Structure of the project

1. `README.md`
   This is the README.md file that explains basically everything one should know about the project, including its structure, how to run it, what's in each file, etc.
2. `__init__`.py
   This is the file that makes Python know that other Python files in the current directory can be considered a package, so that we can import them into other files later.
3. requirements.txt
   This file lists out all the dependencies that we need to install in order to run the project.
4. `create.py`
   This is where all the models for the tables in the database are created.
5. `insert_data.py`
   This is where all the data is inserted into the tables.
6. `query_data.py`
   This is where all the queries are made.

How to run the project

`$ virtualenv venv` - to create a virtual environment

`$ venv/Scripts/activate` - to activate the virtual environment

`$ pip install -r requirements.txt` - to install all the necessary packages

`$ python create.py` - to create the database and the tables in it

`$ python insert_data.py` - to insert the data into the database

`$ python query_data.py` - to query the database

Structure of the database (`create.py`)

There are 8 tables in the database:

- Agent stores all info about the agents, the primary key is agent_id;
- Office stores all info about the offices, the primary key is office_id;
- AgentOffice is the association table to demonstrate the many-to-many relationship between agents and offices;
- Listing stores all the info about the listings, the primary key is listing_id, the foreign keys are agent_id and office_id;
- Sale stores all the info about the sold houses, the primary key is sale_id, the foreign keys are listing_id, agent_id;
- Commission stores the commission that corresponds to each sale, the primary key is sale_id, the foreign key is agent_id;
- AgentCom stores the monthly commissions for each agent, the primary key is agent_id
- Summary stores the total number of sales and the total prices over time

Some assumptions about the system:

- Each property has one listing agent, and if sold, one selling agent.
- The listing agent for one property is not necessarily the same as the selling agent, as long as they all work at the same office in the area of the property (i.e., sharing the same zip code).
- Each area with a distinct zip code only has one office.

Most the tables are in standard 4th normalization form (4NF). Take the table Sale for example. It is in the first normalization form (1NF) because each cell only has one value, all the entries in the same field are the same data type, and each row is uniquely identified with a primary key. It is in the second normalization form because it is already in 1NF, and all the non-key columns are dependent on the key column. If we had added the number of bedrooms to the Sale table, this would have been violated, because the number of bedroom depends on the actual property which is the listing. The table is in the third normal form (3NF)

because it is in 2NF, and all fields only depend on the key, not other fields. For example, by looking at the the sale price or the sale agent, we wouldn't be able to figure out the sale date. Finally, it is in 4NF because it is already in 3NF, and there are no multi-valued dependencies. This would have been violated if I had added agent's emails to the table, so if the agent was changed, the email would have to change. That's why we have a separate table to store all info about agents.

The Listing table is not in 4NF because we have zip_code and office_id in it, which violated the last rule, and it is kinda redundant because from the office_id we can totally figure out the zip_code by using the Office table. But in the instructions, it is said that whenever a listing is made, both info should be captured. This could be useful if an area of a zip code had multiple offices.

It is important to notice that a Commission table was created outside of the Sale table (even though it would make total sense to merge them into one). The reason I separated them was because of the different commission rate. It would be tedious and not efficient to hard-code the commission given a selling price if we had such different rates, and because we can't do the calculations when we insert the data about the sale, a separate commission table was needed. In practice, if we had a mechanism such that once a property is sold, all the info is stored, together with the commission already calculated somewhere before it was put into the database, then we wouldn't need 2 tables. Then I created AgentCom table to store the total commission of each agent for that particular month as instructed.

Inserting and updating (`insert_data.py`)

Each change created to the database, either insertion or update, is wrapped in a transaction with the following structure:

```
try:
  <insertion/update>
  session.commit()
except:
  session.rollback()
  raise
finally:
  session.close()
```

This is to ensure that unless everything in the transaction is successfully carried out, nothing will be carried out and saved at all. This is to ensure the consistency of the database, so that for example, once a house is sold, it would be marked as sold in the Listing table. Without such a transaction, if a disruption happens in the process, such a house might appear in the Sale table but its status in the Listing won't be updated yet.

Querying (`query_data.py`)

All the data is queried for March, 2020 only. All the query functions are neatly constructed and very simple to understand. Most of them take y and m as parameters because we want to do this report monthly, so it makes sense to have the year and the month as parameters so we could easily filter out only the sales that happen in that month.

```python
import sqlalchemy, datetime
from datetime import datetime, date
from sqlalchemy import create_engine, Table, Column, Text, Integer,
ForeignKey, Float, DateTime, MetaData, func, extract
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker

# Create and connect engine to the database
engine = create_engine('sqlite:///:memory:', echo = True)
engine.connect()
connection = engine.connect()
metadata = MetaData()
Base = declarative_base()

# Define Listing table
class Listing(Base):
    __tablename__ = 'Listing'
    listing_id = Column(Integer, primary_key=True)
    seller_id = Column(Integer)
    num_bed = Column(Integer)
    num_bath = Column(Integer)
    listing_price = Column(Float(15, 2))
    zip_code = Column(Integer)
    listing_date = Column(DateTime)
    agent_id = Column(Integer, ForeignKey('Agent.agent_id'))
    office_id = Column(Integer, ForeignKey('Office.office_id'))
    status = Column(Text)

    def __repr__(self):
        return '<Listing(id: {0}, seller: {1}, date: {2}, agent: {3},
        status: {4})>'.format(self.listing_id,\
        self.seller_id, self.listing_date, self.agent_id, self.status)

agent_office = Table('AgentOffice', Base.metadata,
    Column('agent_id', Integer, ForeignKey('Agent.agent_id')),
    Column('office_id', Integer, ForeignKey('Office.office_id'))
)

# Create Office table
class Office(Base):
    __tablename__ = 'Office'
    office_id = Column(Integer, primary_key=True)
    zip_code = Column(Integer)
    listing = relationship("Listing")
    agent = relationship("Agent", secondary='AgentOffice')
    def __repr__(self):
        return '<Office(id: {0}, zip_code: {1})>'.format(self.office_id,
        self.zip_code)

# Create Agent table
class Agent(Base):
    __tablename__ = 'Agent'
    agent_id = Column(Integer, primary_key=True)
    name = Column(Text)
    email = Column(Text)
```

```
53      phone = Column(Text)
54      listing = relationship("Listing")
55      office = relationship("Office", secondary='AgentOffice')
56      commission = relationship("Commission")
57      agentcom = relationship("AgentCom")
58
59      def __repr__(self):
60          return '<Agent(id: {0},name: {1},email: {2})>'\
61          .format(self.agent_id, self.name, self.email)
62
63  # Create AgentCom table
64  class AgentCom(Base):
65      __tablename__ = 'AgentCom'
66      agent_id = Column(Integer, ForeignKey('Agent.agent_id'),
•     primary_key=True)
67      monthly_com = Column(Float(15, 2))
68
69      def __repr__(self):
70          return '<AgentCom(id: {0}, monthly_com:
•         {1})>'.format(self.agent_id, self.monthly_com)
71
72  # Create Sale table
73  class Sale(Base):
74      __tablename__ = 'Sale'
75      sale_id = Column(Integer, primary_key=True)
76      listing_id = Column(Integer, ForeignKey('Listing.listing_id'))
77      sale_price = Column(Float(15,2))
78      sale_date = Column(DateTime)
79      agent_id = Column(Integer, ForeignKey('Agent.agent_id'))
80      commission = relationship('Commission')
81
82      def __repr__(self):
83          return '<Sale(id: {0}, listing: {1}, price: {2},
•         date:{3})>'.format(self.sale_id,\
84          self.listing_id, self.sale_price, self.sale_date)
85
86  # Create Commission table
87  class Commission(Base):
88      __tablename__ = 'Commission'
89      sale_id = Column(Integer, ForeignKey('Sale.sale_id'), primary_key=True)
90      agent_id = Column(Integer, ForeignKey('Agent.agent_id'))
91      commission = Column(Float(15, 2))
92
93      def __repr__(self):
94          return '<Commission(sale: {0}, agent: {1}, commission:
•         {2})>'.format(self.sale_id, self.agent_id, self.commission)
95
96  # Create Summary table
97  class Summary(Base):
98      __tablename__ = 'Summary'
99      id = Column(Integer, primary_key=True)
100     num_sale = Column(Integer)
101     tot_price = Column(Float(15,2))
102
103     def __repr__(self):
104         return '<Summary(num_sale: {0}, total_price:
```

```
104             return '<Summary(num_sale: {0}, total_price:
  •             {1})>'.format(self.num_sale, self.tot_price)
105
106     Base.metadata.create_all(engine)
107
```

```python
from create import Agent, Office, Sale, Listing, Commission, Summary,
engine, AgentCom
import sqlalchemy, datetime
from datetime import datetime, date
from sqlalchemy import create_engine, Table, Column, Text, Integer,
ForeignKey, Float, DateTime, MetaData, func, extract
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker

# Make a session
Session = sessionmaker(bind=engine)
session = Session()
metadata = MetaData()

# Define commission calculation
def commission_cal(x):
    if x < 100000:
        return x*0.01
    elif x < 200000:
        return x*0.075
    elif x < 500000:
        return x*0.06
    elif x < 1000000:
        return x*0.05
    else:
        return x*0.04

# Transaction - insert initial value to the summary
try:
    session.add(Summary(id=1, num_sale=0,tot_price=0))
    session.commit()
except:
    session.rollback()
    raise
finally:
    session.close()

# Transaction - insert agent and office info
try:
    agent_data = [Agent(agent_id=1, name='Harvey Spector',
    email='s.harvey@gmail.com', phone='123-456-789'),\
                Agent(agent_id=2, name='Donna Paulsen',
                email='p.donna@gmail.com', phone='789-456-123'),\
                Agent(agent_id=3, name='Louis Litt',
                email='l.louis@gmail.com', phone='123-456-123'),\
                Agent(agent_id=4, name='Mike Ross',
                email='r.mike@gmail.com', phone='000-456-123'),\
                Agent(agent_id=6, name='Jessica Pearson',
                email='p.jessica@gmail.com', phone='000-416-993'),
                Agent(agent_id=7, name='Rachel Zane',
                email='z.rachel@gmail.com', phone='000-459-993'),
                Agent(agent_id=8, name='Norma',
                email='p.jessica@gmail.com', phone='000-456-983')]

    office_data = [Office(office_id=1, zip_code=555, agent=[agent_data[0],
```

```
    agent_data[1]]),\
    Office(office_id=2, zip_code=777, agent=[agent_data[1],
    agent_data[2]]),\
    Office(office_id=3, zip_code=999, agent=[agent_data[2],
    agent_data[3]]),\
    Office(office_id=4, zip_code=333, agent=[agent_data[0], agent_data[3],
    agent_data[4]]),
    Office(office_id=5, zip_code=111, agent=[agent_data[5],
    agent_data[6]]),
    Office(office_id=6, zip_code=222, agent=[agent_data[4], agent_data[6],
    agent_data[2]])]

    session.add_all(agent_data)
    session.add_all(office_data)
    session.commit()
except:
    session.rollback()
    raise
finally:
    session.close()

# Transaction - insert listing data
try:
    listing_data = [Listing(listing_id=1,
    seller_id=1,num_bed=2,num_bath=1,listing_price=90000.43,zip_code=555,
    listing_date=datetime.strptime('2018-03-03 20:59:29', '%Y-%m-%d
    %H:%M:%S'), agent_id=1,office_id=1,status='Available'),\
    Listing(listing_id=2,
    seller_id=2,num_bed=2,num_bath=2,listing_price=150000.43,zip_code=555,
    listing_date=datetime.strptime('2019-04-06 20:59:29', '%Y-%m-%d
    %H:%M:%S'), agent_id=2,office_id=1,status='Available'),\
    Listing(listing_id=3,
    seller_id=3,num_bed=3,num_bath=2,listing_price=400000.43,zip_code=777,l
    isting_date=datetime.strptime('2019-04-07 20:59:29', '%Y-%m-%d
    %H:%M:%S'), agent_id=3,office_id=2,status='Available'),\
    Listing(listing_id=4,
    seller_id=4,num_bed=3,num_bath=3,listing_price=900000.43,zip_code=555,
    listing_date=datetime.strptime('2019-04-08 20:59:29', '%Y-%m-%d
    %H:%M:%S'), agent_id=1,office_id=1,status='Available'),\
    Listing(listing_id=5,
    seller_id=5,num_bed=4,num_bath=3,listing_price=1300000.43,zip_code=333,
     listing_date=datetime.strptime('2019-04-09 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=5,office_id=4,status='Available'),\
    Listing(listing_id=6,
    seller_id=6,num_bed=2,num_bath=1,listing_price=130000.43,zip_code=999,
    listing_date=datetime.strptime('2019-05-01 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=4,office_id=3,status='Available'),\
    Listing(listing_id=7,
    seller_id=7,num_bed=3,num_bath=3,listing_price=900000.43,zip_code=999,
    listing_date=datetime.strptime('2019-05-02 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=4,office_id=3,status='Available'),\
    Listing(listing_id=8,
    seller_id=8,num_bed=3,num_bath=1,listing_price=1000000.43,zip_code=333,
     listing_date=datetime.strptime('2019-05-03 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=5,office_id=4,status='Available'),\
    Listing(listing_id=9,
```

```
    Listing(listing_id=9,
    seller_id=9,num_bed=2,num_bath=2,listing_price=800000.43,zip_code=555,
    listing_date=datetime.strptime('2019-05-04 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=2,office_id=1,status='Available'),\
    Listing(listing_id=10,
    seller_id=10,num_bed=2,num_bath=3,listing_price=1500000.43,zip_code=777
    , listing_date=datetime.strptime('2019-05-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=3,office_id=2,status='Available'),\
    Listing(listing_id=11,
    seller_id=11,num_bed=2,num_bath=2,listing_price=950000.43,zip_code=111,
     listing_date=datetime.strptime('2019-06-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=6,office_id=5,status='Available'),\
    Listing(listing_id=12,
    seller_id=12,num_bed=4,num_bath=3,listing_price=1400000.43,zip_code=111
    , listing_date=datetime.strptime('2019-07-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=7,office_id=5,status='Available'),\
    Listing(listing_id=13,
    seller_id=13,num_bed=4,num_bath=2,listing_price=1250000.43,zip_code=222
    , listing_date=datetime.strptime('2019-08-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=3,office_id=6,status='Available'),\
    Listing(listing_id=14,
    seller_id=14,num_bed=2,num_bath=1,listing_price=400000.43,zip_code=222,
     listing_date=datetime.strptime('2019-09-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=5,office_id=6,status='Available'),\
    Listing(listing_id=15,
    seller_id=15,num_bed=3,num_bath=1,listing_price=970000.43,zip_code=222,
     listing_date=datetime.strptime('2019-10-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=7,office_id=6,status='Available'),\
    Listing(listing_id=16,
    seller_id=16,num_bed=1,num_bath=1,listing_price=200000.43,zip_code=333,
     listing_date=datetime.strptime('2019-11-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=4,office_id=4,status='Available'),\
    Listing(listing_id=17,
    seller_id=17,num_bed=2,num_bath=3,listing_price=1200000.43,zip_code=999
    , listing_date=datetime.strptime('2019-12-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'),agent_id=3,office_id=3,status='Available')]

    session.add_all(listing_data)
    session.commit()
except:
    session.rollback()
    raise
finally:
    session.close()

# Transaction - insert sale data
try:
    sale_data = [Sale(sale_id=1, listing_id=1,sale_price=85000,
    sale_date=datetime.strptime('2019-05-05 20:59:29', '%Y-%m-%d
    %H:%M:%S'), agent_id=2),\
    Sale(sale_id=2, listing_id=3,sale_price=390000,
    sale_date=datetime.strptime('2020-03-03 20:59:29', '%Y-%m-%d
    %H:%M:%S'), agent_id=2),\
    Sale(sale_id=3, listing_id=4,sale_price=850000,
    sale_date=datetime.strptime('2020-03-06 20:59:29', '%Y-%m-%d
    %H:%M:%S'), agent_id=1),\
```

```
        %H:%M:%S'), agent_id=1),\
95      Sale(sale_id=4, listing_id=5,sale_price=1200000,
        sale_date=datetime.strptime('2020-03-07 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=5),\
96      Sale(sale_id=5, listing_id=6,sale_price=120000,
        sale_date=datetime.strptime('2020-03-08 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=4),\
97      Sale(sale_id=6, listing_id=7,sale_price=850000,
        sale_date=datetime.strptime('2020-03-09 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=3),\
98      Sale(sale_id=7, listing_id=8,sale_price=950000,
        sale_date=datetime.strptime('2020-03-10 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=1),\
99      Sale(sale_id=8, listing_id=9,sale_price=750000,
        sale_date=datetime.strptime('2020-03-11 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=2),\
100     Sale(sale_id=9, listing_id=10,sale_price=1400000,
        sale_date=datetime.strptime('2020-03-12 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=3),\
101     Sale(sale_id=10, listing_id=11,sale_price=900000,
        sale_date=datetime.strptime('2020-03-15 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=6),\
102     Sale(sale_id=11, listing_id=12,sale_price=1330000,
        sale_date=datetime.strptime('2020-03-17 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=7),\
103     Sale(sale_id=12, listing_id=13,sale_price=1200000,
        sale_date=datetime.strptime('2020-03-19 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=3),\
104     Sale(sale_id=13, listing_id=14,sale_price=390000,
        sale_date=datetime.strptime('2020-03-28 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=7),\
105     Sale(sale_id=14, listing_id=15,sale_price=950000,
        sale_date=datetime.strptime('2020-03-30 20:59:29', '%Y-%m-%d
        %H:%M:%S'), agent_id=5)]
106
107     session.add_all(sale_data)
108
109     # update the summary table
110     total_sale = session.query(func.sum(Sale.sale_price)).scalar()
111     summary = session.query(Summary).first()
112     summary.num_sale += len(sale_data)
113     summary.tot_price += total_sale
114
115     # insert commission to the commission table for each sale
116     for sale in session.query(Sale).all():
117         session.add(Commission(sale_id=sale.sale_id,
            agent_id=sale.agent_id,
            commission=commission_cal(sale.sale_price)))
118     # update listing status
119     listing_sold =
        session.query(Listing).join(Sale).filter(Listing.listing_id==Sale.listi
        ng_id).all()
120     for each in listing_sold:
121         each.status = 'Sold'
122
123     session.commit()
```

```
except:
    session.rollback()
    raise
finally:
    session.close()

```

```
from create import Agent, AgentCom, Office, Sale, Listing, Commission,
Summary, engine, metadata
from insert_data import session, engine
import sqlalchemy, datetime
from datetime import datetime, date
from sqlalchemy import create_engine, Table, Column, Text, Integer,
ForeignKey, Float, DateTime, MetaData, func, extract
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker

# Actually create the tables
Agent = Table('Agent', metadata, autoload=True, autoload_with=engine)
Office = Table('Office', metadata, autoload=True, autoload_with=engine)
Listing = Table('Listing', metadata, autoload=True, autoload_with=engine)
Sale = Table('Sale', metadata, autoload=True, autoload_with=engine)
Commission = Table('Commission', metadata, autoload=True,
autoload_with=engine)
Summary = Table('Summary', metadata, autoload=True, autoload_with=engine)

# Print out all the tables
print('Listing Table', session.query(Listing).all())
print('-----------')
print('Office Table', session.query(Office).all())
print('-----------')
print('Agent Table', session.query(Agent).all())
print('-----------')
print('Sale Table', session.query(Sale).all())
print('-----------')
print('Commission Table', session.query(Commission).all())
print('-----------')
print('Summary Table', session.query(Summary).all())
print('-----------')

y = 2020
m = 3
n = 5

# Top 5 offices with most sales for a particular month
def top_office(y, m, n):
    print('Top 5 offices with most sales',
    session.query(Listing.columns.office_id,\
    func.count(Listing.columns.office_id)).join(Sale).\
    filter(Sale.columns.listing_id==Listing.columns.listing_id).\
    filter(extract('year', Sale.columns.sale_date) ==
    2020).filter(extract('month', Sale.columns.sale_date) == 3).\
    group_by(Listing.columns.office_id).order_by(func.count(Listing.columns
    .office_id).desc()).all()[:n])
top_office(y, m, n)
print('-----------')

# Top 5 agents with mose sales and their info
def top_agent(y, m, n):
    print('Top 5 agents with mose sales and their info',
    session.query(Agent.columns.agent_id, Agent.columns.email,\
    func.count(Agent.columns.agent_id)).join(Sale).\
```

```
    filter(Sale.columns.agent_id==Agent.columns.agent_id).\
    filter(extract('year', Sale.columns.sale_date) ==
    y).filter(extract('month', Sale.columns.sale_date) == m).\
    group_by(Agent.columns.agent_id).order_by(func.count(Agent.columns.agen
    t_id).desc()).all()[:n])
top_agent(y, m, n)
print('-----------')

# Calculate the commission for each agent and store in a separate table
def agent_com(y, m):
    agent_com = session.query(Sale.columns.agent_id,
    func.sum(Commission.columns.commission)).\
    join(Commission).filter(Sale.columns.agent_id==Commission.columns.agent
    _id).\
    filter(extract('year', Sale.columns.sale_date) ==
    y).filter(extract('month', Sale.columns.sale_date) == m).\
    group_by(Sale.columns.agent_id).order_by(func.sum(Commission.columns.co
    mmission).desc()).all()

    print('Agents and commission', agent_com)

    agent_ls = []
    for agent in agent_com:
        agent_ls.append(AgentCom(agent_id=agent[0], monthly_com=agent[1]))
    try:
        session.add_all(agent_ls)
        session.commit
    except:
        session.rollback()
        raise
    finally:
        session.close()

    Agentcom = Table('AgentCom', metadata, autoload=True,
    autoload_with=engine)

agent_com(y, m)
print('-----------')

# For all houses that were sold that month, calculate the average number
of days that the house was on the market.
def avg_time(y, m):
    houses = session.query(Listing.columns.listing_date,
    Sale.columns.sale_date).join(Sale).\
    filter(Listing.columns.listing_id==Sale.columns.listing_id).\
    filter(extract('year', Sale.columns.sale_date) ==
    y).filter(extract('month', Sale.columns.sale_date) == m).all()
    time_on_market = 0
    for house in houses:
        time_on_market += (house[1]-house[0]).days
    print('Average time on market', time_on_market / len(houses))
avg_time(y, m)
print('-----------')

# For all houses that were sold that month, calculate the average selling
```

```
price
def avg_price(y, m):
    print('Average selling price',
    session.query(func.avg(Sale.columns.sale_price)).\
    filter(extract('year', Sale.columns.sale_date) == y).\
    filter(extract('month', Sale.columns.sale_date) == m).all()[0][0])
avg_price(y, m)
print('-----------')

# Find the zip codes with the top 5 average sales prices
def top_area(y, m):
    print('Zip codes with the top 5 average sales prices',
    session.query(Listing.columns.zip_code,
    func.avg(Sale.columns.sale_price)).\
    filter(Listing.columns.listing_id==Sale.columns.listing_id).\
    filter(extract('year', Sale.columns.sale_date) ==
    y).filter(extract('month', Sale.columns.sale_date) == m).\
    group_by(Listing.columns.zip_code).order_by(func.avg(Sale.columns.sale_
    price).desc()).all()[:n])
top_area(y, m)
print('-----------')

```