Oct 03, 2025

Chromosome 12 and Environmental Factors in Parkinson's Disease: An All of Us Data Analysis

Genes

DOI
https://dx.doi.org/10.17504/protocols.io.dm6gpq7jdlzp/v1

Kenta Abe[1], karen niemchick[1]

[1]Grand Valley State University

K Kenta Abe
Grand Valley State University

OPEN ACCESS

DOI: https://dx.doi.org/10.17504/protocols.io.dm6gpq7jdlzp/v1

External link: https://doi.org/10.3390/genes16101197

Protocol Citation: Kenta Abe, karen niemchick 2025. Chromosome 12 and Environmental Factors in Parkinson's Disease: An All of Us Data Analysis. **protocols.io** https://dx.doi.org/10.17504/protocols.io.dm6gpq7jdlzp/v1

Manuscript citation:
Abe K, Niemchick K (2025) Chromosome 12 and Environmental Factors in Parkinson's Disease: An All of Us Data Analysis. Genes 16(10). doi: 10.3390/genes16101197

Protocol status: Working
We use this protocol and it's working

Created: June 13, 2025

Last Modified: October 03, 2025

Protocol Integer ID: 220161

Keywords: environmental factors in parkinson, parkinson, including park gene mutation, park8 gene g2019s mutation, genetic dysfunction, neurodegenerative disease, park gene mutation, genetic polymorphism, candidate genetic polymorphism, gps on other chromosome, genetic knowledge, chromosome, disease, genomic, gene

Abstract

Background/Objectives: Parkinson's disease (PD) is a neurodegenerative disease that develops with age and is related to a decline in motor function. Studies suggest that the causes may be based on genetic dysfunction including *PARK* gene mutations and environmental factors. **Methods**: To explore those factors, we used multivariable logistic regression to obtain odds ratios (ORs) and adjusted ORs by using the All of Us Dataset which contains genomic, blood test, and other environmental data. **Results**: On Chromosome 12, there were 3709 candidate genetic polymorphisms (GPs) that are associated with PD. Of those GPs, fourteen GPs had high ORs which are similar to the OR of the *PARK8* gene G2019S mutation. Of those 3709 GPs, a 2.00-fold change in OR was observed in five GPs located at bases 53,711,362 (OR = 4.86, 95% CI [1.46, 16.18]), 31,281,818 (OR = 4.37, 95% CI [1.02, 18.82]), 101,921,705 (OR = 5.38, 95% CI [1.23, 23.51]), 47,968,795 (OR = 7.82, 95% CI [1.81, 33.83]), and 112,791,809 (OR = 8.05, 95% CI [1.85, 35.05]) by calcium, Vitamin D, and alcohol intake and were statistically significant. **Conclusions**: The results suggest that the progression of some PD caused by certain GPs can be delayed or prevented by the environmental factors above. In February 2025, All of Us released the CT Dataset v.8 which has a 50% increase in the number of participants. Potentially, it may be possible to research more GPs and environmental factors. In future studies, we would like to explore other environmental factors and GPs on other chromosomes. It is believed that specific GPs may tailor current treatments and qualify patients for clinical trials. Additionally, genetic knowledge may help increase accuracy in clinical trials.

Data preparation

1 Use *All of Us* dataset (Controlled Tier (CT) v.7 Curated Data Repository (CDR)).

All of Us dataset is allowed to use only in the cloud environment provided by *All of Us* Researcher Workbench. https://workbench.researchallofus.org/login

Preliminary test (Logistic regression (PD(+/-) = GP (+/-) for each locus)

2 Chromosome 12 has over 130 million base pairs and over 1,600 genes [1]. In the Chromosome 12 data, there were 1,576,756 bases. In the MatrixTable, the reference type of gene was recorded as 0/0 and GPs were recorded such as 0/1, 1/1, 0/2, 2/2, etc. The meaning of 0/0 is that both the father and mother of the individual had the reference types of nucleotide and the individual received those; 0/1, 0/2, 0/3, etc. mean that either the father or mother of the individual had a reference type of nucleotide, but another parent had a GP. The meaning of 1/1, 1/2, 2/2, 1/3, etc. is that both parents did not have reference type of nucleotide.

In the MatrixTable of Chromosome 12, data for all 1,576,756 bases, 0/0 was recoded as GP = 0, and all other combinations were recoded as GP = 1. There were 2,429 types of GPs in the dataset. For all 245,394 participants, the total number of those with no GP was 381,055,807,656 bases (99.67%) and those with a GP was 1,260,533,354 bases (0.33%). The total number of missing data was 17,086,615 bases.

Calculate odds ratios (ORs) of Parkinson's disease (PD) and genetic polymorphisms (GPs) on Jupyter Notebook.

```
# Load PD data from All of Us dataset

import pandas
import os

# This query represents dataset "PD" for domain "condition" and
was generated for All of Us Controlled Tier Dataset v7
dataset_72936839_condition_sql = """
    SELECT
        c_occurrence.person_id,
        c_standard_concept.concept_name as standard_concept_name
    FROM
        ( SELECT
            *
        FROM
            `""" + os.environ["WORKSPACE_CDR"] +
""".condition_occurrence` c_occurrence
        WHERE
            (
                condition_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `""" + os.environ["WORKSPACE_CDR"] +
""".cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `""" + os.environ["WORKSPACE_CDR"] +
""".cb_criteria` cr
                    WHERE
                        concept_id IN (381270)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
                c_occurrence.PERSON_ID IN (SELECT
                    distinct person_id
```

```
                FROM
                    `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 ) )
            )) c_occurrence
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept`
c_standard_concept
            ON c_occurrence.condition_concept_id =
c_standard_concept.concept_id"""

dataset_72936839_condition_df = pandas.read_gbq(
    dataset_72936839_condition_sql,
    dialect="standard",
    use_bqstorage_api=("BIGQUERY_STORAGE_API_ENABLED" in
os.environ),
    progress_bar_type="tqdm_notebook")

dataset_72936839_condition_df.head(5)
```

```
# Import modules

import os

import pyspark
import hail as hl
from hail.plot import output_notebook, show
import pandas as pd

hl.init()
```

```
# Import other modules

import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Dict, Any
import warnings
from statsmodels.stats.multitest import multipletests
import time
from tqdm import tqdm
```

```
# Load MatrixTable file from All of Us dataset

mt_path = "gs://fc-aou-datasets-
controlled/v7/wgs/short_read/snpindel/exome_v7.1/multiMT/hail.mt"
mt = hl.read_matrix_table(mt_path)

mt.describe()
```

```
# Choose Chromosome 12 from the MT data

mt12 = mt.filter_rows(mt.locus.contig == "chr12")

mt_entries2 = mt12.select_entries(mt12.GT)
mt_entries2.show(10)
```

```
# Count types of GPs

gt_counts =
mt_entries2.aggregate_entries(hl.agg.counter(mt_entries2.GT))

for gt, count in gt_counts.items():
    print(f"{gt}: {count}")

---------------------------------------
# Results
# 0/0: 381055807656
# 0/1: 575784144
# 1/1: 335171969
# ...
# 52/95: 19
# 52/96: 1
# 0/98: 3
# None: 17086615
```

```
# Convert GPs data
# wildtype: 0
# polytype: 1
# missing: None

mt12 = mt12.annotate_entries(genotype_num = hl.case()
    .when(mt12.GT.is_diploid() & ((mt12.GT ==
hl.parse_call("0|0")) | (mt12.GT == hl.parse_call("0/0"))), 0)
    .when(mt12.GT.is_diploid(), 1)
    .default(hl.missing(hl.tint32)))  # other: Missing
```

```
# Confirm the converted GPs data

gt_counts2 =
mt12.aggregate_entries(hl.agg.counter(mt12.genotype_num))

for gt, count in gt_counts2.items():
    print(f"{gt}: {count}")

---------------------------
# Result
# 0: 381055807656
# 1: 1260533354
# None: 17086615
```

```
# Make PD DataFrame

dataset_72936839_condition_df.to_csv("pd.csv")
parkinson_df = pd.read_csv('pd.csv', index_col = 1)
parkinson_df.groupby("standard_concept_name").size()
```

```
# According to All of Us regulation, the population having minor
disease (< 20) should be handled appropriately because of the
safety of personal information. Therefore, in this study, those
samples were removed. The variable names "minor PD name1" and
"minor PD name2" below must be replaced to the real name if you
like to use this code.

filtered_parkinson_df =
parkinson_df[~parkinson_df["standard_concept_name"].isin(["minor
PD name1", "minor PD name2"])]
filtered_parkinson_df.groupby("standard_concept_name").size()

-------------------
# Result
# standard_concept_name
# Parkinson's disease    39112
# dtype: int64
```

```
import datetime

# Convert condition_start_datetime column to datetime data
filtered_parkinson_df['condition_start_datetime'] =
pd.to_datetime(filtered_parkinson_df['condition_start_datetime'],
format='mixed')

# Count the number of condition_start_datetime for each person_id
distinct_counts = filtered_parkinson_df.groupby('person_id')
['condition_start_datetime'].nunique()

# Filter person_id by multiple datetime data
persons_with_multiple_dates = distinct_counts[distinct_counts > 1]

# Print results
print(f"Number of people who have multiple datetime:
{len(persons_with_multiple_dates)}")
print("List of person_id who have multiple datetime:")
print(persons_with_multiple_dates.index.tolist())
--------------------------------
# result
Number of people who have multiple datetime: 1118 # Number of
people who have multiple onset date
List of person_id who have multiple datetime: # ID number of those
people
[xxx, xxx, xxx, xxx...
```

```
# The result indicates that some patients go to hospitals and
recorded those dates as "condition_start_datetime".
# Therefore, pick the first date up and choose the first row of
each individual

filtered_parkinson_df = filtered_parkinson_df.sort_values(by=
['person_id', 'condition_start_datetime'], ascending=[True, True])
parkinson_df = filtered_parkinson_df.groupby(level=0).first()

parkinson_df
---------------------------------------
# Result
(skip the df screenshot because it is prohibited)
1422 rows × 4 columns

# According to All of Us Controlled Tier Dataset v7, the PD
population in the dataset is 1422, therefore the number is
confirmed.
```

```
# Recode "parkinson's disease" to "1"

pd_df2 = parkinson_df.rename({'standard_concept_name':
'phenotype'}, axis='columns')
pd_df2 = pd_df2.replace({"phenotype": {"Parkinson's disease": 1}})
```

Simple Logistic Regression for PD and GPs

Power analysis

We used G*Power 3.1.9.7 [2] to analyze statistic power for a simple logistic regression. When the conditions are α = 0.05, 1 - β = 0.80, PD positive is 0.33%, the dependent GP positive is 0.58%, and the total sample size is 245,394; an OR of > 2.57 can be detected statistically.

Therefore, an OR of 2.58 was set as a criterion to extract candidate bases for preliminary analysis.

Results that had p>.050 and OR < 2.58 were excluded. After the procedure, we obtained 3,709 candidate bases.

```
# Convert pandas df to Hail table
pd_df2['s'] = pd_df2.index
pd_df2['s'] = pd_df2['s'].astype("str")

ht_phenotype = hl.Table.from_pandas(pd_df2, key='s')

# Merge tables
filtered_mt =
filtered_mt.annotate_cols(parkinson_status=ht_phenotype[filtered_m
t.s].phenotype)

# Confirm
mt_entries =
filtered_mt.select_entries(filtered_mt.genotype_num).entries().sho
w(10)
```

```
# Fill 0 to PD negative
filtered_mt2 =
filtered_mt.select_entries(filtered_mt.genotype_num)

filtered_mt2 = filtered_mt2.annotate_cols(
    parkinson_status=hl.if_else(
        hl.is_missing(ht_phenotype[filtered_mt2.s].phenotype), 0,
ht_phenotype[filtered_mt2.s].phenotype
    )
)

# Confirm
filtered_mt2.cols().select('parkinson_status').show(20)
```

```
# Collect PD positive/negative
status_counts = (
    filtered_mt3.cols()
    .group_by('parkinson_status')
    .aggregate(count=hl.agg.count())
)

status_counts.show()
---------------------------------------
# Result

parkinson_status        count
int32   int64
0       243972
1       1422
```

```
# Execute logistic regression
gwas_results = hl.logistic_regression_rows(
    test="wald",
    y=filtered_mt2.parkinson_status,
    x=filtered_mt2.genotype_num,
    covariates=[1.0]
)

# OR and 95% CI
gwas_results = gwas_results.annotate(
    odds_ratio=hl.exp(gwas_results.beta),  # OR
    ci_lower_or=hl.exp(gwas_results.beta - 1.96 *
gwas_results.standard_error),  # lower CI
    ci_upper_or=hl.exp(gwas_results.beta + 1.96 *
gwas_results.standard_error)   # upper CI
)

# Print result (--value, OR, CI)
gwas_results.select('p_value', 'odds_ratio', 'ci_lower_or',
'ci_upper_or').show()
```

```
# Save the result as a CSV file
gwas_df = gwas_results.select('p_value', 'odds_ratio',
'ci_lower_or', 'ci_upper_or').to_pandas()
gwas_df.to_csv('odds_ratio_with_95%CI_hailversion.csv')
```

3 **Confirm false discovery rate**

The Benjamini-Hochberg method was used to confirm the false discovery rate. As a result of FDR confirmation (total number of tested: 3,709), all 3,709 results were significant.

```
import pandas as pd
from statsmodels.stats.multitest import multipletests

file_path = 'your_data.xlsx'
data = pd.read_excel(file_path)

# Confirm first 5 lines
print("First 5 lines:")
print(data.head())

p_value_column = 'p_value'
locus_column = 'locus'

# Benjamini-Hochberg
alpha = 0.05  # significant level
reject, pvals_corrected, _, _ =
multipletests(data[p_value_column], alpha=alpha, method='fdr_bh')

data['p_adjusted'] = pvals_corrected
data['significant'] = reject

print("\nResult:")
print(f"Number of significant p-values before correction:
{sum(data[p_value_column] < alpha)}")
print(f"Number of significant p-values after correction:
{sum(reject)}")

significant_results = data[data['significant']]
print(f"\nNumber of significant results:
{len(significant_results)}")

print("\nFirst 5 lines of significant results:")
if len(significant_results) > 0:
    print(significant_results.head())
else:
    print("No significant results found.")

output_file = 'bh_corrected_results.xlsx'
data.to_excel(output_file, index=False)
print(f"\nSaved the result file as {output_file}.")
```

4 **Table 1** ORs of *LRRK2* G2019S and Other GPs With Similar ORs and 95% CI Ranges (OR ≥ 5.00, 95% CI ≤ 10 (n = 245,394, Outcome: PD positive/negative)

	A	B	C	D	E
	Base No. (GRCh38.p14)	OR (95% CI)	Original *p*-value	Adjusted *p*-value	Gene Name
	1860203	5.58 (2.76–11.31)	< .000	< .000	*CACNA2D4*
	4628152	5.43 (2.43–12.11)	< .000	< .000	*AKAP3*
	11869533	5.94 (2.93–12.05)	< .000	< .000	*ETV6*
	13537468	5.25 (2.97–9.27)	< .000	< .000	*GRIN2B*
	30983164	5.53 (2.60–11.76)	< .000	< .000	*TSPAN11*
	38321345	5.77 (2.85–11.69)	< .000	< .000	*ALG10B*
	40340400 (G2019S)	5.46 (2.90–10.27)	< .000	< .000	*LRRK2 (PARK8)*
	48569196	5.28 (2.61–10.70)	< .000	< .000	*LALBA*
	49990203	5.09 (2.26–11.48)	< .000	.002	*RACGAP1*
	52107506	5.33 (2.36–12.02)	< .000	.001	*SMIM41*
	65955867	5.06 (2.24–11.42)	< .000	.002	*HMGA2*
	66254622	5.87 (2.89–11.89)	< .000	< .000	*IRAK3*
	81260048	5.30 (2.35–11.96)	< .000	.001	*ACSS3*

	A	B	C	D	E
	108544453	5.01 (2.48–10.15)	< .000	< .000	*SART3*
	120460300	5.69 (2.67–12.10)	< .000	< .000	*GATC*

This is a part of the preliminary analysis resulting in 3,709 bases. At the main analysis, those bases in Table 1 were compared with the main results using the All of Us CD (n = 245,388). Original p-values were calculated by logistic regression. Adjusted p-values are corrected p-values for detection of false discovery rate (FDR) by using Benjamini-Hochberg method. For FDR confirmation, StatsModels on Python was used.

Table 2 Roles of Genes Which Had Similar ORs and 95% CIs of G2019S

	A	B	C	D
			Association	
	Gene Name	Role of Gene	Neurodegeneration or Brain Disorders	PD
	CACNA2D4	Encodes a protein in the voltage-dependent calcium channel complex [3]	ADHD or bipolar disorder [4]	Unclear
	AKAP3	Encodes functionally related proteins that target protein kinase A to specific locations within cells [5]	AD, seizure, mental retardation and drug addiction [6]	Unclear
	ETV6	Encodes an erythroblast transformation-specific family transcription factor [7]	Associated with adult hematopoietic stem cells [8]	Associated with β-synuclein rearrangement which has a similar structure to α-syn [9]
	GRIN2B	Encodes a member of the N-methyl-D-aspartate receptor family [10]	Intellectual disability, developmental delays, autism spectrum disorder, and AD [11]	May be associated with the phenotype of PD such as how or when PD occurs [12]
	TSPAN11	Contributes determination of bone matrix organization direction [13]	The dysfunction of TSPAN6 is associated with AD [14]	Unclear

	A	B	C	D
	ALG10B	Involves a dolichol-linked oligosaccharide biosynthetic process [15]	Dysfunction exacerbates neurodegeneration [16]	Unclear
	LALBA	Encodes the alpha-lactalbumin protein of milk [17]	Unclear	Unclear
	BACGAP1	Associated with roles of cytokinesis, cell growth, and differentiation [18]	Since the protein coded by RACGAP1 plays a part of roles in apoptosis, it may be connected with neurodegenerative disorders.	Unclear
	SMM41	Not been studied very well.	Unclear	Unclear
	HMGA2	Encodes a protein that functions as an architectural factor of the enhanceosome [19]	In mice experiments, AD mice treated by silencing HMGA2 had improved learning and memory ability, alleviated brain injury, and decreased inflammatory and oxidative stress reactions. [20]	Unclear
	IRAK3	Mutations are associated with a susceptibility to asthma [21]		In mice, IRAK3 deficiency exacerbates dopaminergic neuron damage in PD [22]
	ACSS3	Located in the mitochondrial matrix and is predicted to be involved in the ketone body biosynthetic process [23]	AD [24]	Unclear
	SART3	Encodes an RNA-binding nuclear protein that may contribute to tumor rejection and specific immunotherapy [25]	Bi-allelic variants in SART3 are associated with a syndrome characterized by developmental delay [26]	Unclear
	GATC	May have a role in ATP binding activity	Unclear	Unclear

	A	B	C	D
		and glutaminyl-tRNA synthase activity [27]		

AD: Alzheimer's disease.

Demographics

5 **Load demographics data**

```
import pandas
import os

# This query represents dataset "Demographics_All" for domain
"person" and was generated for All of Us Controlled Tier Dataset
v7
dataset_60024656_person_sql = """
    SELECT
        person.person_id,
        p_gender_concept.concept_name as gender,
        person.birth_datetime as date_of_birth,
        p_race_concept.concept_name as race,
        p_ethnicity_concept.concept_name as ethnicity,
        p_sex_at_birth_concept.concept_name as sex_at_birth
    FROM
        `""" + os.environ["WORKSPACE_CDR"] + """.person` person
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept`
p_gender_concept
            ON person.gender_concept_id =
p_gender_concept.concept_id
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept`
p_race_concept
            ON person.race_concept_id = p_race_concept.concept_id
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept`
p_ethnicity_concept
            ON person.ethnicity_concept_id =
p_ethnicity_concept.concept_id
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept`
p_sex_at_birth_concept
            ON person.sex_at_birth_concept_id =
p_sex_at_birth_concept.concept_id
    WHERE
        person.PERSON_ID IN (SELECT
            distinct person_id
        FROM
            `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` cb_search_person
        WHERE
            cb_search_person.person_id IN (SELECT
                person_id
```

```
            FROM
                `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` p
            WHERE
                has_whole_genome_variant = 1 ) )"""

dataset_60024656_person_df = pandas.read_gbq(
    dataset_60024656_person_sql,
    dialect="standard",
    use_bqstorage_api=("BIGQUERY_STORAGE_API_ENABLED" in
os.environ),
    progress_bar_type="tqdm_notebook")

dataset_60024656_person_df.head(5)
```

```
demo_all_df = dataset_60024656_person_df
```

6 **Recode age**

```
from datetime import datetime


# Age at July 1, 2022
reference_date = datetime(2022, 7, 1).date()

# Calculate age
demo_all_df['age_at_20220701'] =
demo_all_df['date_of_birth'].apply(lambda dob: reference_date.year
- dob.year -
                                        ((reference_date.month,
reference_date.day) < (dob.month, dob.day)))
```

```
def recode_age(age):
    if age < 50:
        return 1
    elif age >= 50:
        return 2
    else:
        return None

demo_all_df['age_group'] =
demo_all_df['age_at_20220701'].apply(recode_age)
```

Recode sex at birth

```
def recode_sex(row):
    if row['sex_at_birth'] == 'Male':
        return 0
    elif row['sex_at_birth'] == 'Female':
        return 1
    return np.nan

demo_all_df['sex_recode'] = demo_all_df.apply(recode_sex, axis=1)
```

Calcium

7 **Import calcium data**

```
import pandas
import os

# This query represents dataset "calcium" for domain "measurement"
and was generated for All of Us Controlled Tier Dataset v7
dataset_95848550_measurement_sql = """
    SELECT
        measurement.person_id,
        m_standard_concept.concept_name as standard_concept_name,
        measurement.measurement_datetime,
        measurement.value_as_number,
        m_unit.concept_name as unit_concept_name,
        measurement.unit_source_value
    FROM
        ( SELECT
            *
        FROM
            `""" + os.environ["WORKSPACE_CDR"] + """.measurement`
measurement
        WHERE
            (
                measurement_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `""" + os.environ["WORKSPACE_CDR"] +
""".cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `""" + os.environ["WORKSPACE_CDR"] +
""".cb_criteria` cr
                    WHERE
                        concept_id IN (3006906)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
```

```
                measurement.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 ) )
            )) measurement
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept`
m_standard_concept
            ON measurement.measurement_concept_id =
m_standard_concept.concept_id
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept` m_unit
            ON measurement.unit_concept_id = m_unit.concept_id"""

dataset_95848550_measurement_df = pandas.read_gbq(
    dataset_95848550_measurement_sql,
    dialect="standard",
    use_bqstorage_api=("BIGQUERY_STORAGE_API_ENABLED" in
os.environ),
    progress_bar_type="tqdm_notebook")

dataset_95848550_measurement_df.head(5)
```

```
calcium_df = dataset_95848550_measurement_df
```

Confirm units of calcium

```
calcium_counts =
calcium_df['unit_concept_name'].value_counts(dropna=False)

# Count unique values
unique_calcium_count =
calcium_df['unit_concept_name'].nunique(dropna=False)

print("Numbers of each values:")
print(calcium_counts)
print(f"\nNumbers of unique values: {unique_calcium_count}")
----------------------------------------
# result
Numbers of each values:
unit_concept_name
milligram per deciliter     2584076
No matching concept          123271
None                           8977
millimole per liter             113
no value                         82
milligram per milliliter         20
milligram per 24 hours            1
Name: count, dtype: int64

Numbers of unique values: 7
```

```
def recode_calcium(row):
    if row['unit_concept_name'] == 'milligram per deciliter':
        return "milligram per deciliter"
    return np.nan

# Add new column
calcium_df['calcium_recode'] = calcium_df.apply(recode_calcium,
axis=1)
```

```
calcium_df2 = calcium_df.dropna(subset=["calcium_recode"])
calcium_df2 = calcium_df2.rename({'value_as_number':
'calcium(mg/dl)'}, axis='columns')
calcium_df2 = calcium_df2.dropna(subset=["calcium(mg/dl)"])
calcium_df2
```

Confirm top 10 values

```
if "calcium(mg/dl)" in calcium_df2.columns:
    top_n = calcium_df2.nlargest(20, "calcium(mg/dl)")

    print(top_n[["calcium(mg/dl)"]])
else:
    print("calcium(mg/dl) column does not exist.")
------------------------------
# there were many 10000000.0s
------------------------------
if "calcium(mg/dl)" in calcium_df2.columns:
    top_n = calcium_df2.nsmallest(20, "calcium(mg/dl)")

    print(top_n[["calcium(mg/dl)"]])
else:
    print("calcium(mg/dl) column does not exist.")
------------------------------
# no minus
```

Delete 10000000.0s (because it can be regarded as missing)

```
calcium_df2 = calcium_df2[calcium_df2["calcium(mg/dl)"] !=
10000000]
```

Calculate mean value for each person

```
calcium_df3 = calcium_df2.groupby("person_id")
['calcium(mg/dl)'].mean().to_frame()
calcium_df3
```

Vitamin D

8 **Load vitamin D data**

```
import pandas
import os

# This query represents dataset "vitamin D" for domain
"measurement" and was generated for All of Us Controlled Tier
Dataset v7
dataset_89441313_measurement_sql = """
    SELECT
        measurement.person_id,
        m_standard_concept.concept_name as standard_concept_name,
        measurement.measurement_datetime,
        measurement.value_as_number,
        m_unit.concept_name as unit_concept_name
    FROM
        ( SELECT
            *
        FROM
            `""" + os.environ["WORKSPACE_CDR"] + """.measurement`
measurement
        WHERE
            (
                measurement_concept_id IN (SELECT
                    DISTINCT c.concept_id
                FROM
                    `""" + os.environ["WORKSPACE_CDR"] +
""".cb_criteria` c
                JOIN
                    (SELECT
                        CAST(cr.id as string) AS id
                    FROM
                        `""" + os.environ["WORKSPACE_CDR"] +
""".cb_criteria` cr
                    WHERE
                        concept_id IN (3020149)
                        AND full_text LIKE '%_rank1]%'      ) a
                        ON (c.path LIKE CONCAT('%.', a.id, '.%')
                        OR c.path LIKE CONCAT('%.', a.id)
                        OR c.path LIKE CONCAT(a.id, '.%')
                        OR c.path = a.id)
                WHERE
                    is_standard = 1
                    AND is_selectable = 1)
            )
            AND (
```

```
                measurement.PERSON_ID IN (SELECT
                    distinct person_id
                FROM
                    `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` cb_search_person
                WHERE
                    cb_search_person.person_id IN (SELECT
                        person_id
                    FROM
                        `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` p
                    WHERE
                        has_whole_genome_variant = 1 ) )
            )) measurement
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept`
m_standard_concept
            ON measurement.measurement_concept_id =
m_standard_concept.concept_id
    LEFT JOIN
        `""" + os.environ["WORKSPACE_CDR"] + """.concept` m_unit
            ON measurement.unit_concept_id = m_unit.concept_id"""

dataset_89441313_measurement_df = pandas.read_gbq(
    dataset_89441313_measurement_sql,
    dialect="standard",
    use_bqstorage_api=("BIGQUERY_STORAGE_API_ENABLED" in
os.environ),
    progress_bar_type="tqdm_notebook")

dataset_89441313_measurement_df.head(5)
```

```
vitaminD_df = dataset_89441313_measurement_df
```

Confirm units of vitamin D

```
vitaminD_counts =
vitaminD_df['unit_concept_name'].value_counts(dropna=False)

# Count unique values
unique_vitaminD_count =
vitaminD_df['unit_concept_name'].nunique(dropna=False)

print("Numbers of each values:")
print(vitaminD_counts)
print(f"\nNumbers of unique values: {unique_vitaminD_count}")
----------------------------------------
# result
Numbers of each values:
unit_concept_name
nanogram per milliliter     153187
milliliter per minute         8054
No matching concept           5576
no value                      4998
picogram per milliliter        414
None                           125
ng/mL                           97
milligram per milliliter         1
millimole per liter              1
nanogram per deciliter           1
Name: count, dtype: int64

Numbers of unique values: 10
```

```
vitaminD_df[vitaminD_df['unit_concept_name'] == 'nanogram per
milliliter'].head(5)
-----------------
vitaminD_df[vitaminD_df['unit_concept_name'] == 'picogram per
milliliter'].head(5)
-----------------
# picogram values did not seem to be real picogram but nanogram,
but I was not sure.
# Decided to delete picogram
```

Recode units

```
def recode_vitaminD(row):
    if row['unit_concept_name'] == 'nanogram per milliliter':
        return "nanogram per milliliter"
    elif row['unit_concept_name'] == 'ng/mL':
        return "nanogram per milliliter"
    return np.nan

# Add new column
vitaminD_df['vitaminD_recode'] =
vitaminD_df.apply(recode_vitaminD, axis=1)
vitaminD_df
```

Delete invalid units and values

```
vitaminD_df2 = vitaminD_df.dropna(subset=["vitaminD_recode"])
vitaminD_df2 = vitaminD_df2.rename({'value_as_number':
'vitaminD(ng/ml)'}, axis='columns')
vitaminD_df2 = vitaminD_df2.dropna(subset=["vitaminD(ng/ml)"])
vitaminD_df2
```

Confirm top 10 values

```
if "vitaminD(ng/ml)" in vitaminD_df2.columns:
    top_n = vitaminD_df2.nlargest(20, "vitaminD(ng/ml)")

    print(top_n[["vitaminD(ng/ml)"]])
else:
    print("vitaminD(ng/ml) column does not exist.")
--------------------------
# no strange values
--------------------------
if "vitaminD(ng/ml)" in vitaminD_df2.columns:
    top_n = vitaminD_df2.nsmallest(20, "vitaminD(ng/ml)")

    print(top_n[["vitaminD(ng/ml)"]])
else:
    print("vitaminD(ng/ml) column does not exist.")
-------------------------
# no minus
```

Calculate mean values for each person

```
vitaminD_df3 = vitaminD_df2.groupby("person_id")
['vitaminD(ng/ml)'].mean().to_frame()
vitaminD_df3
```

Alcohol

9 **Load alcohol data**

```
import pandas
import os

# This query represents dataset "alcohol" for domain "survey" and
was generated for All of Us Controlled Tier Dataset v7
dataset_68779372_survey_sql = """
    SELECT
        answer.person_id,
        answer.survey_datetime,
        answer.question,
        answer.answer
    FROM
        `""" + os.environ["WORKSPACE_CDR"] + """.ds_survey` answer
    WHERE
        (
            question_concept_id IN (1586201)
        )
        AND (
            answer.PERSON_ID IN (SELECT
                distinct person_id
            FROM
                `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` cb_search_person
            WHERE
                cb_search_person.person_id IN (SELECT
                    person_id
                FROM
                    `""" + os.environ["WORKSPACE_CDR"] +
""".cb_search_person` p
                WHERE
                    has_whole_genome_variant = 1 ) )
        )"""

dataset_68779372_survey_df = pandas.read_gbq(
    dataset_68779372_survey_sql,
    dialect="standard",
    use_bqstorage_api=("BIGQUERY_STORAGE_API_ENABLED" in
os.environ),
    progress_bar_type="tqdm_notebook")

dataset_68779372_survey_df.head(5)
```

```
alcohol_df = dataset_68779372_survey_df
```

Confirm units of alcohol

```
# Count unique values
unique_alcohol_count = alcohol_df['answer'].nunique(dropna=False)

print("Numbers of each values:")
print(alcohol_counts)
print(f"\nNumbers of unique values: {unique_alcohol_count}")
----------------------------------
# result
Numbers of each values:
answer
Drink Frequency Past Year: Monthly Or Less       71638
Drink Frequency Past Year: 2 to 4 Per Month      44989
Drink Frequency Past Year: Never                 37764
Drink Frequency Past Year: 2 to 3 Per Week       30259
Drink Frequency Past Year: 4 or More Per Week    25952
PMI: Prefer Not To Answer                         2501
PMI: Skip                                         1879
Name: count, dtype: int64

Numbers of unique values: 7
```

Recode alcohol

```
recoding_dict = {
    'Drink Frequency Past Year: Never': 0,
    'Drink Frequency Past Year: Monthly Or Less': 1,
    'Drink Frequency Past Year: 2 to 4 Per Month': 2,
    'Drink Frequency Past Year: 2 to 3 Per Week': 3,
    'Drink Frequency Past Year: 4 or More Per Week': 4,
    'PMI: Prefer Not To Answer': None,
    'PMI: Skip': None,
}

alcohol_df['alcohol_recode'] =
alcohol_df['answer'].replace(recoding_dict)
```

Count values

```
alcohol_counts =
alcohol_df['alcohol_recode'].value_counts(dropna=False)

# Count unique values
unique_alcohol_count =
alcohol_df['alcohol_recode'].nunique(dropna=False)

print("Numbers of each values:")
print(alcohol_counts)
print(f"\nNumbers of unique values: {unique_alcohol_count}")
------------------------------
# result
Numbers of each values:
alcohol_recode
1.0    71638
2.0    44989
0.0    37764
3.0    30259
4.0    25952
NaN     4380
Name: count, dtype: int64

Numbers of unique values: 6
```

Confirm if one person has multiple values

```
col_values = alcohol_df["person_id"]
has_duplicates = col_values.duplicated().any()  # check if one
person has multiple values

if has_duplicates:
    print(f"Column '{'person_id'}' has duplicated value(s).:
{col_values[col_values.duplicated()].unique()}")
else:
    print(f"Column '{'person_id'}' does not have duplicated
value(s)")
-------------------
# Column 'person_id' does not have duplicated value(s)
```

```
alcohol_df2 = alcohol_df[["person_id", "answer",
"alcohol_recode"]]
alcohol_df2
```

Merge dfs of age, sex, calcium, Vitamin D, and alcohol

10

```
demo_all_df = demo_all_df.set_index('person_id')

pd_df2 = pd_df2.rename(columns={'parkinson_status': 'PD'}
demo_all_df = demo_all_df.sort_index()
demo_all_merge_df = demo_all_df.join(pd_df2, how='outer')

calcium_df3 = calcium_df3.set_index("person_id")
demo_all_merge_df2 = demo_all_merge_df.join(calcium_df3,
how='outer')

vitaminD_df3 = vitaminD_df3.set_index("person_id")
demo_all_merge_df2 = demo_all_merge_df.2join(vitaminD_df3,
how='outer')

alcohol_df3 = alcohol_df2.set_index("person_id")
demo_all_merge_df2 = demo_all_merge_df2.join(alcohol_df3,
how='outer')

demo_all_merge_df3 = demo_all_merge_df2.drop(["gender",
"date_of_birth", "race", "ethnicity", "sex_at_birth",
"condition_start_datetime", "condition_end_datetime", "answer",
"calcium(mg/dl)", "vitaminD(ng/ml)", "age_at_20220701"], axis=1)
```

```
logistic_regression_without_gwas_df = demo_all_merge_df3

# If there are still unnecessary columns, drop them
logistic_regression_without_gwas_df =
logistic_regression_without_gwas_df.drop(["xxxxx"], axis=1)

logistic_regression_without_gwas_df =
logistic_regression_without_gwas_df.dropna()

logistic_regression_without_gwas_df['s'] =
logistic_regression_without_gwas_df.index
logistic_regression_without_gwas_df['s'] =
logistic_regression_without_gwas_df['s'].astype("str")
```

Main analysis (Logistic regression (PD = age * sex * calcium * Vitamin D * alcohol)

11 **Count sample numbers of MT**

```
sample_ids = filtered_mt.col_key.collect()
sample_ids_list2 = [x.s for x in sample_ids]
len(sample_ids_list2)
-----------------------------
# 245394
```

Count sample numbers of demographic data

```
df_ids_list = demo_all_merge_df3.index.tolist()
len(df_ids_list)
-----------------------------
# 245388
```

Drop samples from MT files which were not used in CT data

```
sample_ids_list2_int = [int(x) for x in sample_ids_list2]
diff_only_in_samples = list(set(sample_ids_list2_int) -
set(df_ids_list))
diff_only_in_samples
```

```
# Convert keys of diff_only_in_samples from numbers to strings (mt
use strings for s (= sample id))
diff_samples_str = set(str(x) for x in diff_only_in_samples)

# Filtering
filtered_mt2 =
filtered_mt.filter_cols(~hl.set(diff_samples_str).contains(filtere
d_mt.s))

# Confirm
print(f"Previous numbers of samples: {filtered_mt.count_cols()}")
print(f"Current numbers of samples: {filtered_mt2.count_cols()}")
----------------------------------------
# Previous numbers of samples: 245394
# Current numbers of samples: 245388
```

```
sample_ids = filtered_mt2.col_key.collect()
sample_ids_list2 = [x.s for x in sample_ids]
df_ids_list = logistic_regression_without_gwas_df.index.tolist()

sample_ids_list2_int = [int(x) for x in sample_ids_list2]

diff_only_in_samples = list(set(sample_ids_list2_int) -
set(df_ids_list))
diff_samples_str = set(str(x) for x in diff_only_in_samples)

# Filtering
filtered_mt2 =
filtered_mt2.filter_cols(~hl.set(diff_samples_str).contains(filter
ed_mt.s))

# Confirm sample size
print(f"Sample size after filtering: {filtered_mt2.count_cols()}")
```

12 **Logistic regression (using Hail function, no dummy variables)**

```
# Reset index and obtain it as a column (because it will be used
as a key)
logistic_regression_without_gwas_df_reset =
logistic_regression_without_gwas_df.reset_index()
ht =
hl.Table.from_pandas(logistic_regression_without_gwas_df_reset,
key='person_id')  # 'index' should be the original index's name of
index

# Convert key type as strings
logistic_regression_without_gwas_df_reset['person_id'] =
logistic_regression_without_gwas_df_reset['person_id'].astype(str)
ht =
hl.Table.from_pandas(logistic_regression_without_gwas_df_reset,
key='person_id')

# Redifine the key of MatrixTable
annotated_mt = filtered_mt2.key_cols_by()  # Release key
annotated_mt = annotated_mt.annotate_cols(**ht[annotated_mt.s])  #
Add annotations
annotated_mt = annotated_mt.key_cols_by('s')  # Redefine key
```

Execute logistic regression (Hail version, no dummy variables)

```
gwas_results = hl.logistic_regression_rows(
    test="wald",
    y=annotated_mt.PD,
    x=annotated_mt.genotype_num,
    # covariates=[1.0, filtered_mt.factorB, filtered_mt.factorC]
    #covariates=[1.0]
    covariates=[1.0, annotated_mt.age_group,
annotated_mt.sex_recode, annotated_mt.alcohol_recode,
annotated_mt.calcium_recode, annotated_mt.vitaminD_recode]
)

# OR and 95% CI
gwas_results = gwas_results.annotate(
    odds_ratio=hl.exp(gwas_results.beta),  # OR
    ci_lower_or=hl.exp(gwas_results.beta - 1.96 *
gwas_results.standard_error),  # lower CI
    ci_upper_or=hl.exp(gwas_results.beta + 1.96 *
gwas_results.standard_error)   # upper CI
)

# Display results
gwas_results.select('p_value', 'odds_ratio', 'ci_lower_or',
'ci_upper_or').show()
```

```
# Save the results as a csv file
gwas_df = gwas_results.select('p_value', 'odds_ratio',
'ci_lower_or', 'ci_upper_or').to_pandas()
gwas_df.to_csv('odds_ratio_with_95%CI_hailversion.csv')
```

13 **Logistic regression (using StatModels version with dummy variables)**

```
df = logistic_regression_without_gwas_df
df = df.astype('float64')

# Set the references for each variable
#df['FactorA'] = pd.Categorical(df['FactorA'], categories=[3, 1,
2, 4], ordered=True) # 3 is reference
df['age_group'] = pd.Categorical(df['age_group'], categories=[1,
2], ordered=True)
df['sex_recode'] = pd.Categorical(df['sex_recode'], categories=[1,
0], ordered=True)
df['alcohol_recode'] = pd.Categorical(df['alcohol_recode'],
categories=[0, 1, 2, 3, 4], ordered=True)
df['calcium_recode'] = pd.Categorical(df['calcium_recode'],
categories=[2, 1, 3], ordered=True)
df['vitaminD_recode'] = pd.Categorical(df['vitaminD_recode'],
categories=[2, 1, 3, 4], ordered=True)

# Making dummy variables
X = pd.get_dummies(
    df[['age_group', 'sex_recode', 'alcohol_recode',
'calcium_recode', 'vitaminD_recode']],
    drop_first=True
).copy()

X = X.astype(float)
y = df['PD']
X = sm.add_constant(X)

model = sm.Logit(y, X)

result = model.fit()

# OR and 95% CI
odds_ratios = np.exp(result.params)
conf = np.exp(result.conf_int())

# Result
odds_ratio_df = pd.DataFrame({
    'Odds Ratio': odds_ratios,
    '95% CI Lower': conf[0],
    '95% CI Upper': conf[1],
    'p-value': result.pvalues
})
```

```
print("\nOdds Ratios with 95% Confidence Intervals:")
print(odds_ratio_df)
```

Main analysis (Logistic regression (PD = GP * age * sex * calcium * Vitamin D * alcohol)

14 **Chromosome 12 data is too large to convert from MT file into Pandas df, therefore choosing specific base(s) is necessary**

```
# Choose specific base(s)
positions = [53711362, 31281818, 101921705, 47968795, 112791809]
filtered_mt_selected = filtered_mt.filter_rows(
    hl.set(positions).contains(filtered_mt.locus.position)
)
```

```
# Choose variants and phenotype_num and convert it to pandas df
selected_df = (filtered_mt_selected.entries()
               .key_by()
               .select('s', 'genotype_num', 'locus')
               .to_pandas())

selected_df_pivot = selected_df.pivot(index='s', columns='locus',
values='genotype_num')

selected_df_pivot = selected_df.pivot(index='s', columns='locus',
values='genotype_num')
selected_df_pivot.columns = selected_df_pivot.columns.map(lambda
x: x.position)

selected_df_pivot.head()
```

```
logistic_regression_without_gwas_df =
logistic_regression_without_gwas_df.drop("s", axis=1)

selected_df_pivot2 = selected_df_pivot.rename(columns={53711362:
'G53711362',
                                                      31281818:
'G31281818',
                                                      101921705:
'G101921705',
                                                      47968795:
'G47968795',
                                                      112791809:
'G112791809'})

selected_df_pivot2.index =
selected_df_pivot2.index.astype('object')
logistic_regression_without_gwas_df.index =
logistic_regression_without_gwas_df.index.astype('object')
```

```
selected_df_pivot2.sort_index(inplace=True)
logistic_regression_without_gwas_df.sort_index(inplace=True)
logistic_regression_without_gwas_df.index =
logistic_regression_without_gwas_df.index.astype(str)

# Confirm index match
print("df1 index type:", selected_df_pivot2.index.dtype)
print("df2 index type:",
logistic_regression_without_gwas_df.index.dtype)
print("\ndf1's first 5 indexes:", selected_df_pivot2.index[:5])
print("df2's first 5 indexes:",
logistic_regression_without_gwas_df.index[:5])

print("\nDid two indexes match?:", all(selected_df_pivot2.index ==
logistic_regression_without_gwas_df.index))

--------------------------------

df1 index type: object
df2 index type: object

df1's first 5 indexes: Index([("omit")], dtype='object',
name='person_id')
df2's first 5 indexes: Index([(omit)], dtype='object',
name='person_id')

Did two indexes match?: True
```

```
# Merge MT and CT as one pandas df
specific_analysis_df =
logistic_regression_without_gwas_df.merge(selected_df_pivot2,
                     left_index=True,
                     right_index=True,
                     how='outer')
```

```
df = specific_analysis_df
df = df.astype('float64')

# Set the references for each variable
#df['FactorA'] = pd.Categorical(df['FactorA'], categories=[3, 1,
2, 4], ordered=True) # 3 is reference
df['Gxxxxxxx'] = pd.Categorical(df['Gxxxxxxx'], categories=[0, 1],
ordered=True) # GP
df['age_group'] = pd.Categorical(df['age_group'], categories=[1,
2], ordered=True)
df['sex_recode'] = pd.Categorical(df['sex_recode'], categories=[1,
0], ordered=True)
df['alcohol_recode'] = pd.Categorical(df['alcohol_recode'],
categories=[0, 1, 2, 3, 4], ordered=True)
df['calcium_recode'] = pd.Categorical(df['calcium_recode'],
categories=[2, 1, 3], ordered=True)
df['vitaminD_recode'] = pd.Categorical(df['vitaminD_recode'],
categories=[2, 1, 3, 4], ordered=True)

# Making dummy variables
X = pd.get_dummies(
    df[['age_group', 'sex_recode', 'alcohol_recode',
'calcium_recode', 'vitaminD_recode']],
    drop_first=True
).copy()

X = X.astype(float)
y = df['PD']
X = sm.add_constant(X)

model = sm.Logit(y, X)

result = model.fit()

# OR and 95% CI
odds_ratios = np.exp(result.params)
conf = np.exp(result.conf_int())

# Result
odds_ratio_df = pd.DataFrame({
    'Odds Ratio': odds_ratios,
    '95% CI Lower': conf[0],
    '95% CI Upper': conf[1],
    'p-value': result.pvalues
```

```
})

print("\nOdds Ratios with 95% Confidence Intervals:")
print(odds_ratio_df)
```

References

15

1. National Library of Medicine. Chromosome Map. In: Genes and Disease [Internet]. Maryland, United States: National Center for Biotechnology Information; 1998. **https://www.ncbi.nlm.nih.gov/books/NBK22266/**
2. Buchner A, Erdfelder E, Faul F, Lang A-G. G*Power. https://www.psychologie.hhu.de/arbeitsgruppen/allgemeine-psychologie-und-arbeitspsychologie/gpower. Accessed 3 May 2025.
3. National Library of Medicine. CACNA2D4 calcium voltage-gated channel auxiliary subunit alpha2delta 4 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/93589
4. Ablinger C, Geisler SM, Stanika RI, Klein CT, Obermair GJ. Neuronal α2δ proteins and brain disorders. Pflügers Archiv-European Journal of Physiology. 2020;472:845–63. https://doi.org/10.1007/s00424-020-02420-2
5. National Library of Medicine. AKAP3 A-kinase anchoring protein 3 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/10566
6. Tröger J, Moutty MC, Skroblin P, Klussmann E. A-kinase anchoring proteins as potential drug targets. British journal of pharmacology. 2012;166:420–33. https://doi.org/10.1111/j.1476-5381.2011.01796.x
7. National Library of Medicine. ETV6 ETS variant transcription factor 6 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/2120
8. Monovich AC, Gurumurthy A, Ryan RJH. The Diverse Roles of ETV6 Alterations in B-Lymphoblastic Leukemia and Other Hematopoietic Cancers. Transcription factors in blood cell development. 2024;:291–320. https://doi.org/10.1007/978-3-031-62731-6_13
9. Xiao P, Chen N, Shao T, Bian X, Miao J, Zheng J, et al. Intragenic β-synuclein rearrangements in malignancy. Frontiers in Oncology. 2023;13:1167143. https://doi.org/10.3389/fonc.2023.1167143
10. National Library of Medicine. GRIN2B glutamate ionotropic receptor NMDA type subunit 2B [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/2904
11. Myers SJ, Yuan H, Kang J-Q, Tan FCK, Traynelis SF, Low C-M. Distinct roles of GRIN2A and GRIN2B variants in neurological conditions. F1000Research. 2019;8:F1000-Faculty. https://doi.org/10.12688/f1000research.18949.1
12. Hassan A, Heckman MG, Ahlskog JE, Wszolek ZK, Serie DJ, Uitti RJ, et al. Association of Parkinson disease age of onset with DRD2, DRD3 and GRIN2B polymorphisms.

Parkinsonism & related disorders. 2016;22:102–5. https://doi.org/10.1016/j.parkreldis.2015.11.016
13. Becic A, Leifeld J, Shaukat J, Hollmann M. Tetraspanins as potential modulators of glutamatergic synaptic function. Frontiers in Molecular Neuroscience. 2022;14:801882. https://doi.org/10.3389/fnmol.2021.801882
14. Perot BP, Ménager MM. Tetraspanin 7 and its closest paralog tetraspanin 6: membrane organizers with key functions in brain development, viral infection, innate immunity, diabetes and cancer. Medical Microbiology and Immunology. 2020;209:427–36. https://doi.org/10.1007/s00430-020-00681-3
15. National Library of Medicine. ALG10B ALG10 alpha-1,2-glucosyltransferase B [Homo sapiens (human)]. 2025 https://www.ncbi.nlm.nih.gov/gene/144245
16. Cruchaga C, Bradley J, Western D, Wang C, Da Fonseca EL, Neupane A, et al. Novel early-onset Alzheimer-associated genes influence risk through dysregulation of glutamate, immune activation, and intracell signaling pathways. Research square. 2024;:rs-3. https://doi.org/10.21203/rs.3.rs-4480585/v1
17. National Library of Medicine. LALBA lactalbumin alpha [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/3906
18. National Library of Medicine. RACGAP1 Rac GTPase activating protein 1 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/29127
19. National Library of Medicine. HMGA2 high mobility group AT-hook 2 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/8091
20. Liu X, Wang H, Bei J, Zhao J, Jiang G, Liu X. The protective role of miR-132 targeting HMGA2 through the PI3K/AKT pathway in mice with Alzheimer's disease. American Journal of Translational Research. 2021;13:4632. **https://pmc.ncbi.nlm.nih.gov/articles/PMC8205745/**
21. National Library of Medicine. IRAK3 interleukin 1 receptor associated kinase 3 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/11213
22. Deng Y, Liao Y, Huang P, Yao Y, Liu W, Gu Y, et al. IRAK-M deficiency exacerbates dopaminergic neuronal damage in a mouse model of sub-acute Parkinson's disease. Neuroreport. 2023;34:463–70. https://doi.org/10.1097/WNR.0000000000001913
23. National Library of Medicine. ACSS3 acyl-CoA synthetase short chain family member 3 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/79611
24. Sun Y, Jiang M, Long X, Miao Y, Du H, Zhang T, et al. Transcriptomic analysis of lipid metabolism genes in Alzheimer's disease: highlighting pathological outcomes and compartmentalized immune status. Journal of Molecular Neuroscience. 2024;74:55. https://doi.org/10.1007/s12031-024-02225-3
25. National Library of Medicine. SART3 spliceosome associated factor 3, U4/U6 recycling protein [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/9733
26. Ayers KL, Eggers S, Rollo BN, Smith KR, Davidson NM, Siddall NA, et al. Variants in SART3 cause a spliceosomopathy characterised by failure of testis development and neuronal defects. nature communications. 2023;14:3403. https://doi.org/10.1038/s41467-023-39040-0

27. National Library of Medicine. GATC glutamyl-tRNA amidotransferase subunit C [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/283459

Protocol references

1. National Library of Medicine. Chromosome Map. In: Genes and Disease [Internet]. Maryland, United States: National Center for Biotechnology Information; 1998. **https://www.ncbi.nlm.nih.gov/books/NBK22266/**
2. Buchner A, Erdfelder E, Faul F, Lang A-G. G*Power. https://www.psychologie.hhu.de/arbeitsgruppen/allgemeine-psychologie-und-arbeitspsychologie/gpower. Accessed 3 May 2025.
3. National Library of Medicine. CACNA2D4 calcium voltage-gated channel auxiliary subunit alpha2delta 4 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/93589
4. Ablinger C, Geisler SM, Stanika RI, Klein CT, Obermair GJ. Neuronal α2δ proteins and brain disorders. Pflügers Archiv-European Journal of Physiology. 2020;472:845–63. https://doi.org/10.1007/s00424-020-02420-2
5. National Library of Medicine. AKAP3 A-kinase anchoring protein 3 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/10566
6. Tröger J, Moutty MC, Skroblin P, Klussmann E. A-kinase anchoring proteins as potential drug targets. British journal of pharmacology. 2012;166:420–33. https://doi.org/10.1111/j.1476-5381.2011.01796.x
7. National Library of Medicine. ETV6 ETS variant transcription factor 6 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/2120
8. Monovich AC, Gurumurthy A, Ryan RJH. The Diverse Roles of ETV6 Alterations in B-Lymphoblastic Leukemia and Other Hematopoietic Cancers. Transcription factors in blood cell development. 2024;:291–320. https://doi.org/10.1007/978-3-031-62731-6_13
9. Xiao P, Chen N, Shao T, Bian X, Miao J, Zheng J, et al. Intragenic β-synuclein rearrangements in malignancy. Frontiers in Oncology. 2023;13:1167143. https://doi.org/10.3389/fonc.2023.1167143
10. National Library of Medicine. GRIN2B glutamate ionotropic receptor NMDA type subunit 2B [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/2904
11. Myers SJ, Yuan H, Kang J-Q, Tan FCK, Traynelis SF, Low C-M. Distinct roles of GRIN2A and GRIN2B variants in neurological conditions. F1000Research. 2019;8:F1000-Faculty. https://doi.org/10.12688/f1000research.18949.1
12. Hassan A, Heckman MG, Ahlskog JE, Wszolek ZK, Serie DJ, Uitti RJ, et al. Association of Parkinson disease age of onset with DRD2, DRD3 and GRIN2B polymorphisms. Parkinsonism & related disorders. 2016;22:102–5. https://doi.org/10.1016/j.parkreldis.2015.11.016
13. Becic A, Leifeld J, Shaukat J, Hollmann M. Tetraspanins as potential modulators of glutamatergic synaptic function. Frontiers in Molecular Neuroscience. 2022;14:801882. https://doi.org/10.3389/fnmol.2021.801882
14. Perot BP, Ménager MM. Tetraspanin 7 and its closest paralog tetraspanin 6: membrane organizers with key functions in brain development, viral infection, innate immunity, diabetes and cancer. Medical Microbiology and Immunology. 2020;209:427–36. https://doi.org/10.1007/s00430-020-00681-3
15. National Library of Medicine. ALG10B ALG10 alpha-1,2-glucosyltransferase B [Homo sapiens (human)]. 2025 https://www.ncbi.nlm.nih.gov/gene/144245
16. Cruchaga C, Bradley J, Western D, Wang C, Da Fonseca EL, Neupane A, et al. Novel early-onset Alzheimer-associated genes influence risk through dysregulation of glutamate, immune activation, and intracell signaling pathways. Research square. 2024;:rs-3. https://doi.org/10.21203/rs.3.rs-4480585/v1
17. National Library of Medicine. LALBA lactalbumin alpha [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/3906
18. National Library of Medicine. RACGAP1 Rac GTPase activating protein 1 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/29127

19. National Library of Medicine. HMGA2 high mobility group AT-hook 2 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/8091
20. Liu X, Wang H, Bei J, Zhao J, Jiang G, Liu X. The protective role of miR-132 targeting HMGA2 through the PI3K/AKT pathway in mice with Alzheimer's disease. American Journal of Translational Research. 2021;13:4632. **https://pmc.ncbi.nlm.nih.gov/articles/PMC8205745/**
21. National Library of Medicine. IRAK3 interleukin 1 receptor associated kinase 3 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/11213
22. Deng Y, Liao Y, Huang P, Yao Y, Liu W, Gu Y, et al. IRAK-M deficiency exacerbates dopaminergic neuronal damage in a mouse model of sub-acute Parkinson's disease. Neuroreport. 2023;34:463–70. https://doi.org/10.1097/WNR.0000000000001913
23. National Library of Medicine. ACSS3 acyl-CoA synthetase short chain family member 3 [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/79611
24. Sun Y, Jiang M, Long X, Miao Y, Du H, Zhang T, et al. Transcriptomic analysis of lipid metabolism genes in Alzheimer's disease: highlighting pathological outcomes and compartmentalized immune status. Journal of Molecular Neuroscience. 2024;74:55. https://doi.org/10.1007/s12031-024-02225-3
25. National Library of Medicine. SART3 spliceosome associated factor 3, U4/U6 recycling protein [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/9733
26. Ayers KL, Eggers S, Rollo BN, Smith KR, Davidson NM, Siddall NA, et al. Variants in SART3 cause a spliceosomopathy characterised by failure of testis development and neuronal defects. nature communications. 2023;14:3403. https://doi.org/10.1038/s41467-023-39040-0
27. National Library of Medicine. GATC glutamyl-tRNA amidotransferase subunit C [Homo sapiens (human)]. 2025. https://www.ncbi.nlm.nih.gov/gene/283459

Acknowledgements

We gratefully acknowledge *All of Us* participants for their contributions, without whom this research would not have been possible. We also thank the National Institutes of Health's *All of Us* Research Program for making available the participant data examined in this study.