')
plt.show()
```



Feature Selection and Data Cleaning

In [14]:
```
data_file_fm = Path("./", "cleaned_data_2016.hdf")
train_df_fm = pd.read_hdf(data_file_fm, "train")
val_df_fm = pd.read_hdf(data_file_fm, "val")
```

In [15]:

```
# Copied from part 2
def haversine(lat1, lng1, lat2, lng2):
    """
    Compute haversine distance
    """
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    average_earth_radius = 6371
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * average_earth_radius * np.arcsin(np.sqrt(d))
    return h

# Copied from part 2
def manhattan_distance(lat1, lng1, lat2, lng2):
    """
    Compute Manhattan distance
    """
    a = haversine(lat1, lng1, lat1, lng2)
    b = haversine(lat1, lng1, lat2, lng1)
    return a + b

# Copied from part 2
def bearing(lat1, lng1, lat2, lng2):
    """
    Compute the bearing, or angle, from (lat1, lng1) to (lat2, lng2).
    A bearing of 0 refers to a NORTH orientation.
    """
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))

# Copied from part 2
def add_time_columns(df):
    """
    Add temporal features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'month'] = df['tpep_pickup_datetime'].dt.month
    df.loc[:, 'week_of_year'] = df['tpep_pickup_datetime'].dt.weekofyear
    df.loc[:, 'day_of_month'] = df['tpep_pickup_datetime'].dt.day
    df.loc[:, 'day_of_week'] = df['tpep_pickup_datetime'].dt.dayofweek
    df.loc[:, 'hour'] = df['tpep_pickup_datetime'].dt.hour
    df.loc[:, 'week_hour'] = df['tpep_pickup_datetime'].dt.weekday * 24 + df['hour']
    return df

# Copied from part 2
def add_distance_columns(df):
    """
    Add distance features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'manhattan'] = manhattan_distance(lat1=df['pickup_latitude'],
                                                lng1=df['pickup_longitude'],
                                                lat2=df['dropoff_latitude'],
                                                lng2=df['dropoff_longitude'])

    df.loc[:, 'bearing'] = bearing(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    df.loc[:, 'haversine'] = haversine(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    return df

def select_columns(data, *columns):
    return data.loc[:, columns]

def mae(actual, predicted):
    """
    Calculates MAE from actual and predicted values
    Input:
      actual (1D array-like): vector of actual values
      predicted (1D array-like): vector of predicted/fitted values
    Output:
      a float, the MAE
    """

    mae = np.mean(np.abs(actual - predicted))
    return mae
```

In [16]:

```
def add_ifdaytime(data):
    data['ifdaytime'] = (data['hour'] >= 8) & (data['hour'] <= 18)
    return data

def add_ifweekday(data):
    data['ifweekday'] = data['day_of_week'] > 4
    return data

def drop_outlier(data, col, _filter):
    return data.loc[data[col][lambda x: _filter(x)].index]

def replace_outlier(data, col, _filter):
    mean = data[col][lambda x : _filter(x)].mean()
    data[col] = data[col].apply(lambda x : x if _filter(x) else mean)
    return data
```

In [17]:
```
def process_data_fm(data, test=False):
    # Put your final pipeline here

    # data cleaning
    if(test):
        clean_data = replace_outlier
    else:
        clean_data = drop_outlier
        data =  clean_data(data,'duration', lambda x : (x < 8000) & x > 0)

    filter_latitude = lambda x : (x >= 40.63) & (x <= 40.85)

    data = clean_data(data,'pickup_latitude', filter_latitude )
    data = clean_data(data,'dropoff_latitude', filter_latitude )

    filter_longitude  = lambda x : (x >= -74.03) & (x <= -73.75)

    data = clean_data(data,'pickup_longitude', filter_longitude)
    data = clean_data(data,'dropoff_longitude', filter_longitude )

    data = clean_data(data,'total_amount', lambda x: (x>0) & (x <= 90))
    data = clean_data(data,'fare_amount', lambda x: (x>0) & (x <= 80))
    data = clean_data(data,'tip_amount', lambda x :(x>0) & (x <= 20) )

    data = clean_data(data, 'trip_distance', lambda x : x < 50)

    X = (
        data
        # Transform data
        .pipe(add_time_columns)
        .pipe(add_distance_columns)
        .pipe(add_ifdaytime)
        .pipe(add_ifweekday)
        .pipe(select_columns,
              'pickup_longitude',
              'pickup_latitude',
              'dropoff_longitude',
              'dropoff_latitude',
              'manhattan',
              'tip_amount',
              'haversine',
              'trip_distance',
               'ifdaytime',
              'ifweekday',
              'total_amount',
              'fare_amount',
             )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y

# YOUR CODE HERE
#raise NotImplementedError()
```

Parameter Section for Ridge and Lasso and Model Selection

In [18]:
```
import sklearn.linear_model as lm
```

In [19]:
```
# Parameter Section for Ridge
_lambdas = [0.01,0.1,0.2,0.5,1,2,3,5,10]
X_train_fm, y_train_fm = process_data_fm(train_df_fm)
X_val_fm, y_val_fm = process_data_fm(val_df_fm)

#final_model = lm.LinearRegression(fit_intercept=True)
for _lambda in _lambdas:
    final_model = lm.Ridge(alpha = _lambda, fit_intercept=True)

    # Define your final model here, feel free to try other forms of regression
    final_model.fit(X_train_fm, y_train_fm)

    y_val_pred_fm = final_model.predict(X_val_fm)

    print("for lambda " + str(_lambda) + ", validation accuracy: " + str(mae(y_val_pred_fm,y_val_fm)))
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)

for lambda 0.01, validation accuracy: 122.773109937
for lambda 0.1, validation accuracy: 122.715747715
for lambda 0.2, validation accuracy: 122.655868278
for lambda 0.5, validation accuracy: 122.502909548
for lambda 1, validation accuracy: 122.325242267
for lambda 2, validation accuracy: 122.109814835
for lambda 3, validation accuracy: 122.027044081
for lambda 5, validation accuracy: 122.060878093
for lambda 10, validation accuracy: 122.361729584
```

In [20]:

```python
# Parameter Section for Ridge
_lambdas = [0.01,0.1,0.2,0.5,1,2,3,5,10]
X_train_fm, y_train_fm = process_data_fm(train_df_fm)
X_val_fm, y_val_fm = process_data_fm(val_df_fm)

#final_model = lm.LinearRegression(fit_intercept=True)
for _lambda in _lambdas:
    final_model = lm.Lasso(alpha = _lambda, fit_intercept=True)

    # Define your final model here, feel free to try other forms of regression
    final_model.fit(X_train_fm, y_train_fm)

    y_val_pred_fm = final_model.predict(X_val_fm)

    print("for lambda " + str(_lambda) + ", validation accuracy: " + str(mae(y_val_pred_fm,y_val_fm)))
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
/srv/conda/envs/data100/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Fitting data with very small alpha may cause precision problems.
  ConvergenceWarning)
```

```
for lambda 0.01, validation accuracy: 122.720904993
for lambda 0.1, validation accuracy: 122.359093687
for lambda 0.2, validation accuracy: 122.2259669
for lambda 0.5, validation accuracy: 122.37614613
for lambda 1, validation accuracy: 124.429678243
for lambda 2, validation accuracy: 124.532659775
for lambda 3, validation accuracy: 124.788272559
for lambda 5, validation accuracy: 125.625109785
for lambda 10, validation accuracy: 128.680014266
```

So we find using Ridge regression with lambda = 3 is the best

Submission

You're almost done!

Before submitting this assignment, ensure that you have:

1. Restarted the Kernel (in the menubar, select Kernel→Restart & Run All)
2. Validated the notebook by clicking the "Validate" button.

Then,

1. **Submit** the assignment via the Assignments tab in **Datahub**
2. **Upload and tag** the manually reviewed portions of the assignment on **Gradescope**