Quantification

In [1]:
```
import pandas as pd
import numpy as np
from functools import reduce
from collections import defaultdict
import json
from ast import literal_eval
import datetime
```

In [10]:
```
distribution = pd.read_csv('../../Distribution/master csv/distribution_with_rc
epss_time = pd.read_csv('../../Distribution/master csv/percentage_r_score_above
```

Merging EPSS_TIME to distribution

- epss time comes from z scores

In [13]:
```
dist_df_copy = distribution.copy()
dist_df_copy = dist_df_copy.merge(epss_time, on='circuit_protection_zone_id', h
dist_df_copy
```

Out[13]:

	Unnamed: 0.1	Unnamed: 0	PZ_idx	feeder_id	LATITUDE	LONGITUDE	circuit_protection_z
0	0	0	0.0	252681104.0	36.166922	-119.877392	HENRIETTA 11
1	1	36	9041.0	252681104.0	36.165786	-119.878317	HENRIETTA 11
2	2	37	9043.0	252681104.0	36.242067	-119.905048	HENRIETTA
3	3	1182	6915.0	252681104.0	36.123362	-119.879802	HENRIETTA 11
4	4	1	574.0	252681110.0	36.242082	-119.904815	HENRIETTA
...	...	...	...	...	...	...	
11845	11844	11831	NaN	NaN	NaN	NaN	SOLEDAD 2105
11846	11845	11768	NaN	NaN	NaN	NaN	NEWARK 21KV 210
11847	11846	11789	NaN	NaN	NaN	NaN	NEWARK 12KV 110
11848	11847	11845	NaN	NaN	NaN	NaN	NEWARK 21KV 210
11849	11848	11848	NaN	NaN	NaN	NaN	NEWARK 12KV

11850 rows × 33 columns

In [14]:
```
class Quantification:
    def __init__(self, **kwargs):
        self.psps_not_performed=kwargs.get('psps_not_performed', 0.05)
        self.psps=kwargs.get('psps', None)
        self.ignition1=kwargs.get('ignition1', 1)
        self.ignition2=kwargs.get('ignition2', 0)
        self.ignition3=kwargs.get('ignition3', 1)
        self.outage1=kwargs.get('outage1', None)
        self.outage2=kwargs.get('outage2', None)
        self.catastrophe_ac_threshold=kwargs.get('catastrophe_ac_threshold', 30
        self.catastrophe_bu_threshold=kwargs.get('catastrophe_bu_threshold', 50
        self.catastrophe_pop_threshold=kwargs.get('catastrophe_pop_threshold',
        # Totals
        self.ac_total = None
        self.pop_total = None
        self.bu_total = None
        self.financial_total = None
        self.reliability_total = None
        self.safety_total = None


    def log_progress(self, sequence, every=None, size=None, name='Items'):
        from ipywidgets import IntProgress, HTML, VBox
        from IPython.display import display

        is_iterator = False
        if size is None:
            try:
                size = len(sequence)
            except TypeError:
                is_iterator = True
        if size is not None:
            if every is None:
                if size <= 200:
                    every = 1
                else:
                    every = int(size / 200)     # every 0.5%
        else:
            assert every is not None, 'sequence is iterator, set every'

        if is_iterator:
            progress = IntProgress(min=0, max=1, value=1)
            progress.bar_style = 'info'
        else:
            progress = IntProgress(min=0, max=size, value=0)
        label = HTML()
        box = VBox(children=[label, progress])
        display(box)

        index = 0
        try:
            for index, record in enumerate(sequence, 1):
                if index == 1 or index % every == 0:
                    if is_iterator:
                        label.value = '{name}: {index} / ?'.format(
                            name=name,
                            index=index
                        )
                    else:
```

```
                    progress.value = index
                    label.value = u'{name}: {index} / {size}'.format(
                        name=name,
                        index=index,
                        size=size
                    )
                yield record
        except:
            progress.bar_style = 'danger'
            raise
        else:
            progress.bar_style = 'success'
            progress.value = index
            label.value = "{name}: {index}".format(
                name=name,
                index=str(index or '?')
            )


    def get_outage(self, structure_prob, veg_prob):
        return structure_prob + veg_prob - (structure_prob * veg_prob)

    """
    Calculates probability histograms by multiplying the each probability of a
    and with a fire probability, also stores new probabilities in a list based
    """
    def get_prob_hist(self, prob, hist, grouped_hist=None):
        ac_bu_pop_hist = {'AC':{}, 'BU':{}, 'POP':{}}
        for value, probability in hist['AC']:
            product = prob * probability
            ac_bu_pop_hist['AC'][value]=(product)
            if grouped_hist:
                grouped_hist['AC'][value].append(product)

        for value, probability in hist['BU']:
            product = prob * probability
            ac_bu_pop_hist['BU'][value]=(product)
            if grouped_hist:
                grouped_hist['BU'][value].append(product)

        for value, probability in hist['POP']:
            product = prob * probability
            ac_bu_pop_hist['POP'][value]=(product)
            if grouped_hist:
                grouped_hist['POP'][value].append(product)
        return ac_bu_pop_hist


    def get_catastrophic_fire(
        self, high_consequence,
        psps,ineffective_evacuation,
        outage_1, outage_2, epss
        ):

        ineffective_evacuation1 = ineffective_evacuation
        value1 = psps * self.psps_not_performed * epss * outage_1 * self.igniti
        value2 = psps * (1 - self.psps_not_performed)*self.ignition2
        value3 = (1-psps) * epss * outage_2 * self.ignition3
        return (value1 + value2 + value3) * high_consequence * ineffective_evac
```

```
    def get_destructive_fire(
        self, high_consequence,
        psps, ineffective_evacuation,
        outage_1, outage_2, epss
        ):

        ineffective_evacuation1 = ineffective_evacuation
        value1 = psps * self.psps_not_performed * epss * outage_1 * self.igniti
        value2 = psps * (1-self.psps_not_performed) * self.ignition2
        value3 = (1-psps) * epss * outage_2 * self.ignition3
        total = (value1 + value2 + value3) * high_consequence * (1-ineffective_

        return total


    def get_small_fire_prob(
        self, high_consequence, low_consequence,
        psps, ineffective_evacuation,
        outage_1, outage_2, epss
        ):

        ineffective_evacuation2 = ineffective_evacuation
        value1 = psps * self.psps_not_performed * epss * outage_1 * self.igniti
        value2 = psps * (1 - self.psps_not_performed) * self.ignition2
        value3 = (1 - psps) * epss * outage_2 * self.ignition3
        return (value1 + value2 + value3) * (1 - high_consequence) * low_conseq

    def get_ac_greater_than_100_fire_endstate(self, psps, outage_1, outage_2, a

        value1 = psps * self.psps_not_performed * epss * outage_1 * self.igniti
        value2 = psps * (1 - self.psps_not_performed) * self.ignition2
        value3 = (1 - psps) * epss * outage_2 * self.ignition3

        return (value1 + value2 + value3) * ac_greater_than_100

    def get_ac_greater_than_300_fire_endstate(self, psps, outage_1, outage_2, a

        value1 = psps * self.psps_not_performed * epss * outage_1 * self.igniti
        value2 = psps * (1 - self.psps_not_performed) * self.ignition2
        value3 = (1 - psps) * epss * outage_2 * self.ignition3

        return (value1 + value2 + value3) * ac_greater_than_300


    def get_safe_prob(
        self, high_consequence,
        low_consequence, psps,
        outage_1, outage_2, epss
        ):

        value1 = psps * self.psps_not_performed * epss * outage_1 * self.igniti
        value2 = psps * (1 - self.psps_not_performed) * self.ignition2
        value3 = (1 - psps) * epss * outage_2 * self.ignition3
        value4 = (value1 + value2 + value3) * (1 - high_consequence) * (1 - low

        value5 = psps * self.psps_not_performed * epss * outage_1 * (1 - self.i
        value6 = psps * self.psps_not_performed * (1 - (epss * outage_1))
        value7 = psps * (1 - self.psps_not_performed) * (1-self.ignition2)
        value8 = (1 - psps) * epss * outage_2 * (1-self.ignition3)
```

```
        value9 = (1 - psps) * (1 - (epss * outage_2))

        return value4 + value5 + value6 + value7 + value8 + value9


    def get_fire_no_injury(
        self, high_consequence,
        low_consequence, psps,
        ineffective_evacuation,
        outage_1, outage_2, epss
        ):

        ineffective_evacuation2 = ineffective_evacuation
        value1 = psps * self.psps_not_performed * epss * outage_1 * self.igniti
        value2 = psps * (1 - self.psps_not_performed) * self.ignition2
        value3 = (1 - psps) * epss * outage_2 * self.ignition3
        return (value1 + value2 + value3) * (1 - high_consequence) * low_conseq


    def get_quanitification(self, high_consequence, psps, ineffective_evacuatic
        quantification = {}
        quantification['catastrophic_fire'] = self.get_catastrophic_fire(high_c
        quantification['destructive_fire'] = self.get_destructive_fire(high_cor
        quantification['fire_probability'] = self.get_small_fire_prob(high_cons
        quantification['fire_probability_no_injury'] = self.get_fire_no_injury(
        quantification['safe_probability'] = self.get_safe_prob(high_consequenc
        quantification['ac_greater_than_100_fire'] = self.get_ac_greater_than_1
        quantification['ac_greater_than_300_fire'] = self.get_ac_greater_than_3
        return quantification


    def get_consequences(self, ac_bu_pop, grouped_hist=None):
         # collects values for high and low consequences
        high_consequence_sums = {'AC':0, 'BU':0}
        low_consequence_sums = {'AC':0, 'BU':0, 'POP':0}
        ineffective_evacuation = 0
        ac_greater_than_100 = 0
        ac_greater_than_300 = 0
        if 'AC' in ac_bu_pop:
            for hist in ac_bu_pop['AC']:
                if hist[1] is not None and hist[0] is not None:
                    if hist[0] == 0:
                        low_consequence_sums['AC'] += hist[1]/19
                    if hist[0] > 300:
                        high_consequence_sums['AC'] += hist[1]/19
                    if hist[0] > 100:
                        ac_greater_than_100 += hist[1]/19
                    if hist[0]> 300:
                        ac_greater_than_300 += hist[1]/19
                    if grouped_hist:
                        grouped_hist['AC'][hist[0]].append(hist[1]/19)

            for hist in ac_bu_pop['BU']:
                if hist[1] is not None and hist[0] is not None:
                    if hist[0] == 0:
                        low_consequence_sums['BU'] += hist[1]/19
                    if hist[0] > 50:
                        high_consequence_sums['BU'] += hist[1]/19
                    if grouped_hist:
                        grouped_hist['BU'][hist[0]].append(hist[1]/19)
```

```
        for hist in ac_bu_pop['POP']:
            if hist[1] is not None and hist[0] is not None:
                if hist[0] == 0:
                    low_consequence_sums['POP'] += hist[1]/19
                if hist[0]/2000 > 1:
                    ineffective_evacuation += hist[1]/19
                if grouped_hist:
                    grouped_hist['POP'][hist[0]].append(hist[1]/19)

    high_consequence = high_consequence_sums['AC'] * high_consequence_sums[
    low_consequence = 1 - (low_consequence_sums['AC'] * low_consequence_sum
    return high_consequence, low_consequence, ineffective_evacuation, ac_gr


def get_risk_curves(self, hist_dict):
    output={}
    ac = hist_dict['AC']
    bu = hist_dict['BU']
    pop = hist_dict['POP']

    output['ac'] = ac
    output['pop'] = pop
    output['bu'] = bu
    safety_histogram = pop
    safety_curve = []

    if len(pop):
        sorted_keys = sorted(pop.keys())
        starting_sum = sum([pop[key] for key in sorted_keys]) # sum of all
        safety_curve.append([pop[sorted_keys[0]], starting_sum])
        for idx in range(1,len(sorted_keys)):
            prev_prob= pop[sorted_keys[idx-1]]
            current_num = sorted_keys[idx]
            starting_sum -= prev_prob
            safety_curve.append([current_num, starting_sum])

    output['safety'] = safety_curve
    output['safety_histogram'] = pop
    # print(output['safety'])
    financial_bu = [[arr[0] * 10 ** 6, arr[1]] for arr in
                    bu.items() if arr[0] is not None]  # multiplies bu valu
    # print(bu)
    # print(financial_bu)
    financial_ac = [[arr[0] * 1175, arr[1]] for arr in# UPDATE
                    ac.items() if arr[0] is not None]  # multiplies ac valu
    # print(financial_ac)
    merged_hist = {}
    for current_bu in financial_bu:
        if merged_hist.get(current_bu[0]):
            merged_hist[current_bu[0]].append(current_bu[1])
        else:
            merged_hist[current_bu[0]] = [current_bu[1]]
    for current_ac in financial_ac:
        if merged_hist.get(current_ac[0]):
            merged_hist[current_ac[0]].append(current_ac[1])
        else:
            merged_hist[current_ac[0]] = [current_ac[1]]
#     print(merged_hist)
    ordered_hist = []
```

```
        for value in sorted(list(merged_hist)):
            ordered_hist.append([value, sum(merged_hist[value]) / 2])
    #         ordered_hist.append([value, reduce(lambda x, y: x * y, merged_his
        financial_histogram = ordered_hist
        # print(ordered_hist)
        financial_curve = []
        if len(ordered_hist):
            starting_sum = sum([arr[1] for arr in ordered_hist]) # sum of all p
            financial_curve.append([ordered_hist[0][0], starting_sum])
            # starting_sum = starting_sum - a[0][0]
            for idx in range(1,len(ordered_hist)):
                prev_num = ordered_hist[idx-1][1]
                current_num = ordered_hist[idx][0]
                starting_sum -= prev_num
                financial_curve.append([current_num,starting_sum])
    #         sorted_keys = sorted(pop.keys())
        output['financial_histogram'] = financial_histogram
        output['financial'] = financial_curve
        # print(financial_curve)

        probability_1 = sum([current_ac[1] for current_ac in ac.items() if curr
        ac_over_300 = sum([current_ac[1] for current_ac in ac.items() if curren
        bu_under_50 = sum([current_bu[1] for current_bu in bu.items() if curren
        bu_over_50 = sum([current_bu[1] for current_bu in bu.items() if current

        probability_2 = ac_over_300 * bu_under_50
        probability_3 = ac_over_300 * bu_over_50
        reliability_curve = [
            [0.04, probability_3 + probability_2 + probability_1],
            [0.7, (probability_3 + probability_2)],
            [223, (probability_3)]
        ]
        # if incorrect numbers, same x values, y values prob1, prob2, prob 3
        reliability_histogram = [
            [0.04,  probability_1],
            [0.7,  (probability_2)],
            [223, (probability_3)]
        ]
        output['reliability_histogram'] = reliability_histogram
        output['reliability'] = reliability_curve
        return output


    def calculate_endstate(self,):
        pass


    def read_initial_settings(self,):
        configurations =  ['psps_not_performed','psps','ignition1','ignition2',
        config_str = '=============CONFIGURATIONS============\n\n'
        for config in configurations:
            value = self.__getattribute__(config)
            if value is None:
                config_str += '{}\t(NOT SET)\n'.format(config)
            else:
                config_str += '{}\t{}\n'.format(config, value)
        print(config_str, '\n=======================================\n')


    def run(self,):
```

```
        pass
```

In [16]:

```
class DistributionQuantification(Quantification):
    def __init__(self, dist_df, **kwargs):
        Quantification.__init__(self, **kwargs)
        self.dist_df = dist_df
        self.quantifications = {}
        self.distribution_level_endstates = None
        self.prob_hist_by_line = None
        self.risk_curves_by_line = None
        self.config_by_line = None
        self.mitigation_endstates = {
            'evm': {},
            'hardening': {},
            'combined': {},
            'underground': {},
        }
        self.total_fires = None
        self.with_epss_time = kwargs.get('with_epss_time', False)

    def get_hists(self, row):
        """takes a df row and returns the hists"""
        ac = row['AC_2020']
        pop = row['POP_2020']
        bu = row['BU_2020']
        if ac != ac or 'nan' in ac:
            ac = '[]'
        if bu != bu or 'nan' in bu:
            bu = '[]'
        if pop != pop or 'nan' in pop:
            pop = '[]'
        return {'AC': literal_eval(ac), 'BU': literal_eval(bu), 'POP': literal_

    """
    Gets columns from get_outage_inputs 11 cols
    """

    def get_outage(self, **args):
        # 1-((1-p1)*…*(1-p10))
        start = 1
        for arg, value in args.items():
            start *= 1 - value
        return 1 - start

    def get_effective_outage(self, evm_example, system_hardening, **args):
        # (1-effectiveness)*(ignition probability of Driver X)
        evm_outage = 1
        system_hardening_outage = 1
        combined = 1
        underground = 0
        evm_dict = {}
        system_dict = {}
        for arg, value in args.items():
            value = value if value == value else 0
            evm_val = (1 - evm_example.get(arg, 0)) * value
            system_val = (1 - system_hardening.get(arg, 0)) * value
            evm_dict[arg] = evm_val
            system_dict[arg] = system_val
```

```
        evm_outage = self.get_outage(**evm_dict)
        system_hardening_outage = self.get_outage(**system_dict)
        return 1 - evm_outage, 1 - system_hardening_outage, underground, evm_di

    def get_outage_inputs(self, row_dict):
        required_cols = [
            'weighted_support_structure_equipment_cause_probability_ignition_to
            'weighted_support_structure_equipment_electrical_probability_igniti
            'weighted_transformer_equipment_cause_probability_ignition_total',
            'weighted_transformer_equipment_leaking_probability_ignition_total'
            'weighted_animal_bird_probability_ignition_total',
            'weighted_animal_other_probability_ignition_total',
            'weighted_animal_squirrel_probability_ignition_total',
            'weighted_other_equipment_type_probability_ignition_total',
            'weighted_primary_conductor_probability_ignition_total',
            'weighted_secondary_conductor_probability_ignition_total',
            'weighted_third_party_balloon_probability_ignition_total',
            'weighted_third_party_other_probability_ignition_total',
            'weighted_third_party_vehicle_probability_ignition_total',
            'weighted_vegetation_branch_probability_ignition_total',
            'weighted_vegetation_other_probability_ignition_total',
            'weighted_vegetation_trunk_probability_ignition_total',
            'weighted_voltage_control_equipment_type_probability_ignition_total
        ]

        return {col: row_dict[col] for col in required_cols}

    def calculate_endstate(self,):
        quantifications = {}
        config_by_line = {}
        evm_example = {'vegetation_trunk_probability_ignition_ignition_probabil
                       'vegetation_branch_probability_ignition_ignition_probabi
                       'vegetation_other_probability_ignition_ignition_probabil
                       }
        system_hardening = {
            'vegetation_trunk_probability_ignition_ignition_probability': 0.34,
            'vegetation_branch_probability_ignition_ignition_probability': 0.56
            'vegetation_other_probability_ignition_ignition_probability': 0.55,
            'primary_conductor_probability_ignition_ignition_probability': 0.39
            'support_structure_equipment_cause_probability_ignition_ignition_pr
            'secondary_conductor_probability_ignition_ignition_probability': 0.
            'secondary_other_equipment_type_probability_ignition_ignition_proba
            'primary_interrupter_probability_ignition_ignition_probability': 0.
            'transformer_equipment_cause_probability_ignition_ignition_probabil
            'secondary_interrupter_probability_ignition_ignition_probability':
            'primary_other_equipment_type_probability_ignition_ignition_probabi

        }

        for idx in range(len(self.dist_df)):
            row = self.dist_df.iloc[idx]
            quantifications[row.circuit_protection_zone_id] = {}
            histograms = self.get_hists(row)
            outage1 = 1 if self.outage1 is None else self.outage1
            outage2 = self.get_outage(**self.get_outage_inputs(row)) if self.ou
            # EPSS
            if self.with_epss_time and not np.isnan(row.epss_time):
                outage1  = row.epss_time * 0.2 * outage1
                outage2 = row.epss_time * 0.2 * outage2
            high_consequence, low_consequence, ineffective_evacuation, ac_great
```

```
                histograms
                )
            psps = row.psps if self.psps is None else self.psps
            config_by_line[row.circuit_protection_zone_id] = {
                'psps': psps, 'high_consequence': high_consequence, 'low_conseq
                'ineffective_evacuation': ineffective_evacuation, 'outage2': ou
            }
            quantifications[row.circuit_protection_zone_id] = self.get_quanitif
                high_consequence,
                psps,
                ineffective_evacuation,
                outage1,
                outage2,
                low_consequence,
                ac_greater_than_100,
                ac_greater_than_300,
                row.epss
            )
#         print(count)
        self.quantifications = quantifications
        self.config_by_line = config_by_line

    def get_distribution_level_endstate(self,):
        endstate_dict = defaultdict(list)
        for _, endstates in self.quantifications.items():
            for endstate, value in endstates.items():
                endstate_dict[endstate].append(value)

        distribution_level_endstates = {
            'catastrophic_fire': 1 - reduce(lambda x, y: x*y, [1 - value for va
            'destructive_fire': 1 - reduce(lambda x, y: x*y, [1 - value for val
            'fire_probability': 1 - reduce(lambda x, y: x*y, [1 - value for val
            'fire_probability_no_injury': 1 - reduce(lambda x, y: x*y, [1 - val
        }
        distribution_level_endstates['safety_probability'] = reduce(
            lambda x, y: x*y, [1-value for value in distribution_level_endstate
        self.distribution_level_endstates = distribution_level_endstates

    def get_endstate_description(self,):
        endstate_dict = defaultdict(list)
        for _, endstates in self.quantifications.items():
            for endstate, value in endstates.items():
                endstate_dict[endstate].append(value)
        return pd.DataFrame(endstate_dict).describe()

    def calculate_ac_bu_pop_and_risk_totals(self,):
        ac_total = 0
        pop_total = 0
        bu_total = 0
        financial_total = 0
        reliability_total = 0
        safety_total = 0
        for pz in self.log_progress(self.prob_hist_by_line):
            prob_hist = self.prob_hist_by_line[pz]
            for value, prob in prob_hist['AC'].items():
                ac_total += value * prob
            for value, prob in prob_hist['POP'].items():
                pop_total += value * prob
                safety_total += value/2000 * prob
            for value, prob in prob_hist['BU'].items():
```

```
                bu_total += value * prob
            prob_hist = self.risk_curves_by_line[pz]
            for arr in prob_hist['financial_histogram']:
                financial_total += arr[0] * arr[1]
            for arr in prob_hist['reliability_histogram']:
                reliability_total += arr[0] * arr[1]

        self.ac_total = ac_total
        self.pop_total = pop_total
        self.bu_total = bu_total
        self.financial_total = financial_total
        self.reliability_total = reliability_total
        self.safety_total = safety_total

    def calculate_total_fires(self, ):
        fires_by_endstates_dist = defaultdict(int)
        for _, endstates in self.quantifications.items():
            for endstate, value in endstates.items():
                fires_by_endstates_dist[f'{endstate} (total fires)'] += value
        self.total_fires = fires_by_endstates_dist

    def get_prob_hist_and_curves(self,):
        histograms_by_pz = {}
        risk_curves_by_pz = {}
        for pz in self.quantifications:
            fire_prob = 1 - self.quantifications[pz]['safe_probability']
            row = self.dist_df[self.dist_df.circuit_protection_zone_id == pz].i
            histograms = self.get_hists(row)
            hist = self.get_prob_hist(fire_prob, histograms)
            histograms_by_pz[pz] = hist
            risk_curves_by_pz[pz] = {'id': pz, **self.get_risk_curves(hist)}
        self.prob_hist_by_line = histograms_by_pz
        self.risk_curves_by_line = risk_curves_by_pz


    def totals(self, include_configuration=False) -> pd.DataFrame:
        totals_dict = {
            'AC_TOTAL':[self.ac_total],
            'BU_TOTAL': [self.bu_total],
            'POP_TOTAL': [self.pop_total],
            'Financial_Total': [self.financial_total],
            'Reliability_Total': [self.reliability_total],
            'Safety_Total': [self.safety_total],
            **{ key: [value] for key,value in
                self.distribution_level_endstates.items()
            },
            **{ key: [value] for key,value in
               self.total_fires.items()
            }
        }
        if include_configuration:
            totals_dict['psps_not_performed'] = self.psps_not_performed
            totals_dict['psps'] = self.psps
            totals_dict['ignition1'] = self.ignition1
            totals_dict['ignition2'] = self.ignition2
            totals_dict['ignition3'] = self.ignition3
            totals_dict['outage1'] = self.outage1
            totals_dict['outage2'] = self.outage2
            totals_dict['catastrophe_ac_threshold'] = self.catastrophe_ac_thres
            totals_dict['catastrophe_bu_threshold'] = self.catastrophe_bu_thres
```

```
            totals_dict['catastrophe_pop_threshold'] = self.catastrophe_pop_thr
            totals_dict['EPSS MEAN'] = self.dist_df.epss.mean()
        return pd.DataFrame(totals_dict)


    def read_initial_settings(self, ):
        print(f'EPSS AVERAGE \t\t\t {self.dist_df.epss.mean()}')
        super().read_initial_settings()


    def run(self,):
        self.read_initial_settings()
        self.calculate_endstate()
        self.get_distribution_level_endstate()
        self.get_prob_hist_and_curves()
        self.calculate_ac_bu_pop_and_risk_totals()
        self.calculate_total_fires()
```

In [17]:
```
dist = DistributionQuantification(
    dist_df=dist_df_copy,
    with_epss_time=True
)
dist.run()
```

```
EPSS AVERAGE                        1.0
=============CONFIGURATIONS============

psps_not_performed      0.05
psps    (NOT SET)
ignition1       1
ignition2       0
ignition3       1
outage1 (NOT SET)
outage2 (NOT SET)
catastrophe_ac_threshold        300
catastrophe_bu_threshold        50
catastrophe_pop_threshold       1

=========================================

VBox(children=(HTML(value=''), IntProgress(value=0, max=11849)))
```

Totals

In [18]:
```
dist.totals().transpose().to_dict()[0]
```

Out[18]:

```
{'AC_TOTAL': 262327.68399019167,
 'BU_TOTAL': 3692.9804892455445,
 'POP_TOTAL': 6406.648332797897,
 'Financial_Total': 2000607758.9669824,
 'Reliability_Total': 341.08223517075254,
 'Safety_Total': 3.2033241663988252,
 'catastrophic_fire': 6.315954759195108e-09,
 'destructive_fire': 0.0440262268224092,
 'fire_probability': 1.2400264343459533e-05,
 'fire_probability_no_injury': 1.0,
 'safety_probability': 0.0,
 'catastrophic_fire (total fires)': 6.31595475301898e-09,
 'destructive_fire (total fires)': 0.04502255724653958,
 'fire_probability (total fires)': 1.2400316449798387e-05,
 'fire_probability_no_injury (total fires)': 326.651812314017,
 'safe_probability (total fires)': 11522.30315272212,
 'ac_greater_than_100_fire (total fires)': 9.424903399504075,
 'ac_greater_than_300_fire (total fires)': 8.313478958417992}
```


In [ ]: