Q.2 Implementing search algorithms that provide relevant results is known to be a challenging task because of the complexity of any language. Please complete the below:

Part-1:

Return relevant results considering the following points:

- Different words, sometimes, refer to a common concept.
  Eg.: men, man;
  tree, trees;
- The same word has different meanings given the context.
  Eg: Free parking (no cost for the parking);
  Meat free restaurant (no meat dishes served).

Explain theoretically, (without implementation) how would you use machine learning concepts and techniques to deliver relevant results considering the above points in a search engine context.

Solut'n:

Machine learning(ML) is where a computer teaches itself how to do something, rather than being taught by humans or by following detailed programmings.

Step 1: **Query understanding**

Machine Learning can be used for understanding the search query typed by the user.

Paying more attention to each word in a query coming from user is very important. Ensuring that the whole query (the whole sentence or conversation or meaning) is taken into account, rather than focusing on particular words in our search algorithm.

In query we have to focus on few problems, which are solved by machine learning like:

- Query classification
- Spelling suggestion or spelling correction
- Synonyms / Query expansion (synonyms to expand the query keywords and expand the result set
- Intent disambiguation

Our problem is of **Synonyms/ Query Expansion.**

Step 2: **Candidate Generation**

At this step, given a word, we generate all possible candidates that might be synonyms for the word.

Note: what we mean by "synonyms" usually changes a lot based on domain.

For a search engine, our definition of a synonyms can be much broader and include any alternate words that will help us get better search results for our query.

Eg.: Acronym expansions [CS = "computer science"],
synonyms for proper nouns ["city of gold" = "Dubai"],
or even entire phrase substitutions are good "synonyms" for a search engine. Depending on our exact application, here are some of the sources we can use for synonym generation:

1) **Word embeddings:** we can train word vectors for our corpus and then find synonyms for the current word by using nearest neighbors or by defining some notion of "similarity".

   eg. Word2Vec outputs

2) **Historical user data:** we can also look at historical user behavior and generate synonym candidates from that. A simple way to do this for search engines is to look at word and phrase substitutions by looking for the "query, query, click" pattern. So if we see users searching for [Hotels in Dubai], not clicking on anything, then searching for [rooms in Dubai] and clicking on a result, we can consider "rooms" to be a synonym candidate for "hotels". The simple version of this ignores the context (in this case, the previous word being "Dubai"), but to get more accurate synonyms, we do want to use the previous and next *n* words as context. For word processors, we can similarly look at word and phrase substitutions that our users do.

3) **Lexical synonyms:** These are grammatical synonyms as defined by the rules of the language. Wordnet is a popular source for these among others.

If there are other synonym sources outside of these that are better suited for our application, we should use them too.

We can use either of these techniques in isolation to solve our problem. Eg. a synonym generation algorithm using word2vec vectors alone might be sufficient for us. But by using just one source we will miss out on the strengths that the other sources offer.

Step 3: **Synonym detection**

Now that we have a set of synonym candidates for a given word, we need to find out which ones of those are actually synonyms. This can be solved as a classical **supervised learning** problem.

Given our candidate set, we can generate ground-truth training data either using human judges or past user engagement. As I mentioned above, the definition of synonyms is different for different applications, so if we are using human judges, we will need to come up with clear guidelines on what makes a good synonym for them to use.

Once we have a labeled training set, we can generate various lexical and statistical features for our data and train a supervised ML model of our choice on it.

Note: Any features associated with past user behavior such as word substitution, word frequency perform the best for synonyms.

-- continue for **Part-2**

Part-2.

A user might engage with the insydo website through different kind of activities.

- How to take into consideration his activities to recommend relevant content and a personalized experience.
- What activities should be tracked?

Solut’n:

Most Online services use artificial intelligence to personalize user experience. User Activities should be tracked to achive a recomandation engin. Typically, there are two types of entities involved in a recommender system (RS), let’s call them *users* and *items*. we would like to make recommendations on Items which are the things we would like to recommend to them such as Eat, Drink, Party, Adventure, Explore, , Look Good, Kids, Watch, Travel, and News on ‘Insydo’.

A recommendation engine typically processes data through four phases:
1. Collection
2. Storage
3. Analysis
4. Recomandation

1. Collection

A recommendation engine can collect data about users based on their implicit *behavior* or their explicit *input* through searching or "query, query, click" pattern.

Behavioral data is easy to collect because we can keep logs of user activities. Collecting this data is also straightforward because it doesn't require any additional action from the user as they are already using the application. In this approach it's harder to analyze or filtering the interesting logs from the less-interesting users.

Inputted data can be harder to collect because users need to take additional actions, such as writing a review or feedback or experience of the place. Users might not want to provide this data for a variety of reasons. But when it comes to understanding user preferences, such results are quite precise.

2. Storage

The more data we can make available to our algorithms, the better the recommendations will be. This means that any recommendations project can quickly turn into a big data project.

The type of data that we use to create recommendations can help us to decide the type of storage to use. We could choose to use a NoSQL database like MongioDB, or standard SQL database like MySQl, or even some kind of object storage like IBM Cloud object storage.

Each of these options is viable depending on whether we're capturing user input or behavior and on factors such as ease of implementation, the amount of data that the storage can manage, integration with the rest of the environment, and portability.

When saving user ratings or events, a scalable and managed database minimizes the amount of operational tasks needed and helps to focus on the recommendation.

3. Analysis

The first factor to consider in analyzing the data is how quickly we need to present the recommendations to the user.

If we want to present recommendations immediately, such as when the user is viewing a product, we will need a more quick and light in action, type of analysis. Eg., If we want to send the tourist an email that contains recommendations at a later date.

We can follow *Near-real-time* analysis lets us gather data quickly so we can refresh the analytics every few minutes or seconds. A near-real-time system might be good for providing recommendations during the same browsing session.

The platform running the analysis could work directly from a database where the data is saved or on a dump saved periodically in persistent storage.

Another most important aspect of analysis after above disscussed things is ***filtering***. The most common filterning approaches includes:

- **Content-based**: A popular, recommended facts has similar attributes to what the user views or likes.
- **Cluster**: Recommended facts go well together, no matter what other users have done.
- **Collaborative**: Other users, who like the same products the user views or likes, also liked a recommended facts.

Collaborative filtering output is based on the assumption that two different users who liked the same item in the past will probably like the same ones now also is what i feel to be done.

As a Analytic approach, we can represent data about **ratings** or **interactions** as a set of matrices, with iteams and users as dimensions(m rows * n columns). Collaborative filtering tries to predict the missing cells in a matrix for a specific user-product pair.

The process of **Training** the models will be as,

1. **Rank**: The number of unknown factors that lead a user to give a rating. These could include factors such as age, gender, or location etc. The higher the rank, the better the recommendation will be, to some extent. Starting at 5 and increasing by 5 until the recommendation improvement rate slows down, memory and CPU permitting, is a good approach.

2. **Lambda**: A *regularization* parameter to prevent *overfitting*, represented by high *variance*, and low *bias*

Variance :	represents how much the predictions fluctuate at a given point, over multiple runs, compared to the theoretically correct value for that point.
Bias :	represents how far away the generated predictions are from the true value we're trying to predict

The higher the lambda, the lower the overfitting but the greater the bias.

3. **Iteration**: The number of times that the training will run. In this example, we will do 5, 10, and 20 iterations for various combinations of rank and lambda.

And many more aspects to keep in mind in complete analysis process

4. Recomandation

To make the results available to the user quickly and easily, we need to load them into a database that can be queried on demand and provide to user as recomendations.

Practicale implementqtion will be as,
1. Get the data from MySQL / NoSQL / Cloud SQL
2. Convert the DataFrame to fault-tolerant collection of elements that can be operated on in parallel and create the various datasets of it.
3. Train models based on various parameters
4. Calculating top predictions for the user
5. Saving the top predictions
6. Running the solution
7. Monitoring the work of Algorithm

To Track user Activities

We can take help of already availabler User Tracking systems like, Google Analytics, AWStats, etc.

Or for more granularity we can embed our own set of code in backend which captures and stores every click information of user on website and later can be processed.

Eg. We can use MongoDB to store data(as storing every click) and storing will be done on every click and page load fuctions in backend in Django views.

Google Analytics captures data like:

- **Page Information**
    - URL – the URL of the page the user is viewing
    - Title – the title of the page the user is viewing
- **Browser Information**
    - Browser name – the browser the user is using
    - Viewport or Viewing pane – the size of the browser window
    - Screen resolution – the resolution of the user’s screen
    - Java enabled – whether or not the user has Java enabled
    - Flash version – what version of Flash the user is using
- **User Information**
    - Location – this is derived from the IP address where the hit originated. The IP address itself is not available in GA as it is personally identifiable information (PII) which violates the terms of Google Analytics.
    - Language – derived from the language settings of the browser

If we add our set of code to caputer user activities:

- **Features wise tracking**
    - Most viewd feature(user interest) or **Popular Categories.**
    - Repeateadly accesing feature and its sub Categories.
    - Gathering data from social media likes and comments on insydo page and posts.
    - Gettng track of user liking recomendations or not.
      and many more depending on our.

Q.3 Find the attached data.txt file which contains a dump JSON for the Business Data and Keyword Data.

1. Read all the data and index that in Elastic Search.

2. Create a GET type Api in Django which takes a query parameter ('q') and search within the index data in Elastic search. e.g. http://localhost:8000/search/?q=cafe

3. Create one simple HTML page with only 1 search text box in it, and hit the above listed api from the search box text.

Solut'n:

Refer to insydo_test project for pratical code.

File = insydo_test/Templates/elastic_search_template.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Elastic Search From JSON DUMP Given</title>
</head>
<body>
    <form>
      <div>
        <input type="search" id="mySearch" name="q"
        placeholder="Search the site..." size="30">
        <button>Search</button>
      </div>
    </form>
</body>
</html>
```

File = insydo_test/myapp/elastic_search_mapping.py

```python
import django.db.models.options as options
from django.contrib.auth.models import User
import datetime
from django.db import models
options.DEFAULT_NAMES = options.DEFAULT_NAMES + (
    'es_index_name', 'es_type_name', 'es_mapping'
)
class ESearch(models.Model):
    id = models.AutoField(primary_key=True, db_column='id')
    name = models.TextField(db_column='name', default='')
    def __unicode__(self):
        return unicode(self.id)
    class Meta:
        es_index_name = 'django'
        es_type_name = 'esearch'
        es_mapping = {
            'properties': {
                'business': {'type': 'object',
                             'properties': {
                                            'id': {'type': 'long'},
                                            'name': {'type': 'string',
'index': 'not_analyzed'},
                                          }
                              },
                'keyword': {'type': 'object',
                             'properties': {
                                            'id': {'type': 'long'},
                                            'name': {'type': 'string',
'index': 'not_analyzed'},
                                          }
                             },
                            },
        }
```

file = insydo_test/myapp/push-to-index-esearch.py

```
from elasticsearch.client import IndicesClient
from django.conf import settings
from django.core.management.base import BaseCommand
from elasticsearch.helpers import bulk
from elastic_search_mapping import ESearch
class Command(BaseCommand):
    def handle(self, *args, **options):
        help = "My shiny new management command."
        self.recreate_index()
        self.push_db_to_index()
    def recreate_index(self):
        indices_client = IndicesClient(client=settings.ES_CLIENT)
        index_name = ESearch._meta.es_index_name
        if indices_client.exists(index_name):
            indices_client.delete(index=index_name)
        indices_client.create(index=index_name)
        indices_client.put_mapping(
            doc_type=ESearch._meta.es_type_name,
            body=ESearch._meta.es_mapping,
            index=index_name
        )
    def push_db_to_index(self):
        data = [
            self.convert_for_bulk(s, 'create') for s in
ESearch.objects.all()
        ]
        bulk(client=settings.ES_CLIENT, actions=data, stats_only=True)
    def convert_for_bulk(self, django_object, action=None):
        data = django_object.es_repr()
        metadata = {
            '_op_type': action,
            "_index": django_object._meta.es_index_name,
            "_type": django_object._meta.es_type_name,
        }
        data.update(**metadata)
        return data
    def es_repr(self):
        data = {}
        mapping = self._meta.es_mapping
        data['_id'] = self.pk
        for field_name in mapping['properties'].keys():
            data[field_name] = self.field_es_repr(field_name)
        return data
    def field_es_repr(self, field_name):
        config = self._meta.es_mapping['properties'][field_name]
        if hasattr(self, 'get_es_%s' % field_name):
            field_es_value = getattr(self, 'get_es_%s' % field_name)()
        else:
            if config['type'] == 'object':
                related_object = getattr(self, field_name)
                field_es_value = {}
                field_es_value['_id'] = related_object.pk
                for prop in config['properties'].keys():
                    field_es_value[prop] = getattr(related_object, prop)
            else:
                field_es_value = getattr(self, field_name)
        return field_es_value
    def get_es_name_complete(self):
        return {
            "input": [self.id, self.name],
            "output": "%s %s" % (self.id, self.name),
            "payload": {"pk": self.pk},
        }
```

file = insydo_test/myapp/base.py

```
from elasticsearch import Elasticsearch, RequestsHttpConnection
ES_CLIENT = Elasticsearch(
    ['http://127.0.0.1:9200/'],
    connection_class=RequestsHttpConnection
)
```

file = insydo_test/myapp/views.py

```
from urllib import urlencode
from copy import deepcopy
from django.http import HttpResponse
from django.conf import settings
from django.views.generic.base import TemplateView
from elastic_search_mapping import ESearch
from django.contrib.auth.models import User
from django.shortcuts import render
```

```
# from django.views.decorators.csrf import csrf_exempt
import json


##### Elastic Search Fuctionalities #####
client = settings.ES_CLIENT
def autocomplete_view(request):
    query = request.GET.get('term', '')
    resp = client.suggest(
        index='django',
        body={
            'name_complete': {
                "text": query,
                "completion": {
                    "field": 'name_complete',
                }
            }
        }
    )
    options = resp['name_complete'][0]['options']
    data = json.dumps(
        [{'id': i['payload']['pk'], 'value': i['text']} for i in
options]
    )
    mimetype = 'application/json'
    return HttpResponse(data, mimetype)
def user_detail(request):
    user_id = request.GET.get('id')
    student = ESearch.objects.get(pk=user_id)
    return render(request, 'user_details.html', context={'user':
student})
class HomePageView(TemplateView):
    template_name = "elastic_search_template.html"
    def get_context_data(self, **kwargs):
        body = {
            'aggs': {
                'business': {
                    'terms': {
                        'field': 'course_names', 'size': 0
                    }
                },
                'keywords': {
                    'terms': {
                        'field': 'university.name'
                    }
                }
            },
            # 'query': {'match_all': {}}
        }
        es_query = self.gen_es_query(self.request)
        body.update({'query': es_query})
        search_result = client.search(index='django',
doc_type='esearch', body=body)
        context = super(HomePageView, self).get_context_data(**kwargs)
        context['hits'] = [
            self.convert_hit_to_template(c) for c in
search_result['hits']['hits']
        ]
        context['aggregations'] = self.prepare_facet_data(
            search_result['aggregations'],
            self.request.GET
        )
        return context
    def convert_hit_to_template(self, hit1):
        hit = deepcopy(hit1)
        almost_ready = hit['_source']
        almost_ready['pk'] = hit['_id']
        return almost_ready
    def facet_url_args(self, url_args, field_name, field_value):
        is_active = False
        if url_args.get(field_name):
            base_list = url_args[field_name].split(',')
            if field_value in base_list:
                del base_list[base_list.index(field_value)]
                is_active = True
            else:
                base_list.append(field_value)
            url_args[field_name] = ','.join(base_list)
        else:
            url_args[field_name] = field_value
        return url_args, is_active
    def prepare_facet_data(self, aggregations_dict, get_args):
        resp = {}
        for area in aggregations_dict.keys():
            resp[area] = []
            if area == 'age':
                resp[area] = aggregations_dict[area]['buckets']
                continue
            for item in aggregations_dict[area]['buckets']:
                url_args, is_active = self.facet_url_args(
                    url_args=deepcopy(get_args.dict()),
                    field_name=area,
                    field_value=item['key']
                )
                resp[area].append({
                    'url_args': urlencode(url_args),
                    'name': item['key'],
                    'count': item['doc_count'],
                    'is_active': is_active
                })
        return resp
    def gen_es_query(self, request):
        req_dict = deepcopy(request.GET.dict())
        if not req_dict:
            return {'match_all': {}}
        filters = []
        for field_name in req_dict.keys():
            if '__' in field_name:
                filter_field_name = field_name.replace('__', '.')
            else:
                filter_field_name = field_name
            for field_value in req_dict[field_name].split(','):
                if not field_value:
                    continue
                filters.append(
                    {
                        'term': {filter_field_name: field_value},
                    }
                )
        return {
            'filtered': {
                'query': {'match_all': {}},
                'filter': {
                    'bool': {
                        'must': filters
                    }
                }
            }
        }
    def delete(self, es_client=None, *args, **kwargs):
        prev_pk = self.pk
        super(ESearch, self).delete(*args, **kwargs)
        es_client.delete(
            index=self._meta.es_index_name,
            doc_type=self._meta.es_type_name,
            id=prev_pk,
            refresh=True,
        )
```

file = Insydo_test/insydo_test/urls.py

```
from myapp.views import autocomplete_view, user_detail, HomePageView

### URLS for Elasticsearch pages ###
url(r'^autocomplete/', autocomplete_view, name='autocomplete-view'),
url(r'^user', user_detail, name='student-detail'),
url(r'^$', HomePageView.as_view(), name='index-view'),
```

Q.4. Optimize and correct the following Python code snippet.

The following method was implemented to shuffle the elements of a list with some restrictions but it's not working as expected.
Here are the restrictions:

- Elements on the list can move randomly to the left or to the right
- Elements must not move more than one position away from the original one
- The list is non circular.

How would you change the algorithm to work as expected?

Input Value = [21, 22, 23, 24, 25, 26, 27, 28]
Output Value: [22, 21, 24, 23, 26, 25, 28, 27]

```
def _shuffle_list_values(self):
        ids_list = [-1122, 2321, -9023, 2711, 8112, -0912, 2711, 9832]
        random_movements = {}
        for n_id in ids_list:
                movement = random.randrange(-1, 2, 1)
                random_movements[n_id] = movement
        left = -1
        right = 1
        shuffled_list = [x for x in ids_list]
        for n_id in ids_list:
                if not(ids_list.index(n_id) == 0 and random_movements[n_id] == left) or\
                not(ids_list.index(n_id) == len(ids_list) - 1 and random_movements[n_id] ==
                        right):
                        shuffled_list.insert(ids_list.index(n_id) + random_movements[n_id],
                        shuffled_list.pop(ids_list.index(n_id)))
        return shuffled_list
```

Solut'n:

Solution is on insydo_test/list_shuffle_Q4.py

```
def _shuffle_list_values_mine():
     arr = [21, 22, 23, 24, 25, 26, 27, 28]
     arr.insert(0, min(arr) - 1)
     for i in range(len(arr)-1):
       if (i % 2) == 0 and arr[i] > arr[i + 1]:
          arr[i+1], arr[i] = arr[i], arr[i+1]
       if (i % 2 != 0) and arr[i] < arr[i + 1]:
          arr[i+1], arr[i] = arr[i], arr[i+1]
     arr.pop(0)
     return arr

print "Input = [21, 22, 23, 24, 25, 26, 27, 28]"
print "new code = ", _shuffle_list_values_mine()
```

```
Output = [22, 21, 24, 23, 26, 25, 28, 27]
```

Q.5. Share an example of your work (code/repository) that you are proud of and tell us why.

Solut'n:

As i have signed NDA for trendzlink, so i couldnt share even a small bit of code from my workplace. But can answer the query with tweaked code snippet.

```
import os
import django
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'chatbotdemo.settings')
django.setup()
from yowsup.stacks import YowStackBuilder
from yowsup.layers.auth import AuthError
from yowsup.layers import YowLayerEvent
from yowsup.layers.network import YowNetworkLayer
from yowsup.layers.interface import YowInterfaceLayer,
ProtocolEntityCallback
from adminapp.views import *
import re
class EchoLayer(YowInterfaceLayer):
    @ProtocolEntityCallback("message")
    def onMessage(self, messageProtocolEntity):
        if messageProtocolEntity.getType() == 'text':
            self.onTextMessage(messageProtocolEntity)
        elif messageProtocolEntity.getType() == 'media':
            self.onMediaMessage(messageProtocolEntity)
        print "shrutiMessage Received"
        message_body = messageProtocolEntity.getBody()
        a = process_request(message_body,
messageProtocolEntity.getTo(), messageProtocolEntity.getFrom())
        print a
        a = re.sub(u"x_star", u"\u2B50", a)
        response_text = a
        if response_text is not None:
            messageProtocolEntity.setBody(response_text)

self.toLower(messageProtocolEntity.forward(messageProtocolEntity.getFr
om()))
            self.toLower(messageProtocolEntity.ack())
            self.toLower(messageProtocolEntity.ack(True))
    @ProtocolEntityCallback("receipt")
    def onReceipt(self, entity):
        self.toLower(entity.ack())
    def onTextMessage(self,messageProtocolEntity):
        # just print info
        print("Echoing %s to %s" % (messageProtocolEntity.getBody(),
messageProtocolEntity.getFrom(False)))
    def onMediaMessage(self, messageProtocolEntity):
        # just print info
        if messageProtocolEntity.getMediaType() == "image":
            print("Echoing image %s to %s" % (messageProtocolEntity.url,
messageProtocolEntity.getFrom(False)))
```

```
        elif messageProtocolEntity.getMediaType() == "location":
            print("Echoing location (%s, %s) to %s" %
(messageProtocolEntity.getLatitude(),
messageProtocolEntity.getLongitude(),
messageProtocolEntity.getFrom(False)))
        elif messageProtocolEntity.getMediaType() == "vcard":
            print("Echoing vcard (%s, %s) to %s" %
(messageProtocolEntity.getName(), messageProtocolEntity.getCardData(),
messageProtocolEntity.getFrom(False)))
class YowsupEchoStack(object):
    def __init__(self, credentials, encryptionEnabled = True):
        stackBuilder = YowStackBuilder()
        self.stack = stackBuilder\
            .pushDefaultLayers(encryptionEnabled)\
            .push(EchoLayer)\
            .build()
        self.stack.setCredentials(credentials)
    def start(self):

self.stack.broadcastEvent(YowLayerEvent(YowNetworkLayer.EVENT_STATE_CO
NNECT))
        try:
            self.stack.loop()
        except AuthError as e:
            print("Authentication Error: %s" % e.message)
if __name__ == "__main__":
    credentials = ("919011722675", "Uz4WBUunlAOoNxo2ajBBOpjAzF8=")
    echo = YowsupEchoStack(credentials)
    echo.start()
```

Above Code Snippet is a complete implementation of Yowsup library.

Our one of the project need it as the base for our feature to run.

I studied the completely library and implemented above script with unserstanding not only the flow and logic but also the newtwork layer protocols which help user to transfer data from one place to another.

Was a challanging as i did all self. And to tweek the Yowsup API was the tedious job. As it works on only one way . But out requirment was to make it work for tboth the was. I m still working on the project.

Q.6. (Bonus question): Write down the necessary test cases for Q1 and Q3.

Solut'n:

we can write test cases either by code(Test Script)
[i.e. from **django.test import TestCase**] as Django provides us with writing the Test model for our project. Sillenium web drivers plays an important role to our browser and test cases to execute.

other is traditional way of writing hand written test cases [using a standard Test Case Template].

Simple example for test on admin login account
Check on Insydo project folder at **Insydo_test/myapp/tests.py**

```
from django.contrib.auth.models import User
from django.test import TestCase
from selenium import webdriver
from django.test import Client
class MyTestCase(TestCase):

    def setUp(self):
        self.client = Client()
        user = User.objects.create_user('david', 'david@david.com',
'david')
        self.browser = webdriver.Firefox()
    #
    def tearDown(self):
        self.browser.quit()
    def test_redirect(self):
        response = self.client.get('/admin/')
        self.assertRedirects(response, '/admin/login/?next=/admin/')
    def test_login(self):
        # Issue a GET request.
        response = self.client.login(username='david',
password='david')
        print response
```

Due to less time in hand i'm writiing the test cases manually without code implementation, as question doesnt specify to write test cases in which fashion,

Test cases for Q.1 :

Test ID	***Test Title***	***Test Data***	***Defects***	***Status***
T101	Admin Login	username= 'david' password = 'david'	-	Passed
T102	Admin login	username= 'david' password = ''	-	Passed
T103	Updating stored data permission to admin(Superuser)	Add/update/delete actions	-	Passed

T104	Updating stored data permission to admin(staff)	Add/update/delete actions	-	Passed
T105	After adding all permission to staff User (add/update/delete)	Add/Update/delete actions	-	Passed
T106	After adding only view permission to staff User.(no add/update/delete permissions)	Login as a only view permission staff user	-	Passed
T107	Same time login of **two superuser** with all permission on production	Two superuser logged into the system from diffrent PC's	Both have permission to add/update/delete so concurrency issue occures	Failed
T108	User with no added permission but staff user	Logged in as staff	Nothing is shown	passed
T109	Same time login of **one superuser user and one Staff User** with all permission on production	Two loggins 1 as super user 1 as staff user	Both have permission to add/update/delete so concurrency issue occures	Failed
T110	Same time login of **two Staff User** with all permission on production	Both login as Staff user	Both have permission to add/update/delete so concurrency issue occures	Failed

And many more...

Test cases for Q.3 :

Test ID	***Test Title***	***Test Data***	***Defects***	***Status***
T201	Read all the data and index as per elastic search	JSON Dump in Data.txt file	-	passed
T202	Url pattern http://localhost:9200/search/?q=cafe	After inputting data in search text box	-	passed
T203	Elastic local host on port 9200 status 200	On addressbar http://localhost:9200/	-	passed
T204	Html Page with only a seach text box nothing else	http://localhost:9200/search	-	passed
T205	Searching correct data	Passing q= value corectly	-	passed
T206	Searching wrong data	Passing q= value wrong	No results shown	passed
T207	Searching with speacial symbols	Passing q= value with special charecters	exception	Failed

T208	Manually entering url	Input complete URL manually	Shows result	Failed
T209	Testing on multiple browsers	Trying on IE, Firefox, chrome	-	passed
T210	Comapring time for the result on diffrent diffrent queries	Iterating input with multiple currect searches	-	passed

...and many more.

Q1. Create a model in Django with a minimum of 4 attributes. Register that model in Django Admin. Make sure this Model instance is accessible by 1 user (on the admin site) at a time.

Example: User_1 logs in to Django Admin and accesses this model instance. Now User_1 can see all the fields and edit them. Meanwhile, User_2 opens the same model in Django admin. User_2 can't edit this model and all actions are disabled. As soon as User_1 closes this form (navigating away from form) User_2 can edit it.

Solut'n:

[Reffer to insydo_test project for pratical look]

file = insydo_test/insydo_test/admin.py

```
"""
Admin file for insydo_test project
"""
from django.contrib import admin
from django.contrib.auth.models import Group, Permission
from django.contrib.gis import forms
from myapp.models import Tourist
class MyGroupAdminForm(forms.ModelForm):
    class Meta:
        model = Group
        fields = ('name', 'permissions')
    permissions = forms.ModelMultipleChoiceField(
        Permission.objects.exclude(
            content_type__app_label__in=['tourist']),
        widget=admin.widgets.FilteredSelectMultiple(('permissions'),
False))
class MyGroupAdmin(admin.ModelAdmin):
    form = MyGroupAdminForm
    search_fields = ('name',)
    ordering = ('name',)
    def get_actions(self, request):
        actions = super(MyGroupAdmin, self).get_actions(request)
        # if isinstance(obj, Customer):
        print "*****actions", actions
        # if request.user.username[0].upper() != 'J':
        if request.user.username == "user_1":
            # actions =
            # admin.site.disable_action('delete_selected')
            if 'delete_selected' in actions:
                del actions['delete_selected']
        return actions
    def get_form(self, request, *args, **kwargs):
        form = super(MyGroupAdmin, self).get_form(request, *args,
**kwargs)
        form.base_fields['c_customer'].initial = request.user
        return form
class TouristAdmin(admin.ModelAdmin):
    def get_model_perms(self, request):
        """
        Return empty perms dict thus hiding the model from admin index.
        """
        return {}
admin.site.unregister(Group)
admin.site.register(Group, MyGroupAdmin)
admin.site.register(Tourist, TouristAdmin)
```

file = insydo_test/insydo_test/urls.py

```
from django.conf.urls import include, url

from django.contrib import admin

urlpatterns = [

     url(r'^admin/', include(admin.site.urls)),

     url(r'^admin2/', include(admin.site.urls)),

]
```

file = insydo_test/myapp/views.py

```
from django.http import HttpResponse
from django.contrib.auth.models import User
from django.shortcuts import render

def give_user_addon_permission(request):
    """
    fuction will add additional permission soto the user
    :return:
    """
    user = None
    if request.user.is_authenticated():
        if request.user.username == "user_1":
            user = User(username="user_1")
            user.is_staff = True
            user.is_superuser = True
            user.save()
        elif request.user.username == "user_2":
            user = User(username="user_2")
            user.is_staff = True
            user.is_superuser = True
            user.save()
        else:
            pass
    return user
```

```
def check_if_acquired_by_any_superuser():
    if request.user.is_authenticated():
    superusers_emails =
User.objects.filter(is_superuser=True).values_list('email')
    print superusers_emails
    return True
```

file = insydo_test/myapp/models.py

```
"""
Database models for insydo_test project
"""
from django.contrib.auth import get_permission_codename
from django.db import models, connection, transaction
from django.core.validators import MaxValueValidator
from django.contrib.auth.models import Permission
from django.contrib.auth.models import User
class Tourist(models.Model):
    """
    Table to store Tourist Info.
    """
    t_name = models.CharField(db_column='t_name', default='',
max_length=30)
    t_address = models.TextField(db_column='t_address', default='')
    t_city = models.CharField(db_column='t_city', default='',
max_length=50)
    t_email = models.EmailField(db_column='t_email', default=True)
    t_contact = models.BigIntegerField(db_column='t_contact',
default=0, validators=[MaxValueValidator(9999999999)])
    def add_permissions(self, **kwargs):
        if kwargs.has_key('request') and self.user is None:
            request = kwargs.pop('request')
            self.user = request.user
        if self.t_name == "User 1":
            # User.objects.get_or_create(username="user_1",
is_staff=True)
            u = User.objects.get(username="user_1")
            # u.set_password('temporary')
            permission = Permission.objects.get(name='Can view Tourist')
            permission2 = Permission.objects.get(name='Can change
Tourist')
            u.user_permissions.add(permission).remove(permission2)
            print "**** hello"
        return self.t_name
    def locking_using_sql(self, cls):
        with transaction.atomic():
            account = (
                    cls.objects
                        .select_for_update()
                        .get(id=id)
               )
        with connection.cursor() as cursor:
            cursor.execute("LOCK TABLES %s READ", Tourist)
            try:
                if self.t_name == "User 1":
                    # User.objects.get_or_create(username="user_1",
is_staff=True)
                    u = User.objects.get(username="user_1")
                    # u.set_password('temporary')
                    permission = Permission.objects.get(name='Can view
Tourist')
                    permission2 = Permission.objects.get(name='Can change
Tourist')
                    u.user_permissions.add(permission)
                    u.user_permissions.remove(permission2)
                    print "**********hello"
            finally:
                cursor.execute("UNLOCK TABLES;")

    def has_view_permission(self, request, obj=None):
        """
        Returns True if the given request has permission to view an
object.
        Can be overridden by the user in subclasses.
        """
        if request.user.username == "user_1":
            opts = self.c_customer
            codename = get_permission_codename('view', opts)
            return request.user.has_perm("%s.%s" % (opts.app_label,
codename))
        else:
            pass

    def __unicode__(self):
        return self.t_name
    # def has_change_permission(self, request, obj=None):
    #     """
    #     Override this method in order to return True whenever a user
has view
    #     permission and avoid re-implementing the change_view and
    #     changelist_view views. Also, added an extra argument to
determine
    #     whenever this function will return the original response
    #     """
    #     change_permission = super(AdminViewPermissionBaseModelAdmin,
    #
self).has_change_permission(request, obj)
    #     if change_permission or self.has_view_permission(request,
obj):
    #         return True
    #     else:
    #         pass
    #     return change_permission
    #
    # def get_actions(self, request):
```

```
    #     actions = super(Customer, self).get_actions(request)
    #     # if isinstance(obj, Customer):
    #
    #     print "*****actions", actions
    #
    #     # if request.user.username[0].upper() != 'J':
    #     if request.user.username == "user_1":
    #         # actions =
    #         # admin.site.disable_action('delete_selected')
    #         # if 'delete_selected' in actions:
    #         del actions['delete_selected']
    #     return actions
    #
    # def get_form(self, request, *args, **kwargs):
    #     form = super(Customer, self).get_form(request, *args,
**kwargs)
    #     form.base_fields['c_customer'].initial = request.user
    #     return form
    #
    # def refresh(self):
    #     updated = Customer.objects.filter(id=self.c_customer_id)[0]
    #     fields = [f.name for f in self._meta._fields()
    #               if f.name != 'id']
    #     for field in fields:
    #         setattr(self, field, getattr(updated, field))
    #
    # def _mangle_sql_for_locking(self, sql):
    #     # yeah, it's really this difficult
    #     return sql + ' FOR UPDATE'
    #
    # def _get_lock(self):
    #     values = ('c_customer', 'c_address', 'c_city', 'c_email',
'c_contact')
    #     query =
Customer.objects.filter(id=self.c_customer.id).values_list(*values)
    #     sql, params = query._as_sql(connection=connection)
    #     cursor = connection.cursor()
    #     cursor.execute(self._mangle_sql_for_locking(sql), params)
    #     self._concurrency_poison()
    #
    # @classmethod
    # def update(cls, c_customer_id):
    #     with transaction.atomic():
    #         account = (
    #             cls.objects.select_for_update().get(id=c_customer_id)
    #         )
    #         account.save()
    #
    #     return account
    #
    # @classmethod
    # def add(cls, c_customer_id):
    #     with transaction.atomic():
    #         account = (
    #             cls.objects.select_for_update().get(id=c_customer_id)
    #         )
    #         account.save()
    #
    #     return account
    #
    # @classmethod
    # def delete(cls, c_customer_id):
    #     with transaction.atomic():
    #         account = (
    #             cls.objects.select_for_update().get(id=c_customer_id)
    #         )
    #         account.save()
    #
    #     return account
```